Complexity Results and Practical Algorithms for Logics in Knowledge Representation

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

von
Diplom-Informatiker Stephan Tobies
aus Essen

Berichter:
Universitätsprofessor Dr. Franz Baader
Universitätsprofessor Dr. Erich Grädel

Tag der mündlichen Prüfung: 31.05.2001

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek
online verfügbar.

For Antje, Johanna, and Annika.

I would not have been able to write this thesis without the help of my family. I would like to thank my wife Antje, who gave me love and support; my parents, on whose assistance I could always count throughout the years; my sister, who didn't stop nagging me about finishing this thesis; Marianne, who shared Antje's burden of taking care of our twin daughters Johanna and Annika while I was working overtime. And I would like to thank you, Annika and Johanna, for being my favourite distraction. Without you, writing this thesis would have been much less interesting and challenging.

I am also greatly indebted to my colleagues who have contributed to this thesis in numerous ways. Many of the discussion we had have found there way into this thesis. I would like to thank Franz Baader, my thesis advisor, whose encouraging "haeh?" forced me to write better papers; Ulrike Sattler and Carsten Lutz, who never grew too annoyed with my sloppiness; and all my colleagues, who happily endured the constant interruptions when I needed to discuss some new ideas. Thanks for the good time I had while working with you.

Contents

Chapter 1

Introduction

Description Logics (DLs) are used in knowledge-based systems to represent and reason about terminological knowledge of the application domain in a semantically well-defined manner. They allow the definition of complex concepts (i.e., classes, unary predicates) and roles (relations, binary predicates) to be built from atomic ones by the application of a given set of constructors. A DL system allows concept descriptions to be interrelated and implicit knowledge can be derived from the explicitly represented knowledge using inference services.

This thesis is concerned with issues of reasoning with DLs and Guarded Logics, which generalise many of the good properties of DLs to a large fragment of first-order predicate logic. We study inference algorithms for these logics, both from the viewpoint of (worst-case) complexity of the algorithms and their usability in system implementations. This chapter gives a brief introduction to DL systems and reasoning in DLs. After that, we introduce the specific aspects of DLs we will be dealing with and motivate their use in knowledge representation. We also introduce Guarded Logics and describe why they are interesting from the viewpoint of DLs. We finish with an overview of the structure of this thesis and the results we establish.

1.1 Description Logic Systems

Description Logics (DLs) are logical formalisms for representing and reasoning about conceptual and terminological knowledge of a problem domain. They have evolved from the knowledge representation formalism of Quillian's *semantic networks* (1967) and Minsky's *frame systems* (1981), as an answer to the poorly defined semantics of these formalisms (Woods, 1975). Indeed, one of the distinguishing features of DLs is the well-defined—usually Tarski-style, extensional—semantics. DLs are based on the notions of concepts (classes, unary predicates) and roles (binary relations) and are mainly characterized by a set of operators that allow complex concepts and roles to be built from atomic ones. As an example consider the following concept that describes fathers having a daughter whose children are all rich, using concept conjunction ($\sqcap$), and universal ($\forall$) and existential

($\exists$) restriction over the role `has_child`:

$$\mathtt{Male} \sqcap \exists \mathtt{has\_child}.(\mathtt{Female} \sqcap \forall \mathtt{has\_child}.\mathtt{Rich})$$

DL systems (Woods & Schmolze, 1992) employ DLs to represent knowledge of an application domain and offer inference services based on the formal semantics of the DL to derive *implicit* knowledge from the *explicitly* represented facts.

In many DL systems, one can find the following components:

- a *terminological component* or *TBox*, which uses the DL to formalise the terminological domain knowledge. Usually, such a TBox at least allows to introduce abbreviations for complex concepts but also more general statements are available in some systems. As an example consider the following TBox that formalizes some knowledge about relationships of people, where $\bot$ denotes the concept with empty extension (the empty class):

$$\begin{aligned}
\mathtt{Parent} &= \mathtt{Human} \sqcap \exists \mathtt{has_child}.\mathtt{Human} \sqcap \forall \mathtt{has_child}.\mathtt{Human} \\
\mathtt{Husband} &= \mathtt{Male} \sqcap \exists \mathtt{married_to}.\mathtt{Human} \\
\mathtt{Human} &= \mathtt{Male} \sqcup \mathtt{Female} \\
\mathtt{Husband} &\sqsubseteq \forall \mathtt{married_to}.\mathtt{Female} \\
\mathtt{Male} \sqcap \mathtt{Female} &= \bot
\end{aligned}$$

  The first three statements introduce `Parent`, `Husband`, and `Male` as abbreviations of more complex concepts. The fourth statement additionally requires that instances of `Husband` must satisfy $\forall$`married_to.Female`, i.e., that a man, if married, must be married to a woman. Finally, the last statement expresses that the concepts `Male` and `Female` must be disjoint as their intersection is defined to be empty.

- an *assertional component* or *ABox*, which formalizes (parts of) a concrete situation involving certain individuals. A partial description of a concrete family, e.g., might look as this:

$$\begin{aligned}
\mathtt{MARY} &: \mathtt{Female} \sqcap \mathtt{Parent} \\
\mathtt{PETER} &: \mathtt{Husband} \\
(\mathtt{MARY}, \mathtt{PETER}) &: \mathtt{has_child}
\end{aligned}$$

  Note, that it is allowed to refer to concepts mentioned in the TBox.

- an *inference engine*, which allows implicit knowledge to be derived from what has been explicitly stated. One typical inference service is the calculation of the *subsumption hierarchy*, i.e., the arrangement of the concepts that occur in the TBox into a quasi-order according to their specialisation/generalisation relationship. In our example, this service could deduce that both `Male` and `Female` are a specialisation of (are subsumed by) `Human`. Another example of an inference service is *instance* *checking*, i.e., determining, whether an individual of the ABox is an instance of a certain concept. In our example, one can derive that MARY has a daughter in law (i.e., is an instance of $\exists\texttt{has\_child}.\exists\texttt{married\_to}.\texttt{Female}$) and is an instance of $\neg\texttt{Husband}$ because the TBox axiom $\texttt{Male} \sqcap \texttt{Female} = \bot$ require Male and Female to be disjoint. We do not make a *closed world assumption*, i.e., assertions not present in the ABox are not assumed to be false by default. This makes it impossible to infer whether PETER is an instance of Parent or not because the ABox does not contain information that supports or circumstantiates this.

KL-ONE (Brachman & Schmolze, 1985) is usually considered to be the first DL system. Its representation formalism possesses a formal semantics and the system allows for the specification of both terminological and assertional knowledge. The inference services provided by KL-ONE include calculation of the subsumption hierarchy and instance checking. Subsequently, a number of systems has been developed that followed the general layout of KL-ONE.

1.2 Reasoning in Description Logics

To be useful for applications, a DL system must at least satisfy the following three criteria: the implemented DL must be capable of capturing an interesting proportion of the domain knowledge, the system must answer queries in a timely manner, and the inferences computed by the systems should be accurate. At least, the inferences should be *sound*, so that every drawn conclusion is correct. It is also desirable to have *complete* inference, so that every correct conclusion can be drawn. Obviously, some of these requirements stand in conflict, as a greater expressivity of a DL makes sound and complete inference more difficult or even undecidable. Consequently, theoretical research in DL has mainly focused on the expressivity of DLs and decidability and complexity of their inference algorithms.

When developing such inference algorithms, one is interested in their computational complexity, their behaviour for “real life” problems, and, in case of incomplete algorithms, their “degree” of completeness. From a theoretical point of view, it is desirable to have algorithms that match the known worst-case complexity of the problem. From the viewpoint of the application, it is more important to have an easily implementable procedure that is amenable to optimizations and hence has good run-time behaviour in realistic applications.

1.3 Expressive Description Logics

Much research in Description Logic has been concerned with the expressivity and computational properties of various DLs (for an overview of current issues in DL research, e.g., see Baader, McGuiness, Nardi, & Patel-Schneider, 2001). These investigations were often triggered by the provision of certain constructors in implemented systems (Nebel, 1988; Borgida & Patel-Schneider, 1994), or by the need for these operators in specific knowledge representation tasks (Baader & Hanschke, 1993; Franconi, 1994; Sattler, 1998).

In the following we introduce the specific features of the DLs that are considered in this thesis.

1.3.1 Counting

Since people tend to describe objects by the *number* of other objects they are related to ("Cars have four wheels, spiders have eight legs, humans have one head, etc.") it does not come as a surprise that most DL systems offer means to capture these aspects. *Number restrictions*, which allow to specify the number of objects related via certain roles, can already be found in KL-ONE and have subsequently been present in nearly all DL systems. More recent systems, like FaCT (Horrocks, 1998) or iFaCT (Horrocks, 1999) also allow for *qualifying number restrictions* (Hollunder & Baader, 1991), which, additionally, state requirements on the related objects. Using number restrictions, it is possible, e.g., to define the concept of parents having at least two children ($\mathtt{Human} \sqcap (\geq 2\ \mathtt{has\_child})$), or of people having exactly two sisters ($\mathtt{Human} \sqcap (\leq 2\ \mathtt{has\_sibling}\ \mathtt{Female}) \sqcap (\geq 2\ \mathtt{has\_sibling}\ \mathtt{Female})$).

It is not hard to see that, at least for moderately expressive DLs, reasoning with number restrictions is more involved than reasoning with universal or existential restrictions only, as number restrictions enforce interactions between role successors. The following concept describes humans having two daughters and two rich children but at most three children:

$$\mathtt{Human} \sqcap (\geq 2\ \mathtt{has\_child}\ \mathtt{Female}) \sqcap (\geq 2\ \mathtt{has\_child}\ \mathtt{Rich}) \sqcap (\leq 3\ \mathtt{has\_child}),$$

which implies that at least one of the daughters must be rich. This form of interaction between role successors cannot be created without number restrictions and has to be dealt with by inference algorithms.

Number restrictions introduce a form of *local* counting into DLs: for an object it is possible to specify the number of other objects it is related to via a given role. There are also approaches to augment DLs with a form of *global* counting. Baader, Buchheit, and Hollunder (1996) introduce *cardinality restrictions on concepts* as a terminological formalism that allows to express constraints on the number of instances that a specific concept may have in a domain. To stay with our family examples, using cardinality restrictions it is possible to express that there are at most two earliest ancestors:

$$(\leq 2\ (\mathtt{Human} \sqcap (\leq 0\ \mathtt{has\_parent}))).$$

1.3.2 Transitive Roles, Role Hierarchies, and Inverse Roles

In many applications of knowledge representation, like configuration (Wache & Kamp, 1996; Sattler, 1996b; McGuinness & Wright, 1998), ontologies (Rector & Horrocks, 1997) or various applications in the database area (Calvanese, Lenzerini, & Nardi, 1998; Calvanese, De Giacomo, Lenzerini, Nardi, & Rosati, 1998; Calvanese, De Giacomo, & Rosati, 1999; Franconi, Baader, Sattler, & Vassiliadis, 2000), *aggregated objects* are of key importance. Sattler (2000) argues that *transitive roles* and *role hierarchies* provide elegant means to express various kinds of part-whole relations that can be used to model aggregated objects.

Again, to stay with our family example, it would be natural to require the `has_offspring` or `has_ancestor` roles to be transitive as this corresponds to the intuitive understanding of these roles. Without transitivity of the role `has_offspring`, the concept

$$\forall \mathtt{has\_offspring}.\mathtt{Rich} \sqcap \exists \mathtt{has\_offspring}.\exists \mathtt{has\_offspring}.\neg \mathtt{Rich}$$

that describes someone who has only rich offsprings and who has an offspring that has a poor offspring, would not be unsatisfiable, which is counter-intuitive

Role hierarchies (Horrocks & Gough, 1997) provide a mean to state sub-role relationship between roles, e.g., to state that `has_child` is a sub-role of `has_offspring`, which makes it possible to infer that, e.g., a grandchild of someone with only rich offsprings must be rich. Role hierarchies also play an important role when modelling sub-relations of the general part-whole relation (Sattler, 1996a).

Role hierarchies only allow to express an approximation of the intuitive understanding of the relationship between the roles `has_child` and `has_offspring`. Our intuitive understanding is that `has_offspring` is the transitive closure of `has_child`, whereas role hierarchies with transitive roles are limited to state that `has_offspring` is an *arbitrary* transitive super-role of `has_child`. Yet, this approximation is sufficient for many knowledge representation tasks and there is empirical evidence that it allows for faster implementations than inferences that support transitive closure (Horrocks, Sattler, & Tobies, 2000a).

Above we have used the roles `has_offspring` and `has_ancestor` and the intuitive understanding of these roles requires them to be mutually inverse. Without the expressive means of *inverse roles*, this cannot be captured by a DL so that the concept

$$\neg \mathtt{Rich} \sqcap \exists \mathtt{has\_offspring}.\top \sqcap \forall \mathtt{has\_offspring}.\forall \mathtt{has\_ancestor}.\mathtt{Rich}$$

that describes somebody poor who has an offspring and whose offsprings only have rich ancestors would not be unsatisfiable. This shortcoming of expressive power is removed by the introduction of inverse roles into a DL, which would allow to replace `has_ancestor` by $\mathtt{has\_offspring}^{-1}$, which denotes the inverse of `has_offspring`.

1.3.3 Nominals

Nominals, i.e., atomic concepts referring to single individuals of the domain, are studied both in the area of DLs (Schaerf, 1994; Borgida & Patel-Schneider, 1994; De Giacomo & Lenzerini, 1996) and modal logics (Gargov & Goranko, 1993; Blackburn & Seligman, 1995; Areces, Blackburn, & Marx, 2000). Nominals play an important rôle in knowledge representation because they allow to capture the notion of uniqueness and identity. Coming back to the ABox example from above, for a DL with nominals, the names `MARY` or `PETER` may not only be used in ABox assertions but can also be used in place of atomic concept, which, e.g., allows to describe `MARY`'s children by the concept $\exists \mathtt{has\_child}^{-1}.\mathtt{MARY}$. Modeling named individuals by pairwise disjoint atomic concepts, as it is done in the DL system CLASSIC (Borgida & Patel-Schneider, 1994), is not adequate and leads to incorrect inferences. For example, if `MARY` does not name a single individual, it would be impossible to infer that every child of MARY must be a sibling of PETER (or PETER himself), and so the concept

$$\exists \texttt{has\_child}^{-1}.\texttt{MARY} \sqcap \forall \texttt{has\_child}^{-1}.(\forall \texttt{has\_child}.\neg \texttt{PETER})$$

together with the example ABox would be incorrectly satisfiable. It is clear that cardinality restrictions on concepts can be used to express nominals and and we will see in this thesis that also the converse holds.

For decision procedures, nominals cause problems because they destroy the tree model property of a logic, which has been proposed as an explanation for the good algorithmic behaviour of modal and description logics (Vardi, 1996; Grädel, 1999c) and is often exploited by decision procedures.

1.4 Guarded Logics

The *guarded fragment* of first-order predicate logic, introduced by Andréka, van Benthem, and Németi (1998), is a successful attempt to transfer many good properties of modal, temporal, and Description Logics to a larger fragment of predicate logic. Among these are decidability, the finite model property, invariance under an appropriate variant of bisimulation, and other nice model theoretic properties (Andréka et al., 1998; Grädel, 1999b).

The Guarded Fragment (GF) is obtained from full first-order logic through relativization of quantifiers by so-called *guard formulas*. Every appearance of a quantifier in GF must be of the form

$$\exists \mathbf{y}(\alpha(\mathbf{x},\mathbf{y}) \wedge \phi(\mathbf{x},\mathbf{y})) \text{ or } \forall \mathbf{y}(\alpha(\mathbf{x},\mathbf{y}) \rightarrow \phi(\mathbf{x},\mathbf{y})),$$

where α is a positive atomic formula, the *guard*, that contains all free variables of ϕ. This generalizes quantification in description, modal, and temporal logic, where quantification is restricted to those elements reachable via some accessibility relation. For example, in DLs, quantification occurs in the form of existential and universal restrictions like $\forall$`has_child.Rich`, which expresses that those individuals *reachable via the role* (guarded by) `has_child` must be rich.

By allowing for more general formulas as guards while preserving the idea of quantification only over elements that are *close together* in the model, one obtains generalisations of GF which are still well-behaved in the sense of GF. Most importantly, one can obtain the *loosely guarded fragment* (LGF) (van Benthem, 1997) and the *clique guarded fragment* (CGF) (Grädel, 1999a), for which decidability, invariance under clique guarded bisimulation, and some other properties have been shown by Grädel (1999a). For other extension of GF the picture is irregular. While GF remains decidable under the extension with fixed point operators (Grädel & Walukiewicz, 1999), adding counting constructs or transitivity statements leads to undecidability (Grädel, 1999b; Ganzinger, Meyer, & Veanes, 1999).

Guarded fragments are of interest for the DL community because many DLs are readily embeddable into suitable guarded fragments. This allows the transfer of results for guarded fragments to DLs. For example, Goncalves and Grädel (2000) show decidability of the guarded fragment μAGFCI, into which a number of expressive DLs is easily embeddable, yielding decidability for these DLs.

1.5 Outline and Structure of this Thesis

This thesis deals with reasoning in expressive DLs and Guarded Logics. We supply a number of novel complexity results and practical algorithms for inference problems. Generally, we are more interested in the algorithmic properties of the logics we study than their application in concrete knowledge representation tasks. Consequently, the examples given in this thesis tend to be terse and abstract and are biased towards computational characteristics. For more information on how to use DLs for specific knowledge representation tasks, e.g., refer to (Brachman, McGuinness, Patel-Schneider, Resnick, & Borgida, 1991; Borgida, 1995; Calvanese et al., 1998; Sattler, 2000).

This thesis is structured as follows:

- We start with a more formal introduction to DLs in Chapter 2. We introduce the standard DL $\mathcal{ALC}$ and define its syntax and semantics. We specify the relevant inference problems and show how they are interrelated.

- Chapter 3 briefly surveys techniques employed for reasoning with DLs. We then describe a tableau algorithm that decides satisfiability of $\mathcal{ALC}$-concepts with optimum worst-case complexity (PSpace) to introduce important notions and methods for dealing with tableau algorithms.

- In Chapter 4 we consider the complexity of a number of DLs that allow for qualifying number restrictions. The DL $\mathcal{ALCQ}$ is obtained from $\mathcal{ALC}$ by, additionally, allowing for qualifying number restrictions. We give a tableau algorithm that decides concept satisfiability for $\mathcal{ALCQ}$. We show how this algorithms can be modified to run in PSpace, which fixes the complexity of the problem as PSpace-complete. Previously, the exact complexity of the problem was only known for the (unnaturally) restricted case of unary coding of numbers (Hollunder & Baader, 1991) and the problem was conjectured to be ExpTime-hard for the unrestricted case (van der Hoek & de Rijke, 1995). We use the methods developed for $\mathcal{ALCQ}$ to obtain a tableau algorithm that decides concept satisfiability for the DL $\mathcal{ALCQI}b$, which adds expressive role expressions to $\mathcal{ALCQ}$, in PSpace, which solves an open problem from (Donini, Lenzerini, Nardi, & Nutt, 1997).

  We show that, for $\mathcal{ALCQI}b$, reasoning w.r.t. general TBoxes and knowledge bases is ExpTime-complete. This extends the known result for $\mathcal{ALCQI}$ (De Giacomo, 1995) to a more expressive DL and, unlike the proof in (De Giacomo, 1995), our proof is not restricted to the case of unary coding of numbers in the input.

- The next chapter deals with the complexity of reasoning with cardinality restrictions on concepts. We study the complexity of the combination of the DLs $\mathcal{ALCQ}$ and

$\mathcal{ALCQI}$ with cardinality restrictions. These combinations can naturally be embedded into C^2, the two variable fragment of predicate logic with counting quantifiers (Grädel, Otto, & Rosen, 1997), which yields decidability in NExpTime (Pacholski, Szwast, & Tendera, 1997) (in the case of unary coding of numbers). We show that this is a (worst-case) optimal solution for $\mathcal{ALCQI}$, as $\mathcal{ALCQI}$ with cardinality restrictions is already NExpTime-hard. In contrast, we show that for $\mathcal{ALCQ}$ with cardinality restrictions, all standard inferences can be solved in ExpTime. This result is obtained by giving a mutual reduction from reasoning with cardinality restrictions and reasoning with nominals. Based on the same reduction, we show that already concept satisfiability for $\mathcal{ALCQI}$ extended with nominals is NExpTime-complete. The results for $\mathcal{ALCQI}$ can easily be generalised to $\mathcal{ALCQI}b$.

- In Chapter 6 we study DLs with transitive and inverse roles. For the DL $\mathcal{SI}$—the extension of $\mathcal{ALC}$ with inverse and transitive roles—we describe a tableau algorithm that decides concept satisfiability in PSpace, which matches the known lower bound for the worst-case complexity of the problem and extends Spaan's results for the modal logic $\mathsf{K4}_t$(1993b).

  $\mathcal{SI}$ is then extended to $\mathcal{SHIQ}$, a DL which, additionally, allows for role hierarchies and qualifying number restrictions. We determined the worst-case complexity of reasoning with $\mathcal{SHIQ}$ as ExpTime-complete. The ExpTime upper bound has been an open problem so far. Moreover, we show that reasoning becomes NExpTime-complete if nominals are added to $\mathcal{SHIQ}$.

  The algorithm used to establish the ExpTime-bound for $\mathcal{SHIQ}$ employs a highly inefficient automata construction and cannot be used for efficient implementations. Instead, we describe a tableau algorithm for $\mathcal{SHIQ}$ that promises to be amenable to optimizations and forms the basis of the highly-optimized DL system iFaCT (Horrocks, 1999).

- In Chapter 7 we develop a tableau algorithm for the clique guarded fragment of FOL, based on the same ideas usually found in algorithms for modal logics or DLs. Since tableau algorithms form the basis of some of the fastest implementations of DL systems, we believe that this algorithm is a viable starting point for an efficient implementation of a decision procedure for CGF. Since many DLs are embeddable into CGF, such an implementation would be of high interest.

- In a final chapter, we conclude.

Some of the results in this thesis have previously been published. The PSpace-algorithm for $\mathcal{ALCQ}$ has been reported in (Tobies, 1999b) and is extended to deal with inverse roles and conjunction of roles in (Tobies, 2001). NExpTime-completeness of $\mathcal{ALCQI}$ with cardinality restrictions is presented in (Tobies, 1999a, 2000), where the latter publication establishes the connection of reasoning with nominals and with cardinality restrictions. The $\mathcal{SI}$-algorithm is presented in (Horrocks, Sattler, & Tobies, 2000a), a description of the tableau algorithm for $\mathcal{SHIQ}$ can be found in (Horrocks, Sattler, & Tobies, 1999). Finally, the tableau algorithm for CGF has previously been published in (Hirsch & Tobies, 2000).

Chapter 2

Preliminaries

In this chapter we give a more formal introduction to Description Logics and their inference problems. We define syntax and semantics of the "basic" DL $\mathcal{ALC}$ and of the terminological and assertional formalism used in this thesis. Based on these definitions, we introduce a number of inference problems and show how they are interrelated.

2.1 The Basic DL $\mathcal{ALC}$

Schmidt-Schauß and Smolka (1991) introduce the DL $\mathcal{ALC}$, which is distinguished in that it is the "smallest" DL that is closed under all Boolean connectives, and give a sound and complete subsumption algorithm. Unlike the other DL inference algorithms developed at that time, they deviated from the structural paradigm and used a new approach, which, due to its close resemblance to first-order logic tableau algorithms, was later also called tableau algorithm. Later, Schild's (1991) discovery that $\mathcal{ALC}$ is a syntactic variant of the basic modal logic K made it apparent that Schmidt-Schauß and Smolka had re-invented in DL notation the tableau-approach that had been successfully applied to modal inference problems (see, e.g., Ladner, 1977; Halpern & Moses, 1992; Goré, 1998).

The DL $\mathcal{ALC}$ allows complex concepts to be built from concept and relation names using the propositional constructors $\sqcap$ (*and, class intersection*), $\sqcup$ (*or, class union*), and $\neg$ (*not, class complementation*). Moreover, concepts can be related using universal and existential quantification along role names.

Definition 2.1 (Syntax of $\mathcal{ALC}$)
Let NC *be a set of* concept names *and* NR *be a set of* role names*. The set of* $\mathcal{ALC}$-concepts *is built inductively from these using the following grammar, where* $A \in \mathsf{NC}$ *and* $R \in \mathsf{NR}$*:*

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

$\diamond$

For now, we will use an informal definition of the *size* $|C|$ of a concept C: we define $|C|$ to be the number of symbols necessary to write down C over the alphabet $\mathsf{NC} \cup \mathsf{NR} \cup \{\sqcap, \sqcup, \neg, \forall, \exists, (,)\}$. This will not be the definitive definition of the size of the concept because it relies on an unbounded alphabet (NC and NR are arbitrary sets), which makes it unsuitable for complexity considerations. We will clarify this issue in Definition 3.11.

Starting with (Brachman & Levesque, 1984), semantics of DLs model concepts as sets and roles as binary relations. Starting from an interpretation of the concept and role names, the semantics of arbitrary concepts are defined by induction over their syntactic structure. For $\mathcal{ALC}$, this is done as follows.

Definition 2.2 (Semantics of $\mathcal{ALC}$)
The semantics of $\mathcal{ALC}$-concepts is defined relative to an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, *where $\Delta^\mathcal{I}$ is a non-empty set, called the* domain *of $\mathcal{I}$, and $\cdot^\mathcal{I}$ is a* valuation *that defines the interpretation of concept and relation names by mapping every concept name to a subset of $\Delta^\mathcal{I}$ and every role name to a subset of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$. To obtain the semantics of a complex concept this valuation is inductively extended by setting:*

$$
\begin{aligned}
&(\neg C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I} \qquad (C_1 \sqcap C_2)^\mathcal{I} = C_1^\mathcal{I} \cap C_2^\mathcal{I} \qquad (C_1 \sqcup C_2)^\mathcal{I} = C_1^\mathcal{I} \cup C_2^\mathcal{I}\\
&(\forall R.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \textit{for all } y \in \Delta^\mathcal{I}, (x,y) \in R^\mathcal{I} \textit{ implies } y \in C^\mathcal{I}\}\\
&(\exists R.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \textit{there is a } y \in \Delta^\mathcal{I} \textit{ with } (x,y) \in R^\mathcal{I} \textit{ and } y \in C^\mathcal{I}\}.
\end{aligned}
$$

A concept C is satisfiable *iff there is an interpretation $\mathcal{I}$ such that $C^\mathcal{I} \neq \emptyset$. A concept C is* subsumed *by a concept D (written $C \sqsubseteq D$) iff, for every interpretation $\mathcal{I}$, $C^\mathcal{I} \subseteq D^\mathcal{I}$. Two concepts C, D are* equivalent *(written $C \equiv D$) iff $C \sqsubseteq D$ and $D \sqsubseteq C$.* ◇

From this definition it is apparent, as noticed by Schild (1991), that $\mathcal{ALC}$ is a syntactic variant of the propositional (multi-) modal logic K_m. More precisely, for a set of concept names NC and role names NR, the logic $\mathcal{ALC}$ corresponds to the modal logic K_m with propositional atoms NC and modal operators $\{\langle R\rangle, [R] \mid R \in \mathsf{NR}\}$ where the Boolean operators of $\mathcal{ALC}$ $(\sqcap, \sqcup, \neg)$ correspond to the Boolean operators of K_m $(\wedge, \vee, \neg)$, existential restrictions over a role R to the diamond modality $\langle R\rangle$, and universal restrictions over a role R to the box modality $[R]$. Applying this syntactic transformation in either direction yields, for every $\mathcal{ALC}$-concept C, an equivalent K_m-formula ϕ_C and, for every K_m-formula ϕ, an equivalent $\mathcal{ALC}$-concept C_ϕ. A similar correspondence exists also for more expressive DLs.

We will often use $\top$ as an abbreviation for an arbitrary tautological concept, i.e., a concept with $\top^\mathcal{I} = \Delta^\mathcal{I}$ for every interpretation $\mathcal{I}$. E.g., $\top = A \sqcup \neg A$ for an arbitrary concept name $A \in \mathsf{NC}$. Similarly, we use $\bot$ as an abbreviation for an unsatisfiable concept ($\bot^\mathcal{I} = \emptyset$ for every interpretation $\mathcal{I}$). E.g., $\bot = A \sqcap \neg A$ for an arbitrary $A \in \mathsf{NC}$. Also, we will use the standard logical abbreviations $C \rightarrow D$ for $\neg C \sqcup D$ and $C \leftrightarrow D$ for $C \rightarrow D \sqcap D \rightarrow C$.

2.2 Terminological and Assertional Formalism

Starting from the initial KL-ONE system (Brachman & Schmolze, 1985), DL systems allow to express two categories of knowledge about the application domain:

- *terminological* knowledge, which is stored in a so-called *TBox* and consists of general definition of concepts and knowledge about their interrelation, and
- *assertional* knowledge, which is stored in a so-called *ABox* and consist of a (partial) description of a specific situation consisting of elements of the application domains.

It should be noted that there are DL systems, e.g., FaCT (Horrocks, 1998), that do not support ABoxes but are limited to TBoxes only. In contrast to this, all systems that support ABoxes also have some kind of support for TBoxes.

Different DL systems allow for different kinds of TBox formalism, which has an impact on the difficulty of the various inference problems. Here, we define the most general form of TBox formalism usually studied—*general axioms*—and describe other possibilities as a restriction of this formalism.

Definition 2.3 (General Axioms, TBox)
A general axiom *is an expression of the form $C \sqsubseteq D$ or $C \doteq D$ where C and D are concepts. A* TBox *$\mathcal{T}$ is a finite set of general axioms.*

An interpretation $\mathcal{I}$ satisfies a general axiom $C \sqsubseteq D$ ($C \doteq D$) iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ ($C^\mathcal{I} = D^\mathcal{I}$). It satisfies $\mathcal{T}$ iff it satisfies every axiom in $\mathcal{T}$. In this case, $\mathcal{T}$ is called satisfiable, *$\mathcal{I}$ is called* a model *of $\mathcal{T}$ and we write $\mathcal{I} \models \mathcal{T}$.*

Satisfiability, subsumption and equivalence of concepts can also be defined w.r.t. TBoxes: a concept C is satisfiable w.r.t. $\mathcal{T}$ *iff there is a model $\mathcal{I}$ of $\mathcal{T}$ with $C^\mathcal{I} \neq \emptyset$. C is* subsumed by D w.r.t. $\mathcal{T}$ *iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{T}$. Equivalence w.r.t. $\mathcal{T}$ is defined analogously and denoted with $\sqsubseteq_\mathcal{T}$.* ◇

Most DL systems, e.g., KRIS (Baader & Hollunder, 1991), allow only for a limited form of TBox that essentially contains only macro definitions. This is captured by the following definition.

Definition 2.4 (Simple TBox)
A TBox $\mathcal{T}$ is called simple *iff*

- *the left-hand side of axioms consist only of concept names, that is, $\mathcal{T}$ consists only of axioms of the form $A \sqsubseteq D$ and $A \doteq D$ for $A \in \mathsf{NC}$,*
- *a concept name occurs at most once as the left-hand side of an axiom in $\mathcal{T}$, and*
- *$\mathcal{T}$ is* acyclic. *Acyclicity is defined as follows: $A \in \mathsf{NC}$ is said to* "directly use" *$B \in \mathsf{NC}$ if $A \doteq D \in \mathcal{T}$ or $A \sqsubseteq D \in \mathcal{T}$ and B occurs in D;* "uses" *is the transitive closure of* "directly uses"*. We say that $\mathcal{T}$ is* acyclic *if there is no $A \in \mathsf{NC}$ that uses itself.*

◇

Partial descriptions of the application domain can be given as an ABox.

Definition 2.5 (ABox)
Let NI *be a set of individual names. For individual names* $x, y \in \mathsf{NI}$*, a concept* C*, and a role name* R*, the expressions* $x : C$*,* $(x, y) : R$ *and* $x \not\doteq y$ *are* assertional axioms*. An* ABox $\mathcal{A}$ *is a finite set of assertional axioms.*

To define the semantics of ABoxes we require interpretations, additionally, to map every individual name $x \in \mathsf{NI}$ *to an element* $x^{\mathcal{I}}$ *of the domain* $\Delta^{\mathcal{I}}$*.*

An interpretation $\mathcal{I}$ *satisfies an assertional axiom* $x : C$ *iff* $x^{\mathcal{I}} \in C^{\mathcal{I}}$*, it satisfies* $(x, y) : R$ *iff* $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}}$*, and it satisfies* $x \not\doteq y$ *iff* $x^{\mathcal{I}} \neq y^{\mathcal{I}}$*.* $\mathcal{I}$ *satisfies* $\mathcal{A}$ *iff it satisfies every assertional axiom in* $\mathcal{A}$*. If such an interpretation* $\mathcal{I}$ *exists, then* $\mathcal{A}$ *is called* satisfiable*,* $\mathcal{I}$ *is called a* model *of* $\mathcal{A}$*, and we write* $\mathcal{I} \models \mathcal{A}$*.* ◇

To decide whether $\mathcal{I} \models \mathcal{A}$ for an interpretation $\mathcal{I}$ and an ABox $\mathcal{A}$, the interpretation of those individuals that do not occur in $\mathcal{A}$ is irrelevant (Nebel, 1990a; Buchheit, Donini, & Schaerf, 1993). Thus, to define a model of an ABox $\mathcal{A}$ it is sufficient to specify the interpretation of those individuals occurring in $\mathcal{A}$. Our definition of ABoxes is slightly different from what can usually be found in the literature, in that we do not impose the *unique name assumption*. The unique name assumption requires that every two distinct individuals must be mapped to distinct elements of the domain. We do not have this requirement but include explicit inequality assertions between two individuals as assertional axioms. It is clear that our approach is more general than the unique name assumption because inequality can be asserted selectively only for some individual names. We use this approach due to its greater flexibility and since it allows for a more uniform treatment of ABoxes in the context of tableau algorithms, which we will encounter in Chapter 3.

Definition 2.6 (Knowledge Base)
A knowledge base *(KB)* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ *consists of a TBox* $\mathcal{T}$ *and an ABox* $\mathcal{A}$*. An interpretation* $\mathcal{I}$ *satisfies* $\mathcal{K}$ *iff* $\mathcal{I} \models \mathcal{T}$ *and* $\mathcal{I} \models \mathcal{A}$*. In this case,* $\mathcal{K}$ *is called* satisfiable*,* $\mathcal{I}$ *is called a* model *of* $\mathcal{K}$ *and we write* $\mathcal{I} \models \mathcal{K}$*.* ◇

2.3 Inference Problems

From the previous definitions, one can immediately derive a number of (so called *standard*) inference problems for DL systems that are commonly studied. Here, we quickly summarize the most important of them and show how they are interrelated.

- **Concept satisfiability**, i.e., given a concept C, is C satisfiable (maybe w.r.t. a TBox $\mathcal{T}$)? This inference allows to determine if concepts in the KB are contradictory (describe the empty class).

- **Concept subsumption**, i.e., given two concepts C, D, is C subsumed by D (maybe w.r.t. a TBox $\mathcal{T}$)? Using this inference, concepts defined in a TBox can be arranged in a subsumption quasi-order that reflects the specialisation/generalisation hierarchy of the concepts. Calculation of the subsumption hierarchy is one of the main inferences used by many applications of DL systems (e.g., Rector & Horrocks, 1997; Schulz & Hahn, 2000; Bechhofer & Horrocks, 2000; Franconi & Ng, 2000).

  For any DL that is closed under Boolean operations, subsumption and (un-)satisfiability are mutually reducible: a concept C is unsatisfiable w.r.t. a TBox $\mathcal{T}$ iff $C \sqsubseteq_{\mathcal{T}} \bot$. Conversely, $C \sqsubseteq_{\mathcal{T}} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{T}$.

  Concept satisfiability and subsumption are problems that are usually considered only w.r.t. TBoxes rather than KBs. The reason for this is the fact (Nebel, 1990a; Buchheit et al., 1993) that the ABox does not interfere with these problems as long as the KB is satisfiable. W.r.t. unsatisfiable KBs, obviously every concept is unsatisfiable and every two concepts mutually subsume each other.

- **Knowledge Base Satisfiability**, i.e., given a KB $\mathcal{K}$, is $\mathcal{K}$ satisfiable? This inference allows to check whether the knowledge stored in the KB is free of contradictions, which is maybe the most fundamental requirement for knowledge in DL systems. For a KB that contains a contradiction, i.e., is not satisfiable, arbitrary conclusion can be drawn.

  Concept satisfiability (and hence concept subsumption) can be reduced to KB satisfiability: a concept C is satisfiable w.r.t. a (possibly empty) TBox $\mathcal{T}$ iff the KB $(\mathcal{T}, \{x : C\})$ is satisfiable.

- **Instance Checking**, i.e., given a KB $\mathcal{K}$, an individual name x, and a concept C, is $x^{\mathcal{I}} \in C^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$? In this case, x is called an *instance* of C w.r.t. $\mathcal{K}$. Using this inference it is possible to deduce knowledge from a KB that is only implicitly present, e.g., it can be deduced that an individual x is an instance of a concept C in every model of the knowledge base even though $x : C$ is not asserted explicitly in the ABox—it follows from the other assertions in the KB.

  Instance checking can be reduced to KB (un-)satisfiability. For a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, x is an instance of C w.r.t. $\mathcal{K}$ iff the KB $(\mathcal{T}, \mathcal{A} \cup \{x : \neg C\})$ is unsatisfiable.

All the mentioned reductions are obviously computable in linear time. Hence, KB satisfiability can be regarded as the most general of the mentioned inference problems. As we will see in a later chapter, for some DLs, it is also possible to polynomially reduce KB satisfiability to concept satisfiability.

Chapter 3

Reasoning in Description Logics

This chapter starts with an overview of methods that have been developed to solve DL inference problems. We then describe a tableau algorithm that decides concept satisfiability and subsumption for $\mathcal{ALC}$ and which can be implemented to run in PSPACE. Albeit this is a well-known result (Schmidt-Schauß & Smolka, 1991), it is repeated here because it allows us to introduce important notions and methods for dealing with tableau algorithms before these are applied to obtain results for more expressive logics in the subsequent chapters.

3.1 Reasoning Paradigms

Generally speaking, there are four major and some minor approaches to reasoning with DLs that will be briefly described here. Refer to (Baader & Sattler, 2000) for a more history-oriented introduction to reasoning with DLs.

Structural algorithms The early DL systems like KL-ONE (Brachman & Schmolze, 1985) and its successor systems BACK (Quantz & Kindermann, 1990), K-REP (Mays, Dionne, & Weida, 1991), or LOOM (MacGregor, 1991) used *structural algorithms* that rely on syntactic comparison of concepts in a suitable normal form to decide subsumption. Nebel (1990a) gives a formal description of an algorithm based on this approach. Usually, these algorithms had very good (polynomial) run-time behaviour. Tractability was a major concern in the development of DL systems and algorithms with super-polynomial runtime were considered unusable in practical applications (Levesque & Brachman, 1987). Yet, as it turned out, even DLs with very limited expressive power prohibit tractable inference algorithms (Brachman & Levesque, 1984; Nebel, 1990b) and for some, like KL-ONE, subsumption is even undecidable (Schmidt-Schauß, 1989). Consequently, complete structural algorithms are known only for DLs of very limited expressivity.

This limitations were addressed by DL researchers in three general ways: some system developers deliberately committed to incomplete algorithms to preserve the good run-time behaviour of their systems. Others proceeded by carefully tailoring the DL to maximise its expressivity while maintaining sound and complete structural algorithms. Representatives of the former approach are the BACK and the LOOM system while CLASSIC(Patel-Schneider, McGuiness, Brachman, Resnick, & Borgida, 1991) follows the latter approach with a “nearly” complete structural subsumption algorithm (Borgida & Patel-Schneider, 1994). The third approach was to develop algorithms that are capable to deal with more expressive DLs despite the higher complexity. This required departure from the methods employed so far.

Tableau algorithms The first such algorithm was developed by Schmidt-Schauß and Smolka (1991) for the DL $\mathcal{ALC}$, and the employed methodology proved to be useful to decide subsumption and other inference problems like concept satisfiability also for other DLs (Hollunder, Nutt, & Schmidt-Schauß, 1990; Hollunder & Baader, 1991; Baader, 1991; Hanschke, 1992). Due to their close resemblance to tableau algorithms for first-order predicate logic (FOL) they were also called *tableau algorithms*. For many DLs, it was possible to obtain algorithms based on the tableau approach that match the known worst-case complexity of the problem (see, e.g., Donini, Lenzerini, Nardi, & Nutt, 1991a, 1991b; Donini, Hollunder, Lenzerini, Spaccamela, Nardi, & Nutt, 1992, for a systematic study). Although the inference problems for these DLs are usually at least NP- or even PSPACE-hard, systems implementing the tableau approach, like KRIS (Baader & Hollunder, 1991) or CRACK (Bresciani, Franconi, & Tessaris, 1995), show reasonable runtime performance on application problems and more recent systems that employ sophisticated optimization techniques, like Horrock’s FaCT system (1998) or Patel-Schneider’s DLP (2000), can deal with problems of considerable size, even for EXPTIME-hard DLs. For theses logics, the employed tableau algorithms exceed the known worst-case complexity of the problems, but are rather biased towards optimizability for “practical” cases. Indeed, for EXPTIME-complete DLs, it turns out to be very involved to obtain tableau algorithms with optimum worst-case complexity (Donini & Massacci, 2000).

Translational approaches Schild’s discovery (1991) that DLs are syntactic variants of modal logics made it possible to obtain inference procedures for DLs by simply borrowing the methods from the corresponding modal logic. This approach has been refined for more expressive DLs and a number of (worst-case) optimal decision procedures for very expressive—usually EXPTIME-complete—DLs were obtained by sophisticated translation into PDL (De Giacomo & Lenzerini, 1994a; De Giacomo & Lenzerini, 1994c, 1996; De Giacomo, 1995) or the modal μ-calculus (Schild, 1994; De Giacomo & Lenzerini, 1994b). While many interesting complexity results could be obtained in this manner, there exists no implementation of a DL system that utilizes this approach. Experiments indicate (Horrocks et al., 2000a) that it will be very hard to obtain efficient implementations based on this kind of translations. More recently, modal logicians like Areces and de Rijke (2000) have advocated hybrid modal logics (Areces et al., 2000; Areces, 2000) as a suitable target for the translation of DLs and obtain novel theoretical results and decision procedures. It is unclear if these decision procedures can be implemented efficiently.

A different approach utilizes translation into FOL. Already Brachman and Levesque

(1984) use FOL to specify the semantics of their DL and inference problems for nearly all DLs (and their corresponding modal logics) are easily expressible in terms of satisfiability problems for (extensions of) FOL. Since FOL is undecidable, one does not immediately obtain a decision procedure in this manner. So, these approaches use more restricted and hence decidable fragments of FOL as target of their translation. Borgida (1996) uses the two-variable fragment of FOL to prove decidability of an expressive DL in NExpTime while De Nivelle (2000) gives a translation of a number of modal logics into the guarded fragment to facilitate the application of FOL theorem proving methods to these logics. Schmidt (1998) uses a non-standard translation into a fragment of FOL for which decision procedures based on a FOL theorem prover exist. Areces et. al (2000) show that careful tuning of standard FOL theorem proving methods also yields a decision procedure for the standard translation. The latter approaches are specifically biased towards FOL theorem provers and make it possible to utilize the massive effort spent on the implementation and optimization of FOL theorem provers to reason with DLs. It seems though, that the translation approach leads to acceptable but inferior runtime when compared with tableau systems (Massacci & Donini, 2000; Horrocks, Patel-Schneider, & Sebastiani, 2000).

Automata based methods Many DL and modal logics possess the so-called *tree model property*, i.e., every satisfiable concept has—under a suitable abstraction—a tree-shaped model. This makes it possible to reduced the satisfiability of a concept to the existence of a tree with certain properties dependent on the formula. If it is possible to capture these properties using a *tree automaton* (Gécseg & Steinby, 1984; Thomas, 1992), satisfiability and hence subsumption of the logic can be reduce to the emptiness problem of the corresponding class of tree automata (Vardi & Wolper, 1986). Especially for DLs with ExpTime-complete inference problems, where it is difficult to obtain tableau algorithms with optimum complexity, exact complexity results can be obtained elegantly using the automaton approach (Calvanese, De Giacomo, & Lenzerini, 1999; Lutz & Sattler, 2000). Yet, so far it seems impossible to obtain efficient implementations from automata-based algorithms. The approach usually involves an exponential step that occurs in *every* case independent of the "difficulty" of the input concept and cannot be avoided by existing methods. This implies that such an algorithm will exhibit exponential behaviour even for "easy" instances, which so far prohibits the use of the approach in practice.

Other approaches In addition to these approaches, there also exist further, albeit less influential, approaches. Instead of dealing with DLs as fragments of a more expressive formalism, the *SAT-based method* developed by Giunchiglia and Sebastiani (1996) uses the opposite approach and extend reasoning procedures for the less expressive formalism of propositional logic to DLs. Since highly sophisticated SAT-solvers are available, this approach has proven to be rather successful. Yet, it cannot compete with tableau based algorithms (Massacci, 1999) and so far is not applicable to DLs more expressive than $\mathcal{ALC}$.

The *inverse method* (Voronkov, 2000) takes a radically different approach to satisfiability testing. It tries to prove unsatisfiability of a formula in a bottom-up manner, by trying to derive the input formula starting from "atomic" contradictions. There exists only a very early implementation of the inverse method (Voronkov, 1999), which shows promising run-time performance but at the moment it cannot compete with tableau based systems (Massacci, 1999).

Both the SAT-based approach and the inverse method have so far not been studied with respect to their worst-case complexity.

3.2 Tableau Reasoning for $\mathcal{ALC}$-satisfiability

Even though KB satisfiability is the most general standard inference problem, it is also worthwhile to consider solutions for the less general problems. For many applications of DL systems, the ABox does not play a role and reasoning is performed solely on the terminological level (Rector & Horrocks, 1997; Schulz & Hahn, 2000; Bechhofer & Horrocks, 2000; Franconi & Ng, 2000). For these applications, the additional overhead of dealing with ABoxes is unnecessary. Additionally, ABoxes do not have a resemblance in the modal world (with the exception of *hybrid modal logics*, see (Areces & de Rijke, 2000)) and hence theoretical results obtained for KB inferences do not transfer as easily as results for reasoning with TBoxes, which often directly apply to modal logics. From a pragmatic point of view, since full KB reasoning is at least as hard as reasoning w.r.t. TBoxes, it is good to know how to deal (efficiently) with the latter problem before trying to solve the former. Finally, as we will see in Section 3.2.3, sometimes concept satisfiability suffices to solve the more complicated inference problems.

Schmidt-Schauß and Smolka (1991) were the first to give a complete subsumption algorithm for $\mathcal{ALC}$. The algorithm they used followed a new paradigm for the development of inference algorithms for DLs that proved to be applicable to a vast range of DL inference problems and, due to its resemblance to tableau algorithms for FOL, was later called the tableau approach (see Baader & Sattler, 2000, for an overview of tableau algorithms for DLs). After the correspondence of DLs and modal logics had been pointed out by Schild (1991), it became apparent that the tableau algorithms developed for DLs also closely resembled those used by modal logicians. The tableau approach has turned out to be particularly amenable to optimizations and some of the most efficient DL and modal reasoner currently available are based on tableau algorithms (see Massacci & Donini, 2000, for a system comparison).

Generally speaking, a tableau algorithm for a DL tries to prove satisfiability of a concept (or a knowledge base) by trying to explicitly construct a model or some kind of structure that induces the existence of a model (a *pre-model*). This is done by manipulating a *constraint system*—some kind of data structure that contains a partial description of a model or pre-model—using a set of *completion rules*. Such constraint systems usually consist of a number of individuals for which role relationships and membership in the extension of concepts are asserted, much like this is done in an ABox. Indeed, for $\mathcal{ALC}$ and the DLs considered in the next chapter, it is convenient to use the ABox formalism to capture the constraints. For more expressive DLs, it will be more viable to use a different data structure, e.g., to emphasise the graph structure underlying the ABox.

Independent of the formalism used to express the constraints, completion of such a constraint system is performed starting from an initial constraint system, which depends on the input concept (or knowledge base), until either an obvious contradiction (a clash) has been generated or no more rules can be applied. In the latter case, the rules have been chosen in a way that a model of the concept (or knowledge base) can be immediately derived from the constraint system.

Definition 3.1 (Negation Normal Form)
In the following, we will consider concepts in negation normal form *(NNF), a form in which negations (¬) appear only in front of concept names. Every $\mathcal{ALC}$-concept can be equivalently transformed into NNF by pushing negation inwards using the following equivalences:*

$$\begin{array}{rlrl} \neg(C_1 \sqcap C_2) & \equiv \neg C_1 \sqcup \neg C_2 & \qquad \neg\forall R.C & \equiv \exists R.\neg C \\ \neg(C_1 \sqcup C_2) & \equiv \neg C_1 \sqcap \neg C_2 & \qquad \neg\exists R.C & \equiv \forall R.\neg C \\ \neg\neg C & \equiv C & & \end{array}$$

Note that every $\mathcal{ALC}$-concept can be transformed into NNF in linear time.

For a concept C in NNF, we denote the set of sub-concepts of C by $sub(C)$. Obviously, the size of $sub(C)$ is bounded by $|C|$. ◇

3.2.1 Deciding Concept Satisfiability for $\mathcal{ALC}$

We will now describe the $\mathcal{ALC}$-algorithm that decides concept satisfiability (and hence concept subsumption) for $\mathcal{ALC}$. As mentioned before, we use ABoxes to capture the constraint systems generated by the $\mathcal{ALC}$-algorithm.

Algorithm 3.2 ($\mathcal{ALC}$-algorithm)
An ABox $\mathcal{A}$ contains a clash *iff, for an individual name $x \in \mathsf{NI}$ and a concept name $A \in \mathsf{NC}$, $\{x : A, x : \neg A\} \subseteq \mathcal{A}$. Otherwise, $\mathcal{A}$ is called* clash-free.

To test the satisfiability of an $\mathcal{ALC}$-concept C in NNF, the $\mathcal{ALC}$-algorithm works as follows. Starting from the initial ABox *$\mathcal{A}_0 = \{x_0 : C\}$ it applies the* completion rules *from Figure 3.1, which modify the ABox. It stops when a clash has been generated or when no rule is applicable. In the latter case, the ABox is* complete. *The algorithm answers "C is satisfiable" iff a complete and clash-free ABox has been generated.*

From the rules in Figure 3.1, the $\rightarrow_\sqcup$-rule is called non-deterministic *while the other rules are called* deterministic. *The $\rightarrow_\exists$-rule is called* generating, *while the other rules are called* non-generating. ◇

Figure 3.1 The completion rules for $\mathcal{ALC}$

$\rightarrow_{\sqcap}$:	**if**	1.	$x : C_1 \sqcap C_2 \in \mathcal{A}$ and
		2.	$\{x : C_1, x : C_2\} \not\subseteq \mathcal{A}$
	then		$\mathcal{A} \rightarrow_{\sqcap} \mathcal{A} \cup \{x : C_1, x : C_2\}$
$\rightarrow_{\sqcup}$:	**if**	1.	$x : C_1 \sqcup C_2 \in \mathcal{A}$ and
		2.	$\{x : C_1, x : C_2\} \cap \mathcal{A} = \emptyset$
	then		$\mathcal{A} \rightarrow_{\sqcap} \mathcal{A} \cup \{x : D\}$ for some $D \in \{C_1, C_2\}$
$\rightarrow_{\exists}$:	**if**	1.	$x : \exists R.D \in \mathcal{A}$ and
		2.	there is no y with $\{(x, y) : R, y : D\} \subseteq \mathcal{A}$
	then		$\mathcal{A} \rightarrow_{\exists} \mathcal{A} \cup \{(x, y) : R,\ y : D\}$ for a fresh individual y
$\rightarrow_{\forall}$:	**if**	1.	$x : \forall R.D \in \mathcal{A}$ and
		2.	there is a y with $(x, y) : R \in \mathcal{A}$ and $y : D \notin \mathcal{A}$
	then		$\mathcal{A} \rightarrow_{\forall} \mathcal{A} \cup \{y : D\}$

The $\mathcal{ALC}$-algorithm is a non-deterministic algorithm due to the $\rightarrow_{\sqcup}$-rule, which non-deterministically chooses which disjunct to add for a disjunctive concept. Also, we have not specified a precedence that determines which rule to apply if there is more than one possibility. To prove that such a non-deterministic algorithm is indeed a decision procedure for satisfiability of $\mathcal{ALC}$-concepts, we have to establish three things:

1. **Termination**, i.e., every sequence of rule-applications terminates after a finite number of steps.
2. **Soundness**, i.e., if the algorithm has generated a complete and clash-free ABox for C, then C is satisfiable.
3. **Completeness**, i.e., for a satisfiable concept C there is a sequence of rule applications that leads to a complete and clash-free ABox for C.

When dealing with non-deterministic algorithms, one can distinguish two different kinds of non-determinism, namely *don't-know* and *don't-care* non-determinism. Choices of an algorithm that may affect the result are called don't-know non-deterministic. For the $\mathcal{ALC}$-algorithm, the choice of which disjunct to add by the $\rightarrow_{\sqcup}$-rule is don't-know non-deterministic. When dealing with the initial ABox

$$\mathcal{A} = \{x_0 : A \sqcup (B \sqcap \neg B)\},$$

the algorithm will only find a clash-free completion of $\mathcal{A}$ if the $\rightarrow_{\sqcup}$-rule chooses to add the assertion $x_0 : A$. In this sense, adding $x_0 : A$ is a "good" choice while adding $x_0 : B \sqcap \neg B$ would be a "bad" choice because it prevents the discovery of a clash-free completion of $\mathcal{A}$ even though there is one. For a (necessarily deterministic) implementation of the $\mathcal{ALC}$-algorithm, this implies that exhaustive search over all possibilities of don't-know non-deterministic choices is required to obtain a complete algorithm.

Non-deterministic choices that don't effect the outcome of the algorithm in the sense that any choice is a "good" choice are called don't-care non-deterministic. Don't-care non-determinism is also (implicitly) present in the $\mathcal{ALC}$-algorithm. Even though in an ABox several rules might be applicable at the same time, the algorithm does not specify which rule to apply to which constraint in which order. On the contrary, it will turn out that, whenever a rule is applicable, it can be applied in a way that leads to the discovery of a complete and clash-free ABox for a satisfiable concept (Lemma 3.9). This implies that in case of a (deterministic) implementation of the $\mathcal{ALC}$-algorithm one is free to choose an arbitrary strategy which rule to apply where and when without sacrificing the completeness of the algorithm, although the efficiency of the implementation might depend on the choice of the employed strategy.

Termination

The general idea behind the termination proofs of the tableau algorithms we will encounter in this thesis is the following:

- The concepts and roles appearing in a constraint are taken from a finite set.
- Paths in the constraint system are of bounded length and every individual has a bounded number of successors.
- The application of a rule either adds a constraint for an individual already present in the constraint system, or it adds new individuals. No constraints or individuals are ever deleted, or, if deletion takes place, the number of deletions is bounded.

Together, this implies termination of the tableau algorithm since an infinite sequence of rule applications would either lead to an unbounded number of constraints for a single individual or to infinitely many individuals in the constraint system. Both stand in contradiction to the mentioned properties.

To prove the termination of the $\mathcal{ALC}$-algorithm, it is convenient to "extract" the underlying graph-structure from an ABox and to view it as an edge and node labelled graph.

Definition 3.3
Let $\mathcal{A}$ be an ABox. The graph $G_\mathcal{A}$ induced by $\mathcal{A}$ *is an edge and node labelled graph $G_\mathcal{A} = (V, E, \mathbf{L})$ defined by*

$$\begin{aligned} V &= \{x \in \mathsf{NI} \mid x \text{ occurs in } \mathcal{A}\}, \\ E &= \{(x,y) \mid (x,y) : R \in \mathcal{A}\}, \\ \mathbf{L}(x) &= \{D \mid x : D \in \mathcal{A}\}, \\ \mathbf{L}(x,y) &= \{R \mid (x,y) : R \in \mathcal{A}\}. \end{aligned}$$

◇

It is easy to see that, for any ABox $\mathcal{A}$ generated by a sequence of applications of the completion rules for $\mathcal{ALC}$ from an initial ABox $\{x_0 : C\}$, the induced graph $G_{\mathcal{A}}$ satisfies the following properties:

- $G_{\mathcal{A}}$ is a tree rooted at x_0.
- For any node $x \in V$, $\mathbf{L}(x) \subseteq sub(C)$.
- For any edge $(x, y) \in E$, $\mathbf{L}(x, y)$ is a singleton $\{R\}$ for a role R that occurs in C.

A proof of this properties can easily be given by induction on the number of rule applications and is left to the reader. Moreover, it is easy to show the following lemma that states that the graph generated by the $\mathcal{ALC}$-algorithm is bounded in the size of the input concept.

Lemma 3.4
Let C be an $\mathcal{ALC}$-concept in NNF and $\mathcal{A}$ an ABox generated by the $\mathcal{ALC}$-algorithm by a sequence of rule applications from the initial ABox $\{x_0 : C\}$. Then the following holds:

1. *For every node x, the size of $\mathbf{L}(x)$ is bounded by $|C|$.*
2. *The length of a directed path in $G_{\mathcal{A}}$ is bounded by $|C|$.*
3. *The out-degree of $G_{\mathcal{A}}$ is bounded by $|C|$.*

Proof.

1. For every node x, $\mathbf{L}(x) \subseteq sub(C)$. Hence, $|\mathbf{L}(x)| \leq |sub(C)| \leq |C|$.
2. For every node x we define $\ell(x)$ as the maximum nesting of existential or universal restrictions in a concept in $\mathbf{L}(x)$. Obviously, $\ell(x_0) \leq |C|$. Also, if $(x, y) \in E$, then $\ell(x) > \ell(y)$. Hence, any path $x_1, \ldots x_k$ in $G_{\mathcal{A}}$ induces a sequence $\ell(x_1) > \cdots > \ell(x_k)$ of non-negative integers. Since $G_{\mathcal{A}}$ is a tree rooted at x_0, the longest path starts with x_0 and is bounded by $|C|$.
3. Successors of a node x are only generated by an application of the $\rightarrow_{\exists}$-rule, which generates at most one successor for each concept of the form $\exists R.D$ in $\mathbf{L}(x)$. Together with (1), this implies that the out-degree is bounded by $|C|$. ∎

From this lemma, termination of the $\mathcal{ALC}$-algorithm is a simple corollary:

Corollary 3.5 (Termination)
Any sequence of rule-applications of the $\mathcal{ALC}$-algorithm terminates after a finite number of steps.

Proof. A sequence of rule-applications induces a sequence of trees whose depth and out-degree is bounded by the size of the input concept by Lemma 3.4. Moreover, every rule application adds a concept to the label of a node or adds a node to the tree. No nodes are ever deleted from the tree and no concepts are ever deleted from the label of a node.

Hence, an unbounded sequence of rule-applications would either lead to an unbounded number of nodes or to an unbounded label of one of the nodes. Both cases contradict Lemma 3.4. ∎

Soundness and Completeness

Soundness and completeness of a tableau algorithm is usually proved by establishing the following properties of the algorithm based on an appropriate notion of satisfiability of constraint systems, which is tailored for the needs of every specific DL and tableau algorithm.

1. A constraint system that contains a clash is necessarily unsatisfiable.
2. The initial constraint system is satisfiable iff the input concept (or knowledge base) is satisfiable.
3. A complete and clash-free constraint systems is satisfiable.
4. For every applicable deterministic rule, its application preserves satisfiability of the constraint systems. For every applicable non-deterministic rule, there is a way of applying the rule that preserves satisfiability.
5. For every rule, no satisfiable constraint system can be generated from an unsatisfiable one, or, alternatively,

5'. a complete and clash-free constraint system implies satisfiability of the initial constraint system.

Property 4 and 5 together are often referred to as *local correctness* of the rules.

Theorem 3.6 (Generic Correctness of Tableau Algorithms)
A terminating tableau algorithm that satisfies the properties mentioned above is correct.

Proof. Termination is required as a precondition of the theorem. The tableau algorithm is sound because a complete and clash-free constraint system is satisfiable (Property 3) which implies satisfiability of the initial constraint system (either by Property 5 and induction over the number of rule applications of directly by Property 5') and hence (by Property 2) the satisfiability of the input concept (or knowledge base).

It is complete because, given a satisfiable input concept (or knowledge base), the initial constraint system is satisfiable (Property 2). Each rule can be applied in a way that maintains the satisfiability of the constraint system (Property 4) and, since the algorithm terminates, any sequence of rule-applications is finite. Hence, after finitely many steps a satisfiable and complete constraint system can be derived from the initial one. This constraint system must be clash-free because (by Property 1) a clash would imply unsatisfiability. ■

Specifically, for $\mathcal{ALC}$ we use the usual notion of satisfiability of ABoxes. Clearly, for a satisfiable concept C, the initial ABox $\{x_0 : C\}$ is satisfiable and a clash in an ABox implies unsatisfiability.

It remains to prove that a complete and clash-free ABox is satisfiable and that the rules preserve satisfiability in the required manner. The following definition extracts a model from a complete and clash-free ABox.

Definition 3.7 (Canonical Interpretation)
For an ABox $\mathcal{A}$, the canonical interpretation $\mathcal{I}_\mathcal{A} = (\Delta^{\mathcal{I}_\mathcal{A}}, \cdot^{\mathcal{I}_\mathcal{A}})$ *is defined by*

$$\begin{aligned}
\Delta^{\mathcal{I}_\mathcal{A}} &= \{x \in \mathsf{NI} \mid x \textit{ occurs in } \mathcal{A}\},\\
A^{\mathcal{I}_\mathcal{A}} &= \{x \mid x : A \in \mathcal{A}\} \quad \textit{for every } A \in \mathsf{NC},\\
R^{\mathcal{I}_\mathcal{A}} &= \{(x,y) \mid (x,y) : R \in \mathcal{A}\} \quad \textit{for every } R \in \mathsf{NR},\\
x^{\mathcal{I}_\mathcal{A}} &= x \quad \textit{for every individual } x \textit{ that occurs in } \mathcal{A}.
\end{aligned}$$

◇

Lemma 3.8
Let $\mathcal{A}$ be a complete and clash-free ABox. Then $\mathcal{A}$ has a model.

Proof. It is obvious that, for an arbitrary ABox $\mathcal{A}$, the canonical interpretation satisfies all assertion of the form $(x, y) : R \in \mathcal{A}$. $\mathcal{A}$ does not contain any assertions of the form $x \dot{\neq} y$.

By induction on the structure of concepts occurring in $\mathcal{A}$, we show that the canonical interpretation $\mathcal{I}_\mathcal{A}$ satisfies any assertion of the form $x : D \in \mathcal{A}$ and hence is a model of $\mathcal{A}$.

- For the base case $x : A$ with $A \in \mathsf{NC}$, this holds by definition of $\mathcal{I}_\mathcal{A}$.
- For the case $x : \neg A$, since $\mathcal{A}$ is clash free, $x : A \notin \mathcal{A}$ and hence $x \notin A^{\mathcal{I}_\mathcal{A}}$.
- If $x : C_1 \sqcap C_2 \in \mathcal{A}$, then, since $\mathcal{A}$ is complete, also $\{x : C_1, x : C_2\} \subseteq \mathcal{A}$. By induction this implies $x \in C_1^{\mathcal{I}_\mathcal{A}}$ and $x \in C_2^{\mathcal{I}_\mathcal{A}}$ and hence $x \in (C_1 \sqcap C_2)^{\mathcal{I}_\mathcal{A}}$.

- If $x : C_1 \sqcup C_2 \in \mathcal{A}$, then, again due the completeness of $\mathcal{A}$, either $x : C_1 \in \mathcal{A}$ or $x : C_2 \in \mathcal{A}$. By induction this yields $x \in C_1^{\mathcal{I}_\mathcal{A}}$ or $x \in C_2^{\mathcal{I}_\mathcal{A}}$ and hence $x \in (C_1 \sqcup C_2)^{\mathcal{I}_\mathcal{A}}$.
- If $x : \exists R.D \in \mathcal{A}$, then completeness yields $\{(x, y) : R, y : D\} \subseteq \mathcal{A}$ for some y. By construction of $\mathcal{I}_\mathcal{A}$, $(x, y) \in R^{\mathcal{I}_\mathcal{A}}$ holds and by induction we have $y \in D^{\mathcal{I}_\mathcal{A}}$. Together this implies $x \in (\exists R.D)^{\mathcal{I}_\mathcal{A}}$.
- If $x : \forall R.D \in \mathcal{A}$, then, for any y with $(x, y) \in R^{\mathcal{I}_\mathcal{A}}$, $(x, y) : R \in \mathcal{A}$ must hold due to the construction of $\mathcal{I}_\mathcal{A}$. Then, due to completeness, $y : D \in \mathcal{A}$ must hold and induction yields $y \in D^{\mathcal{I}_\mathcal{A}}$. Since this holds for any such y, $x \in (\forall R.D)^{\mathcal{I}_\mathcal{A}}$. ∎

Lemma 3.9 (Local Correctness)

1. *If $\mathcal{A}$ is an ABox and $\mathcal{A}'$ is obtained from $\mathcal{A}$ by an application of a completion rule, then satisfiability of $\mathcal{A}'$ implies satisfiability of $\mathcal{A}$.*
2. *If $\mathcal{A}$ is satisfiable and $\mathcal{A}'$ is obtained from $\mathcal{A}$ by an application of a deterministic rule, then $\mathcal{A}'$ is satisfiable.*
3. *If $\mathcal{A}$ is satisfiable and the $\rightarrow_\sqcup$-rule is applicable, then there is a way of applying the $\rightarrow_\sqcup$-rule such that the obtained ABox $\mathcal{A}'$ is satisfiable.*

Proof.

1. Since $\mathcal{A}$ is a subset of $\mathcal{A}'$, satisfiability of $\mathcal{A}'$ immediately implies satisfiability of $\mathcal{A}$.
2. Let $\mathcal{I}$ be a model of $\mathcal{A}$. We distinguish the different rules:
   - The application of the $\rightarrow_\sqcap$-rule is triggered by an assertion $x : C_1 \sqcap C_2 \in \mathcal{A}$. Since $x^\mathcal{I} \in (C_1 \sqcap C_2)^\mathcal{I}$, also $x^\mathcal{I} \in C_1^\mathcal{I} \cap C_2^\mathcal{I}$. Hence, $\mathcal{I}$ is also a model for $\mathcal{A}' = \mathcal{A} \cup \{x : C_1, x : C_2\}$.
   - The $\rightarrow_\exists$-rule is applied due to an assertion $x : \exists R.D \in \mathcal{A}$. Since $\mathcal{I}$ is a model of $\mathcal{A}$, there exists an $a \in \Delta^\mathcal{I}$ with $(x^\mathcal{I}, a) \in R^\mathcal{I}$ and $a \in D^\mathcal{I}$. Hence, the interpretation $\mathcal{I}[y \mapsto a]$, which maps y to a and behaves like $\mathcal{I}$ on all other names, is a model of $\mathcal{A}' = \mathcal{A} \cup \{(x, y) : R, y : D\}$. Note, that this requires y to be fresh.
   - The $\rightarrow_\forall$-rule is applied due to an assertions $\{x : \forall R.D, (x, y) : R\} \subseteq \mathcal{A}$. Since $\mathcal{I} \models \mathcal{A}$, $y^\mathcal{I} \in D^\mathcal{I}$ must hold. Hence, $\mathcal{I}$ is also a model of $\mathcal{A}' = \mathcal{A} \cup \{y : D\}$.
3. Again, let $\mathcal{I}$ be a model of $\mathcal{A}$. If an assertion $x : C_1 \sqcup C_2$ triggers the application of the $\rightarrow_\sqcup$-rule, then $x^\mathcal{I} \in (C_1 \sqcup C_2)^\mathcal{I}$ must hold. Hence, at least for one of the possible choices for $D \in \{C_1, C_2\}$, $x^\mathcal{I} \in D^\mathcal{I}$ holds. For this choice, adding $x : D$ to $\mathcal{A}$ leads to an ABox that is satisfied by $\mathcal{I}$. ∎

Theorem 3.10 (Correctness of the $\mathcal{ALC}$-algorithm)
The $\mathcal{ALC}$-algorithm is a non-deterministic decision procedure for satisfiability of $\mathcal{ALC}$-concepts.

Proof. Termination was shown in Corollary 3.5. In Lemma 3.8 and Lemma 3.9, we have established the conditions required to apply Theorem 3.6, which yields correctness of the $\mathcal{ALC}$-algorithm. ∎

3.2.2 Complexity

Now that we know that the $\mathcal{ALC}$-algorithm is a non-deterministic decision procedure for satisfiability of $\mathcal{ALC}$-concepts, we want to analyse the computational complexity of the algorithm to make sure that it matches the known worst-case complexity of the problem.

Some Basics from Complexity Theory

First, we briefly introduce the notions from complexity theory that we will encounter in this thesis. For a thorough introduction to complexity theory we refer to (Papadimitriou, 1994).

Let M be a Turing Machine (TM) with input alphabet Σ. For a function $f : \mathbb{N} \to \mathbb{N}$, we say that M *operates within time* $f(n)$ if, for any input string $x \in \Sigma^*$, M terminates on input x after at most $f(|x|)$ steps, where $|x|$ denotes the length of x. M *operates within space* $f(n)$ if, for any input $x \in \Sigma^*$, M requires space at most $f(|x|)$. For an arbitrary function $f(n)$ we define the following classes of languages:

$$\begin{aligned}
\textsc{Time}(f(n)) &= \{L \subseteq \Sigma^* \mid L \text{ is decided by a deterministic TM that operates within time } f(n)\},\\
\textsc{NTime}(f(n)) &= \{L \subseteq \Sigma^* \mid L \text{ is decided by a non-deterministic TM that operates within time } f(n)\},\\
\textsc{Space}(f(n)) &= \{L \subseteq \Sigma^* \mid L \text{ is decided by a deterministic TM that operates within space } f(n),\\
\textsc{NSpace}(f(n)) &= \{L \subseteq \Sigma^* \mid L \text{ is decided by a non-deterministic TM that operates within space } f(n)\}.
\end{aligned}$$

Since every deterministic TM is a non-deterministic TM, $\textsc{Time}(f(n)) \subseteq \textsc{NTime}(f(n))$ and $\textsc{Space}(f(n)) \subseteq \textsc{NSpace}(f(n))$ hold trivially for an arbitrary function f. Also, $\textsc{Time}(f(n)) \subseteq \textsc{Space}(f(n))$ and $\textsc{NTime}(f(n)) \subseteq \textsc{NSpace}(f(n))$ hold trivially for an arbitrary f because within time $f(n)$ a TM can consume at most $f(n)$ units of space.

In this thesis, we will encounter complexity classes shown in Figure 3.2.

It is known that the following relationships hold for these classes:

$$\textsc{PSpace} = \textsc{NPSpace} \subseteq \textsc{ExpTime} \subseteq \textsc{NExpTime} \subseteq 2\text{-}\textsc{ExpTime} \subseteq 2\text{-}\textsc{NExpTime},$$

where the fact that $\textsc{PSpace} = \textsc{NPSpace}$ is a corollary of Savitch's theorem (Savitch, 1970).

We employ the usual notion of polynomial many-to-one reductions and completeness: let $L_1, L_2 \subseteq \Sigma^*$ be two languages. A function $r : \Sigma^* \to \Sigma^*$ is a *polynomial reduction from* L_1 *to* L_2 iff there exists a $k \in \mathbb{N}$ such that $r(x)$ can be compute within time $\mathcal{O}(|x|^k)$ and $x \in L_1$ iff $r(x) \in L_2$. A language L is *hard* for a complexity class $\mathcal{C}$ if, for any $L' \in \mathcal{C}$,

Figure 3.2 Some complexity classes

$$\begin{aligned}
\textsc{PSpace} &= \bigcup_{k\in\mathbb{N}} \textsc{Space}(n^k)\\
\textsc{NPSpace} &= \bigcup_{k\in\mathbb{N}} \textsc{NSpace}(n^k)\\
\textsc{ExpTime} &= \bigcup_{k\in\mathbb{N}} \textsc{Time}(2^{n^k})\\
\textsc{NExpTime} &= \bigcup_{k\in\mathbb{N}} \textsc{NSpace}(2^{n^k})\\
2\text{-}\textsc{ExpTime} &= \bigcup_{k\in\mathbb{N}} \textsc{Time}(2^{2^{n^k}})\\
2\text{-}\textsc{NExpTime} &= \bigcup_{k\in\mathbb{N}} \textsc{NTime}(2^{2^{n^k}})
\end{aligned}$$

there exists a polynomial reduction from L' to L. The language L is *complete* for $\mathcal{C}$ if it is $\mathcal{C}$-hard and $L \in \mathcal{C}$.

All these definitions are dependent on the arbitrary but fixed finite input alphabet Σ. The choice of this alphabet is inessential as long as it contains at least two symbols. This allows for succinct encoding of arbitrary problems and a larger input alphabet can reduce the size of the encoding of a problem only by a polynomial amount. All defined complexity classes are insensitive to these changes. From now on, we assume an arbitrary but fixed finite input alphabet Σ with at least two symbols.

Note that this implies that there is not necessarily a distinct symbol for every concept, role, or individual name in Σ. Instead, we assume that the names appearing in concepts are suitably numbered. The results we are going to present are insensitive to this (logarithmic) overhead and so we ignore this issue from now on.

Definition 3.11
For an arbitrary syntactic entity X, like a concept, TBox assertion, knowledge base, etc., we denote the length of a suitable encoding of X in the alphabet Σ with $|X|$. ◇

The Complexity of $\mathcal{ALC}$-Satisfiability

Fact 3.12 (Schmidt-Schauß & Smolka, 1991, Theorem 6.3)
Satisfiability of $\mathcal{ALC}$-concepts is PSpace*-complete.*

Since we are aiming for a PSpace-algorithm, we do not have to deal explicitly with the non-determinism because PSpace = NPSpace. Yet, if naively executed, the algorithm behaves worse because it generates a model for a satisfiable concept and there are $\mathcal{ALC}$-concepts that are only satisfiable in exponentially large interpretations, i.e., it is possible to give a concept C_n of size polynomially in n such that any model of C_n essentially contains a full binary tree of depth n and hence at least $2^n - 1$ nodes (Halpern & Moses, 1992). Since the tableau generates a full description of a model, a naive implementation would require exponential space.

To obtain an algorithm with optimal worst case complexity, the $\mathcal{ALC}$-algorithm has to be implemented in a certain fashion using the so-called *trace technique*. The key idea behind this technique is that instead of keeping the full ABox $\mathcal{A}$ in memory simultaneously, it is sufficient to consider only a single path in $G_\mathcal{A}$ at one time. In Lemma 3.4 we have seen that the length of such a path is linearly bounded in the size of the input concept and there are only linearly many constraints for every node on such a path. Hence, if it is possible to explore $G_\mathcal{A}$ one path at a time, then polynomial storage suffices. This can be achieved by a depth-first expansion of the ABox that selects the rule to apply in a given situation according to a specific strategy (immediately stopping with the output "unsatisfiable" if a clash is generated).

Figure 3.3 A non-deterministic PSPACE decision procedure for $\mathcal{ALC}$.

$\mathcal{ALC}$-SAT(C) := sat$(x_0, \{x_0 : C\})$
sat$(x, \mathcal{A})$:
  while (the $\rightarrow_\sqcap$- or the $\rightarrow_\sqcup$-rule can be applied) and ($\mathcal{A}$ is clash-free) do
    apply the $\rightarrow_\sqcap$- or the $\rightarrow_\sqcup$-rule to $\mathcal{A}$.
  od
  if $\mathcal{A}$ contains a clash then return "not satisfiable".
  $\mathbf{E} := \{x : \exists R.D \mid x : \exists R.D \in \mathcal{A}\}$
  while $\mathbf{E} \neq \emptyset$ do
    pick an arbitrary $x : \exists R.D \in \mathbf{E}$
    $\mathcal{A}_{new} := \{(x, y) : R, y : D\}$ where y is a fresh individual
    while (the $\rightarrow_\forall$-rule can be applied to $\mathcal{A} \cup \mathcal{A}_{new}$) do
      apply the $\rightarrow_\forall$-rule and add the new constraints to $\mathcal{A}_{new}$
    od
    if $\mathcal{A} \cup \mathcal{A}_{new}$ contains a clash then return "not satisfiable".
    if sat$(y, \mathcal{A} \cup \mathcal{A}_{new})$ = "not satisfiable" then return "not satisfiable"
    $\mathbf{E} := \mathbf{E} \setminus \{x : \exists R.D \mid y : D \in \mathcal{A}_{new}\}$
    discard $\mathcal{A}_{new}$ from memory
  od
  return "satisfiable"

Lemma 3.13
The $\mathcal{ALC}$-algorithm can be implemented in PSPACE.

Proof. Let C be the $\mathcal{ALC}$-concept to be tested for satisfiability. We can assume C to be in NNF because transformation into NNF can be performed in linear time. Figure 3.3 sketches an implementation of the $\mathcal{ALC}$-algorithm that uses the trace-technique to preserve memory and runs in polynomial space.

The algorithm generates the constraint system in a depth-first manner: before generating any successors for an individual x, the $\rightarrow_\sqcap$- and $\rightarrow_\sqcup$-rule are applied exhaustively. Then successors are considered for every existential restriction in $\mathcal{A}$ one after another re-using space. This has the consequence that a clash involving an individual x must be present in $\mathcal{A}$ by the time generation of successors for x is initiated or will never occur. This also implies that it is safe to delete parts of the constraint system for a successor y as soon as the existence of a complete and clash-free "sub" constraint system has been determined. Of course, it then has to be ensured that we do not consider the same existential restriction $x : \exists R.D$ more than once because this might lead to non-termination. Here, we do this using the set $\mathbf{E}$ that records which constraints still have to be considered. Hence, the algorithm is indeed an implementation of the $\mathcal{ALC}$-algorithm.

Space analysis of the algorithm is simple: since $\mathcal{A}_{new}$ is reset for every successor that is generated, this algorithm stores only a single path at any given time, which, by Lemma 3.4, can be done using polynomial space only. ∎

As a corollary, we get an exact classification of the complexity of satisfiability of $\mathcal{ALC}$-concepts.

Theorem 3.14
Satisfiability of $\mathcal{ALC}$-concepts is PSPACE*-complete.*

Proof. Satisfiability of $\mathcal{ALC}$-concepts is known to be PSPACE-hard (Schmidt-Schauß & Smolka, 1991), which is shown by reduction from the well-known PSPACE-complete problem QBF (Stockmeyer & Meyer, 1973). Lemma 3.13 together with the fact that PSPACE = NPSPACE (Savitch's theorem (1970)) yields the corresponding upper complexity bound. ∎

It is possible to give an even tighter bound for the complexity of $\mathcal{ALC}$-concept satisfiability and to show that the problem is solvable in deterministic linear space. This was already claimed in (Schmidt-Schauß & Smolka, 1991), but a closer inspection of that algorithm by Hemaspaandra reveals that it consumes memory in the order of $\mathcal{O}(n \log n)$ for a concept with length $|C| = n$. Hemaspaandra (2000) gives an algorithm that decides satisfiability for the modal logic K in deterministic linear space and which is easily applicable to $\mathcal{ALC}$.

3.2.3 Other Inference Problems for $\mathcal{ALC}$

Concept satisfiability is only one inference that is of interest for DL systems. In the remainder of this chapter we give a brief overview of solutions for the other standard inferences for $\mathcal{ALC}$.

Reasoning with ABoxes

To decide ABox satisfiability of an $\mathcal{ALC}$-ABox $\mathcal{A}$ (w.r.t. an empty TBox), one can simply apply the $\mathcal{ALC}$-algorithm starting with $\mathcal{A}$ as the initial ABox. One can easily see that the proofs of soundness and completeness uniformly apply also to this case. Yet, since the generated constraint system is no longer of tree-shape, termination and complexity have to be reconsidered. Hollunder (1996) describes *pre-completion*—a technique that allows reduction of ABox satisfiability directly to $\mathcal{ALC}$-concept satisfiability. The general idea is as follows: all non-generating rules are applied to the input ABox $\mathcal{A}$ exhaustively yielding a pre-completion $\mathcal{A}'$ of $\mathcal{A}$. After that, the $\mathcal{ALC}$-algorithm is called for every individual x of $\mathcal{A}'$ to decide satisfiability of the concept

$$C_x := \bigsqcap_{x:D \in \mathcal{A}'} D.$$

It can be shown that $\mathcal{A}'$ is satisfiable iff C_x is satisfiable for every individual x in $\mathcal{A}'$ and that $\mathcal{A}$ is satisfiable iff the non-generating rules can be applied in a way that yields a satisfiable pre-completion. Since ABox satisfiability is at least as hard as concept satisfiability, we get:

Corollary 3.15
Consistency of $\mathcal{ALC}$-ABoxes w.r.t. an empty TBox is PSPACE*-complete.*

Reasoning with Simple TBoxes

For a simple TBox $\mathcal{T}$, concept satisfiability w.r.t. $\mathcal{T}$ can be reduced to concept satisfiability by a process called *unfolding*:

Let C be an $\mathcal{ALC}$-concept and $\mathcal{T}$ a simple TBox. The *unfolding $C_\mathcal{T}$ of C w.r.t. $\mathcal{T}$* is obtained by successively replacing every defined name in C by its definition from $\mathcal{T}$ until only primitive (i.e., undefined) names occur. It can easily be shown that C is satisfiable w.r.t. $\mathcal{T}$ iff $C_\mathcal{T}$ is satisfiable. Unfortunately, this does not yield a PSPACE-algorithm, as the size of $C_\mathcal{T}$ may be exponential in the size of C and $\mathcal{T}$. Lutz (1999) describes a technique called *lazy unfolding* that performs the unfolding of C w.r.t. $\mathcal{T}$ *on demand*, which yields:

Fact 3.16 (Lutz, 1999, Theorem 1)
Satisfiability of $\mathcal{ALC}$-concepts w.r.t. to a simple TBox is PSPACE*-complete.*

Finally, the techniques of pre-completion and lazy-unfolding can be combined, which yields:

Corollary 3.17
Consistency of $\mathcal{ALC}$ knowledge bases with a simple TBox is PSPACE*-complete.*

Reasoning with General TBoxes

If general TBoxes are considered instead of simple ones, the complexity of the inference problems rises.

Theorem 3.18
Satisfiability of $\mathcal{ALC}$-concepts (and hence of ABoxes) w.r.t. general TBoxes is ExpTime-*hard.*

Proof. As mentioned before, $\mathcal{ALC}$ is a syntactic variant of the propositional modal logic K_m (Schild, 1991). As a simple consequence of the proof of ExpTime-completeness of K with a universal modality (Spaan, 1993a)] (i.e., in DL terms, a role linking every two individuals), one obtains that the *global satisfaction problem* for K is an ExpTime-complete problem. The global satisfaction problem is defined as follows:

> Given a K-formula ϕ, is there a Kripke model $\mathfrak{M}$ such that ϕ holds at every world in $\mathfrak{M}$?

Using the correspondence between $\mathcal{ALC}$ and K_m, this can be re-stated as an ExpTime-complete problem for $\mathcal{ALC}$:

> Given an $\mathcal{ALC}$-concept C, is there an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$?

Obviously, this holds iff the tautological concept $\top$ is satisfiable w.r.t. the (non-simple) TBox $\mathcal{T} = \{\top \doteq C\}$, which implies that satisfiability of $\mathcal{ALC}$-concepts (and hence of ABoxes) w.r.t. general TBoxes is ExpTime-hard. ∎

A matching upper bound for $\mathcal{ALC}$ is given by De Giacomo and Lenzerini (1996) by a reduction to PDL, which yields:

Corollary 3.19
Satisfiability and subsumption w.r.t. general TBoxes, knowledge base satisfiability and instance checking for $\mathcal{ALC}$ are ExpTime-*complete problems.*

Chapter 4

Qualifying Number Restrictions

In this chapter we study the complexity of reasoning with $\mathcal{ALCQ}$, the extension of $\mathcal{ALC}$ with qualifying number restrictions. While for $\mathcal{ALC}$, or, more precisely, for its syntactic variant K, PSPACE-completeness has already been established quite some time ago by Ladner (1977), the situation is entirely different for $\mathcal{ALCQ}$ or its corresponding (multi-)modal logic $\mathbf{Gr}(\mathsf{K}_{\mathcal{R}})$. For $\mathcal{ALCQ}$, decidability of concept satisfiability has been shown only rather recently by Baader and Hollunder (1991) and the known PSPACE upper complexity bound for $\mathcal{ALCQ}$ is only valid if we assume unary coding of numbers in the input, which is an unnatural restriction. For binary coding no upper bound was known and the problem had been conjectured to be EXPTIME-hard by van der Hoek and de Rijke (1995). This coincides with the observation that a straightforward adaptation of the translation technique leads to an exponential blow-up in the size of the first-order formula. This is because it is possible to store the number n in $\log_k n$ bits if numbers are represented in k-ary coding.

We show that reasoning for $\mathcal{ALCQ}$ is not harder than reasoning for $\mathcal{ALC}$ (w.r.t. worst-case complexity) by presenting an algorithm that decides satisfiability in PSPACE, even if the numbers in the input are binary coded. It is based on the tableau algorithm for $\mathcal{ALC}$ and tries to prove the satisfiability of a given concept by explicitly constructing a model for it. When trying to generalise the tableau algorithms for $\mathcal{ALC}$ to deal with $\mathcal{ALCQ}$, there are some difficulties: (1) the straightforward approach leads to an incorrect algorithm; (2) even if this pitfall is avoided, special care has to be taken in order to obtain a space-efficient solution. As an example for (1), we will show that the algorithm presented in (van der Hoek & de Rijke, 1995) to decide satisfiability of $\mathbf{Gr}(\mathsf{K}_{\mathcal{R}})$, a syntactic variant of $\mathcal{ALCQ}$, is incorrect. Nevertheless, this algorithm will be the basis of our further considerations. Problem (2) is due to the fact that tableau algorithms try to prove the satisfiability of a concept by explicitly building a model for it. If the tested formula requires the existence of n accessible role successors, a tableau algorithm will include them in the constructed model, which leads to exponential space consumption, at least if the numbers in the input are not unarily coded or memory is not re-used. An example for a correct algorithm which suffers from this problem can be found in (Hollunder & Baader, 1991) and is briefly presented in this thesis. As we will see, the trace technique alone is not sufficient to obtain an algorithm that runs in polynomial space. Our algorithm overcomes this additional problem by organising the search for a model in a way that allows for the re-use of space *for each successor*, thus being capable of deciding satisfiability of $\mathcal{ALCQ}$ in PSPACE.

Using an extension of these techniques we obtain a PSPACE algorithm for the logic $\mathcal{ALCQIb}$, which extends $\mathcal{ALCQ}$ by expressive role expressions. This solves an open problem from (Donini et al., 1997).

Finally, we study the complexity of reasoning w.r.t. general knowledge bases for $\mathcal{ALCQIb}$ and establish EXPTIME-completeness. This extends the EXPTIME-completeness result for the more "standard" DL $\mathcal{ALCQI}$ (De Giacomo, 1995). Moreover, the proof in (De Giacomo, 1995) is only valid in case of unary coding of numbers in the input whereas our proof also applies in the case of binary coding.

4.1 Syntax and Semantics of $\mathcal{ALCQ}$

The DL $\mathcal{ALCQ}$ is obtained from $\mathcal{ALC}$ by adding so-called *qualifying number restrictions*, i.e., concepts restricting the number of individuals that are related via a given role instead of allowing for existential or universal restrictions only like in $\mathcal{ALC}$. $\mathcal{ALCQ}$ is a syntactic variant of the graded propositional modal logic $\mathbf{Gr}(\mathsf{K}_{\mathcal{R}})$.

Definition 4.1 (Syntax of $\mathcal{ALCQ}$)
Let NC *be a set of atomic* concept names *and* NR *be a set of atomic* role names. *The set of* $\mathcal{ALCQ}$-concepts *is built inductively from these according to the following grammar, where* $A \in \mathsf{NC}$, $R \in \mathsf{NR}$, *and* $n \in \mathbb{N}$:

$$C ::= A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall R.C \mid \exists R.C \mid ({\leq} n\ R\ C) \mid ({\geq} n\ R\ C).$$

◇

Thus, the set of $\mathcal{ALCQ}$-concepts is defined similar to the set of $\mathcal{ALC}$-concepts, with the additional rule that, if $R \in \mathsf{NR}$, C is an $\mathcal{ALCQ}$-concept, and $n \in \mathbb{N}$, then also $({\leq} n\ R\ C)$ and $({\geq} n\ R\ C)$ are $\mathcal{ALCQ}$-concepts. To define the semantics of $\mathcal{ALCQ}$-concepts, we extend Definition 2.2 to deal with these additional concept constructors:

Definition 4.2 (Semantics of $\mathcal{ALCQ}$)
For an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, *the semantics of* $\mathcal{ALCQ}$*-concepts is defined inductively as for* $\mathcal{ALC}$*-concepts with the additional rules:*

$$({\leq} n\ R\ C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp R^{\mathcal{I}}(x, C) \leq n\} \text{ and}$$
$$({\geq} n\ R\ C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp R^{\mathcal{I}}(x, C) \geq n\},$$

where $\sharp R^{\mathcal{I}}(x, C) = \{y \mid (x, y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$ *and* $\sharp$ *denotes set cardinality.* ◇

From these semantics, it is immediately clear that we can dispose of existential and universal restriction in the syntax without changing the expressiveness of $\mathcal{ALCQ}$, since the following equivalences allow the elimination of universal and existential restrictions in linear time:

$$\exists R.C \equiv (\geq 1\ R\ C) \qquad \forall R.C \equiv (\leq 0\ R\ \neg C)$$

In the following, we assume that $\mathcal{ALCQ}$-concepts are built without existential or universal restrictions. To obtain the NNF of an $\mathcal{ALCQ}$-concept, one can "apply" the following equivalences (in addition to de Morgan's laws):

$$\neg(\leq n\ R\ C) \equiv (\geq (n+1)\ R\ C)$$
$$\neg(\geq n\ R\ C) \equiv \begin{cases} \bot & \text{if } n = 0, \\ (\geq (n+1)\ R\ C) & \text{otherwise.} \end{cases}$$

Like for $\mathcal{ALC}$, one can obtain the NNF of an $\mathcal{ALCQ}$-concept in linear time. For an $\mathcal{ALCQ}$-concept C, we denote the NNF of $\neg C$ by $\sim C$.

4.2 Counting Pitfalls

Before we present our algorithm for deciding satisfiability of $\mathcal{ALCQ}$, for historic and didactic reasons, we present two other solutions: an incorrect one (van der Hoek & de Rijke, 1995), and a solution that is less efficient (Hollunder & Baader, 1991).

4.2.1 An Incorrect Solution

Van der Hoek and de Rijke (1995) give an algorithm for deciding satisfiability of the graded modal logic $\mathbf{Gr}(\mathsf{K}_\mathcal{R})$. Since $\mathbf{Gr}(\mathsf{K}_\mathcal{R})$ is a notational variant of $\mathcal{ALCQ}$, such an algorithm could also be used to decide concept satisfiability for $\mathcal{ALCQ}$. Unfortunately, the given algorithm is incorrect. Nevertheless, it will be the basis for our further considerations and thus it is presented here. It will be referred to as the *incorrect* algorithm. It is based on a tableau algorithm given in (Donini et al., 1997) to decide the satisfiability of the DL $\mathcal{ALCN}$, but overlooks an important pitfall that distinguishes reasoning for qualifying number restrictions from reasoning with number restrictions. This mistake leads to the incorrectness of the algorithm. To fit our presentation, we use DL syntax in the presentation of the algorithm. Refer to (Tobies, 1999b) for a presentation in modal syntax.

Similar to the $\mathcal{ALC}$-algorithm presented in Section 2.1, the flawed solution is a tableau algorithm that tries to build a model for a concept C by manipulating sets of constraints with certain completion rules. Again, ABoxes are used to capture constraint systems.

Algorithm 4.3 (Incorrect Algorithm for $\mathcal{ALCQ}$, van der Hoek & de Rijke, 1995) *For an ABox $\mathcal{A}$, a role name R, an individual x, and a concept D, let $\sharp R^{\mathcal{A}}(x, D)$ be the number of individuals y for which $\{(x, y) : R, y : D\} \subseteq \mathcal{A}$. The ABox $[z/y]\mathcal{A}$ is obtained* *from $\mathcal{A}$ by replacing every occurrence of y by z; this replacement is said to be* safe *iff, for every individual x, concept C, and role name R with $\{x : (\geq n\ R\ D), (x, y) : R, (x, z) : R\} \subseteq S$ we have $\sharp R^{[z/y]\mathcal{A}}(x, D) > n$.*

The definition of a clash is slightly extended from the $\mathcal{ALC}$-case to deal with obviously contradictory number restrictions: An ABox $\mathcal{A}$ is said to contain a clash*, iff*

$$\{x : A, x : \neg A\} \subseteq \mathcal{A} \ \text{ or } \ \{x : (\leq m\ R\ D), x : (\geq n\ R\ D)\} \subseteq \mathcal{A}.$$

for a concept name A, a concept D, and two integers $m < n$. Otherwise, $\mathcal{A}$ is called clash-free*. An ABox $\mathcal{A}$ is called* complete *iff none of the rules given in Fig. 4.1 is applicable to $\mathcal{A}$.*

To test the satisfiability of a concept C, the incorrect algorithm works as follows: it starts with the initial ABox $\{x_0 : C\}$ and successively applies the rules given in Fig. 4.1, stopping when a clash occurs. Both the rule to apply and the concept to add (in the $\rightarrow_\sqcup$-rule) or the individuals to identify (in the $\rightarrow_\leq$-rule) are selected non-deterministically. The algorithm answers "C is satisfiable" iff the rules can be applied in a way that yields a complete and clash-free ABox. ◇

The notion of *safe* replacement of variables is needed to ensure the termination of the rule application (see Hollunder & Baader, 1991). The same purpose could be achieved by explicitly asserting all successors generated to satisfy an at-least restriction to be unequal and preventing the identification of unequal elements. Yet, since this notion of safe replacement recurs in the algorithm of Baader and Hollunder (1991), which we are going to describe later on, and since we want to outline an error in the incorrect algorithm, we stay as close to the original description as possible.

Since we are interested in PSPACE algorithms, as for $\mathcal{ALC}$, non-determinism poses no problem due to Savitch's Theorem, which implies that deterministic and non-deterministic polynomial space coincide (Savitch, 1970).

As described in Section 2.1, to prove the correctness of such a tableau algorithm, we need to show three properties of the completion:

1. Termination: Any sequence of rule applications is finite.
2. Soundness: If the algorithm terminates with a complete and clash-free ABox $\mathcal{A}$, then the tested concept is satisfiable.
3. Completeness: If the concept is satisfiable, then there is a sequence of rule applications that yields a complete and clash-free ABox.

The error of the incorrect algorithm is, that is does not satisfy Property 2, even though the opposite is claimed:

> CLAIM (van der Hoek & de Rijke, 1995): (*Restated in DL terminology*) Let C be an $\mathcal{ALCQ}$-concept in NNF. C is satisfiable iff $\{x_0 : C\}$ can be transformed into a clash-free complete ABox using the rules from Figure 4.1.

Figure 4.1 The incorrect completion rules for $\mathcal{ALCQ}$

$\rightarrow_\sqcap$:	**if** 1.	$x : C_1 \sqcap C_2 \in \mathcal{A}$ and
	2.	$\{x : C_1, x : C_2\} \not\subseteq \mathcal{A}$
	then	$\mathcal{A} \rightarrow_\sqcap \mathcal{A} \cup \{x : C_1, x : C_2\}$
$\rightarrow_\sqcup$:	**if** 1.	$x : C_1 \sqcup C_2 \in \mathcal{A}$ and
	2.	$\{x : C_1, x : C_2\} \cap \mathcal{A} = \emptyset$
	then	$\mathcal{A} \rightarrow_\sqcup \mathcal{A} \cup \{x : D\}$ for some $D \in \{C_1, C_2\}$
$\rightarrow_\geq$:	**if** 1.	$x : (\geq n\ R\ D)$ and
	2.	$\sharp R^{\mathcal{A}}(x, D) < n$
	then	$\mathcal{A} \rightarrow_\geq \mathcal{A} \cup \{(x, y) : R, y : D\}$ where y is a fresh variable.
$\rightarrow_{\leq 0}$:	**if** 1.	$\{x : (\leq 0\ R\ D), (x, y) : R\} \subseteq \mathcal{A}$ and
	2.	$y : \sim D \notin \mathcal{A}$
	then	$\mathcal{A} \rightarrow_{\leq 0} \mathcal{A} \cup \{y : \sim D\}$
$\rightarrow_\leq$:	**if** 1.	$x : (\leq n\ R\ D) \in \mathcal{A}$, $R^{\mathcal{A}}(x, D) > n > 0$ and
	2.	$\{(x, y) : R, (x, z) : R\} \subseteq \mathcal{A}$ for some $y \neq z$ and[a]
	3.	replacing y by z is safe in $\mathcal{A}$
	then	$\mathcal{A} \rightarrow_\leq [z/y]\mathcal{A}$

[a]The rules in (van der Hoek & de Rijke, 1995) do not require $\{y : D, z : D\} \in \mathcal{A}$, as one might expect.

Unfortunately, the *if*-direction of this claim is not true. The problem lies in the fact that, while a clash causes unsatisfiability, a complete and clash-free ABox is not necessarily satisfiable. The following counterexample exhibits this problem. Consider the concept

$$C = (\geq 3\ R\ A) \sqcap (\leq 1\ R\ B) \sqcap (\leq 1\ R\ \neg B).$$

On the one hand, C is clearly not satisfiable. Assume an interpretation $\mathcal{I}$ with $x \in C^{\mathcal{I}}$. This implies the existence of at least three R-successors y_1, y_2, y_3 of x. For each of the y_i either $y_i \in B^{\mathcal{I}}$ or $y_i \in (\neg B)^{\mathcal{I}}$ holds by the definition of $\cdot^{\mathcal{I}}$. Without loss of generality, there are two elements y_{i_1}, y_{i_2} such that $\{y_{i_1}, y_{i_2}\} \subseteq B^{\mathcal{I}}$, which implies $x \notin (\leq 1\ R\ B)^{\mathcal{I}}$ and hence $x \notin C^{\mathcal{I}}$.

On the other hand, the ABox $\mathcal{A} = \{x_0 : C\}$ can be turned into a complete and clash-free ABox using the rules from Fig. 4.1, as is shown in Fig. 4.2. Clearly this invalidates the claim and thus its proof.

To understand the mistake of the incorrect algorithm, it is useful to recall how soundness is usually established for tableau algorithms. The central idea is that a complete and clash-free ABox $\mathcal{A}$ is "obviously" satisfiable, in the sense that a model of $\mathcal{A}$ can directly be constructed from $\mathcal{A}$. For a complete and clash-free $\mathcal{ALCQ}$-ABox $\mathcal{A}$ we define the canonical interpretation $\mathcal{I}_{\mathcal{A}}$ as in Definition 3.7.

The mistake of the incorrect algorithm is due to the fact that it did not take into account that, in the canonical interpretation induced by a complete and clash-free ABox, there are concepts satisfied by the individuals even though these concepts do not appear as

Figure 4.2 A run of the incorrect algorithm.

$$\{x : C\} \rightarrow_\sqcap \cdots \rightarrow_\sqcap \underbrace{\{x : C,\ x : (\geq 3\ R\ A),\ x : (\leq 1\ R\ B),\ x : (\leq 1\ R\ \neg B)\}}_{=\mathcal{A}_1}$$

$$\rightarrow_\geq \cdots \rightarrow_\geq \underbrace{\mathcal{A}_1 \cup \{(x, y_i) : R,\ y_i : A \mid i = 1, 2, 3\}}_{=\mathcal{A}_2}$$

$\mathcal{A}_2$ is clash-free and complete, because $\sharp R^{\mathcal{A}_2}(x, A) = 3$ and $\sharp R^{\mathcal{A}_2}(x, B) = 0$.

constraints in the ABox. In our example, all of the y_i, for which B is not explicitly asserted, satisfy $\neg B$ in the canonical interpretation but this is not reflected in the generated ABox.

4.2.2 A Correct but Inefficient Solution

This problem has already been noticed in (Hollunder & Baader, 1991), where an algorithm very similar to the incorrect one is presented that correctly decides the satisfiability of $\mathcal{ALCQ}$-concepts.

The algorithm essentially uses the same definitions and rules. The only substantial difference is the introduction of the $\rightarrow_{\text{choose}}$-rule, which makes sure that all "relevant" concepts that are implicitly satisfied by an individual are made explicit in the ABox. Here, relevant concepts for an individual y are those occurring in qualifying number restrictions in constraints for variables x such that $(x, y) : R$ appears in the ABox.

Algorithm 4.4 (The Standard Algorithm for $\mathcal{ALCQ}$, Hollunder & Baader, 1991)
The rules of the standard algorithm *are given in Figure 4.3. The definition of* clash *is modified as follows: an ABox $\mathcal{A}$ contains a clash iff*

- $\{x : A, x : \neg A\} \subseteq \mathcal{A}$ *for some individual x and a concept name A, or*
- $x : (\leq n\ R\ D) \in \mathcal{A}$ *and* $\sharp R^{\mathcal{A}}(x, D) > n$ *for some variable x, relation R, concept D, and $n \in \mathbb{N}$.*

The algorithm works like the incorrect algorithm with the following differences: (1) it uses the completion rules from Fig. 4.3 (where $\bowtie$ is used as a placeholder for either $\leq$ or $\geq$); (2) it uses the definition of clash from above; and (3) it does not immediately stop when a clash has been generated but always generates a complete ABox. ◇

The standard algorithm is a decision procedure for $\mathcal{ALCQ}$-concept satisfiability:

Theorem 4.5 (Hollunder & Baader, 1991)
Let C be an $\mathcal{ALCQ}$-concept in NNF. C is satisfiable iff $\{x_0 : C\}$ can be transformed into a clash-free complete ABox using the rules in Figure 4.3. Moreover, each sequence of these rule-applications is finite.

Figure 4.3 The standard completion rules for $\mathcal{ALCQ}$

$\rightarrow_\sqcap$, $\rightarrow_\sqcup$: see Fig. 4.1

$\rightarrow_{\text{choose}}$: if 1. $\{x : (\bowtie n\ R\ D), (x, y) : R\} \subseteq \mathcal{A}$ and
2. $\{y : D, y : {\sim}D\} \cap \mathcal{A} = \emptyset$
then $\mathcal{A} \rightarrow_{\text{choose}} \mathcal{A} \cup \{y : E\}$ where $E \in \{D, {\sim}D\}$

$\rightarrow_\geq$: if 1. $x : (\geq n\ R\ D) \in \mathcal{A}$ and
2. $\sharp R^{\mathcal{A}}(x, D) < n$
then $\mathcal{A} \rightarrow_\geq \mathcal{A} \cup \{(x, y) : R, y : D\}$ where y is a new variable.

$\rightarrow_\leq$: if 1. $x : (\leq n\ R\ D) \in \mathcal{A}$, $\sharp R^{\mathcal{A}}(x, D) > n$, and
2. $\{(x, y) : R, (x, z) : R, y : D, z : D\} \subseteq \mathcal{A}$, for some $y \neq z$ and
3. for every u with $(x, u) : R \in \mathcal{A}$, $\{u : D, u : {\sim}D\} \cap \mathcal{A} \neq \emptyset$, and
4. the replacement of y by z is safe in $\mathcal{A}$
then $\mathcal{A} \rightarrow_\leq [y/z]\mathcal{A}$

While no complexity result is explicitly given in (Hollunder & Baader, 1991), it is easy to see that a PSpace result could be derived from the algorithm using the trace technique from Section 2.1.

Unfortunately this is only true if we assume the numbers in the input to be unary coded. The reason for this lies in the $\rightarrow_\geq$-rule, which generates n successors for a concept of the form $(\geq n\ R\ D)$. If n is unary coded, these successors consume at least polynomial space in the size of the input concept. If we assume binary (or k-ary with $k > 1$) encoding, the space consumption is exponential in the size of the input because a number n can be represented in $\log_k n$ bits in k-ary coding. This blow-up cannot be avoided because the completeness of the standard algorithm relies on the generation *and identification* of these successors, which makes it necessary to keep them in memory *at one time*.

4.3 An Optimal Solution

In the following, we will now present the algorithm with optimal worst case complexity, which will be used to prove the exact complexity result for $\mathcal{ALCQ}$:

Theorem 4.6
Satisfiability of $\mathcal{ALCQ}$-concepts is PSpace*-complete, even if numbers in the input are represented using* ***binary*** *coding.*

When aiming for a PSpace algorithm, it is impossible to generate all successors of an individual in the ABox simultaneously at a given stage as this may consume space that is exponential in the size of the input concept. We will give an optimal rule set for $\mathcal{ALCQ}$-satisfiability that does not rely on the identification of successors. Instead we will make stronger use of non-determinism to guess the assignment of the relevant concepts to the successors by the time of their generation. This will make it possible to generate the completion tree in a depth-first manner, which facilitates re-use of space.

Algorithm 4.7 (The Optimal Algorithm for $\mathcal{ALCQ}$)
The definition of clash *is taken from Algorithm 4.4.*

To test the satisfiability of a concept C, the optimal algorithm starts with the initial ABox $\{x_0 : C\}$ and successively applies the rules given in Fig. 4.4, stopping when a clash occurs. The algorithm answers "C is satisfiable" iff the rules can be applied in a way that yields a complete and clash-free ABox. $\diamond$

Figure 4.4 The optimal completion rules for $\mathcal{ALCQ}$.

$\to_\sqcap$-, $\to_\sqcup$: see Fig. 4.1

$\to_\geq$: if 1. $x : (\geq n\ R\ D) \in \mathcal{A}$, and
2. $\sharp R^{\mathcal{A}}(x, D) < n$, and
3. neither the $\to_\sqcap$- nor the $\to_\sqcup$-rule apply to a constraint for x
then $\mathcal{A} \to_\geq \mathcal{A} \cup \{(x, y) : R, y : D, y : D_1, \ldots, y : D_k\}$ where
$\{E_1, \ldots, E_k\} = \{E \mid x : (\bowtie m\ R\ E) \in \mathcal{A}\}$, $D_i \in \{E_i, \sim E_i\}$, and
y is a fresh individual.

For the different kinds on non-determinism present in this algorithm, compare the discussion below Algorithm 3.2. In the proof of Lemma 4.14, it is shown that the choice of which rule to apply when is don't-care non-deterministic. Any strategy that decides which rule to apply if more than one is applicable will yield a complete algorithm.

At first glance, the $\to_\geq$-rule may appear to be complicated and therefore it is explained in more detail: like the standard $\to_\geq$-rule, it is applicable to an ABox that contains the constraint $x : (\geq n\ R\ D)$ if there are less than n R-successors y of x with $y : D \in \mathcal{A}$. The rule then adds a new successor y of x to $\mathcal{A}$. Unlike the standard algorithm, the optimal algorithm also adds additional constraints of the form $y : (\sim)E$ to $\mathcal{A}$ for each concept E appearing in a constraint of the form $x : (\bowtie m\ R\ E)$. Since application of the $\to_\geq$-rule is suspended until no other rule applies to x, by this time $\mathcal{A}$ contains all constraints of the form $x : (\bowtie m\ R\ E)$ it will ever contain. This combines the effects of both the $\to_{\text{choose}}$- and the $\to_\leq$-rule of the standard algorithm.

4.3.1 Correctness of the Optimized Algorithm

To establish the correctness of the optimal algorithm, we will show its termination, soundness, and completeness. Again, it is convenient to view $\mathcal{A}$ as the graph $G_{\mathcal{A}} = (V, E, \mathbf{L})$ as defined in Section 2.1. Since the $\to_\geq$-rule not only adds sub-concepts of C but in some cases also the NNF of sub-concepts, the label $\mathbf{L}(x)$ of a node x is no longer a subset of $sub(C)$ but rather of the larger set $clos(C)$ defined below.

Definition 4.8
For an $\mathcal{ALCQ}$-concept C we define the closure $clos(C)$ as the smallest set of $\mathcal{ALCQ}$-concepts that

- *contains C,*
- *is closed under sub-concepts, and*
- *is closed under the application of $\sim$.*

◇

It is easy to see that the size of $clos(C)$ is linearly bounded in $|C|$:

Lemma 4.9
For an $\mathcal{ALCQ}$-concept C in NNF,

$$\sharp clos(C) \leq 2 \times |C|$$

Proof. This is an immediate consequence of the fact that

$$clos(C) \subseteq sub(C) \cup \{\sim D \mid D \in sub(C)\}$$

which can be shown as follows. Obviously, the set $sub(C) \cup \{\sim D \mid D \in sub(C)\}$ contains C and is closed under the application of $\sim$ (Note that, for a sub-concept D of a concept in NNF, $\sim\sim D = D$). Closure under sub-concepts for the concepts in $sub(C)$ is also immediate, and can be established for $\{\sim D \mid D \in sub(C)\}$ by considering the various possibilities for $\mathcal{ALCQ}$-concepts. ■

Similar to $\mathcal{ALC}$, it is easy to show that the graph $G_{\mathcal{A}}$ for an ABox $\mathcal{A}$ generated by the optimal algorithm from an initial ABox $\{x_0 : C\}$ is a tree with root x_0, and for each edge $(x, y) \in E$, the label $\mathbf{L}(x, y)$ is a singleton. Moreover, for each node x it holds that $\mathbf{L}(x) \subseteq clos(C)$.

Termination

First, we will show that the optimal algorithm always terminates, i.e., each sequence of rule applications starting from the ABox $\{x_0 : C\}$ is finite. The next lemma will also be helpful when we consider the complexity of the algorithm.

Lemma 4.10
Let C be a concept in NNF and $\mathcal{A}$ an ABox that is generated by the optimal algorithm starting from $\{x_0 : C\}$.

- *The length of a path in $G_{\mathcal{A}}$ is limited by $|C|$.*
- *The out-degree of $G_{\mathcal{A}}$ is bounded by $|C| \times 2^{|C|}$.*

Proof. The linear bound on the length of a path in $G_{\mathcal{A}}$ is established as for the $\mathcal{ALC}$-algorithm using the fact that the nesting of qualifying number restrictions strictly decreases along a path in $G_{\mathcal{A}}$.

Successors in $G(\mathcal{A})$ are only generated by the $\rightarrow_{\geq}$-rule. For an individual x this rule will generate at most n successors for each $(\geq n\ R\ D) \in \mathbf{L}(x)$. There are at most $|C|$ such concepts in $\mathbf{L}(x)$. Hence the out-degree of x is bounded by $|C| \times 2^{|C|}$, where $2^{|C|}$ is a limit for the biggest number that may appear in C if binary coding is used. ∎

Corollary 4.11 (Termination)
Any sequence of rule applications starting from an ABox $\mathcal{A} = \{x_0 : C\}$ of the optimal algorithm is finite.

Proof. The sequence of rules induces a sequence of trees. The depth and the out-degree of these trees is bounded by some function in $|C|$ by Lemma 4.10. For each individual x the label $\mathbf{L}(x)$ is a subset of the finite set $\mathit{clos}(C)$. Each application of a rule either

- adds a new constraint of the form $x : D$ and hence adds an element to $\mathbf{L}(x)$, or
- adds fresh individuals to $\mathcal{A}$ and hence adds additional nodes to the tree $G_{\mathcal{A}}$.

Since constraints are never deleted and individuals are never deleted or identified, an infinite sequence of rule application must either lead to an infinite number of nodes in the trees which contradicts their boundedness, or it leads to an infinite label of one of the nodes x which contradicts $\mathbf{L}(x) \subseteq \mathit{clos}(C)$. ∎

Soundness and Completeness

We establish soundness and completeness of the optimal algorithm along the lines of Theorem 3.6. We use a slightly modified notion of ABox satisfiability, which is already implicitly present in the definition of clash. If we want to apply Theorem 3.6 to prove the correctness of the algorithm, then we need that a clash in an ABox causes unsatisfiability of that ABox. For an arbitrary ABox and the definition of clash used by the optimal algorithm, this is not the case. For example the ABox

$$\mathcal{A} = \{x : (\leq 1\ R\ A), (x, y) : R, (x, z) : R, y : A, z : A\}$$

contains a clash but is satisfiable. Yet, if we require, that for all individuals x, y, z, if $(x, y) : R, (x, z) : R \in \mathcal{A}$ and $y \neq z$, then y and z must be interpreted with different elements of the domain, then a clash obviously implies unsatisfiability. This is captured by the definition of the function $\widehat{\cdot}$ that maps an $\mathcal{ALCQ}$-ABox to its *differentiation* $\widehat{\mathcal{A}}$ defined by

$$\widehat{\mathcal{A}} = \mathcal{A} \cup \{y \neq z \mid \{(x, y) : R, (x, z) : R\} \subseteq \mathcal{A}, y \neq z\}.$$

For the proof of soundness and completeness of Algorithm 4.7, an ABox $\mathcal{A}$ is called satisfiable iff $\widehat{\mathcal{A}}$ is satisfiable in the (standard) sense of Definition 2.5. Since $\widehat{\cdot}$ is idempotent, the standard and the modified notion of satisfiability coincide for a differentiated ABox $\widehat{\mathcal{A}}$ and we can unambiguously speak of the satisfiability of $\widehat{\mathcal{A}}$ without specifying if we refer to the modified or the standard notion.

Consider the properties required by Theorem 3.6. As discussed before, with this definition of satisfiability of ABoxes, it is obvious that, for an ABox $\mathcal{A}$ generated by the optimal algorithm that contains a clash, $\widehat{\mathcal{A}}$ (and hence, by our definition, $\mathcal{A}$) must be unsatisfiable (Property 1) and that $\{x_0 : C\}$ is satisfiable iff C is satisfiable (Property 2). It remains to establish Property 3 (a clash-free and complete ABox is satisfiable, Lemma 4.13) and the local correctness (Properties 4,5) of the rules (Lemma 4.14).

For $\mathcal{ALC}$, to prove satisfiability of a complete and clash-free ABox $\mathcal{A}$, we used induction over the structure of concepts appearing in constraints in $\mathcal{A}$. This was possible because the $\mathcal{ALC}$-rules, when triggered by an assertion $x : D$, only add constraints to $\mathcal{A}$ that involve sub-concepts of D. For $\mathcal{ALCQ}$, and specifically for the $\rightarrow_{\geq}$-rule, this is no longer true and hence a proof by induction on the structure of concepts is not feasible. Instead, we will use induction on following *norm* of concepts.

Definition 4.12
For an $\mathcal{ALCQ}$-concept D in NNF, then norm $\|D\|$ is inductively defined by:

$$\begin{array}{llll} \|A\| & := \|\neg A\| & := & 0 \quad \text{for } A \in \mathsf{NC} \\ \|C_1 \sqcap C_2\| & := \|C_1 \sqcup C_2\| & := & 1 + \|C_1\| + \|C_2\| \\ \|(\bowtie n\ R\ D)\| & & := & 1 + \|D\| \end{array}$$

◇

The reader may verify that this norm satisfies $\|D\| = \|{\sim}D\|$ for every concept D.

Lemma 4.13
Let $\mathcal{A}$ be a complete and clash-free ABox generated by the optimal algorithm. Then $\widehat{\mathcal{A}}$ is satisfiable.

Proof. Let $\mathcal{A}$ be a complete and clash-free ABox generated by applications of the optimal rules and $\widehat{\mathcal{A}}$ its differentiation. We show that the canonical interpretation $\mathcal{I}_\mathcal{A}$, as defined in Definition 3.7, is a model of $\widehat{\mathcal{A}}$.

By definition of $\mathcal{I}_\mathcal{A}$, all constraints of the form $(x, y) : R$ are trivially satisfied. Also, $y \neq z$ implies $y^{\mathcal{I}_\mathcal{A}} \neq z^{\mathcal{I}_\mathcal{A}}$ by construction of $\mathcal{I}_\mathcal{A}$. Thus, all remaining assertions in $\widehat{\mathcal{A}}$ are of the form $x : D$ and are also present in $\mathcal{A}$. Thus, it is sufficient to show that $x : D \in \mathcal{A}$ implies $x^{\mathcal{I}_\mathcal{A}} \in D^{\mathcal{I}_\mathcal{A}}$, which we will do by induction on the norm $\|\cdot\|$ of a concept D. Note that, by the definition of $\mathcal{I}_\mathcal{A}$, $x^{\mathcal{I}_\mathcal{A}} = x$ for every individual x that occurs in $\mathcal{A}$.

- The first base case is $D = A$ for $A \in \mathsf{NC}$. $x : A \in \mathcal{A}$ immediately implies $x \in A^{\mathcal{I}_\mathcal{A}}$ by the definition of $\mathcal{I}_\mathcal{A}$. The second base case is $x : \neg A \in \mathcal{A}$. Since $\mathcal{A}$ is clash-free, this implies $x : A \notin \mathcal{A}$ and hence $x \notin A^{\mathcal{I}_\mathcal{A}}$. This implies $x \in (\neg A)^{\mathcal{I}_\mathcal{A}}$

- For the conjunction and disjunction of concepts this follows exactly as in the proof of Lemma 3.8.

- $x : (\geq n\ R\ D) \in \mathcal{A}$ implies $\sharp R^{\mathcal{A}}(x, D) \geq n$ because otherwise the $\rightarrow_{\geq}$-rule would be applicable and $\mathcal{A}$ would not be complete. By induction, we have $y \in D^{\mathcal{I}_{\mathcal{A}}}$ for each y with $y : D \in \mathcal{A}$. Hence $\sharp R^{\mathcal{I}_{\mathcal{A}}}(x, D) \geq n$ and thus $x \in (\geq n\ R\ D)^{\mathcal{I}_{\mathcal{A}}}$.

- $x : (\leq n\ R\ D) \in \mathcal{A}$ implies $\sharp R^{\mathcal{A}}(x, D) \leq n$ because $\mathcal{A}$ is clash-free. Hence it is sufficient to show that $\sharp R^{\mathcal{I}_{\mathcal{A}}}(x, D) \leq \sharp R^{\mathcal{A}}(x, D)$ holds. On the contrary, assume $\sharp R^{\mathcal{I}_{\mathcal{A}}}(x, D) > \sharp R^{\mathcal{A}}(x, D)$ holds. Then there is an individual y such that $(x, y) : R \in \mathcal{A}$ and $y \in D^{\mathcal{I}_{\mathcal{A}}}$ but $y : D \notin \mathcal{A}$. The application of the $\rightarrow_{\geq}$-rule is suspended until the propositional rules are no longer applicable to x and hence, by the time y is generated by an application of the $\rightarrow_{\geq}$-rule, $\mathcal{A}$ contains the assertion $x : (\leq n\ R\ D)$. Hence, the $\rightarrow_{\geq}$-rule ensures $y : D \in \mathcal{A}$ or $y : {\sim}D \in \mathcal{A}$. Since we have assumed that $y : D \notin \mathcal{A}$, this implies $y : {\sim}D \in \mathcal{A}$ and, by the induction hypothesis, $y \in ({\sim}D)^{\mathcal{I}_{\mathcal{A}}}$ holds, which is a contradiction. ■

Lemma 4.14 (Local Correctness)
Let $\mathcal{A}, \mathcal{A}'$ be ABoxes generated by the optimal algorithm from an ABox of the form $\{x_0 : C\}$.

1. *If $\mathcal{A}'$ is obtained from $\mathcal{A}$ by application of the (deterministic) $\rightarrow_{\sqcap}$-rule, then $\widehat{\mathcal{A}}$ is satisfiable iff $\widehat{\mathcal{A}'}$ is satisfiable.*

2. *If $\mathcal{A}'$ is obtained from $\mathcal{A}$ by application of the (non-deterministic) $\rightarrow_{\sqcup}$- or $\rightarrow_{\geq}$-rule, then $\widehat{\mathcal{A}}$ is satisfiable if $\widehat{\mathcal{A}'}$ is satisfiable. Moreover, if $\widehat{\mathcal{A}}$ is satisfiable, then the rule can always be applied in such a way that it yields a satisfiable $\widehat{\mathcal{A}''}$.*

Proof. $\mathcal{A} \rightarrow \mathcal{A}'$ for any rule $\rightarrow$ implies $\mathcal{A} \subseteq \mathcal{A}'$ and, by the definition of $\widehat{\cdot}$, $\widehat{\mathcal{A}} \subseteq \widehat{\mathcal{A}'}$, hence, if $\widehat{\mathcal{A}'}$ is satisfiable then so is $\widehat{\mathcal{A}}$. For the other direction, the $\rightarrow_{\sqcap}$- and $\rightarrow_{\sqcup}$-rule can be handled as in the proof for $\mathcal{ALC}$ in Lemma 3.9.

It remains to consider the $\rightarrow_{\geq}$-rule. Let $\mathcal{I}$ be a model of $\widehat{\mathcal{A}}$ and let $x : (\geq n\ R\ D)$ be the constraint that triggers the application of the $\rightarrow_{\geq}$-rule. Since the $\rightarrow_{\geq}$-rule is applicable, we have $\sharp R^{\mathcal{A}}(x, D) < n$. We claim that there is an $a \in \Delta^{\mathcal{I}}$ with

$$(x^{\mathcal{I}}, a) \in R^{\mathcal{I}}, a \in D^{\mathcal{I}} \text{ and } a \notin \{z^{\mathcal{I}} \mid (x, z) : R \in \mathcal{A}\}. \qquad (*)$$

Before we prove this claim, we show how it can be used to finish the proof. The element a is used to "select" a choice of the $\rightarrow_{\geq}$-rule that preserves satisfiability: let $\{E_1, \ldots, E_k\}$ be an enumeration of the set $\{E \mid x : (\bowtie m\ R\ E) \in \mathcal{A}\}$. We set

$$\mathcal{A}'' = \mathcal{A} \cup \{(x, y) : R, y : D\} \cup \{y : E_i \mid a \in E_i^{\mathcal{I}}\} \cup \{y : {\sim}E_i \mid a \notin E_i^{\mathcal{I}}\}$$

Obviously, $\mathcal{I}[y \mapsto a]$, the interpretation that maps y to a and agrees with $\mathcal{I}$ on all other names, is a model for $\widehat{\mathcal{A}''}$, since y is a fresh individual and a satisfies $(*)$. The ABox $\mathcal{A}''$

is a possible result of the application of the $\rightarrow_{\geq}$-rule to $\mathcal{A}$, which proves that the $\rightarrow_{\geq}$-rule can indeed be applied in a way that maintains satisfiability of the ABox.

We will now come back to the claim. It is obvious that there is an a with $(x^{\mathcal{I}}, a) \in R^{\mathcal{I}}$ and $a \in D^{\mathcal{I}}$ that is not contained in $\{z^{\mathcal{I}} \mid (x,z) : R, z : D \in \mathcal{A}\}$, because $\sharp R^{\mathcal{I}}(x, D) \geq n > \sharp R^{\mathcal{A}}(x, D)$. Yet a might appear as the image of an individual z such that $(x,z) : R \in \mathcal{A}$ but $z : D \notin \mathcal{A}$.

Now, $(x,z) : R \in \mathcal{A}$ and $z : D \notin \mathcal{A}$ implies $z : \sim D \in \mathcal{A}$. This is due to the fact that the constraint $(x,z) : R$ must have been generated by an application of the $\rightarrow_{\geq}$-rule because it has not been an element of the initial ABox. The application of this rule was suspended until neither the $\rightarrow_{\sqcap}$- nor the $\rightarrow_{\sqcup}$-rule were applicable to x. Hence, if $x : ({\geq} n\ R\ D)$ is an element of $\mathcal{A}$ now, then it has already been in $\mathcal{A}$ when the $\rightarrow_{\geq}$-rule that generated z was applied. The $\rightarrow_{\geq}$-rule guarantees that either $z : D$ or $z : \sim D$ is added to $\mathcal{A}$, hence $z : \sim D \in \mathcal{A}$. This is a contradiction to $z^{\mathcal{I}} = a$ because under the assumption that $\mathcal{I}$ is a model of $\mathcal{A}$ this would imply $a \in (\sim D)^{\mathcal{I}}$ while we initially assumed $a \in D^{\mathcal{I}}$. ■

As an immediate consequence of the Lemmas 4.11, 4.13, and 4.14 together with Theorem 3.6 we get:

Corollary 4.15
The optimal algorithm is a non-deterministic decision procedure for satisfiability of $\mathcal{ALCQ}$-concepts.

4.3.2 Complexity of the Optimal Algorithm

The optimal algorithm will enable us to prove Theorem 4.6. We will give a proof by sketching an implementation of this algorithm that runs in polynomial space.

Lemma 4.16
The optimal algorithm can be implemented in PSpace

Proof. Let C be an $\mathcal{ALCQ}$-concept to be tested for satisfiability. We can assume C to be in NNF because the transformation of a concept to NNF can be performed in linear time.

The key idea for the PSpace implementation is the *trace technique* (Schmidt-Schauß & Smolka, 1991) we have already used for the $\mathcal{ALC}$-algorithm in Section 3.2.2, and which is based on the fact that it is sufficient to keep only a single path (a trace) of $G_{\mathcal{A}}$ in memory at a given stage if $\mathcal{A}$ is generated in a depth-first manner. This idea has been the key to a PSpace upper bound for K_m and $\mathcal{ALC}$ in (Ladner, 1977; Schmidt-Schauß & Smolka, 1991; Halpern & Moses, 1992). To do this we need to store the values for $\sharp R^{\mathcal{A}}(x, D)$ for each individual x in the path, each R that appears in $\mathit{clos}(C)$, and each $D \in \mathit{clos}(C)$. By storing these values in binary form, we are able to keep information *about* exponentially many successors in memory while storing only a single path at a given stage.

Consider the algorithm in Fig. 4.5, where NR_C denotes the set of role names that appear in $\mathit{clos}(C)$. It re-uses the space needed to check the satisfiability of a successor y of x once the existence of a complete and clash-free "subtree" for the constraints on y

has been established. This is admissible since, as was the case for $\mathcal{ALC}$, the optimal rules will never modify this subtree once it is completed. Constraints in this subtree also have no influence on the completeness or the existence of a clash in the rest of the tree, with the exception that constraints of the form $y : D$ for R-successors y of x contribute to the value of $\sharp R^{\mathcal{A}}(x, D)$. These numbers play a role both in the definition of a clash and for the applicability of the $\rightarrow_{\geq}$-rule. Hence, in order to re-use the space occupied by the subtree for y, it is necessary and sufficient to store these numbers.

Figure 4.5 A non-deterministic PSpace decision procedure for $\mathcal{ALCQ}$.

$\mathcal{ALCQ}$-Sat(C) := `sat`$(x_0, \{x_0 : C\})$
`sat`$(x, \mathcal{A})$:
  allocate counters $\sharp R^{\mathcal{A}}(x, D) := 0$ for all $R \in \mathsf{NR}_C$ and $D \in \mathit{clos}(C)$.
  `while` (the $\rightarrow_{\sqcap}$- or the $\rightarrow_{\sqcup}$-rule can be applied) `and` ($\mathcal{A}$ is clash-free) `do`
    apply the $\rightarrow_{\sqcap}$- or the $\rightarrow_{\sqcup}$-rule to $\mathcal{A}$.
  `od`
  `if` $\mathcal{A}$ contains a clash `then return` "not satisfiable".
  `while` (the $\rightarrow_{\geq}$-rule applies to a constraint $x : (\geq n\ R\ D) \in \mathcal{A}$) `do`
    $\mathcal{A}_{new} := \{(x, y) : R, y : D, y : D_1, \ldots, y : D_k\}$
    `where`
      y is a fresh individual,
      $\{E_1, \ldots, E_k\} = \{E \mid x : (\bowtie m\ R\ E) \in \mathcal{A}\}$, and
      D_i is chosen non-deterministically from $\{E_i, \sim E_i\}$
    `for each` $y : E \in \mathcal{A}_{new}$ `do` increment $\sharp R^{\mathcal{A}}(x, E)$
    `if` $x : (\leq m\ R\ E) \in \mathcal{A}$ and $\sharp R^{\mathcal{A}}(x, E) > m$ `then return` "not satisfiable".
    `if sat`$(y, \mathcal{A}_{new})$ = "not satisfiable" `then return` "not satisfiable"
    discard $\mathcal{A}_{new}$ from memory
  `od`
  discard the counters for x from memory.
  `return` "satisfiable"

Let us examine the space usage of this algorithm. Let $n = |C|$. The algorithm is designed to keep only a single path of $G_{\mathcal{A}}$ in memory at a given stage. For each individual x on a path, constraints of the form $x : D$ have to be stored for concepts $D \in \mathit{clos}(C)$. The size of $\mathit{clos}(C)$ is bounded by $2n$ and hence the constraints for a single individual can be stored in $\mathcal{O}(n)$ bits. For each individual, there are at most $|\mathsf{NR}_C| \times |\mathit{clos}(C)| = \mathcal{O}(n^2)$ counters to be stored. The numbers to be stored in these counters do not exceed the out-degree of x, which, by Lemma 4.10, is bounded by $|\mathit{clos}(C)| \times 2^{|C|}$. Hence each counter can be stored using $\mathcal{O}(n^2)$ bits when binary coding is used to represent the counters, and all counters for a single individual require $\mathcal{O}(n^4)$ bits. Due to Lemma 4.10, the length of a path is limited by n, which yields an overall memory consumption of $\mathcal{O}(n^5 + n^2) = \mathcal{O}(n^5)$. ∎

Theorem 4.6 now is a simple Corollary from the PSpace-hardness of $\mathcal{ALC}$, Lemma 4.16, and Savitch's Theorem (Savitch, 1970).

4.4 Extensions of $\mathcal{ALCQ}$

It is possible to augment the DL $\mathcal{ALCQ}$ without loosing the PSpace property of the concept satisfiability problem. In this section we extend the techniques to obtain a PSpace algorithm for the logic $\mathcal{ALCQIb}$, which extends $\mathcal{ALCQ}$ with inverse roles and *safe* Boolean combinations of roles. This extends the results from (Tobies, 2001) for the modal logic $\mathbf{Gr}(\mathsf{K}_{\mathcal{R}_{\cap}^{-1}})$, which corresponds to $\mathcal{ALCQ}$ extended with inverse roles and intersection of roles.

Definition 4.17 (Syntax of $\mathcal{ALCQIb}$)
Let NC *be a set of atomic* concept names *and* NR *be a set of atomic* role names. *With* $\overline{\mathsf{NR}} := \mathsf{NR} \cup \{R^{-1} \mid R \in \mathsf{NR}\}$ *we denote the set of* $\mathcal{ALCQIb}$-roles.

A role S *of the form* $S = R^{-1}$ *with* $R \in \mathsf{NR}$ *is called* inverse role.

An $\mathcal{ALCQIb}$-role expression ω *is built from* $\mathcal{ALCQIb}$*-roles using the operators* $\sqcap$ *(role intersection),* $\sqcup$ *(role union), and* $\neg$ *(role complement), with the restriction that, when transformed into disjunctive normal form, every disjunct contains at least one non-negated conjunct. A role expression that satisfies this constraint is called* safe.

The set of $\mathcal{ALCQIb}$-concepts *is built inductively from these using the following grammar, where* $A \in \mathsf{NC}$, ω *is an* $\mathcal{ALCQIb}$*-role expression, and* $n \in \mathbb{N}$*:*

$$C ::= A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid (\leq n\ \omega\ C) \mid (\geq n\ \omega\ C).$$

$\mathcal{ALCQI}$ *is the fragment of* $\mathcal{ALCQIb}$*, where every role expression in a number restriction consists of a single (possibly inverse)* $\mathcal{ALCQIb}$*-role.* $\diamond$

The role-expressions $\neg(\neg R_1 \sqcup (R_2^{-1} \sqcap \neg R_3)) \sqcup (\neg R_2 \sqcap R_2^{-1})$ is safe (its DNF is $(R_1 \sqcap \neg R_2^{-1}) \sqcup (R_1 \sqcap R_3) \sqcup (\neg R_2 \sqcap R_2^{-1})$) while $R \sqcup \neg R$ is not an $\mathcal{ALCQIb}$ role expression since it is already in DNF and $\neg R$ occurs as single element in one of the disjuncts. The latter example also shows that some kind of restrictions on role expressions is indeed necessary if we want to obtain a PSpace algorithm: the concept $(\leq 0\ R \sqcup \neg R\ \neg C)$ is satisfiable iff C is globally satisfiable, which is an ExpTime-complete problem (see the proof of Theorem 3.18. Indeed, for unrestricted role expressions, the problem in the presence of qualifying number restrictions is of even higher complexity. It is NExpTime-complete (see Corollary 5.34).

The syntactic restriction we have chosen enforces that, for a pair (x, y) to appear in the extension of a role expression ω, they must occur at least in the extension of one of the roles that occur in ω. Hence, if no role relation holds between x and y, concepts asserted for x do not impose any restrictions on y.

A similar restriction can be found in the database world in conjunction with the notion of *safe-range queries* (Abiteboul, Hull, & Vianu, 1995, Chapter 5). To decide whether a role expression ω is safe, it is not necessary to calculate its DNF (which might require exponential time). One can rather use the following algorithm: first, compute the NNF ω' of ω by pushing negation inwards using de Morgan's law. Second, test whether *safe*(ω') holds, where the function *safe* is defined inductively on the structure of role expressions as follows (compare Abiteboul et al., 1995, Algorithm 5.4.3):

$$\begin{aligned} safe(R) &= \texttt{true} \text{ for } R \in \overline{\mathsf{NR}} \\ safe(\neg R) &= \texttt{false} \text{ for } R \in \overline{\mathsf{NR}} \\ safe(\omega_1 \sqcap \omega_2) &= safe(\omega_1) \vee safe(\omega_2) \\ safe(\omega_1 \sqcup \omega_2) &= safe(\omega_1) \wedge safe(\omega_2) \end{aligned}$$

It is easy to see that a role expression is safe iff this algorithm yields `true`. Hence, a role expression can be tested for safety in polynomial time.

The semantics of $\mathcal{ALCQ}$-concepts can be extended to $\mathcal{ALCQIb}$-concepts by fixing the interpretations of the role expressions. This is done in the obvious way.

Definition 4.18 (Semantics of $\mathcal{ALCQIb}$)
For an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, the semantics of $\mathcal{ALCQIb}$-concepts is defined inductively as for $\mathcal{ALCQ}$-concepts with the additional rules:

$$\begin{aligned} ({\leq}n\ \omega\ C)^\mathcal{I} &= \{x \in \Delta^\mathcal{I} \mid \sharp\{y \mid (x,y) \in \omega^\mathcal{I} \text{ and } y \in C^\mathcal{I}\} \leq n\}, \\ ({\geq}n\ \omega\ C)^\mathcal{I} &= \{x \in \Delta^\mathcal{I} \mid \sharp\{y \mid (x,y) \in \omega^\mathcal{I} \text{ and } y \in C^\mathcal{I}\} \geq n\}, \end{aligned}$$

where the interpretation of a role expression ω is obtained by extending the valuation $\mathcal{I}$ inductively to role expressions by setting:

$$\begin{aligned} R^{-1} &= \{(y,x) \mid (x,y) \in R^\mathcal{I}\}, \\ (\neg\omega)^\mathcal{I} &= (\Delta^\mathcal{I} \times \Delta^\mathcal{I}) \setminus \omega^\mathcal{I}, \\ (\omega_1 \sqcap \omega_2)^\mathcal{I} &= \omega_1^\mathcal{I} \cap \omega_2^\mathcal{I}, \\ (\omega_1 \sqcup \omega_2)^\mathcal{I} &= \omega_1^\mathcal{I} \cup \omega_2^\mathcal{I}. \end{aligned}$$

◇

Obviously every $\mathcal{ALCQ}$ concept is also a $\mathcal{ALCQIb}$ concept. We will use the letters ω, σ to range over $\mathcal{ALCQIb}$-role-expressions. To avoid dealing with roles of the form $(R^{-1})^{-1}$ we use the convention that $(R^{-1})^{-1} = R$ for any $R \in \overline{\mathsf{NR}}$. This is justified by the semantics. The definition of NNF and $clos(\cdot)$ can be extended from $\mathcal{ALCQ}$ to $\mathcal{ALCQIb}$ in a straightforward manner. Moreover, we use the following notation:

Definition 4.19
Let $\mathbf{R}$ a set of (possibly inverse) roles and ω a role expression. We view then roles in $\overline{\mathsf{NR}}$ as propositional variables and $\mathbf{R}$ as the propositional interpretation that maps exactly the elements of $\mathbf{R}$ to true and all other roles to false. We write $\mathbf{R} \models \omega$ iff ω, viewed as a propositional formula, evaluates to true under $\mathbf{R}$. ◇

The intended use of this definition is captured by the following simple lemma:

Lemma 4.20
Let $\mathcal{I}$ be an interpretation, $x, y \in \Delta^{\mathcal{I}}$ and ω a role expression.

$$(x, y) \in \omega^{\mathcal{I}} \text{ iff } \{R \in \overline{\mathsf{NR}} \mid (x, y) \in R^{\mathcal{I}}\} \models \omega.$$

For two individuals x, y in an ABox $\mathcal{A}$ and a role expression ω,

$$\{R \mid (x, y) : R \in \mathcal{A}\} \models \omega$$

implies $(x, y) \in \omega^{\mathcal{I}_{\mathcal{A}}}$ for the canonical interpretation $\mathcal{I}_{\mathcal{A}}$.

4.4.1 Reasoning for $\mathcal{ALCQIb}$

We will use similar techniques as in the previous section to obtain a PSPACE-algorithm for $\mathcal{ALCQIb}$. We still use ABoxes to capture the constraints generate by completion rules, with the only change that we allow inverse roles R^{-1} to appear in role assertions and require that, for any $R \in \overline{\mathsf{NR}}$, an ABox contains the constraint $(x, y) : R$ iff it contains the constraint $(y, x) : R^{-1}$. For an ABox $\mathcal{A}$, a role-expression ω, and a concept D, let $\sharp\omega^{\mathcal{A}}(x, D)$ be the number of individuals y such that $\{R \mid (x, y) : R \in \mathcal{A}\} \models \omega$ and $y : D \in \mathcal{A}$. Due to the syntactic restriction on role expressions, an individual y may only contribute to $\sharp\omega^{\mathcal{A}}(x, D)$ if $(x, y) : R \in \mathcal{A}$ for some (possibly inverse) role R that occurs in ω.

Algorithm 4.21 (The $\mathcal{ALCQIb}$-algorithm)
We modify the definition of clash *to deal with safe role expressions as follows. An ABox $\mathcal{A}$ contains a clash iff*

- *$\{x : A, x : \neg A\} \subseteq \mathcal{A}$ for some individual x and $A \in \mathsf{NC}$, or*
- *$x : (\leq n\ \omega\ D) \in \mathcal{A}$ and $\sharp\omega^{\mathcal{A}}(x, D) > n$ for some individual x, role expression ω, concept D, and $n \in \mathbb{N}$.*

The set of rules dealing with $\mathcal{ALCQIb}$ is shown in Figure 4.6. The algorithm maintains a binary relation $\prec_{\mathcal{A}}$ between the individuals in an ABox $\mathcal{A}$ with $x \prec_{\mathcal{A}} y$ iff y was inserted by the $\rightarrow_{\geq}$-rule to satisfy a constraint for x. When considering the graph $G_{\mathcal{A}}$, the relation $\prec_{\mathcal{A}}$ corresponds to the successor relation between nodes. Hence, when $x \prec_{\mathcal{A}} y$ holds we will call y a successor of x and x a predecessor of y. We denote the transitive closure of $\prec_{\mathcal{A}}$ by $\prec^{+}_{\mathcal{A}}$.

For a set of individuals $\mathcal{X}$ and an ABox $\mathcal{A}$, we denote the subset of $\mathcal{A}$ in which no individual from $\mathcal{X}$ occurs in a constraint by $\mathcal{A} - \mathcal{X}$. The $\rightarrow_{\sqcap}$-, $\rightarrow_{\sqcup}$- and $\rightarrow_{choose}$-rule are called non-generating rules *while the $\rightarrow_{\geq}$-rule is called* a generating rule.

Let C be an $\mathcal{ALCQIb}$-concept in NNF and $\overline{\mathsf{NR}}_C$ the set of roles that occur in C together with their inverses. To test the satisfiability of C, the $\mathcal{ALCQIb}$-algorithm starts with the *initial ABox $\{x_0 : C\}$ and successively applies the rules from Figure 4.6. stopping when a clash occurs or the $\rightarrow_{\geq}$-rule fails. The algorithm answers "C is satisfiable" iff the rule can be applied in a way that yields a complete ABox.* $\diamond$

Figure 4.6 The completion rules for $\mathcal{ALCQIb}$.

$\rightarrow_{\sqcap}$, $\rightarrow_{\sqcup}$: see Fig. 4.1

$\rightarrow_{\text{choose}}$: if 1. $x : (\bowtie n\ \omega\ D) \in \mathcal{A}$ and
2. for some R that occurs in ω there is a y with $(x, y) : R \in \mathcal{A}$, and $\{y : D, y : \sim D\} \cap \mathcal{A} = \emptyset$

then $\mathcal{A} \rightarrow_{\text{choose}} \mathcal{A}' \cup \{y : E\}$ where $E \in \{D, \sim D\}$ and $\mathcal{A}' = \mathcal{A} - \{z \mid y \prec^{+}_{\mathcal{A}} z\}$

$\rightarrow_{\geq}$: if 1. $x : (\geq n\ \omega\ D) \in \mathcal{A}$, and
2. $\sharp\omega^{\mathcal{A}}(x, D) < n$, and
3. no non-generating rule can be applied to a constraint for x

then guess a set $\mathbf{R} = \{R_1, \ldots, R_m\} \subseteq \overline{\mathsf{NR}}_C$

if $\mathbf{R} \not\models \omega$ then *fail*

else $\mathcal{A} \rightarrow_{\geq} \mathcal{A} \cup \{y : D\} \cup \mathcal{A}' \cup \mathcal{A}''$ and set $x \prec_{\mathcal{A}} y$ where

$\mathcal{A}' = \{y : D_1, \ldots, y : D_k\}$, $D_i \in \{E_i, \sim E_i\}$, and $\{E_1, \ldots, E_k\} = \{E \mid x : (\bowtie m\ \sigma\ E) \in S\}$

$\mathcal{A}'' = \{(x, y) : R_1, (y, x) : R_1^{-1}, \ldots, (x, y) : R_m, (y, x) : R_m^{-1}\}$

y is a fresh individual

For the different kinds on non-determinism present in this algorithm, compare the discussion below Algorithm 3.2. Similar to the case for $\mathcal{ALCQ}$, it is shown in the proof of Lemma 4.25 that the choice of which rule to apply when is don't-care non-deterministic. This implies that one is free to choose an arbitrary strategy that decides which rule to apply if more than one is applicable.

For the different kinds of non-determinism present in the $\mathcal{ALCQIb}$-algorithm, refer to the discussion below

The $\rightarrow_{\geq}$-rule, while looking complicated, is a straightforward extension of the $\rightarrow_{\geq}$-rule for $\mathcal{ALCQ}$, which takes into account that we need to guess a set of roles between the old individual x and the freshly introduced individual y such that these roles satisfy the role expression ω currently under consideration. The $\rightarrow_{\text{choose}}$-rule requires more explanation.

For $\mathcal{ALCQ}$, the optimal algorithm generates an ABox $\mathcal{A}$ in a way that, whenever $x : (\bowtie n\ R\ D) \in \mathcal{A}$, then, for any y with $(x, y) : R \in \mathcal{A}$, either $y : D$ or $y : \sim D \in \mathcal{A}$. This was achieved by suspending the generation of any successors y of x until $\mathcal{A}$ contained all constraints of the from $x : D$ it would ever contain. In the presence of inverse relations, this is no longer possible because y might have been generated as a predecessor of x and hence before it was possible to know which concepts D might be relevant. There are at least two possible ways to overcome this problem. One is, to guess, for every x and *every*

$D \in clos(C)$, whether $x : D$ or $x : {\sim}D$. In this case, since the termination of the optimal algorithm as proved in Lemma 4.11 relies on the fact that the nesting of qualifying number restrictions strictly decreases along a path in the induced graph $G_\mathcal{A}$, termination would no longer be guaranteed. It would have to be enforced by different means.

Here, we use another approach. We can distinguish two different situations where $\{x : (\bowtie n\ \omega\ D), (x, y) : R\} \subseteq \mathcal{A}$ for some R that occurs in ω, and $\{y : D, y : {\sim}D\} \cap \mathcal{A} = \emptyset$: y is a predecessor of x ($y \prec_\mathcal{A} x$) or a successor of x ($x \prec_\mathcal{A} y$). The second situation will never occur. This is due to the interplay of the $\rightarrow_\geq$-rule and the $\rightarrow_{\text{choose}}$-rule. The former is suspended until all known relevant information has been added for x, the latter deletes certain parts of the ABox whenever new constraints are added for predecessor individuals.

The first situation is resolved by non-deterministically adding either $y : D$ or $y : {\sim}D$ to $\mathcal{A}$. The subsequent deletion of all constraints involving individuals from $\{z \mid y \prec_\mathcal{A}^+ z\}$, which correspond to the deletion of all subtrees of $G_\mathcal{A}$ rooted below y, is necessary to make this rule "compatible" with the trace technique we want to employ in order to obtain a PSpace-algorithm. The correctness of the trace approach relies on the property that, once we have established the existence of a complete and clash-free "subtree" for a node x, we can remove this tree from memory because it will not be modified by the algorithm. In the presence of inverse roles this can be no longer taken for granted as can be illustrated by the concept

$$C = ({\leq}0\ R_1\ B)\ \sqcap\ ({\geq}1\ R_1\ A \sqcup B)\ \sqcap\ ({\geq}1\ R_2\ ({\leq}0\ R_2^{-1}\ ({\geq}1\ R_1\ A))).$$

Figure 4.7 shows the beginning of a run of the $\mathcal{ALCQIb}$-algorithm. After a number of steps, a successor y of x has been generated and the expansion of constraints has produced a complete and clash-free subtree for y. Nevertheless, the concept C is not satisfiable. The expansion of $({\geq}1\ R_2\ ({\leq}0\ R_2^{-1}\ ({\geq}1\ R_1\ A)))$ will eventually lead to the generation of the constraint $x : {\sim}({\geq}1\ R_1\ A) = ({\leq}0\ R_1\ A)$ in $\mathcal{A}_5$, which disallows R_1-successors that satisfy A. This conflicts with the constraints $x : ({\leq}0\ R_1\ B)$ and $x : ({\geq}1\ R_1\ A \sqcup B)$, which require a successor of x that satisfies A. Consider an implementation of the algorithm that employs tracing: the ABox $\mathcal{A}_3$ contains a complete and clash-free subtree for y, which is deleted from memory and it is recorded that the constraint $x : ({\geq}1\ R_1\ A \sqcup B)$ has been satisfied and this constraint is never reconsidered—the conflict goes undetected. To make tracing possible, the $\rightarrow_{\text{choose}}$-rule deletes all information about y when stepping from $\mathcal{A}_4$ to $\mathcal{A}_5$, which, while duplicating some work, makes it possible to detect this conflict even when tracing through the ABox. An implementation that uses tracing can safely discard the information about y from memory once the existence of a complete and clash-free subtree has been established in $\mathcal{A}_3$ because, whenever the effect of an application of the $\rightarrow_{\text{choose}}$-rule might conflict with assertions for a successor y, all required successors of x have to be re-generated anyway.

A similar technique will be used in a subsequent chapter to obtain a PSpace-result for another DL with inverse roles.

Figure 4.7 Inverse roles make tracing difficult.

$$
\begin{aligned}
\{x : C\} &\rightarrow_{\sqcap} \dots \\
&\rightarrow_{\sqcap} \underbrace{\{x : C, x : (\leq 0\ R_1\ B), x : (\geq 1\ R_1\ A \sqcup B), x : (\geq 1\ R_2\ (\leq 0\ R_2^{-1}\ (\geq 1\ R_1\ A)))\}}_{\mathcal{A}_1} \\
&\rightarrow_{\geq} \underbrace{\mathcal{A}_1 \cup \{(x, y) : R_1, (y, x) : R_1^{-1}, y : A \sqcup B, y : \neg B\}}_{\mathcal{A}_2} \rightarrow_{\sqcup} \underbrace{\mathcal{A}_2 \cup \{y : A\}}_{\mathcal{A}_3} \\
&\rightarrow_{\sqcap} \underbrace{\mathcal{A}_3 \cup \{(x, z) : R_2, (z, x) : R_2^{-1}, z : (\leq 0\ R_2^{-1}\ (\geq 1\ R_1\ A))\}}_{\mathcal{A}_4} \\
&\rightarrow_{\text{choose}} \underbrace{\mathcal{A}_1 \cup \{(\leq 0\ R_1\ A)\}}_{\mathcal{A}_5}
\end{aligned}
$$

4.4.2 Correctness of the Algorithm

Like for $\mathcal{ALCQ}$, we show correctness of the $\mathcal{ALCQI}b$-algorithm along the lines of Theorem 3.6.

Termination

Obviously, the deletion of constraints in $\mathcal{A}$ makes a new proof of termination necessary, since the proof of Lemma 4.11 relied on the fact that constraints were never removed from the ABox. Note, however, that the Lemma 4.10 still holds for $\mathcal{ALCQI}b$.

Lemma 4.22 (Termination)
Any sequence of rule applications starting from an ABox $\mathcal{A} = \{x_0 : C\}$ of the $\mathcal{ALCQI}b$ algorithm is finite.

Proof. The sequence of rule applications induces a sequence of trees. As before, the depth and out-degree of this tree is bounded in $|C|$ by Lemma 4.10. For each individual x, $\mathbf{L}(x)$ is a subset of the finite set $clos(C)$. Each application of a rule either

- adds a constraint of the form $x : D$ and hence adds an element to $\mathbf{L}(x)$, or
- adds fresh individuals to $\mathcal{A}$ and hence adds additional nodes to the tree $G_{\mathcal{A}}$, or
- adds a constraint to a node y and deletes all subtrees rooted below y.

Assume that algorithm does not terminate. Due to the mentioned facts this can only be because of an infinite number of deletions of subtrees. Each node can of course only be deleted once, but the successors of a single node may be deleted several times. The root of the constraint system cannot be deleted because it has no predecessor. Hence there are nodes that are never deleted. Choose one of these nodes y with maximum distance from the root, i.e., which has a maximum number of ancestors in $\prec_{\mathcal{A}}$. Suppose that y's successors are deleted only finitely many times. This can not be the case because, after the last deletion of y's successors, the "new" successors were never deleted and thus y would not have maximum distance from the root. Hence y triggers the deletion of its successors infinitely many times. However, the $\rightarrow_{\text{choose}}$-rule is the only rule that leads to a deletion, and it simultaneously leads to an increase of $\mathbf{L}(y)$, namely by the missing concept which caused the deletion of y's successors. This implies the existence of an infinitely increasing chain of subsets of $clos(C)$, which is clearly impossible. ∎

Soundness and Completeness

We start by proving an important property of the interplay of the $\rightarrow_{\geq}$-rule and the $\rightarrow_{\text{choose}}$-rule.

Lemma 4.23
Let $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ be ABoxes generated by the $\mathcal{ALCQI}b$-algorithm, such that $\mathcal{A}_2$ is derived from $\mathcal{A}_1$ by application of the $\rightarrow_{\geq}$-rule to an individual x in a way that creates the new successor y of x, and $\mathcal{A}_3$ is derived from $\mathcal{A}_2$ by zero or more rule applications. If both x, y occur in $\mathcal{A}_3$, then $\{D \mid x : D \in \mathcal{A}_1\} = \{D \mid x : D \in \mathcal{A}_3\}$ and the $\rightarrow_{\text{choose}}$-rule is not applicable to x in $\mathcal{A}_3$ in a way that adds a concept assertion for y.

Proof. Assume that x, y occur in $\mathcal{A}_3$. Then they also occur in all intermediate ABoxes because, once an individual is deleted from the constraint system, it is never re-introduced. The proof is by induction on the number of rule applications necessary to derive $\mathcal{A}_3$ from $\mathcal{A}_2$. If no rule must be applied, then $\mathcal{A}_2 = \mathcal{A}_3$ holds, and since application of the $\rightarrow_{\geq}$-rule to x does not alter the concepts asserted for x, we are done. Now assume that the lemma holds for every ABox $\mathcal{A}'$ derivable from $\mathcal{A}_2$ by n rule applications.

Let $\mathcal{A}_3$ be derivable from $\mathcal{A}_2$ in $n+1$ steps and let $\mathcal{A}'$ be an ABox such that $\mathcal{A}_2 \rightarrow^n \mathcal{A}' \rightarrow \mathcal{A}_3$. Since $\{D \mid x : D \in \mathcal{A}_1\} = \{D \mid x : D \in \mathcal{A}'\}$ holds by induction, also $\{D \mid x : D \in \mathcal{A}_1\} = \{D \mid x : D \in \mathcal{A}_3\}$ holds as long as the rule application that derives $\mathcal{A}_3$ from $\mathcal{A}'$ does not alter the concepts asserted for x.

The $\rightarrow_{\geq}$-rule does not alter the constraints for any individual that is already present in the ABox because it introduces a fresh individual.

The $\rightarrow_{\sqcap}$- or $\rightarrow_{\sqcup}$-rule cannot be applicable to x because, if the rule is applicable in $\mathcal{A}'$, then, since $\{D \mid x : D \in \mathcal{A}_1\} = \{D \mid x : D \in \mathcal{A}'\}$, it is also applicable in $\mathcal{A}_1$ and the $\rightarrow_{\geq}$-rule that creates y is not applicable. Assume that an application of the $\rightarrow_{\text{choose}}$-rule asserts an additional concept for x. Any application of the $\rightarrow_{\text{choose}}$-rule that adds a constraint for x removes the individuals $\{z \mid x \prec_{\mathcal{A}'} z\}$ from $\mathcal{A}'$. This includes y and hence y would not occur in $\mathcal{A}_3$, in contradiction to the assumption that x, y occur in $\mathcal{A}_3$.

Since the concept assertions for x have not changed since the generation of y, it holds that $x : (\bowtie n\ \omega\ D) \in \mathcal{A}_3$ iff $x : (\bowtie n\ \omega\ D) \in \mathcal{A}_1$ and so $\{y : D, y : \sim D\} \cap \mathcal{A}_1$ is ensured by the $\rightarrow_{\geq}$-rule that creates y. The individual y still occurs in $\mathcal{A}_3$ and hence $\{y : D, y : \sim D\} \cap \mathcal{A}_3$ holds, which implies that the $\rightarrow_{\text{choose}}$-rule cannot be applied for the constraint $x : (\bowtie n\ \omega\ D) \in \mathcal{A}_3$ in a way that adds $y : D$ or $y : \sim D$ to $\mathcal{A}_3$. ∎

The correctness of the $\mathcal{ALCQIb}$-algorithm is again proved along the lines of Theorem 3.6, but in a slightly different manner than it was proved for $\mathcal{ALCQ}$. Instead of proving local correctness of the rules, which is difficult to establish due to the deletion of constraints by the $\rightarrow_{\text{choose}}$-rule, we use Property 5'. Additionally, we require a stronger notion of satisfiability than standard ABox satisfiability. Similar as for $\mathcal{ALCQ}$, we define the differentiation $\widehat{\mathcal{A}}$ of an ABox $\mathcal{A}$ by setting

$$\widehat{\mathcal{A}} = \mathcal{A} \cup \{y \not\doteq z \mid \{(x, y) : R, (x, z) : S\} \subseteq \mathcal{A}, y \neq z\}.$$

Note the slight difference to the definition of $\mathcal{ALCQ}$, where only those individuals reachable from x via the same role R were asserted to be distinct. Here, all individuals reachable from x via an arbitrary role are asserted to be distinct. We say that an ABox $\mathcal{A}$ is satisfiable iff there exists a model $\mathcal{I}$ of its differentiation $\widehat{\mathcal{A}}$ that, in addition to what is required by the standard notion of ABox satisfiability from Definition 2.5, satisfies:

$$\text{if } (x, y) : R \in \widehat{\mathcal{A}} \text{ then } \{R \mid (x, y) : R \in \widehat{\mathcal{A}}\} = \{R \mid (x^{\mathcal{I}}, y^{\mathcal{I}})) \in R^{\mathcal{I}}\} \cap \overline{\mathsf{NR}}_C. \tag{§}$$

Note that this additional property is trivially satisfied by a canonical interpretation.

Obviously, Properties 1 and 2 of Theorem 3.6 hold for every ABox generated by the $\mathcal{ALCQIb}$-algorithm.

Lemma 4.24 (Soundness)
Let $\mathcal{A}$ be a complete and clash-free ABox generated by the $\mathcal{ALCQIb}$-algorithm. Then $\mathcal{A}$ is satisfiable, i.e., there exists a model $\mathcal{I}$ of $\widehat{\mathcal{A}}$ that additionally satisfies (§).

Proof. Let $\mathcal{A}$ be a complete and clash-free ABox obtained by a sequence of rule applications starting from $\{x_0 : C\}$. We show that the canonical interpretation $\mathcal{I}_\mathcal{A}$ (as defined in Definition 3.7) is indeed a model of $\widehat{\mathcal{A}}$ that satisfies (§). Please note that we need the condition "$(x, y) : R \in \mathcal{A}$ iff $(y, x) : R^{-1} \in \mathcal{A}$", which is maintained by the algorithm, to make sure that all information from the ABox is reflected in the canonical interpretation.

Every canonical interpretation trivially satisfies (§) and also every two different individuals are interpreted differently, which takes care of the additional assertions in $\widehat{\mathcal{A}}$. So, it remains to show that $x : D \in \mathcal{A}$ implies $x \in D^{\mathcal{I}_\mathcal{A}}$ for all individuals x in $\mathcal{A}$ and all concepts $D \in \mathit{clos}(C)$. This is done by induction over the norm of concepts $\|\cdot\|$. The only interesting cases that are different from the $\mathcal{ALCQ}$-case are the qualifying number restrictions.

- $x : (\geq n\ \omega\ D) \in \mathcal{A}$ implies $\omega^\mathcal{A}(x, D) \geq n$ because $\mathcal{A}$ is complete. Hence, there are n distinct individuals $y_1, \ldots, y_n$ with $y_i : D \in \mathcal{A}$ and $\{R \mid (x, y_i) : R \in \mathcal{A}\} \models \omega$ for each $1 \leq i \leq n$. By induction and Lemma 4.20, we have $y_i \in D^{\mathcal{I}_\mathcal{A}}$ and $(x, y_i) \in \omega^{\mathcal{I}_\mathcal{A}}$ and hence $x \in (\geq n\ \omega\ D)^{\mathcal{I}_\mathcal{A}}$.

- $x : (\leq n\ \omega\ D) \in \mathcal{A}$ implies, for any R that occurs in ω and any y with $(x, y) : R \in \mathcal{A}$, $y : D \in \mathcal{A}$ or $y : \sim D \in \mathcal{A}$. For any predecessor of x, this is guaranteed by the $\rightarrow_{\text{choose}}$-rule. For any successor, this follows from Lemma 4.23. Hence, $x : (\leq n\ \omega\ D)$

is present in $\mathcal{A}$ by the time y is generated and the $\rightarrow_{\geq}$-rule ensures $y : D \in \mathcal{A}$ or $y : \sim D \in \mathcal{A}$.

We show that $\sharp\omega^{\mathcal{I}_\mathcal{A}}(x, D) \leq \sharp\omega^{\mathcal{A}}(x, D)$: assume $\sharp\omega^{\mathcal{I}_\mathcal{A}}(x, D) > \sharp\omega^{\mathcal{A}}(x, D)$. This implies the existence of some y with $(x, y) \in \omega^{\mathcal{I}_\mathcal{A}}$ and $y \in D^{\mathcal{I}_\mathcal{A}}$ but $y : D \notin \mathcal{A}$. Due to the syntactic restriction on role expressions, $(x, y) \in \omega^{\mathcal{I}_\mathcal{A}}$ implies $(x, y) \in R^{\mathcal{I}_\mathcal{A}}$ for some R that occurs in ω and and hence $(x, y) : R \in \mathcal{A}$ must hold by construction of $\mathcal{I}_\mathcal{A}$. The $\rightarrow_{\text{choose}}$-rule and the $\rightarrow_{\geq}$-rule then guarantee that $y : D \notin \mathcal{A}$ implies $y : \sim D \in \mathcal{A}$. By induction this yields $y \in (\sim D)^{\mathcal{I}_\mathcal{A}}$ in contradiction to $y \in D^{\mathcal{I}_\mathcal{A}}$. ∎

Lemma 4.25 (Local Completeness)
If $\mathcal{A}$ is a satisfiable ABox generated by the $\mathcal{ALCQI}b$-algorithm and a rule is applicable to $\mathcal{A}$, then it can be applied in a way that yields a satisfiable $\mathcal{A}'$.

Proof. Let $\mathcal{I}$ be a model of $\widehat{\mathcal{A}}$ that satisfies (§), as required by our notion of satisfiability. We distinguish the different rules. For most rules $\mathcal{I}$ can remain unchanged, in all other cases we explicitly state how $\mathcal{I}$ must be modified in order to witness the satisfiability of the modified ABox.

- The $\rightarrow_{\sqcap}$-rule: if $x : C_1 \sqcap C_2 \in \mathcal{A}$, then $x^{\mathcal{I}} \in (C_1 \sqcap C_2)^{\mathcal{I}}$. This implies $x^{\mathcal{I}} \in C_i^{\mathcal{I}}$ for $i = 1, 2$, and hence satisfiability is preserved.

- The $\rightarrow_{\sqcup}$-rule: if $x : C_1 \sqcup C_2 \in \mathcal{A}$, then $x^{\mathcal{I}} \in (C_1 \sqcup C_2)^{\mathcal{I}}$. This implies $x^{\mathcal{I}} \in C_1^{\mathcal{I}}$ or $x^{\mathcal{I}} \in C_2^{\mathcal{I}}$. Hence the $\rightarrow_{\sqcup}$-rule can add a constraint $x : D$ with $D \in \{C_1, C_2\}$ and maintains satisfiability.

- The $\rightarrow_{\text{choose}}$-rule: obviously, either $y^{\mathcal{I}} \in D^{\mathcal{I}}$ or $y^{\mathcal{I}} \notin D^{\mathcal{I}}$ for any individual y in $\mathcal{A}$. Hence, the rule can always be applied in a way that maintains satisfiability. Deletion of constraints as performed by the $\rightarrow_{\text{choose}}$-rule cannot cause unsatisfiability.

- The $\rightarrow_{\geq}$-rule: if $x : (\geq n\ \omega\ D) \in \mathcal{A}$, then $x^{\mathcal{I}} \in (\geq n\ \omega\ D)^{\mathcal{I}}$. This implies $\sharp\omega^{\mathcal{I}}(x^{\mathcal{I}}, D) \geq n$. We claim that there is an element $a \in \Delta^{\mathcal{I}}$ such that

$$\begin{array}{l} (x^{\mathcal{I}}, a) \in \omega^{\mathcal{I}},\ a \in D^{\mathcal{I}},\ \text{and} \\ a \notin \{z^{\mathcal{I}} \mid (x, z) : S \in \mathcal{A} \text{ for some } S \in \overline{\mathsf{NR}}_C\}. \end{array} \tag{$*$}$$

  We will prove this claim later. Let $E_1, \ldots, E_k$ be an enumeration of the set $\{E \mid x : (\bowtie m\ \sigma\ E) \in \mathcal{A}\}$. The $\rightarrow_{\geq}$-rule can add the constraints

$$\begin{aligned} \mathcal{A}' &= \{y : E_i \mid a \in E_i^{\mathcal{I}}\} \cup \{y : \sim E_i \mid a \notin E_i^{\mathcal{I}}\} \\ \mathcal{A}'' &= \{(x, y) : R \mid R \in \overline{\mathsf{NR}}_C, (x^{\mathcal{I}}, a) \in R^{\mathcal{I}}\} \cup \{(y, x) : R \mid R \in \overline{\mathsf{NR}}_C, (a, x^{\mathcal{I}}) \in R^{\mathcal{I}}\} \end{aligned}$$

  as well as $\{y : D\}$ to $\mathcal{A}$. If we set $\mathcal{I}' := \mathcal{I}[y \mapsto a]$, then $\mathcal{I}'$ is a model of the differentiation of the ABox obtained this way that satisfies (§).

  Why does there exists an element a that satisfies $(*)$? Let $b \in \Delta^{\mathcal{I}}$ be an individual with $(x^{\mathcal{I}}, b) \in \omega^{\mathcal{I}}$ and $b \in D^{\mathcal{I}}$ that appears as an image of an arbitrary element z

with $(x, z) : S \in \mathcal{A}$ for some $S \in \overline{\mathsf{NR}}_C$. The requirement (§) implies that $\{R \mid (x, z) : R \in \mathcal{A}\} \models \omega$ and also $z : D \in \mathcal{A}$ must hold. This can be shown as follows:

Assume $z : D \notin \mathcal{A}$. This implies $z : {\sim}D \in \mathcal{A}$: either $z \prec_{\mathcal{A}} x$, then in order for the $\rightarrow_{\geq}$-rule to be applicable, no non-generating rules and especially the $\rightarrow_{\text{choose}}$-rule is not applicable to x and its ancestor, which implies $\{z : D, z : {\sim}D\} \cap \mathcal{A} \neq \emptyset$. If not $z \prec_{\mathcal{A}} x$, then z must have been generated by an application of the $\rightarrow_{\geq}$-rule to x. Lemma 4.23 implies that at the time of the generation of z already $x : ({\geq}n\ \omega\ D) \in \mathcal{A}$ held and hence the $\rightarrow_{\geq}$-rule ensures $\{z : D, z : {\sim}D\} \cap \mathcal{A} \neq \emptyset$.

In any case $z : {\sim}D \in \mathcal{A}$ holds, which implies $b \notin D^{\mathcal{I}}$, in contradiction to $b \in D^{\mathcal{I}}$.

Together this implies that, whenever an element b with $(x^{\mathcal{I}}, b) \in \omega^{\mathcal{I}}$ and $b \in D^{\mathcal{I}}$ is assigned to an individual z with $(x, z) : S \in \mathcal{A}$, then it must be assigned to an individual that contributes to $\sharp\omega^{\mathcal{A}}(x, D)$. Since the $\rightarrow_{\geq}$-rule is applicable, there are less than n such individuals and hence there must be an unassigned element a as required by $(*)$. ∎

The $\rightarrow_{\text{choose}}$-rule deletes only assertions for successors of a node and hence never deletes any assertions for the root x_0. Hence, for any ABox $\mathcal{A}$ generated by application of the completion rules from an initial ABox $\{x_0 : C\}$, $\{x_0 : C\} \subseteq \mathcal{A}$ holds and hence we get the following.

Lemma 4.26
If a complete and clash-free ABox $\mathcal{A}$ can be generated from an initial ABox $\mathcal{A}_0$, then $\mathcal{A}_0$ is satisfiable.

Proof. From Lemma 4.25, it follows that $\mathcal{A}$ is satisfiable and every model of $\mathcal{A}$ is also a model of $\mathcal{A}_0 = \{x_0 : C\}$ because $\mathcal{A}_0 \subseteq \mathcal{A}$ and $\mathcal{A}_0$ contains no role assertions, which implies $\widehat{\mathcal{A}}_0 = \mathcal{A}_0$ and every interpretation trivially satisfies (§) for $\widehat{\mathcal{A}}_0$. ∎

Hence, we can apply Theorem 3.6 and get:

Corollary 4.27
The $\mathcal{ALCQI}b$-algorithm is a non-deterministic decision procedure for satisfiability of $\mathcal{ALCQI}b$-concepts.

Proof. Termination has been shown in Lemma 4.22. As mentioned before, Property 1 and 2 of Theorem 3.6 are trivially satisfied due to the chosen notion of ABox satisfiability. Property 3 has been shown in Lemma 4.24, Property 4 in Lemma 4.25 and Property 5' in Lemma 4.26. ∎

4.4.3 Complexity of the Algorithm

Like for the optimal algorithm for $\mathcal{ALCQ}$, we have to show that the $\mathcal{ALCQIb}$-algorithm can be implemented in a way that consumes only polynomial space. This is done similarly to the $\mathcal{ALCQ}$-case, but we have to deal with two additional problems: we have to find a way to implement the "reset-restart" caused by the $\rightarrow_{\text{choose}}$-rule, and we have to store the values of the relevant counters $\omega^{\mathcal{A}}(x, D)$. It is impossible to store the values for every possible role expression ω because there are exponentially many inequivalent of these. Fortunately, storing only the values for those ω that actually appear in C is sufficient.

Lemma 4.28
The $\mathcal{ALCQIb}$-algorithm can be implemented in PSPACE.

Proof. Consider the algorithm in Figure 4.8, where Ω_C denotes all role expressions that occur in the input concept C. Like the algorithm for $\mathcal{ALCQ}$, the $\mathcal{ALCQIb}$-algorithm re-uses the space used to check for the existence of a complete and clash-free "subtree" for each successor y of an individual x and keeps only a single path in memory at one time. Counter variables are used to keep track of the values $\sharp\omega^{\mathcal{A}}(x, D)$ for all $\omega \in \Omega_C$ and $D \in \mathit{clos}(C)$.

Resetting a node and restarting the generation of its successors is achieved by jumping to the label restart in the algorithm, which re-initializes all successor counters for a node x. Note, how the predecessor of a node is taken into account when initializing the counter individuals. Since $G_{\mathcal{A}}$ is a tree, every newly generated node has a uniquely determined predecessor and since only safe role expressions occur in Ω_C, it is sufficient to take only this predecessor node into account when initializing the counter.

Let $n = |C|$. For every node x of a path in $G_{\mathcal{A}}$, $\mathcal{O}(n)$ bits suffice to store the constraints of the form $x : D$ and $\mathcal{O}(n^4)$ suffice to store the counters (in binary representation) because $\sharp\Omega_C = \mathcal{O}(n)$, $\sharp\mathit{clos}(C) = \mathcal{O}(n)$, and the out-degree of $G_{\mathcal{A}}$ is bounded by $\mathcal{O}(n) \times 2^n$ (by Lemma 4.10, which also holds for $\mathcal{ALCQIb}$). Also by Lemma 4.10, the length of a path in $G_{\mathcal{A}}$ is bounded by $\mathcal{O}(n)$, which yields an overall memory requirement of $\mathcal{O}(n^5)$ for a path. ∎

Obviously, satisfiability of $\mathcal{ALCQIb}$-concepts is PSPACE-hard, hence Lemma 4.28 and Savitch's Theorem (Savitch, 1970) yield:

Theorem 4.29
Satisfiability of $\mathcal{ALCQIb}$-concepts is PSPACE*-complete if the numbers in the input are represented using binary coding.*

As a simple corollary, we get the solution of an open problem in (Donini et al., 1997):

Corollary 4.30
Satisfiability of $\mathcal{ALCNR}$-concepts is PSPACE*-complete if the numbers in the input are represented using binary coding.*

Figure 4.8 A non-deterministic PSPACE decision procedure for $\mathcal{ALCQIb}$-satisfiability.

```
ALCQIb-Sat(C) := sat(x₀, {x₀ : C})
sat(x, S):
    allocate counters ♯ω^A(x, D) for all ω ∈ Ω_C and D ∈ clos(C).
  restart:
    for each counter ♯ω^A(x, D):
        if (x has a predecessor y ≺_A x with {R | (x, y) : R ∈ A} ⊨ ω and y : D ∈ A)
            then ♯ω^A(x, D) := 1 else ♯ω^A(x, D) := 0
    while (the →⊓- or the →⊔-rule can be applied at x) and (A is clash-free) do
        apply the →⊓- or the →⊔-rule to A.
    od
    if A contains a clash then return "not satisfiable".
    if the →choose-rule is applicable to the constraint x : (⋈n ω D) ∈ A
        then return "restart with D"
    while (the →≥-rule applies to a constraint x : (≥n ω D) ∈ A) do
        non-deterministically choose R ⊆ NR̄_C
        if R ⊭ ω then return "not satisfiable"
        A_new := {y : D} ∪ A′ ∪ A″
        where
            y is a fresh individual
            {E₁, ..., E_k} = {E | x : (⋈m σ E) ∈ A}
            A′ = {y : D₁, ..., y : D_k}, and
            D_i is chosen non-deterministically from {E_i, ∼E_i}
            A″ = {(x, y) : R, (y, x) : R⁻¹ | R ∈ R}
        for each E with y : E ∈ A′ and σ ∈ Ω_C with R ⊨ σ do
            increment ♯σ^A(x, E)
        if x : (≤m σ E) ∈ A and ♯σ^A(x, E) > m
            then return "not satisfiable".
        result := sat(y, A ∪ A_new)
        if result = "not satisfiable" then return "not satisfiable"
        if result = "restart with D" then
            A := A ∪ {x : E}
            where E is chosen non-deterministically from {D, ∼D}
            goto restart
    od
    discard the counters for x from memory.
    return "satisfiable"
```

Proof. The DL $\mathcal{ALCNR}$ is a syntactic restriction of the DL $\mathcal{ALCQIb}$, where we do not allow for inverse roles and in number restrictions $(\bowtie n\ \omega\ D)$, ω must be a conjunction of positive roles and D the tautological concept $\top$. Hence, the $\mathcal{ALCQIb}$-algorithm can immediately be applied to $\mathcal{ALCNR}$-concepts, which yields decidability in PSPACE. ∎

4.5 Reasoning with $\mathcal{ALCQIb}$-Knowledge Bases

So far, we have only dealt with the problem of concept satisfiability rather than satisfiability of knowledge bases. In this section, we will examine the complexity of reasoning with knowledge bases for the DL $\mathcal{ALCQIb}$. For the more "standard" DL $\mathcal{ALCQI}$, this problem has been shown to be EXPTIME-complete by De Giacomo (1995), but this result does not easily transfer to $\mathcal{ALCQIb}$ because of the role expressions and the proof in (De Giacomo, 1995) is only valid in case of unary coding of numbers in the input. Here, we are aiming for a proof that is valid also in the case of binary coding of numbers.

In a first step, we deal with concept satisfiability w.r.t. general TBoxes and prove that this problem can be solved in EXPTIME using an automata approach. ABoxes are then handled by a pre-completion algorithm similar to the one presented by Hollunder (1996) (see also Section 3.2.3). It should be mentioned that the algorithms developed in this section are by no means intended for implementation. They are used only to obtain tight worst-case complexity results. We are also very generous in size or time estimates.

The lower complexity bound for $\mathcal{ALCQIb}$ with general TBoxes is an immediate consequence of Theorem 3.18 because $\mathcal{ALC}$ is strictly contained in $\mathcal{ALCQIb}$.

Lemma 4.31
Satisfiability of $\mathcal{ALCQIb}$-concepts (and hence of ABoxes) w.r.t. general TBoxes is EXPTIME-*hard.*

To establish a matching upper complexity bound, we employ an automata approach, where (un-)satisfiability of concepts is reduced to emptiness of suitable finite automata, usually Büchi word or tree automata (Thomas, 1992). This approach is a valuable tool to establish exact complexity results for DLs and modal logics (Vardi & Wolper, 1986; Lutz & Sattler, 2000), particularly for EXPTIME-complete logics, where tableau approaches—due to their non-deterministic nature—either fail entirely or require very sophisticated techniques (Donini & Massacci, 2000) to prove decidability of the decision problem in EXPTIME.

In general, the automata approach works as follows. To test satisfiability of a concept C w.r.t. a TBox $\mathcal{T}$, an automaton $\mathfrak{A}_{C,\mathcal{T}}$ is constructed that accepts exactly (abstractions of) models of C and $\mathcal{T}$, so that $\mathfrak{A}_{C,\mathcal{T}}$ accepts a non-empty language iff C is satisfiable w.r.t. $\mathcal{T}$. For $\mathcal{ALCQIb}$, we do not require the full complexity of Büchi tree automata—the simpler formalism of *looping tree automata* (Vardi & Wolper, 1994) suffices.

Definition 4.32 (Looping Tree Automata)
For a natural number n, let $[n]$ denote the set $\{1, \ldots, n\}$. An n-ary infinite tree over the alphabet Σ *is a mapping $t : [n]^* \to \Sigma$, where $[n]^*$ denotes the set of finite strings over $[n]$.*

An n-ary looping tree automaton *is a tuple $\mathfrak{A} = (Q, \Sigma, I, \delta)$, where Q is a finite set of states, Σ is a finite alphabet, $I \subseteq Q$ is the set of initial states, and $\delta \subseteq Q \times \Sigma \times Q^n$ is the transition relation. Sometimes, we will view δ as a function from $Q \times \Sigma$ to 2^{Q^n} and write $\delta(q, \sigma)$ for the set of tuples $\{\mathbf{q} \mid (q, \sigma, \mathbf{q}) \in \delta\}$.*

A run *of $\mathfrak{A}$ on an n-ary infinite tree t over Σ is an n-ary infinite tree r over Q such that, for every $p \in [n]^*$,*

$$(r(p), t(p), (r(p1), \ldots, r(pn))) \in \delta.$$

An automaton $\mathfrak{A}$ accepts *t iff there is a run r of $\mathfrak{A}$ on t with $r(\epsilon) \in I$. With $L(\mathfrak{A})$ we denote the* language accepted *by $\mathfrak{A}$ defined by $L(\mathfrak{A}) := \{t \mid \mathfrak{A} \text{ accepts } t\}$.* $\diamond$

For a looping automaton $\mathfrak{A}$, emptiness of $L(\mathfrak{A})$ can be decided efficiently.

Fact 4.33
Let $\mathfrak{A} = (Q, \Sigma, I, \delta)$ be an n-ary looping tree automaton. Emptiness of $L(\mathfrak{A})$ can be decided in time $\mathcal{O}(\sharp Q + \sharp\delta)$.

A polynomial bound directly follows from the quadratic time algorithm for Büchi tree automata (Vardi & Wolper, 1986) of which looping tree automata are special cases. A closer inspection of this algorithm shows that one can even obtain a linear algorithm using the techniques from (Dowling & Gallier, 1984). For our purposes also the mentioned quadratic and really every *polynomial* algorithm suffices.

Before we formally define $\mathfrak{A}_{C,\mathcal{T}}$ we give an informal description of the employed construction of the automaton and the abstraction from an interpretation $\mathcal{I}$ to a tree T we use. Generally speaking, nodes of T correspond to elements of an unraveling of $\mathcal{I}$. In the label of the node, we record the relevant (sub-)concepts from C and $\mathcal{T}$ that are satisfied by this element, and also which roles connect the element to its unique predecessor in $\mathcal{I}$. This information has to be recorded at the node since edges of a tree accepted by a looping automaton are unlabelled. Hence, the label of a node is a *locally consistent* set of "relevant" concepts (as defined below) paired with a set of "relevant" roles.

For now, we fix an $\mathcal{ALCQIb}$-concept C in NNF and an $\mathcal{ALCQIb}$-TBox $\mathcal{T}$. Let $\overline{\mathsf{NR}}_{C,\mathcal{T}}$ be the set of role names that occur in C and $\mathcal{T}$ together with their inverse and $\Omega_{C,\mathcal{T}}$ the set of role-expressions that occur in C and $\mathcal{T}$. The closure $clos(C, \mathcal{T})$ of "relevant" concepts is defined as the smallest set X of concepts such that

- $C \in X$ and $\mathrm{NNF}(\neg C_1 \sqcup C_2) \in X$ for every $C_1 \sqsubseteq C_2 \in \mathcal{T}$
- X is closed under sub-concepts and the application of $\sim$, the operator that maps ever concept C to $\mathrm{NNF}(\neg C)$.

Obviously, $\sharp clos(C, \mathcal{T}) = \mathcal{O}(|C| + |\mathcal{T}|)$ (compare Lemma 4.9).

A subset $\Phi \subseteq clos(C, \mathcal{T})$ is *locally consistent* iff

- for every $D \in clos(C, \mathcal{T})$, $\Phi \cap \{D, \sim D\} \neq \emptyset$ and $\{D, \sim D\} \not\subseteq \Phi$,
- for every $C_1 \sqsubseteq C_2 \in \mathcal{T}$, $\text{NNF}(\neg C_1 \sqcup C_2) \in \Phi$,
- if $C_1 \sqcap C_2 \in \Phi$ then $\{C_1, C_2\} \subseteq \Phi$, and
- if $C_1 \sqcup C_2 \in \Phi$ then $\Phi \cap \{C_1, C_2\} \neq \emptyset$.

The set of locally consistent subsets of $clos(C, \mathcal{T})$ is defined by $\mathsf{lc}(C, \mathcal{T}) = \{\Phi \subseteq clos(C, \mathcal{T}) \mid \Phi \text{ is l.c.}\}$. Obviously, for every element x in a model of $\mathcal{T}$, there exists a set $\Phi \subseteq \mathsf{lc}(C, \mathcal{T})$ such that all concepts from Φ are satisfied by x.

It remains to describe how the role relationships in $\mathcal{I}$ are mapped to T. Unfortunately, it is not possible to simply map successors in $\mathcal{I}$ to successors in T due to the presence of binary coding of numbers in number restrictions. A number restriction of the form $(\geq n\ \omega\ D)$ requires the existence of n successors, where n may be exponential in the size of C if numbers are coded binarily. In this case, the transition table of the corresponding automaton requires double-exponential space in the size of C and the automata approach would not yield the ExpTime-result we desire.

We overcome this problem as follows. Instead of using a k-ary tree, where k somehow depends on the input C and $\mathcal{T}$, we use a binary tree. Required successors t_i of an element s in $\mathcal{I}$ are not mapped to direct successors of the node corresponding to s but rather to nodes that are reachable by zero or more steps to the left and a single step to the right. The dummy label $\langle *, * \rangle$ is used for the auxiliary states that are reachable by left-steps only because these do not correspond to any elements of $\mathcal{I}$. If n successors must be mapped, the subtree rooted n left-steps from the current node is not needed to map any more successors and hence is labelled entirely with $\langle *, * \rangle$. Figure 4.9 illustrates this construction, where Φ_x denotes the concepts from $clos(C, \mathcal{T})$ that are satisfied by x and $\mathbf{R}_x$ the set of roles connecting x with its predecessor.

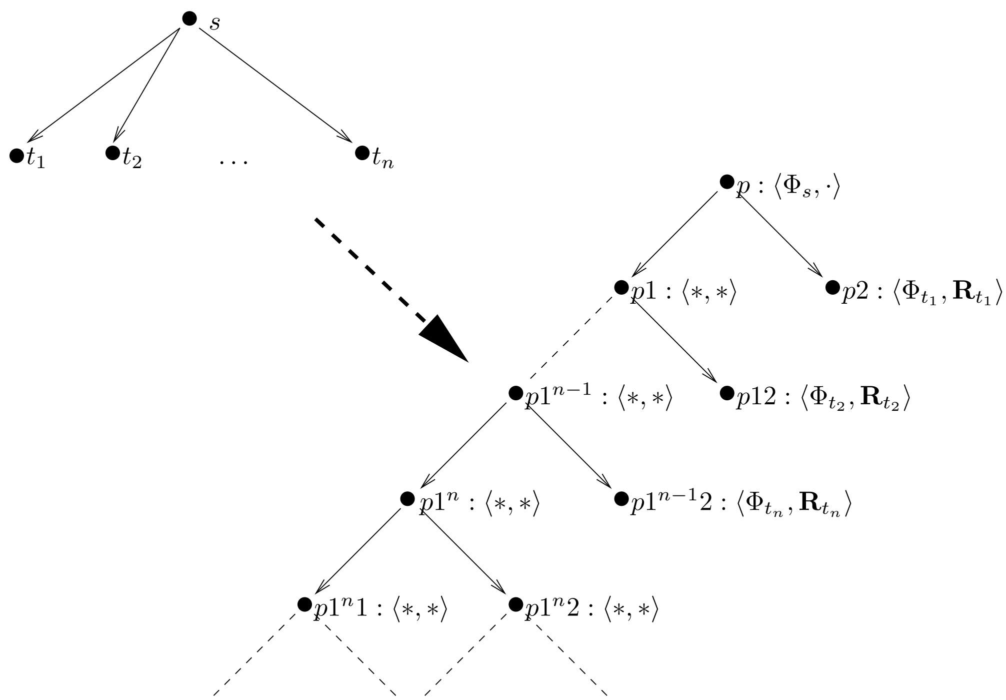
In order to accept exactly the abstractions of models generated by this transformation, it is necessary to perform additional book-keeping in the states. Since successors of the element s are spread through the tree, we must equip the states of $\mathfrak{A}_{C,\mathcal{T}}$ "responsible" for the auxiliary nodes with enough information to ensure that the number restrictions are "obeyed". For this purpose, we use counters to record the minimal and maximal number of ω-successors satisfying D that a node may have. This information is initialized whenever stepping to a right successor and updated when moving to a left successor in the tree. The counters are modelled as functions as follows.

The maximum number $n_{\max}(C, \mathcal{T})$ occurring in a qualifying number restriction in $clos(C, \mathcal{T})$ is defined by $n_{\max}(C, \mathcal{T}) = \max\{n \in \mathbb{N} \mid (\bowtie n\ \omega\ D) \in clos(C, \mathcal{T})\}$ with $\max(\emptyset) := 0$.

The set of concepts that occur in number restrictions and hence must be considered at successor and predecessor nodes is defined by

$$\mathsf{succ}(C, \mathcal{T}) = \{D \mid (\bowtie n\ \omega\ D) \in clos(C, \mathcal{T})\}.$$

Figure 4.9 Transforming a model for C into a tree accepted by $\mathfrak{A}_{C,\mathcal{T}}$



In the automaton, we keep track of the numbers of witnesses for every occurring role expression $\omega \in \Omega_{C,\mathcal{T}}$ and concept from $\mathsf{succ}(C,\mathcal{T})$. This is done using a set of *limiting functions* $\mathsf{limit}(C,\mathcal{T})$ defined by

$$\mathsf{limit}(C,\mathcal{T}) := \{f \mid f : \Omega_{C,\mathcal{T}} \times \mathsf{succ}(C,\mathcal{T}) \to \{0,\dots,n_{\max}(C,\mathcal{T}),\infty\}\}.$$

The maximum/minimum number of allowed/required ω-successors satisfying a certain concept D imposed by number restrictions in a set of concepts is captured by the functions

$$\min,\max : \mathsf{lc}(C,\mathcal{T}) \times \overline{\mathsf{NR}}_{C,\mathcal{T}} \times \mathsf{succ}(C,\mathcal{T}) \to \{0,\dots,n_{\max}(C,\mathcal{T}),\infty\}$$

defined by

$$\begin{aligned}\max(\Phi,\omega,D) &= \min\{n \mid ({\leq} n\ \omega\ D) \in \Phi\}\\ \min(\Phi,\omega,D) &= \max\{n \mid ({\geq} n\ \omega\ D) \in \Phi\}\end{aligned}$$

with $\min(\emptyset) := \infty$.

In the automaton $\mathfrak{A}_{C,\mathcal{T}}$, each state consists of a locally consistent set, a set of roles, and two limiting functions for the upper and lower bounds. There are three kinds of states.

- states that label nodes of T corresponding to elements of the interpretation. These states record the locally consistent set Φ labelling that node, the set of roles that connect the corresponding element to its unique predecessor in $\mathcal{I}$ and the appropriate initial values of the counters for this node—taking into account the concepts satisfied by the predecessor. This is necessary due to the presence of inverse roles.

- states labelling nodes that are reachable from a node s corresponding to an element of $\mathcal{I}$ by one or more steps to the left. These states are marked by an empty set of roles and record the locally consistent set labelling s to allow for the correct initialization of the counters for nodes corresponding to successors of s. Moreover, their limiting functions record the upper and lower bound of ω-successors of s still allowed/required. According to these functions, their right successor state "expects" a node corresponding to a successor of s and their left successor state a further auxiliary node. The limiting functions of this auxiliary state are adjusted according to the right successor. Once sufficiently many successors have been generated, the automaton switches to the following dummy state.
- a dummy state $\langle *, *, *, * \rangle$, which reproduces itself and accepts a tree entirely labelled with $\langle *, * \rangle$.

For a role $R \in \overline{\mathsf{NR}}_{C,\mathcal{T}}$, we define $\mathsf{Inv}(R)$ by setting

$$\mathsf{Inv}(R) = \begin{cases} R^{-1} & \text{if } R \in \mathsf{NR}, \\ S & \text{if } R = S^{-1} \text{ for some } S \in \mathsf{NR}, \end{cases}$$

and for a set of roles $\mathbf{R}$ we define $\mathsf{Inv}(\mathbf{R}) = \{\mathsf{Inv}(R) \mid R \in \mathbf{R}\}$. We are now ready to define the automaton $\mathfrak{A}_{C,\mathcal{T}}$.

Definition 4.34
Let C be an $\mathcal{ALCQIb}$-concept in NNF and $\mathcal{T}$ an $\mathcal{ALCQIb}$-TBox. The binary looping tree automaton $\mathfrak{A}_{C,\mathcal{T}} = (Q, \Sigma, I, \delta)$ for C and $\mathcal{T}$ is defined by

$$\begin{aligned}
Q &= \Big(\mathit{lc}(C,\mathcal{T}) \times 2^{\overline{\mathsf{NR}}_{C,\mathcal{T}}} \times \mathit{limit}(C,\mathcal{T}) \times \mathit{limit}(C,\mathcal{T})\Big) \cup \{\langle *, *, *, * \rangle\} \\
\Sigma &= \Big(\mathit{lc}(C,\mathcal{T}) \times 2^{\overline{\mathsf{NR}}_{C,\mathcal{T}}}\Big) \cup \{\langle *, * \rangle\} \\
I &= \{\langle \Phi, \overline{\mathsf{NR}}_{C,\mathcal{T}}, \ell, h \rangle \in Q \mid C \in \Phi, \ell = \lambda\omega D.\min(\Phi,\omega,D), h = \lambda\omega D.\max(\Phi,\omega,D)\} \\
\delta &\subseteq Q \times \Sigma \times Q^2,
\end{aligned}$$

such that δ is the maximal transition relation with $(\langle *,*,*,* \rangle, \langle *,* \rangle, \langle *,*,*,* \rangle, \langle *,*,*,* \rangle) \in \delta$ and if $(q_0, \sigma, q_1, q_2) \in \delta$ with $q_0 \neq \langle *, *, *, * \rangle$ and $q_i = \langle \Phi_i, \mathbf{R}_i, \ell_i, h_i \rangle$ then

(A1) *if $\mathbf{R}_0 \neq \emptyset$ then $\sigma = \langle \Phi_0, \mathbf{R}_0 \rangle$ else $\sigma = \langle *, * \rangle$*

(A2) *if, for all $\omega \in \Omega_{C,\mathcal{T}}$ and $D \in \mathit{succ}(C,\mathcal{T})$, $\ell_0(\omega, D) = 0$, then $q_1 = q_2 = \langle *, *, *, * \rangle$*

(A3) *otherwise,* $\Phi_2 \in \mathsf{lc}(C, \mathcal{T})$ *and* $\mathbf{R}_2 \subseteq \overline{\mathsf{NR}}_{C,\mathcal{T}}$ *such that there is a* $\omega \in \Omega_{C,\mathcal{T}}$ *and a* $D \in \Phi_2$ *with* $\mathbf{R}_2 \models \omega$ *and* $\ell_0(\omega, D) > 0$. *As an auxiliary function, we define*

$$e(\Phi, \mathbf{R}, \omega, D) = \begin{cases} , 1 & \text{if } \mathbf{R} \models \omega \text{ and } D \in \Phi \\ 0 & \text{otherwise,} \end{cases}$$

and require, for all $\omega \in \Omega_{C,\mathcal{T}}$ *and* $D \in \mathit{clos}(C, \mathcal{T})$,

$$\begin{array}{llll} \text{if} & \max(\Phi_2, \omega, D) = 0 & \text{then} & e(\Phi_0, \mathsf{Inv}(\mathbf{R}_2), \omega, D) = 0 \\ \text{if} & h_0(\omega, D) = 0 & \text{then} & e(\Phi_2, \mathbf{R}_2, \omega, D) = 0. \end{array} \quad (*)$$

Finally, $\Phi_1 = \Phi_0, \mathbf{R}_1 = \emptyset$ *and*

$$\begin{aligned} \ell_1 &= \lambda\omega D.\ell_0(\omega, D) \dot{-} e(\Phi_2, \mathbf{R}_2, \omega, D), \\ h_1 &= \lambda\omega D.h_0(\omega, D) - e(\Phi_2, \mathbf{R}_2, \omega, D), \\ \ell_2 &= \lambda\omega D.\min(\Phi_2, \omega, D) \dot{-} e(\Phi_0, \mathsf{Inv}(\mathbf{R}_2), \omega, D), \text{ and} \\ h_2 &= \lambda\omega D.\max(\Phi_2, \omega, D) - e(\Phi_0, \mathsf{Inv}(\mathbf{R}_2), \omega, D) \end{aligned}$$

must hold, where $\dot{-}$ *denotes subtraction in* $\mathbb{N}$, *i.e.,* $x \dot{-} y = \max(0, x - y)$. ◇

The choice of $\overline{\mathsf{NR}}_{C,\mathcal{T}}$ as the role component of the initial states in I is arbitrary and indeed every non-empty set of could be used instead of $\overline{\mathsf{NR}}_{C,\mathcal{T}}$. Note that the subtraction in the requirements for h_1 and h_2 never yields a negative value because of $(*)$. Moreover, $\mathfrak{A}_{C,\mathcal{T}}$ is small enough (i.e., exponential in the input) to be of use in our further considerations:

Lemma 4.35
Let C *be a* $\mathcal{ALCQI}b$*-concept in NNF,* $\mathcal{T}$ *an* $\mathcal{ALCQI}b$*-TBox,* $m = |C| + |\mathcal{T}|$, *and* $\mathfrak{A}_{C,\mathcal{T}} = (Q, \Sigma, I, \delta)$ *the looping tree automaton for* C *and* $\mathcal{T}$. *Then*

$$\sharp Q + \sharp\delta = \mathcal{O}(2^{m^5}).$$

Proof. The cardinality of $\mathsf{lc}(C, \mathcal{T})$ is bounded by $2^{\sharp \mathit{clos}(C,\mathcal{T})} = \mathcal{O}(2^m)$. The cardinality of $\overline{\mathsf{NR}}_{C,\mathcal{T}}$ is bounded by $2m$ and hence $\sharp 2^{\overline{\mathsf{NR}}_{C,\mathcal{T}}} = \mathcal{O}(2^m)$. Finally, the cardinality of $\mathsf{limit}(C, \mathcal{T})\{f \mid f : \Omega_{C,\mathcal{T}} \times \mathsf{succ}(C, \mathcal{T}) \to \{0, \ldots, n_{\max}(C, \mathcal{T}), \infty\}\}$ is bounded by $(n_{\max}(C, \mathcal{T}) + 2)^{(\sharp\Omega_{C,\mathcal{T}} \times \sharp\mathsf{succ}(C,\mathcal{T}))} = \mathcal{O}((2^m)^{m^2}) = \mathcal{O}(2^{m^3})$, where 2^m is an upper bound for $n_{\max}(C, \mathcal{T})$ if numbers are coded binarily in the input. Summing up, we get $\mathcal{O}(2^m \times 2^m \times 2^{m^3}) = \mathcal{O}(2^{m^4})$ as a bound for $\sharp Q$ and $\mathcal{O}(2^{m^5})$ as a bound for $\sharp\delta$, which dominates $\sharp Q$. ∎

We now show that emptiness of $L(\mathfrak{A}_{C,\mathcal{T}})$ is indeed equivalent to unsatisfiability of C w.r.t. $\mathcal{T}$.

Lemma 4.36
For an $\mathcal{ALCQI}b$*-concept* C *in NNF and a* $\mathcal{ALCQI}b$*-TBox* $\mathcal{T}$, $L(\mathfrak{A}_{C,\mathcal{T}}) \neq \emptyset$ *iff* C *is satisfiable w.r.t.* $\mathcal{T}$.

Proof. Assume $L(\mathfrak{A}_{C,\mathcal{T}}) \neq \emptyset$, T is a tree accepted by $\mathfrak{A}_{C,\mathcal{T}}$, and r is an arbitrary run of $\mathfrak{A}_{C,\mathcal{T}}$ on T with $r(\epsilon) \in I$. From T, we will construct a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for C and $\mathcal{T}$, which proves satisfiability of C w.r.t. $\mathcal{T}$. For every path $p \in \{1,2\}^*$ with $r(p) = \langle \Phi, \mathbf{R}, \ell, h\rangle$, we define $\Phi_p := \Phi, \mathbf{R}_p := \mathbf{R}, \ell_p := \ell$, and $h_p := h$.

The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is defined by $\Delta^{\mathcal{I}} = \{p \in \{1,2\}^*2 \mid T(p) \neq \langle *, *\rangle\} \cup \{\epsilon\}$. Hence, $\Delta^{\mathcal{I}}$ contains only "right successors" and the root. For concept names A, we define

$$A^{\mathcal{I}} = \{p \in \Delta^{\mathcal{I}} \mid A \in \Phi_p\}.$$

For the interpretation of roles, we define

$$\begin{aligned} R^{\mathcal{I}} = &\{(p,p') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid p' \in p1^*2, R \in \mathbf{R}_{p'}\} \cup \\ &\{(p',p) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid p' \in p1^*2, R \in \mathsf{Inv}(\mathbf{R}_{p'})\}. \end{aligned}$$

Before we prove that $\mathcal{I}$ is indeed a model for C and $\mathcal{T}$, we state some general properties of the automaton and this construction.

(R1) Due to the construction of $\Delta^{\mathcal{I}}$, for every $p \in \Delta^{\mathcal{I}}$, $\mathbf{R}_p \neq \emptyset$ and hence $r(p) \neq \langle *,*,*,*\rangle$.

(R2) "Once $\langle *,*,*,*\rangle$, always $\langle *,*,*,*\rangle$." For a path $p \in \{1,2\}^*$, if $r(p) = \langle *,*,*,*\rangle$, then, for all p' with $p' \in p\{1,2\}^*$, $T(p') = \langle *,*\rangle$ and $r(p') = \langle *,*,*,*\rangle$.

(R3) "A left successor is either $\langle *,*,*,*\rangle$ or an auxiliary state, in which case it is labelled with the same set from $\mathsf{lc}(C,\mathcal{T})$." For a path $p \in \{1,2\}^*$, if $r(p) = \langle \Phi, \mathbf{R}, \ell, h\rangle$, then, for all $p' \in p1^*$, if $r(p') \neq \langle *,*,*,*\rangle$ then $r(p')$ is of the form $r(p') = \langle \Phi, \emptyset, \ell', h'\rangle$.

(R4) "h and ℓ are a lower an upper bounds on the number of successors of a node." For a path $p \in \{1,2\}^*$ with $T(p) \neq \langle *,*\rangle$, $\omega \in \Omega_{C,\mathcal{T}}$, and $D \in \mathsf{succ}(C,\mathcal{T})$,

$$\ell_p(\omega, D) \leq \sharp\{p' \in p1^*2 \mid \mathbf{R}_{p'} \models \omega, D \in \Phi_{p'}\} \leq h_p(\omega, D).$$

This property is less obvious than the others and we give a proof by induction on

$$\|p\| = \sum_{\omega \in \Omega_{C,\mathcal{T}}, D \in \mathsf{succ}(C,\mathcal{T})} \ell_p(\omega, D).$$

If $\|p\| = 0$, then $\ell_p(\omega, D) = 0$ for all $\omega \in \Omega_{C,\mathcal{T}}$ and $D \in \mathsf{succ}(C,\mathcal{T})$ and hence, by (A2) and (R2), for all ancestors $p' \in p1^*2$ of p, $r(p') = \langle *,*,*,*\rangle$ and $T(p') = \langle *,*\rangle$. Thus

$$0 = \ell_p(\omega, D) = \sharp\{p' \in p1^*2 \mid \mathbf{R}_{p'} \models \omega, D \in \Phi_{p'}\} \leq h_p(\omega, D)$$

holds for all $\omega \in \Omega_{C,\mathcal{T}}$ and $D \in \mathsf{succ}(C,\mathcal{T})$.

If $\|p\| > 0$ then there is an $\omega \in \Omega_{C,\mathcal{T}}$ and a $D \in \mathsf{succ}(C,\mathcal{T})$ with $\ell_p(\omega, D) > 0$, $\mathbf{R}_{p2} \models \omega$, and $D \in \Phi_{p2}$. Hence, $\ell_{p1}(\omega, D) = \ell_p(\omega, D) - 1$ and $\|p1\| < \|p\|$ by (A3) and we can use the induction hypothesis for $p1$.

For all $\omega \in \Omega_{C,\mathcal{T}}$ and $D \in \mathsf{succ}(C, \mathcal{T})$,

$$\begin{aligned}
\ell_p(\omega, D) &\leq \ell_{p1}(\omega, D) + e(\Phi_{p2}, \mathbf{R}_{p2}, \omega, D) \\
&\leq^{(*)} \sharp\{p' \in p11^*2 \mid \mathbf{R}_{p'} \models \omega, D \in \Phi_{p'}\} + \sharp\{p' \in p2 \mid \mathbf{R}_{p'} \models \omega, D \in \Phi_{p'}\} \\
&= \sharp\{p' \in p1^*2 \mid \mathbf{R}_{p'} \models \omega, D \in \Phi_{p'}\} \\
&= \sharp\{p' \in p11^*2 \mid \mathbf{R}_{p'} \models \omega, D \in \Phi_{p'}\} + \sharp\{p' \in p2 \mid \mathbf{R}_{p'} \models \omega, D \in \Phi_{p'}\} \\
&\leq^{(*)} h_{p1}(\omega, D) + e(\Phi_{p2}, \mathbf{R}_{p2}, \omega, D) \\
&= h_p(\omega, D),
\end{aligned}$$

where the steps marked with $(*)$ use the induction hypothesis. This is what we needed to show.

(R5) For two paths $p, q \in \Delta^{\mathcal{I}}$ and a role expression $\omega \in \Omega_{C,\mathcal{T}}$, if $(p, q) \in \omega^{\mathcal{I}}$ then $q \in p1^*2$ or $p \in q1^*2$.

Because of the syntactic restriction to safe role expressions in $\mathcal{ALCQIb}$, for $(p, q) \in \omega^{\mathcal{I}}$ to hold there must be a role $R \in \overline{\mathsf{NR}}_{C,\mathcal{T}}$ such that $(p, q) \in R^{\mathcal{I}}$. By construction of $R^{\mathcal{I}}$, this can only be the case if $q \in p1^*2$ or $p \in q1^*2$.

(R6) For two paths $p, q \in \Delta^{\mathcal{I}}$ with $q \in p1^*2$ and a role ω, $(p, q) \in \omega^{\mathcal{I}}$ iff $\mathbf{R}_q \models \omega$ and $(q, p) \in \omega^{\mathcal{I}}$ iff $\mathsf{Inv}(\mathbf{R}_q) \models \omega$.

For every $R \in \overline{\mathsf{NR}}_{C,\mathcal{T}}$, $(p, q) \in R^{\mathcal{I}}$ iff $R \in \mathbf{R}_q$ holds as follows. For a (non-inverse) role $R \in \overline{\mathsf{NR}}_{C,\mathcal{T}} \cap \mathsf{NR}$, immediately by the construction of $R^{\mathcal{I}}$, $(p, q) \in R^{\mathcal{I}}$ iff $R \in \mathbf{R}_q$. For an inverse role $R = S^{-1}$ with $S \in \overline{\mathsf{NR}}_{C,\mathcal{T}} \cap \mathsf{NR}$, $(p, q) \in R^{\mathcal{I}}$ iff $(q, p) \in S^{\mathcal{I}}$ iff $\mathsf{Inv}(S) = R \in \mathbf{R}_q$. Hence, $(p, q) \in \omega^{\mathcal{I}}$ iff $\mathbf{R}_q \models \omega$. Similarly, for every $R \in \overline{\mathsf{NR}}_{C,\mathcal{T}}$, $(q, p) \in R^{\mathcal{I}}$ iff $\mathsf{Inv}(R) \in \mathbf{R}_q$, and hence $(q, p) \in \omega^{\mathcal{I}}$ iff $\mathsf{Inv}(\mathbf{R}_q) \models \omega$.

Using these properties we can now show:

Claim 4.37 For all $p \in \Delta^{\mathcal{I}}$ and $D \in \Phi_p$, $p \in D^{\mathcal{I}}$.

The proof is by induction on the norm $\|\cdot\|$ of the concepts (as defined Definition 4.12). The base cases are $D = A$ or $D = \neg A$ for a concept name $A \in \mathsf{NC}$. For $D = A$ this is immediate by the definition of $A^{\mathcal{I}}$. For the case $D = \neg A$, since $\Phi_p \in \mathsf{lc}(C, \mathcal{T})$, $\neg A \in \Phi_p$ implies $A \notin \Phi_p$ and hence $p \in (\neg A)^{\mathcal{I}}$. For the induction step, we distinguish the different concept operators of $\mathcal{ALCQIb}$.

- If $D = C_1 \sqcap C_2 \in \Phi_p$ then, since $\Phi_p \in \mathsf{lc}(C, \mathcal{T})$, also $\{C_1, C_2\} \subseteq \Phi_p$. Hence, by induction, $p \in C_1^{\mathcal{I}}$, $p \in C_2^{\mathcal{I}}$ and thus $p \in D^{\mathcal{I}}$.
- The case $D = C_1 \sqcup C_2$ is similar to the previous one.
- Now assume $D = (\bowtie n\ \omega\ E)$. For every $q \in \Delta^{\mathcal{I}}$, $\Phi_q \in \mathsf{lc}(C, \mathcal{T})$ and hence $E \in \Phi_q$ iff $\sim E \notin \Phi_q$. Since $\|E\| = \|\sim E\| < \|D\|$, by induction, $q \in E^{\mathcal{I}}$ iff $E \in \Phi_q$ holds for every $q \in \Delta^{\mathcal{I}}$.

If $p = \epsilon$ is the root of T then, by (R5) and (R6),

$$\sharp\{q \mid (p,q) \in \omega^{\mathcal{I}}, q \in E^{\mathcal{I}}\} = \sharp\{q \in p1^*2 \mid \mathbf{R}_q \models \omega, E \in \Phi_q\}$$

and hence, by (R4),

$$\min(\Phi_p, \omega, E) \leq \sharp\{q \mid (p,q) \in \omega^{\mathcal{I}}, q \in E^{\mathcal{I}}\} \leq \max(\Phi_p, \omega, E).$$

If $p \neq \epsilon$, then $p \in \{1,2\}^*2$ is a "right successor". Let q_0 be the unique path in $\{1,2\}^*2 \cup \{\epsilon\}$ with $p = q_0 1^k 2$, i.e., p's "predecessor" in $\mathcal{I}$.

$$\begin{aligned}
&\sharp\{q \mid (p,q) \in \omega^{\mathcal{I}}, q \in E^{\mathcal{I}}\}\\
=&\sharp\{q \in p1^*2 \mid \mathbf{R}_q \models \omega, E \in \Phi_q\} + e(\Phi_{q_0}, \mathsf{Inv}(\mathbf{R}_p), \omega, E)\\
=&\sharp\{q \in p1^*2 \mid \mathbf{R}_q \models \omega, E \in \Phi_q\} + e(\Phi_{q_0 1^k}, \mathsf{Inv}(\mathbf{R}_p), \omega, E).
\end{aligned}$$

If $E \notin \Phi_{q_0}$ or $\mathsf{Inv}(\mathbf{R}_p) \not\models \omega$, then $e(\Phi_{q_0 1^k}, \mathsf{Inv}(\mathbf{R}_p), \omega, E) = 0$ and

$$\begin{aligned}
\min(\Phi_p, \omega, E) &= \ell_p(\omega, E)\\
&\leq \sharp\{q \mid q \in p1^*2, \mathbf{R}_q \models \omega, E \in \Phi_q\}\\
&= \sharp\{q \mid (p,q) \in \omega^{\mathcal{I}}, q \in E^{\mathcal{I}}\}\\
&\leq h_p(\omega, E) = \max(\Phi_p, \omega, E)
\end{aligned}$$

holds because of induction, (R4), (R5), and (R6). If $E \in \Phi_{q_0}$ and $\mathsf{Inv}(\mathbf{R}_p) \models \omega$, then $e(\Phi_{q_0 1^k}, \mathsf{Inv}(\mathbf{R}_p), \omega, E) = 1$ and

$$\begin{aligned}
\min(\Phi_p, \omega, E) &\leq \ell_p(\omega, E) + 1\\
&\leq \sharp\{q \in p1^*2 \mid \mathbf{R}_q \models \omega, E \in \Phi_q\} + 1\\
&= \sharp\{q \mid (p,q) \in \omega^{\mathcal{I}}, q \in E^{\mathcal{I}}\}\\
&\leq h_p(\omega, E) + 1 = \max(\Phi_p, \omega, E)
\end{aligned}$$

again holds by (R4), (R5), and (R6).

If $D = ({\geq} n\ \omega\ E)$ then $n \leq \min(\Phi_p, \omega, E) \leq \sharp\{q \mid (p,q) \in \omega^{\mathcal{I}}, q \in E^{\mathcal{I}}\}$ and hence $p \in D^{\mathcal{I}}$. If $D = ({\leq} n\ \omega\ E)$ then $n \geq \max(\Phi_p, \omega, E) \geq \sharp\{q \mid (p,q) \in \omega^{\mathcal{I}}, q \in E^{\mathcal{I}}\}$ and hence $p \in D^{\mathcal{I}}$.

This finishes the proof of the claim, which yields the *only-if* direction of the lemma: if $L(\mathfrak{A}_{C,\mathcal{T}}) \neq \emptyset$ then there exists a tree $T \in L(\mathfrak{A}_{C,\mathcal{T}})$ and a corresponding interpretation $\mathcal{I}$ that satisfies the claim. Since $C \in \Phi_\epsilon$, $\epsilon \in C^{\mathcal{I}}$ and hence $C^{\mathcal{I}} \neq \emptyset$. Also, for every $p \in \Delta^{\mathcal{I}}$ and every $C_1 \sqsubseteq C_2 \in \mathcal{T}$, $\mathrm{NNF}(\neg C_1 \sqcup C_2) \in \Phi_p$. Hence $(\neg C_1 \sqcup C_2)^{\mathcal{I}} = \Delta^{\mathcal{I}}$ and $\mathcal{I} \models \mathcal{T}$.

For the *if*-direction, let C be satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ a model of $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. We construct a tree T from $\mathcal{I}$ that is accepted by $\mathfrak{A}_{C,\mathcal{T}}$. To this purpose, we define a function $\pi : \{1,2\}^* \rightarrow \Delta^{\mathcal{I}} \dot\cup \{*\}$ and maintain an agenda of paths $p \in \{1,2\}$ whose successors still need consideration.

Let $s \in \Delta^{\mathcal{I}}$ be an arbitrary element such that $s \in C^{\mathcal{I}}$. Set $\pi(\epsilon) = s$ and $T(\epsilon) = \langle \Phi_s, \overline{\mathsf{NR}}_{C,\mathcal{T}} \rangle$ with $\Phi_s = \{D \in \mathit{clos}(C, \mathcal{T}) \mid s \in D^{\mathcal{I}}\}$. Initialize the agenda with ϵ.

Pick the first element $p \in \{1,2\}^*$ off the agenda. For $s = \pi(p)$, let $\Phi_s = \{D \in \mathit{clos}(C, \mathcal{T}) \mid s \in D^{\mathcal{I}}\}$ and let $X \subseteq \Delta^{\mathcal{I}}$ be a set such that

- $X \subseteq \{t \in \Delta^{\mathcal{I}} \mid (s,t) \in R^{\mathcal{I}}$ for some $R \in \overline{\mathsf{NR}}_{C,\mathcal{T}}\}$.
- For every $(\geq n\ \omega\ D) \in \Phi_s$ there are $t_1, \ldots t_n \in X$ with $(s,t_i) \in \omega^{\mathcal{I}}$, $t_i \in D^{\mathcal{I}}$ for $1 \leq i \leq n$ and $t_i \neq t_j$ for $1 \leq i < j \leq n$.
- X is minimal w.r.t. set cardinality with these properties.

Such a set X exists, is finite, possibly empty, and not necessarily uniquely defined. Let $\{t_1, \ldots, t_n\}$ be an enumeration of X.

- For every $1 < i \leq n$, we set $\pi(p1^{i-1}) = *$ and $T(p1^{i-1}) = \langle *, * \rangle$.
- For every $1 \leq i \leq n$, we set $\pi(p1^{i-1}2) = t_i$ and

  $$T(p1^{i-1}2) = \langle \Phi_{t_i}, \mathbf{R}_{t_i} \rangle$$

  where

  $$\begin{aligned} \Phi_{t_i} &= \{D \in \mathit{clos}(C,\mathcal{T}) \mid t_i \in D^{\mathcal{I}}\} \\ \mathbf{R}_{t_i} &= \{R \in \overline{\mathsf{NR}}_{C,\mathcal{T}} \mid (s,t_i) \in R^{\mathcal{I}}\}. \end{aligned}$$

  Put $p1^{i-1}2$ at the end of the agenda.
- Finally, for all $p' \in p1^n\{1,2\}^*$ we define $\pi(p') = *$ and $T(p') = \langle *, * \rangle$.

Figure 4.9 illustrates this construction.

Continuing this process until the agenda runs empty (or indefinitely if it never does) eventually defines $T(p)$ for every $p \in \{1,2\}^*$ (since the agenda is organised as a queue, every element will eventually be taken off the agenda). The proof that $T \in L(\mathfrak{A}_{C,\mathcal{T}})$ (and hence $L(\mathfrak{A}_{C,\mathcal{T}}) \neq \emptyset$) is relatively simple and omitted here. ∎

Theorem 4.38
Satisfiability of $\mathcal{ALCQI}b$-concepts w.r.t. general TBoxes is ExpTime*-complete, even if numbers in the input are represented in binary coding.*

Proof. ExpTime-hardness was established in Lemma 4.31. By Lemma 4.36, generating $\mathfrak{A}_{C,\mathcal{T}}$ and testing $L(\mathfrak{A}_{C,\mathcal{T}})$ for emptiness decides satisfiability of C w.r.t. $\mathcal{T}$. Due to Lemma 4.35 and Fact 4.33 this can be done in time exponential in $|C| + |\mathcal{T}|$. ∎

Now that we know how to deal with satisfiability of $\mathcal{ALCQIb}$-concept w.r.t. TBoxes, we show how satisfiability of full knowledge bases can reduced to that problem using a pre-completion technique similar to the one in (Hollunder, 1996) for $\mathcal{ALCQ}$-knowledge bases (see also Section 3.2.3).

The definition of $clos(\cdot)$ is extended to $\mathcal{ALCQIb}$-knowledge bases as follows. For a $\mathcal{ALCQIb}$-knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we define $clos(\mathcal{K})$ as the smallest set X that satisfies the following properties:

- for every $x : D \in \mathcal{A}$, $\mathrm{NNF}(D) \in X$
- for every $C_1 \sqsubseteq C_2 \in \mathcal{T}$, $\mathrm{NNF}(\neg C_1 \sqcup C_2) \in X$
- X is closed under sub-concepts and the application of $\sim$.

Again, $\sharp clos(\mathcal{K}) = \mathcal{O}(|\mathcal{K}|)$ holds (compare Lemma 4.9).

Definition 4.39
Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ALCQIb}$-knowledge base. A knowledge base $\mathcal{K}' = (\mathcal{T}, \mathcal{A}')$ is a pre-completion *of $\mathcal{K}$, if*

1. *there is a surjective function*

$$f : \{x \in \mathsf{NI} \mid x \text{ occurs in } \mathcal{A}\} \to \{x \in \mathsf{NI} \mid x \text{ occurs in } \mathcal{A}'\}$$

   such that

   - *if $x : C \in \mathcal{A}$ then $f(x) : C \in \mathcal{A}'$*
   - *if $(x, y) : R \in \mathcal{A}$ then $(f(x), f(y)) : R \in \mathcal{A}'$*

2. *for every x that occurs in $\mathcal{A}'$ and every $D \in clos(\mathcal{K})$, $x : D \in \mathcal{A}'$ or $x : {\sim}D \in \mathcal{A}'$*
3. *for every x that occurs in $\mathcal{A}'$, if $x : C_1 \sqcap C_2 \in \mathcal{A}'$ then $x : C_1 \in \mathcal{A}'$ and $x : C_2 \in \mathcal{A}'$*
4. *for every x that occurs in $\mathcal{A}'$, if $x : C_1 \sqcup C_2 \in \mathcal{A}'$ then $x : C_1 \in \mathcal{A}'$ or $x : C_2 \in \mathcal{A}'$*
5. *for every two distinct x, y that occur in $\mathcal{A}'$, $x \neq y \in \mathcal{A}'$*

A knowledge base $\mathcal{K}'$ that satisfies 2–5 is called pre-completed. $\diamond$

It is easy to see that a knowledge base is satisfiable iff it has as pre-completion that has a model that *exactly* satisfies the role assertions:

Lemma 4.40
Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ALCQIb}$-knowledge base and $\overline{\mathsf{NR}}_\mathcal{K}$ the set of roles that occur in $\mathcal{K}$ together with their inverse. $\mathcal{K}$ is satisfiable iff there exists pre-completion $\mathcal{K}' = (\mathcal{T}, \mathcal{A}')$ of $\mathcal{K}$ and a model $\mathcal{I}$ of $\mathcal{K}'$ such that, for every $x, y \in \mathsf{NI}$ that occur in $\mathcal{A}'$,

$$\{R \in \overline{\mathsf{NR}}_\mathcal{K} \mid (x,y) : R \in \mathcal{A}'\} = \{R \in \overline{\mathsf{NR}}_\mathcal{K} \mid (x,y) \in R^\mathcal{I}\}.$$

For a pre-completion $\mathcal{K}' = (\mathcal{T}, \mathcal{A}')$, the existence of such a model can be reduced to concept satisfiability w.r.t. $\mathcal{T}$. For an individual x that occurs in $\mathcal{A}'$, we define C_x by

$$\begin{aligned} C_x = & \textstyle\bigsqcap\{A \mid A \in \mathsf{NC}, x : A \in \mathcal{A}'\} \sqcap \\ & \textstyle\bigsqcap\{\neg A \mid A \in \mathsf{NC}, x : \neg A \in \mathcal{A}'\} \sqcap \\ & \textstyle\bigsqcap\{(\geq(n-m)\ \omega\ D) \mid x : (\geq n\ \omega\ D) \in \mathcal{A}', m = \sharp\omega^{\mathcal{A}'}(x,D)\} \sqcap \\ & \textstyle\bigsqcap\{(\leq(n-m)\ \omega\ D) \mid x : (\leq n\ \omega\ D) \in \mathcal{A}', m = \sharp\omega^{\mathcal{A}'}(x,D)\}. \end{aligned}$$

Lemma 4.41
Let $\mathcal{K}' = (\mathcal{T}, \mathcal{A}')$ be a pre-completed $\mathcal{ALCQIb}$-knowledge base. $\mathcal{K}'$ has a model that satisfies,

$$\{R \in \overline{\mathsf{NR}}_\mathcal{K} \mid (x,y) : R \in \mathcal{A}'\} = \{R \in \overline{\mathsf{NR}}_\mathcal{K} \mid (x,y) \in R^\mathcal{I}\},$$

for every $x, y \in \mathsf{NI}$ that occur in $\mathcal{A}'$ iff, for every x that occurs in $\mathcal{A}'$, the concept C_x is satisfiable w.r.t. $\mathcal{T}$.

The proof of this lemma is straightforward and omitted here.

Putting together Lemma 4.40 and Lemma 4.41, we have the steps of a reduction from knowledge-base satisfiability to concept satisfiability w.r.t. general TBoxes—a problem that we know how to solve in ExpTime (Theorem 4.38). But how do we obtain an ExpTime-algorithm from these lemmas? Lemma 4.40 involves a non-deterministic step since it talks about the *existence* of a completion. Since it is generally assumed that ExpTime $\neq$ NExpTime we have to show how to search for such a completion in exponential time.

Theorem 4.42
Knowledge base satisfiability and instance checking for $\mathcal{ALCQIb}$ are ExpTime*-complete, even if numbers in the input are represented using binary coding.*

Proof. ExpTime-hardness is immediate from Theorem 4.38. It remains to show that these problems can be decided in exponential time.

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ALCQIb}$-knowledge base, $\overline{\mathsf{NR}}_\mathcal{K}$ the set of roles that occur in $\mathcal{K}$ together with their inverse, and $clos(\mathcal{K})$ defined as above. Let $m = |\mathcal{K}|$. Only ABoxes $\mathcal{A}'$ with no more individuals than $\mathcal{A}$ are candidates for pre-completions because the mapping f must be surjective. The number of individuals in $\mathcal{A}$ is bounded by m.

For an ABox $\mathcal{A}'$ with $i \leq m$ individuals, concept assertions ranging over $clos(\mathcal{K})$, and role assertions ranging over $\overline{\mathsf{NR}}_{\mathcal{K}}$, there are at most $2^{i \times m} \times 2^{i^2 \times 2m} = \mathcal{O}(2^{m^5})$ different possibilities, and each such ABox contains at most $i \times m + i^2 \times 2m + i^2 = \mathcal{O}(m^3)$ assertions. For an ABox $\mathcal{A}'$ with i individuals there are at most $i^m = \mathcal{O}(2^{m^2})$ different possibilities of mapping the individuals from $\mathcal{A}$ (of which there are at most m many) into the i individuals of $\mathcal{A}'$. Given a fixed $\mathcal{A}'$ and a fixed mapping f, testing whether the requirement of Definition 4.39 are satisfied can be done in polynomial time in m and hence certainly in time $\mathcal{O}(2^m)$.

Summing up, it is possible to enumerate all potential pre-completions of $\mathcal{K}$, generate all possible mappings f, and test whether all requirements from Definition 4.39 are satisfied in time bounded in

$$\sum_{i=1}^{m} \Big(\mathcal{O}(2^{m^5}) \times \mathcal{O}(2^{m^2}) \times \mathcal{O}(2^m) \Big) = \mathcal{O}(2^{m^6}).$$

Due to Lemma 4.40 and Lemma 4.41, $\mathcal{A}$ is satisfiable iff this enumeration yields a pre-completion $\mathcal{A}'$ such that C_x is satisfiable w.r.t. $\mathcal{T}$ for every x that occurs in $\mathcal{A}'$. Since all candidate pre-completions $\mathcal{A}'$ from the enumeration contain at most $\mathcal{O}(m^3)$ assertions, this can be checked for in time exponential in m for every candidate pre-completion $\mathcal{A}'$. This yields an overall decision procedure that runs in time exponentially bounded in m.

Instance checking is at least as hard as concept satisfiability w.r.t. general TBoxes and not harder than knowledge base satisfiability, hence ExpTime-completeness of instance checking for $\mathcal{ALCQIb}$ is immediate from what we have just proved. ∎

Chapter 5

Cardinality Restrictions and Nominals

In this chapter, we study the complexity of the combination of the DLs $\mathcal{ALCQ}$ and $\mathcal{ALCQI}$ with a terminological formalism based on cardinality restrictions on concepts. Cardinality restrictions were first introduced by Baader, Buchheit, and Hollunder (1996) as a terminological formalism that is particularly useful for configuration applications. They allow to restrict the number of instances of a (possibly complex) concept C *globally* using expressions of the form $(\geq n\ C)$ or $(\leq n\ C)$. In a configuration application, the cardinality restriction $(\geq 100$ `Parts`$)$ can be used to limit the overall number of `Parts` by 100, the cardinality restrictions $(\geq 1$ `PowerSource`$)$ and $(\leq 1$ `PowerSource`$)$ together state that there must be exactly one `PowerSource`, etc.

As it turns out, cardinality restrictions are closely connected to nominals, i.e., atomic concepts referring to single individuals of the domain. Nominals are studied both in the context of DLs (Borgida & Patel-Schneider, 1994; De Giacomo & Lenzerini, 1996) and of modal logics (Gargov & Goranko, 1993; Blackburn & Seligman, 1995; Areces et al., 2000).

After introducing cardinality restrictions and nominals, we show that, in the presence of nominals, reasoning w.r.t. cardinality restrictions can be polynomially reduced to reasoning w.r.t. TBoxes. In general the latter is a simpler problem. This allows to determine the complexity of $\mathcal{ALCQ}$ with cardinality restrictions as ExpTime-complete as a corollary of a result in (De Giacomo, 1995), if unary coding of numbers in the input is assumed. For binary coding, we will show that the problem becomes NExpTime-hard. Of all logics studied in this thesis, $\mathcal{ALCQ}$ with number restrictions is the only logic for which it has been shown that the coding of numbers effects the complexity of the inference problems.

For $\mathcal{ALCQI}$ with cardinality restrictions, we show that reasoning becomes NExpTime-hard and is NExpTime-complete if unary coding of numbers is assumed. By the connection to reasoning with nominals, this implies that reasoning w.r.t. general TBoxes for $\mathcal{ALCQI}$ with nominals has the same complexity and we sharpen this result to pure concept satisfiability.

Finally, we generalise the results for reasoning with $\mathcal{ALCQI}$ to $\mathcal{ALCQI}b$, with little effort, and show that, for $\mathcal{ALCQIB}$, i.e., $\mathcal{ALCQI}b$ without the restriction to safe role expressions, concept satisfiability is NExpTime-complete (this is also a simple corollary of the NExpTime-completeness of Boolean Modal Logic (Lutz & Sattler, 2000)).

5.1 Syntax and Semantics

Cardinality restrictions can be defined independently of a particular DL as long as it has standard extensional semantics. In this thesis, we will mainly study cardinality restrictions in combination with the DLs $\mathcal{ALCQ}$ and $\mathcal{ALCQI}$. To make our considerations here easier, we assume that concepts are built using only the restricted set of concept constructors $\neg, \sqcap, (\geq n\ R\ C)$. Using de Morgan's laws and the duality of the at-least and at-most-restriction (see below Definition 4.2) the other constructors can be defined as abbreviations.

Definition 5.1 (Cardinality Restrictions)
A cardinality restriction *is an expression of the form* $(\leq n\ C)$ *or* $(\geq n\ C)$ *where* $n \in \mathbb{N}$ *and* C *is a concept.*

A CBox *is a finite set of cardinality restrictions.*

An interpretation $\mathcal{I}$ satisfies *a cardinality restriction* $(\leq n\ C)$ *iff* $\sharp(C^{\mathcal{I}}) \leq n$*, and it* satisfies $(\geq n\ C)$ *iff* $\sharp(C^{\mathcal{I}}) \geq n$*. It satisfies a CBox* $\mathcal{C}$ *iff it satisfies all cardinality restrictions in* $\mathcal{C}$*; in this case,* $\mathcal{I}$ *is called a* model *of* $\mathcal{C}$ *and we will denote this fact by* $\mathcal{I} \models \mathcal{C}$*. A CBox that has a model is called* satisfiable*.*

Since $\mathcal{I} \models (\leq 0\ \neg C)$ *iff* C *is satisfied by all elements of* $\mathcal{I}$*, we will use* $(\forall\ C)$ *as an abbreviation for the cardinality restriction* $(\leq 0\ \neg C)$*.* ◇

It is obvious that, for DLs that are closed under Boolean combinations of concepts, reasoning with cardinality restrictions is at least as hard as reasoning with TBoxes, as $\mathcal{I} \models C \sqsubseteq D$ iff $\mathcal{I} \models (\leq 0\ (C \sqcap \neg D))$. As we will see, CBoxes can also be used to express ABoxes and even the stronger formalism of *nominals*. In this thesis, we have already encountered nominals in a restricted form, namely, as individuals that may occur in ABox assertions. DLs that allow for nominals allow those individuals to appear in arbitrary concept expressions, which, e.g., makes it possible to define the concept of parents of $\mathtt{BOB}$ by $\exists\mathtt{has\_child}.\mathtt{BOB}$ or the concept of $\mathtt{BOB}$'s siblings by $\neg\mathtt{BOB} \sqcap \exists\mathtt{has\_child}^{-1}.\exists\mathtt{has\_child}.\mathtt{BOB}$.

Definition 5.2 (Nominals)
Let NI *be a set of* individual names *or* nominals*. For an arbitrary DL* $\mathcal{L}$*, its extension with nominals (usually denoted by* $\mathcal{LO}$*) is obtained by, additionally, defining that every* $i \in \mathsf{NI}$ *is a concept.*

For the semantics, we require an interpretation $\mathcal{I}$ *to map every* $i \in \mathsf{NI}$ *to a singleton set* $i^{\mathcal{I}}$ *and extend the semantics of* $\mathcal{L}$ *to* $\mathcal{LO}$ *canonically.* ◇

Nominals in a DL makes ABoxes superfluous, since these can be captured using nominals. Indeed, in the presence of nominals, it suffices to consider satisfiability of TBoxes as the "strongest" inference required.

Lemma 5.3
For an arbitrary DL $\mathcal{L}$, KB-satisfiability can be polynomially reduced to satisfiability of $\mathcal{LO}$-TBoxes.

Proof. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{L}$-knowledge base, where the individuals in the ABox coincide with the individuals of $\mathcal{LO}$. The ABox $\mathcal{A}$ is transformed into a TBox as follows. We define

$$\mathcal{T}_{\mathcal{A}} = \{i \sqsubseteq C \mid i : C \in \mathcal{A}\} \cup \{i \sqsubseteq \exists R.j \mid (i,j) : R \in \mathcal{A}\} \cup \{i \sqsubseteq \neg j \mid i \not\doteq j \in \mathcal{A}\}.$$

Claim 5.4 $\mathcal{K}$ is satisfiable iff $\mathcal{T} \cup \mathcal{T}_{\mathcal{A}}$ is satisfiable.

If $\mathcal{K}$ is satisfiable with $\mathcal{I} \models \mathcal{K}$, it is easy to verify that $\mathcal{I}'$, which is obtained from $\mathcal{I}$ by setting $i^{\mathcal{I}'} = \{i^{\mathcal{I}}\}$ and preserving the interpretation of the concept and role names, is a model for $\mathcal{T} \cup \mathcal{T}_{\mathcal{A}}$.

Conversely, any model $\mathcal{I}$ of $\mathcal{T} \cup \mathcal{T}_{\mathcal{A}}$ can be turned into a model $\mathcal{I}'$ of $\mathcal{K}$ by setting, for every individual $i \in \mathsf{NI}$, $i^{\mathcal{I}'} = x$ for the unique $x \in i^{\mathcal{I}}$ and preserving the interpretation of concept and role names. ∎

Now that we have seen how to get rid of ABoxes in the presence of nominals, we show how cardinality restrictions and nominals can emulate each other.

Lemma 5.5
For an arbitrary DL $\mathcal{L}$, satisfiability of $\mathcal{L}$-CBoxes and $\mathcal{LO}$-TBoxes are mutually reducible. The reduction from $\mathcal{LO}$ to $\mathcal{L}$ is polynomial. The reduction from $\mathcal{L}$ to $\mathcal{LO}$ is polynomial if unary coding of numbers in the cardinality restrictions is assumed.

Proof. It is obvious that the cardinality restrictions $(\leq 1\ C)$ and $(\geq 1\ C)$ enforce the interpretation of a concept name C to be a singleton, which can now serve as a substitute for a nominal. Also, an interpretation satisfies a general axiom $C \sqsubseteq D$ iff it satisfies $(\leq 0\ (C \sqcap \neg D))$. In this manner, every nominal can be replaced by a concept and every general axiom by a cardinality restriction, which yields the reduction from reasoning with nominals and TBoxes to reasoning with cardinality restrictions. For the converse direction, the reduction works as follows.

Let $\mathcal{C} = \{(\bowtie_1\ n_1\ C_1), \dots (\bowtie_k\ n_k\ C_k)\}$ be an $\mathcal{L}$-CBox. W.l.o.g., we assume that $\mathcal{C}$ contains no cardinality restriction of the form $(\geq 0\ C)$ because these are trivially satisfied by any interpretation. The translation of $\mathcal{C}$, denoted by $\Phi(\mathcal{C})$, is the $\mathcal{LO}$-TBox defined by:

$$\Phi(\mathcal{C}) = \bigcup \{\Phi(\bowtie_i\ n_i\ C_i) \mid 1 \leq i \leq k\},$$

where $\Phi(\bowtie_i\ n_i\ C_i)$ is defined depending on whether $\bowtie_i = \leq$ or $\bowtie_i = \geq$.

$$\Phi(\bowtie_i\ n_i\ C_i) = \begin{cases} \{C_i \sqsubseteq o_i^1 \sqcup \cdots \sqcup o_i^{n_i}\} & \text{if } \bowtie_i = \leq \\ \{o_i^j \sqsubseteq C_i \mid 1 \leq j \leq n_i\} \cup \{o_i^j \sqsubseteq \neg o_i^\ell \mid 1 \leq j < \ell \leq n_i\} & \text{if } \bowtie_i = \geq \end{cases},$$

where $o_i^1, \dots, o_i^{n_i}$ are fresh and distinct nominals and we use the convention that the empty disjunction is interpreted as $\perp$ to deal with the case $n_i = 0$.

Assuming unary coding of numbers, the translation of a CBox $\mathcal{C}$ is obviously computable in polynomial time.

CLAIM 5.6 $\mathcal{C}$ is satisfiable iff $\Phi(\mathcal{C})$ is satisfiable.

If $\mathcal{C}$ is satisfiable then there is a model $\mathcal{I}$ of $\mathcal{C}$ and $\mathcal{I} \models (\bowtie_i n_i\ C_i)$ for each $1 \leq i \leq k$. We show how to construct a model $\mathcal{I}'$ of $\Phi(\mathcal{C})$ from $\mathcal{I}$. $\mathcal{I}'$ will be identical to $\mathcal{I}$ in every respect except for the interpretation of the nominals o_i^j (which do not appear in $\mathcal{C}$).

If $\bowtie_i = \leq$, then $\mathcal{I} \models \mathcal{C}$ implies $\sharp C_i^{\mathcal{I}} \leq n_i$. If $n_i = 0$, then we have not introduced new nominals, and $\Phi(\mathcal{C})$ contains $C_i \sqsubseteq \perp$. Otherwise, we define $(o_i^j)^{\mathcal{I}'}$ such that $C_i^{\mathcal{I}} \subseteq \{(o_i^j)^{\mathcal{I}'} \mid 1 \leq j \leq n_i\}$. This implies $C_i^{\mathcal{I}'} \subseteq (o_i^1)^{\mathcal{I}'} \cup \dots \cup (o_i^{n_i})^{\mathcal{I}'}$ and hence, in either case, $\mathcal{I}' \models \Phi(\leq n_i\ C_i)$.

If $\bowtie_i = \geq$, then $n_i > 0$ must hold, and $\mathcal{I} \models \mathcal{C}$ implies $\sharp C_i^{\mathcal{I}} \geq n_i$. Let $x_1, \dots x_{n_i}$ be n_i distinct elements from $\Delta^{\mathcal{I}}$ with $\{x_1, \dots, x_{n_i}\} \subseteq C_i^{\mathcal{I}}$. We set $(o_i^j)^{\mathcal{I}'} = \{x_j\}$. Since we have chosen distinct individuals to interpret different nominals, we have $\mathcal{I}' \models o_i^j \sqsubseteq \neg o_i^\ell$ for every $1 \leq i < \ell \leq n_i$. Moreover, $x_j \in C_i^{\mathcal{I}}$ implies $\mathcal{I}' \models o_i^j \sqsubseteq C_i$ and hence $\mathcal{I}' \models \Phi(\geq n_i\ C_i)$.

We have chosen distinct nominals for every cardinality restrictions, hence the previous construction is well-defined and, since $\mathcal{I}'$ satisfies $\Phi(\bowtie_i n_i\ C_i)$ for every i, $\mathcal{I}' \models \Phi(\mathcal{C})$.

For the converse direction, let $\mathcal{I}$ be a model of $\Phi(\mathcal{C})$. The fact that $\mathcal{I} \models \mathcal{C}$ (and hence the satisfiability of $\mathcal{C}$) can be shown as follows: let $(\bowtie_i n_i\ C_i)$ be an arbitrary cardinality restriction in $\mathcal{C}$. If $\bowtie_i = \leq$ and $n_i = 0$, then we have $\Phi(\leq 0\ C_i) = \{C_i \sqsubseteq \perp\}$ and, since $\mathcal{I} \models \Phi(\mathcal{C})$, we have $C_i^{\mathcal{I}} = \emptyset$ and $\mathcal{I} \models (\leq 0\ C_i)$. If $\bowtie_i = \leq$ and $n_i > 0$, we have $\{C_i \sqsubseteq o_i^1 \sqcup \dots \sqcup o_i^{n_i}\} \subseteq \Phi(\mathcal{C})$. From $\mathcal{I} \models \Phi(\mathcal{C})$ follows $\sharp C_i^{\mathcal{I}} \leq \sharp (o_i^1 \sqcup \dots \sqcup o_i^{n_i})^{\mathcal{I}} \leq n_i$. If $\bowtie_i = \geq$, then we have $\{o_i^j \sqsubseteq C_i \mid 1 \leq j \leq n_i\} \cup \{o_i^j \sqsubseteq \neg o_i^\ell \mid 1 \leq j < \ell \leq n_i\} \subseteq \Phi(\mathcal{C})$. From the first set of axioms we get $\{(o_i^j)^{\mathcal{I}} \mid 1 \leq j \leq n_i\} \subseteq C_i^{\mathcal{I}}$. From the second set of axioms we get that, for every $1 \leq j < \ell \leq n_i$, $(o_i^j)^{\mathcal{I}} \neq (o_i^\ell)^{\mathcal{I}}$. This implies that $n_i = \sharp \bigcup \{(o_i^j)^{\mathcal{I}} \mid 1 \leq j \leq n_i\} \leq \sharp C_i^{\mathcal{I}}$. ∎

5.2 The Complexity of Cardinality Restrictions and Nominals

We will now study the complexity of reasoning with cardinality restrictions both for $\mathcal{ALCQ}$ and $\mathcal{ALCQI}$. Baader, Buchheit and Hollunder (1996) give an algorithm that decides satisfiability of CBoxes for $\mathcal{ALCQ}$ but they do not give complexity results. Yet, it is easy to see that their algorithm runs in non-deterministic exponential time, which gives us a first upper bound for the complexity of the problem. For the lower bound, it is obvious that the problem is at least ExpTime-hard, due to Lemma 5.5 and Theorem 3.18 Lemma 5.5 also yields ExpTime as an upper bound for the complexity of this problem using the following result established by De Giacomo (1995).

Fact 5.7 (De Giacomo, 1995, Section 7.3)
Satisfiability and logical implication for $\mathcal{CNO}$ knowledge bases (TBox and ABox) are ExpTime*-complete.*

The DL $\mathcal{CNO}$ studied by the author is a strict extension of $\mathcal{ALCQO}$. Unary coding of numbers is assumed throughout his thesis. Although the author imposes a unique name assumption, it is not inherent to the utilized reduction and must be explicitly enforced. It is thus possible to eliminate the formulas that require a unique interpretation of individuals from the reduction. Hence, according to Lemma 5.5, reasoning with cardinality restrictions for $\mathcal{ALCQ}$ can be reduced to $\mathcal{CNO}$, which yields:

Corollary 5.8
Consistency of $\mathcal{ALCQO}$-CBoxes is ExpTime*-complete if unary coding of number is assumed.*

For binary coding of numbers, the reduction used in the proof of Lemma 5.5 is no longer polynomial and, indeed, reasoning for $\mathcal{ALCQ}$-CBoxes becomes at least NExpTime-hard if binary coding is assumed (Corollary 5.20).

5.2.1 Cardinality Restrictions and $\mathcal{ALCQI}$

The algorithm developed by Baader et. al. (1996) for $\mathcal{ALCQ}$ with number restrictions cannot easily be extended to $\mathcal{ALCQI}$ with cardinality restrictions. One indication for this is that the algorithm from (Baader et al., 1996) is a tableau algorithm that always constructs a finite model for a satisfiable CBox; yet, $\mathcal{ALCQI}$ with cardinality restriction no longer has the finite model property. The CBox

$$(\geq 1\ \neg A),\ (\forall\ (\exists R.\top \sqcap (\leq 1\ R^{-1}) \sqcap \forall R.A))$$

is satisfiable, but does not have a finite model. The first cardinality restriction requires the existence of an instance x of $\neg A$ in the model. The second cardinality restriction requires every element of the model to have an R-successor, so from x there starts an infinite path of R-successors. This path must either run into a cycle or there must be infinitely many elements in the model. It cannot cycle back to x because this would conflict with the requirement that every element satisfies $\forall R.A$. It cannot cycle back to another element of the path because in that case, this element would have two incoming R-edges, which conflicts with $(\leq 1\ R^{-1})$.

There exists no dedicated decision procedure for $\mathcal{ALCQI}$ with number restrictions, but it is easy to see that the problem can be solved by a reduction to C^2, the two-variable fragment of FOL extended with counting quantifiers. Let L^2 denote the fragment of FOL that only has the variable symbols x and y. Then C^2 is the extension of L^2 that admits all counting quantifiers $\exists^{\geq m}$ and $\exists^{\leq m}$ for $m \geq 1$, rather than only $\exists$. Grädel, Otto, and Rosen (1997) show that C^2 is decidable. Based on their decision procedure (Pacholski et al., 1997) determine the complexity of C^2:

Figure 5.1 The translation from $\mathcal{ALCQI}$ into C^2

$$
\begin{array}{lll}
\Psi_x(A) & := Ax & \text{for } A \in N_C \\
\Psi_x(\neg C) & := \neg\Psi_x(C) & \\
\Psi_x(C_1 \sqcap C_2) & := \Psi_x(C_1) \wedge \Psi_x(C_2) & \\
\Psi_x({\geq}n\ R\ C) & := \exists^{\geq n} y.(Rxy \wedge \Psi_y(C)) & \\
\Psi_x({\geq}n\ R^{-1}\ C) & := \exists^{\geq n} y.(Ryx \wedge \Psi_y(C)) & \\
\Psi_y(A) & := Ay & \text{for } A \in N_C \\
\Psi_y(\neg C) & := \neg\Psi_y(C) & \\
\Psi_y(C_1 \sqcap C_2) & := \Psi_y(C_1) \wedge \Psi_y(C_2) & \\
\Psi_y({\geq}n\ R\ C) & := \exists^{\geq n} y.(Ryx \wedge \Psi_x(C)) & \\
\Psi_y({\geq}n\ R^{-1}\ C) & := \exists^{\geq n} y.(Rxy \wedge \Psi_x(C)) & \\
\Psi(\bowtie n\ C) & := \exists^{\bowtie n} x.\Psi_x(C) \quad \text{for } \bowtie \in \{\geq, \leq\} & \\
\Psi(\mathcal{C}) & := \bigwedge\{\Psi(\bowtie\ n\ C) \mid (\bowtie\ n\ C) \in \mathcal{C}\} &
\end{array}
$$

Fact 5.9 (Pacholski et al., 1997)
*Satisfiability of C^2 is decidable in 2-*NExpTime *for binary coding of number and is* NExpTime*-complete for unary coding of numbers.*

Figure 5.1 shows how the standard translation of $\mathcal{ALCQI}$ into C^2 due to Borgida (1996) can be extended to cardinality restrictions. It is obviously a satisfiability preserving translation, which yields:

Lemma 5.10
An-$\mathcal{ALCQI}$ CBox is satisfiable iff $\Psi(\mathcal{C})$ is satisfiable.

The translation from Figure 5.1 is obviously polynomial, and so we obtain, from Lemma 5.10 and Fact 5.9:

Lemma 5.11
Satisfiability of $\mathcal{ALCQI}$-CBoxes can be decided in NExpTime*, if unary coding of numbers in the input is assumed.*

We will see that, from the viewpoint of worst-case complexity, this is an optimal result, as the problem is also NExpTime hard. To prove this, we use a bounded version of the domino problem. Domino problems (Wang, 1963; Berger, 1966) have successfully been employed to establish undecidability and complexity results for various description and modal logics (Spaan, 1993a; Baader & Sattler, 1999).

Domino Systems

Definition 5.12
For $n \in \mathbb{N}$, let $\mathbb{Z}_n$ denote the set $\{0, \dots, n-1\}$ and $\oplus_n$ denote the addition modulo n. A domino system *is a triple $\mathcal{D} = (D, H, V)$, where D is a finite set (of tiles) and $H, V \subseteq D \times D$*

are relations expressing horizontal and vertical compatibility constraints between the tiles. For $s, t \in \mathbb{N}$, let $U(s,t)$ be the torus $\mathbb{Z}_s \times \mathbb{Z}_t$, and let $w = w_0 \dots w_{n-1}$ be a word over D of length n (with $n \leq s$). We say that $\mathcal{D}$ tiles *$U(s,t)$* with initial condition *w iff there exists a mapping $\tau : U(s,t) \to D$ such that, for all $(x,y) \in U(s,t)$,*

- *if $\tau(x,y) = d$ and $\tau(x \oplus_s 1, y) = d'$, then $(d,d') \in H$ (horizontal constraint);*
- *if $\tau(x,y) = d$ and $\tau(x, y \oplus_t 1) = d'$, then $(d,d') \in V$ (vertical constraint);*
- *$\tau(i,0) = w_i$ for $0 \leq i < n$ (initial condition).* ◇

Bounded domino systems are capable of expressing the computational behaviour of restricted, so-called *simple*, Turing Machines (TM). This 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, as long as $S(n), T(n) \geq 2n$ (Börger, Grädel, & Gurevich, 1997).

Theorem 5.13 (Börger et al., 1997, Theorem 6.1.2)
Let M be a simple TM with input alphabet Σ. Then there exists a domino system $\mathcal{D} = (D, H, V)$ and a linear time reduction which takes any input $x \in \Sigma^*$ to a word $w \in D^*$ with $|x| = |w|$ such that

- *If M accepts x in time t_0 with space s_0, then $\mathcal{D}$ tiles $U(s,t)$ with initial condition w 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 w for any $s, t \geq 2$.*

Corollary 5.14
There is a domino system $\mathcal{D}$ such that the following is a NExpTime*-hard problem:*

> *Given an initial condition $w = w_0 \dots w_{n-1}$ of length n. Does $\mathcal{D}$ tile the torus $U(2^{n+1}, 2^{n+1})$ with initial condition w?*

Proof. Let M be a (w.l.o.g. simple) non-deterministic TM with time- (and hence space-) bound 2^n deciding an arbitrary NExpTime-complete language $\mathcal{L}(M)$ over the alphabet Σ. Let $\mathcal{D}$ be the according domino system and *trans* the reduction from Theorem 5.13.

The function *trans* is a linear reduction from $\mathcal{L}(M)$ to the problem above: For $v \in \Sigma^*$ with $|v| = n$, it holds that $v \in \mathcal{L}(M)$ iff M accepts v in time and space $2^{|v|}$ iff $\mathcal{D}$ tiles $U(2^{n+1}, 2^{n+1})$ with initial condition *trans*(v). ∎

Defining a Torus of Exponential Size

Similar to proving undecidability by reduction of unbounded domino problems, where defining infinite grids is the key problem, defining a torus of exponential size is the key to obtain a NExpTime-completeness proof by reduction of bounded domino problems.

To be able to apply Corollary 5.14 to CBox satisfiability for $\mathcal{ALCQI}$, we must characterize the torus $\mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n}$ with a CBox of polynomial size. To characterize this torus, we use $2n$ concepts $X_0, \ldots, X_{n-1}$ and $Y_0, \ldots, Y_{n-1}$, where X_i (Y_i) codes the ith bit of the binary representation of the X-coordinate (Y-coordinate) of an element a.

For an interpretation $\mathcal{I}$ and an element $a \in \Delta^{\mathcal{I}}$, we define $\mathit{pos}(a)$ by

$$\mathit{pos}(a) := (\mathrm{xpos}(a), \mathrm{ypos}(a)) := \left(\sum_{i=0}^{n-1} x_i \cdot 2^i, \quad \sum_{i=0}^{n-1} y_i \cdot 2^i \right), \text{ where}$$

$$x_i = \begin{cases} 0, & \text{if } a \notin X_i^{\mathcal{I}} \\ 1, & \text{otherwise} \end{cases} \qquad y_i = \begin{cases} 0, & \text{if } a \notin Y_i^{\mathcal{I}} \\ 1, & \text{otherwise .} \end{cases}$$

We use a well-known characterization of binary addition (see, e.g., Börger et al., 1997) to interrelate the positions of the elements in the torus:

Lemma 5.15
Let x, x' be natural numbers with binary representations

$$x = \sum_{i=0}^{n-1} x_i \cdot 2^i \quad \text{and} \quad x' = \sum_{i=0}^{n-1} x'_i \cdot 2^i.$$

Then

$$x' \equiv x + 1 \pmod{2^n} \quad \textit{iff} \quad \bigwedge_{k=0}^{n-1} (\bigwedge_{j=0}^{k-1} x_j = 1) \rightarrow (x_k = 1 \leftrightarrow x'_k = 0)$$
$$\wedge \bigwedge_{k=0}^{n-1} (\bigvee_{j=0}^{k-1} x_j = 0) \rightarrow (x_k = x'_k) \ ,$$

where the empty conjunction and disjunction are interpreted as true and false, respectively.

The CBox $\mathcal{C}_n$ is defined in Figure 5.2. The concept $C_{(0,0)}$ is satisfied by all elements a of the domain for which $\mathit{pos}(a) = (0, 0)$ holds. $C_{(2^n-1,2^n-1)}$ is a similar concept, whose instances a satisfy $\mathit{pos}(a) = (2^n - 1, 2^n - 1)$.

The concept D_{north} is similar to D_{east} where the role *north* has been substituted for *east* and variables X_i and Y_i have been swapped. The concept D_{east} (D_{north}) enforces that, along the role *east* (*north*), the value of xpos (ypos) increases by one while the value of ypos (xpos) is unchanged. They are analogous to the formula in Lemma 5.15.

The following lemma is a consequence of the definition of *pos* and Lemma 5.15.

Figure 5.2 A CBox defining a torus of exponential size

$$
\begin{array}{rcl}
\mathcal{C}_n & = & \{\ (\forall\ \exists east.\top), \qquad (\forall\ \exists north.\top), \\
& & \quad (\forall\ (=1\ east^{-1}\ \top)), \quad (\forall\ (=1\ north^{-1}\ \top)), \\
& & \quad (\geqslant 1\ C_{(0,0)}), \qquad (\geqslant 1\ C_{(2^n-1,2^n-1)}), \\
& & \quad (\leqslant 1\ C_{(2^n-1,2^n-1)}), \quad (\forall\ D_{east} \sqcap D_{north})\ \} \\
C_{(0,0)} & = & \prod_{k=0}^{n-1} \neg X_k \sqcap \prod_{k=0}^{n-1} \neg Y_k \\
C_{(2^n-1,2^n-1)} & = & \prod_{k=0}^{n-1} X_k \sqcap \prod_{k=0}^{n-1} Y_k \\
D_{east} & = & \prod_{k=0}^{n-1} (\prod_{j=0}^{k-1} X_j) \rightarrow ((X_k \rightarrow \forall east.\neg X_k) \sqcap (\neg X_k \rightarrow \forall east.X_k)) \sqcap \\
& & \prod_{k=0}^{n-1} (\bigsqcup_{j=0}^{k-1} \neg X_j) \rightarrow ((X_k \rightarrow \forall east.X_k) \sqcap (\neg X_k \rightarrow \forall east.\neg X_k)) \sqcap \\
& & \prod_{k=0}^{n-1} ((Y_k \rightarrow \forall east.Y_k) \sqcap (\neg Y_k \rightarrow \forall east.\neg Y_k)) \\
D_{north} & = & \prod_{k=0}^{n-1} (\prod_{j=0}^{k-1} Y_j) \rightarrow ((Y_k \rightarrow \forall north.\neg Y_k) \sqcap (\neg Y_k \rightarrow \forall north.Y_k)) \sqcap \\
& & \prod_{k=0}^{n-1} (\bigsqcup_{j=0}^{k-1} \neg Y_j) \rightarrow ((Y_k \rightarrow \forall north.Y_k) \sqcap (\neg Y_k \rightarrow \forall north.\neg Y_k)) \sqcap \\
& & \prod_{k=0}^{n-1} ((X_k \rightarrow \forall north.X_k) \sqcap (\neg X_k \rightarrow \forall north.\neg X_k))
\end{array}
$$

Lemma 5.16
Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation, D_{east}, D_{north} defined as in Figure 5.2, and $a, b \in \Delta^{\mathcal{I}}$.

$$
\begin{array}{rll}
(a,b) \in east^{\mathcal{I}} \text{ and } a \in D^{\mathcal{I}}_{east} \text{ implies:} & xpos(b) & \equiv xpos(a) + 1 \pmod{2^n} \\
& ypos(b) & = ypos(a) \\
(a,b) \in north^{\mathcal{I}} \text{ and } a \in D^{\mathcal{I}}_{north} \text{ implies:} & xpos(b) & = xpos(a) \\
& ypos(b) & \equiv ypos(a) + 1 \pmod{2^n}
\end{array}
$$

The CBox $\mathcal{C}_n$ defines a torus of exponential size in the following sense:

Lemma 5.17
Let $\mathcal{C}_n$ be the CBox as defined in Figure 5.2. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of $\mathcal{C}_n$. Then

$$(\Delta^{\mathcal{I}}, east^{\mathcal{I}}, north^{\mathcal{I}}) \cong (U(2^n, 2^n), S_1, S_2)\ ,$$

where $U(2^n, 2^n)$ is the torus $\mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n}$ and S_1, S_2 are the horizontal and vertical successor relations on this torus.

Proof. We show that the function *pos* is an isomorphism from $\Delta^\mathcal{I}$ to $U(2^n, 2^n)$. Injectivity of *pos* is shown by induction on the "Manhattan distance" $d(a)$ of the *pos*-value of an element a to the *pos*-value of the upper right corner.

For an element $a \in \Delta^\mathcal{I}$ we define $d(a)$ by

$$d(a) = (2^n - 1 - \mathrm{xpos}(a)) + (2^n - 1 - \mathrm{ypos}(a)).$$

Note that $pos(a) = pos(b)$ implies $d(a) = d(b)$. Since $\mathcal{I} \models (\leqslant 1\ C_{(2^n-1,2^n-1)})$, there is at most one element $a \in \Delta^\mathcal{I}$ such that $d(a) = 0$. Hence, there is at most one element a such that $pos(a) = (2^n - 1, 2^n - 1)$. Now assume there are elements $a, b \in \Delta^\mathcal{I}$ such that $pos(a) = pos(b)$ and $d(a) = d(b) > 0$. Then $\mathrm{xpos}(a) < 2^n - 1$ or $\mathrm{ypos}(a) < 2^n - 1$. W.l.o.g., we assume $\mathrm{xpos}(a) < 2^n - 1$. From $\mathcal{I} \models \mathcal{C}_n$, it follows that $a, b \in (\exists east.\top)^\mathcal{I}$. Let a_1, b_1 be elements such that $(a, a_1) \in east^\mathcal{I}$ and $(b, b_1) \in east^\mathcal{I}$. From Lemma 5.16, it follows that

$$\begin{aligned} \mathrm{xpos}(a_1) &\equiv \mathrm{xpos}(b_1) \equiv \mathrm{xpos}(a) + 1 \pmod{2^n} \\ \mathrm{ypos}(a_1) &= \mathrm{ypos}(b_1) = \mathrm{ypos}(a). \end{aligned}$$

This implies $pos(a_1) = pos(b_1)$ and, since $\mathrm{xpos}(a) < 2^n - 1$, it holds that $\mathrm{xpos}(a_1) = \mathrm{xpos}(b_1) = \mathrm{xpos}(a) + 1 > \mathrm{xpos}(a)$. Hence, $d(a_1) = d(b_1) < d(a)$ and the induction hypothesis is applicable, which yields $a_1 = b_1$. This also implies $a = b$ because $a_1 \in (= 1\ east^{-1}.\top)^\mathcal{I}$ and $\{(a, a_1), (b, a_1)\} \subseteq east^\mathcal{I}$. Hence *pos* is injective.

To prove that *pos* is also *surjective* we use a similar technique. This time, we use an induction on the distance from the lower left corner. For each element $(x, y) \in U(2^n, 2^n)$, we define:

$$d'(x, y) = x + y.$$

We show by induction that, for each $(x, y) \in U(2^n, 2^n)$, there is an element $a \in \Delta^\mathcal{I}$ such that $pos(a) = (x, y)$. If $d'(x, y) = 0$, then $x = y = 0$. Since $\mathcal{I} \models (\geqslant 1\ C_{(0,0)})$, there is an element $a \in \Delta^\mathcal{I}$ such that $pos(a) = (0, 0)$. Now consider $(x, y) \in U(2^n, 2^n)$ with $d'(x, y) > 0$. Without loss of generality we assume $x > 0$ (if $x = 0$ then $y > 0$ must hold). Hence $(x - 1, y) \in U(2^n, 2^n)$ and $d'(x - 1, y) < d'(x, y)$. From the induction hypothesis, it follows that there is an element $a \in \Delta^\mathcal{I}$ such that $pos(a) = (x - 1, y)$. Then there must be an element a_1 such that $(a, a_1) \in east^\mathcal{I}$ and Lemma 5.16 implies that $pos(a_1) = (x, y)$. Hence *pos* is also surjective.

Finally, *pos* is indeed a homomorphism as an immediate consequence of Lemma 5.16.

■

It is interesting to note that we need inverse roles only to guarantee that the function *pos* is injective. The same can be achieved by adding the cardinality restriction $(\leqslant (2^n \cdot 2^n)\ \top)$ to $\mathcal{C}_n$, from which the injectivity of *pos* follows from its surjectivity and simple cardinality considerations. Of course, the size of this cardinality restriction is polynomial in n only if we assume binary coding of numbers. This has consequences for the complexity of $\mathcal{ALCQ}$-CBoxes if binary coding of numbers in the input is assumed (see Corollary 5.20).

Also note that we have made explicit use of the special expressive power of cardinality restrictions by stating that, in any model of $\mathcal{C}_n$, the extension of $C_{(2^n-1,2^n-1)}$ must have *at most* one element. This cannot be expressed with a $\mathcal{ALCQI}$-TBox consisting of terminological axioms.

Reducing Domino Problems to CBox Satisfiability

Once Lemma 5.17 has been proved, it is easy to reduce the bounded domino problem to CBox satisfiability. We use the standard reduction that has been applied in the DL context, e.g., by Baader and Sattler (1999).

Lemma 5.18
Let $\mathcal{D} = (D, V, H)$ be a domino system. Let $w = w_0 \dots w_{n-1} \in D^*$. There is a CBox $\mathcal{C}(n, \mathcal{D}, w)$ such that:

- *$\mathcal{C}(n, \mathcal{D}, w)$ is satisfiable iff $\mathcal{D}$ tiles $U(2^n, 2^n)$ with initial condition w, and*
- *$\mathcal{C}(n, \mathcal{D}, w)$ can be computed in time polynomial in n.*

Proof. We define $\mathcal{C}(n, \mathcal{D}, w) := \mathcal{C}_n \cup \mathcal{C}_{\mathcal{D}} \cup \mathcal{C}_w$, where $\mathcal{C}_n$ is defined in Figure 5.2, $\mathcal{C}_{\mathcal{D}}$ captures the vertical and horizontal compatibility constraints of the domino system $\mathcal{D}$, and $\mathcal{C}_w$ enforces the initial condition. We use an atomic concept C_d for each tile $d \in D$. $\mathcal{C}_{\mathcal{D}}$ consists of the following cardinality restrictions:

$$(\forall \bigsqcup_{d \in D} C_d), \quad (\forall \bigsqcap_{d \in D} \bigsqcap_{d' \in D \setminus \{d\}} \neg(C_d \sqcap C_{d'})),$$
$$(\forall \bigsqcap_{d \in D} (C_d \rightarrow (\forall \mathit{east}. \bigsqcup_{(d,d') \in H} C_{d'}))), \quad (\forall \bigsqcap_{d \in D} (C_d \rightarrow (\forall \mathit{north}. \bigsqcup_{(d,d') \in V} C_{d'}))).$$

$\mathcal{C}_w$ consists of the cardinality restrictions

$$(\forall \ (C_{(0,0)} \rightarrow C_{w_0})), \dots, (\forall \ (C_{(n-1,0)} \rightarrow C_{w_{n-1}}),$$

where, for each x, y, $C_{(x,y)}$ is a concept that is satisfied by an element a iff $\mathit{pos}(a) = (x, y)$, defined similarly to $C_{(0,0)}$ and $C_{(2^n-1,2^n-1)}$.

From the definition of $\mathcal{C}(n, \mathcal{D}, w)$ and Lemma 5.17, it follows that each model of $\mathcal{C}(n, \mathcal{D}, w)$ immediately induces a tiling of $U(2^n, 2^n)$ and vice versa. Also, for a fixed domino system $\mathcal{D}$, $\mathcal{C}(n, \mathcal{D}, w)$ is obviously polynomially computable. ∎

The main result of this section is now an immediate consequence of Lemma 5.11, Lemma 5.18, and Corollary 5.14:

Theorem 5.19
Satisfiability of $\mathcal{ALCQI}$-CBoxes is NExpTime*-hard. It is* NExpTime*-complete if unary coding of numbers is used in the input.*

Recalling the note below the proof of Lemma 5.17, we see that the same reduction also applies to $\mathcal{ALCQ}$ if we allow binary coding of numbers.

Corollary 5.20
Satisfiability of $\mathcal{ALCQ}$-CBoxes is NExpTime*-hard if binary coding is used to represent numbers in cardinality restrictions.*

It should be noted that it is open whether the problem can be decided in NExpTime, if binary coding of numbers is used. In fact, the reduction to C^2 only yields decidability in 2-NExpTime if binary coding is assumed.

We have already seen that, for unary coding of numbers, deciding satisfiability of $\mathcal{ALCQ}$-CBoxes can be done in ExpTime (Corollary 5.8). This shows that the coding of numbers indeed has an influence on the complexity of the reasoning problem. For the problem of concept satisfiability in $\mathcal{ALCQ}$ this is not the case; in Chapter 4 we have shown that the complexity of the problem does not rise when going from unary to binary coding.

For unary coding, we needed both inverse roles and cardinality restrictions for the reduction. This is consistent with the fact that satisfiability for $\mathcal{ALCQI}$ concepts with respect to TBoxes that consist of terminological axioms is still in ExpTime. This can be shown by a reduction to the ExpTime-complete logics $\mathcal{CIN}$ (De Giacomo, 1995) or CPDL (Pratt, 1979). This shows that cardinality restrictions on concepts are an additional source of complexity.

Using Lemma 5.5 it is now also possible to determine the complexity of reasoning with $\mathcal{ALCQIO}$ TBoxes:

Corollary 5.21
Satisfiability of $\mathcal{ALCQIO}$-TBoxes is NExpTime*-hard. It is* NExpTime*-complete if unary coding of numbers in the input is assumed.*

Proof. Lemma 5.5 states that satisfiability of $\mathcal{ALCQIO}$-TBoxes and satisfiability of $\mathcal{ALCQI}$-CBoxes are mutually polynomially reducible problems. Hence, both the lower and the upper complexity bound follow from Theorem 5.19. ∎

This result explains a gap in (De Giacomo, 1995). There the author establishes the complexity of satisfiability of knowledge bases consisting of TBoxes and ABoxes both for $\mathcal{CNO}$, which allows for qualifying number restrictions, and for $\mathcal{CIO}$, which allows for inverse roles, by reduction to the ExpTime-complete logic PDL. No results are given for the combination $\mathcal{CINO}$, which is a strict extension of $\mathcal{ALCQIO}$. Corollary 5.21 shows that, assuming ExpTime $\neq$ NExpTime, there cannot be a polynomial reduction from the satisfiability problem of $\mathcal{CINO}$ knowledge bases to PDL. A possible explanation for this leap in complexity is the loss of the tree model property, which has been proposed by Vardi (1996) and Grädel (1999c) as an explanation for good algorithmic properties of a logic. While, for $\mathcal{CIO}$ and $\mathcal{CNO}$, satisfiability is decided by searching for tree-like pseudo-models even in the presence of nominals, this seems no longer to be possible in the case of knowledge bases for $\mathcal{CINO}$.

Unique Name Assumption

It should be noted that our definition of nominals is non-standard for DLs in the sense that we do not impose the unique name assumption that is widely made, i.e., for any two individual names $o_1, o_2 \in \mathsf{NI}$, $o_1^{\mathcal{I}} \neq o_2^{\mathcal{I}}$ is required. Even without a unique name assumption, it is possible to enforce distinct interpretation of nominals by adding axioms of the form $o_1 \sqsubseteq \neg o_2$, which we have already used in the proof of Lemma 5.3. Moreover, imposing a unique name assumption in the presence of inverse roles and number restriction leads to peculiar effects. Consider the following TBox:

$$\mathcal{T} = \{o \sqsubseteq ({\leq}k\ R\ \top),\ \top \sqsubseteq \exists R^{-1}.o\}$$

Under the unique name assumption, $\mathcal{T}$ is satisfiable iff NI contains at most k individual names, because each individual name must be interpreted by a unique element of the domain, every element of the domain must be reachable from $o^{\mathcal{I}}$ via the role R, and $o^{\mathcal{I}}$ may have at most k R-successors. We believe that this dependency of the satisfiability of a TBox on constraints that are not explicit in the TBox is counter-intuitive and hence have not imposed the unique name assumption.

Nevertheless, it is possible to obtain a tight complexity bound for satisfiability of $\mathcal{ALCQIO}$-TBoxes with unique name assumption without using Lemma 5.5, but by an immediate adaptation of the proof of Theorem 5.19.

Corollary 5.22
Satisfiability of $\mathcal{ALCQIO}$-TBoxes with the unique name assumption *is* NExpTime-*hard. It is* NExpTime-*complete if unary coding of numbers in the input is assumed.*

Proof. A simple inspection of the reduction used to prove Theorem 5.19, and especially of the proof of Lemma 5.17 shows that only a single nominal, which marks the upper right corner of the torus, is sufficient to perform the reduction. If o is an individual name and *create* is a role name, then the following TBox defines a torus of exponential size:

$$\begin{array}{rll}\mathcal{T}_n = \{ & \top \sqsubseteq \exists \mathit{east}.\top, & \top \sqsubseteq \exists \mathit{north}.\top, \\ & \top \sqsubseteq (= 1\ \mathit{east}^{-1}\ \top), & \top \sqsubseteq (= 1\ \mathit{north}^{-1}\ \top), \\ & \top \sqsubseteq \exists \mathit{create}.C_{(0,0)}, & \top \sqsubseteq D_{\mathit{east}} \sqcap D_{\mathit{north}}, \\ & C_{(2^n-1,2^n-1)} \sqsubseteq o, & o \sqsubseteq C_{(2^n-1,2^n-1)}\ \} \end{array}$$

Since this reduction uses only a single individual name, the unique name assumption is irrelevant in this case. ∎

Internalization of Axioms

In the presence of inverse roles and nominals, it is possible to *internalise* general inclusion axioms into concepts (Baader, 1991; Schild, 1991; Baader, Bürckert, Nebel, Nutt, & Smolka, 1993) using the *spy-point* technique used, e.g., by Blackburn and Seligman (1995) and Areces, Blackburn, and Marx (1999). The main idea of this technique is to enforce that all elements in the model of a concept are reachable in a single step from a distinguished point (the spy-point) marked by an individual name.

Definition 5.23
Let $\mathcal{T}$ be an $\mathcal{ALCQIO}$-TBox. W.l.o.g., we assume that $\mathcal{T}$ contains only a single axioms $\top \sqsubseteq D$. Let spy denote a fresh role name and i a fresh individual name. We define the function $\cdot^{spy}$ inductively on the structure of concepts by setting $A^{spy} = A$ for all $A \in \mathsf{NC}$, $o^{spy} = o$ for all $o \in \mathsf{NI}$, $(\neg C)^{spy} = \neg C^{spy}$, $(C_1 \sqcap C_2)^{spy} = C_1^{spy} \sqcap C_2^{spy}$, and $(\geq n\ R\ C)^{spy} = (\geq n\ R\ (\exists spy^{-1}.i) \sqcap C^{spy})$.

The internalization *$C_\mathcal{T}$ of $\mathcal{T}$ is defined as follows:*

$$C_\mathcal{T} = i \sqcap D^{spy} \sqcap \forall spy.D^{spy}$$

Lemma 5.24
Let $\mathcal{T}$ be an $\mathcal{ALCQIO}$-TBox. $\mathcal{T}$ is satisfiable iff $C_\mathcal{T}$ is satisfiable.

Proof. For the *if*-direction let $\mathcal{I}$ be a model of $C_\mathcal{T}$ with $a \in (C_\mathcal{T})^\mathcal{I}$. This implies $i^\mathcal{I} = \{a\}$. Let $\mathcal{I}'$ be defined by

$$\Delta^{\mathcal{I}'} = \{a\} \cup \{x \in \Delta^\mathcal{I} \mid (a,x) \in spy^\mathcal{I}\}$$

and $\cdot^{\mathcal{I}'} = \cdot^{\mathcal{I}}|_{\Delta^{\mathcal{I}'}}$.

Claim 5.25 For every $x \in \Delta^{\mathcal{I}'}$ and every $\mathcal{ALCQIO}$-concept C, we have $x \in (C^{spy})^\mathcal{I}$ iff $x \in C^{\mathcal{I}'}$.

We proof this claim by induction on the structure of C. The only interesting case is $C = (\geq n\ R\ D)$. In this case $C^{spy} = (\geq n\ R\ (\exists spy^{-1}.i) \sqcap D^{spy})$. We have

$$\begin{aligned} & x \in (\geq n\ R\ (\exists spy^{-1}.i) \sqcap D^{spy})^\mathcal{I} \\ \text{iff}\ \ & \sharp\{y \in \Delta^\mathcal{I} \mid (x,y) \in R^\mathcal{I} \text{ and } y \in (\exists spy^{-1}.i)^\mathcal{I} \cap (D^{spy})^\mathcal{I}\} \geqslant n \\ (*)\ \text{iff}\ \ & \sharp\{y \in \Delta^{\mathcal{I}'} \mid (x,y) \in R^{\mathcal{I}'} \text{ and } y \in D^{\mathcal{I}'}\} \geqslant n \\ \text{iff}\ \ & x \in (\geq n\ R\ D)^{\mathcal{I}'}, \end{aligned}$$

where the equivalence $(*)$ holds because, if $y \in (\exists spy^{-1}.i)^\mathcal{I} \cap (D^{spy})^\mathcal{I}$ then $y \in \Delta^{\mathcal{I}'}$ and $y \in D^{\mathcal{I}'}$ by induction. Also, if $y \in \Delta^{\mathcal{I}'}$, then $(x,y) \in R^\mathcal{I}$ iff $(x,y) \in R^{\mathcal{I}'}$ and hence the sets $\{y \in \Delta^\mathcal{I} \mid (x,y) \in R^\mathcal{I} \text{ and } y \in (\exists spy^{-1}.i)^\mathcal{I} \cap (D^{spy})^\mathcal{I}\}$ and $\{y \in \Delta^{\mathcal{I}'} \mid (x,y) \in R^{\mathcal{I}'} \text{ and } y \in D^{\mathcal{I}'}\}$ are equal.

By construction, for every $x \in \Delta^{\mathcal{I}'}$, $x \in (D^{spy})^\mathcal{I}$. Due to Claim 5.25, this implies $x \in D^{\mathcal{I}'}$ and hence $\mathcal{I}' \models \top \sqsubseteq D$.

For the *only-if*-direction, let $\mathcal{I}$ be an interpretation with $\mathcal{I} \models \mathcal{T}$. We pick an arbitrary element $a \in \Delta^\mathcal{I}$ and define an extension $\mathcal{I}'$ of $\mathcal{I}$ by setting $i^{\mathcal{I}'} = \{a\}$ and $spy^{\mathcal{I}'} = \{(a,x) \mid x \in \Delta^\mathcal{I}\}$. Since i and *spy* do not occur in $\mathcal{T}$, we still have that $\mathcal{I}' \models \mathcal{T}$.

Claim 5.26 For every $x \in \Delta^{\mathcal{I}'}$ and every $\mathcal{ALCQIO}$-concept C that does not contain i or *spy*, $x \in C^{\mathcal{I}'}$ iff $x \in (C^{spy})^{\mathcal{I}'}$.

Again, this claim is proved by induction on the structure of concepts and the only interesting case is $C = (\geq n\ R\ E)$.

$$\begin{array}{rl} & x \in (\geq n\ R\ E)^{\mathcal{I}'} \\ \text{iff} & \sharp\{y \in \Delta^{\mathcal{I}'} \mid (x,y) \in R^{\mathcal{I}'} \text{ and } y \in E^{\mathcal{I}'}\} \geqslant n \\ (*)\ \text{iff} & \sharp\{y \in \Delta^{\mathcal{I}'} \mid (x,y) \in R^{\mathcal{I}'}, (a,y) \in spy^{\mathcal{I}'}, \text{ and } y \in (E^{spy})^{\mathcal{I}'}\} \geqslant n \\ \text{iff} & x \in (\geq n\ R\ (\exists spy^{-1}.i) \sqcap E^{spy})^{\mathcal{I}'}. \end{array}$$

The equivalence $(*)$ holds because, by construction of $\mathcal{I}'$, $(a,y) \in spy^{\mathcal{I}'}$ holds for every element y of the domain and $y \in E^{\mathcal{I}'}$ iff $y \in (E^{spy})^{\mathcal{I}'}$ holds by induction.

Since, $\mathcal{I}' \models \top \sqsubseteq D$, Claim 5.26 yields that $(D^{spy})^{\mathcal{I}'} = \Delta^{\mathcal{I}'}$ and hence $a \in (C_{\mathcal{T}})^{\mathcal{I}'}$ ■

As a consequence, we obtain the sharper result that already pure concept satisfiability for $\mathcal{ALCQIO}$ is a NExpTime-complete problem.

Corollary 5.27
Concept satisfiability for $\mathcal{ALCQIO}$ is NExpTime*-hard. It is* NExpTime*-complete if unary coding of numbers in the input is assumed.*

Proof. From Lemma 5.24, we get that the function mapping a $\mathcal{ALCQIO}$-TBox $\mathcal{T}$ to $C_{\mathcal{T}}$ is a reduction from satisfiability of $\mathcal{ALCQIO}$-TBoxes to satisfiability of $\mathcal{ALCQIO}$ concepts. From Corollary 5.21 we know that the former problem is NExpTime-complete. Obviously, $C_{\mathcal{T}}$ can be computed from $\mathcal{T}$ in polynomial time. Hence, the lower complexity bound transfers. The NExpTime upper bound is a consequence of Corollary 5.21 and the fact that an $\mathcal{ALCQIO}$ concept C is satisfiable iff, for an individual j that does not occur in C, the TBox $\{j \sqsubseteq C\}$ is satisfiable. ■

5.2.2 Boolean Role Expressions

In Chapter 4, we have studied the DL $\mathcal{ALCQI}b$, which allowed for a restricted—so called safe—form of Boolean combination of roles, and for which concept satisfiability is decidable in polynomial space. It is easy to see that the results established for $\mathcal{ALCQI}$ in this chapter all transfer to $\mathcal{ALCQI}b$ and we state them here as (indeed, trivial) corollaries.

We have already argued that the restriction to safe role expressions is necessary to obtain a DL for which satisfiability is still decidable in polynomial space: the concept $(\leq 0\ (R \sqcup \neg R)\ \neg C)$ is satisfiable iff C is globally satisfiable, which is an ExpTime-complete problem (see, Theorem 3.18). Indeed, as a corollary of Theorem 5.19, it can be shown that concept satisfiability becomes a NExpTime-hard in the presence of arbitrary Boolean operations on roles.

Definition 5.28
The DL $\mathcal{ALCQIB}$ is defined as $\mathcal{ALCQI}b$ with the exception that arbitrary role expressions are allowed. The DL $\mathcal{ALCQB}$ is the restriction of $\mathcal{ALCQIB}$ that disallows inverse roles. The semantics of $\mathcal{ALCQIB}$ and $\mathcal{ALCQB}$ are define as for $\mathcal{ALCQI}b$. ◇

Decidability of concept and CBox satisfiability for $\mathcal{ALCQI}b$, $\mathcal{ALCQB}$, and $\mathcal{ALCQIB}$ in NExpTime can easily be shown by extending the embedding Ψ_x into C^2 from Figure 5.1 to deal with Boolean combination of roles.

Lemma 5.29
Satisfiability of $\mathcal{ALCQIB}$-concepts and $\mathcal{ALCQIB}$-CBoxes is polynomially reducible to C^2-satisfiability.

Proof. For a role expression ω, we define $\Psi_{xy}(\omega)$ inductively by

$$\begin{aligned}
\Psi_{xy}(R) &= Rxy \\
\Psi_{xy}(R^{-1}) &= Ryx \\
\Psi_{xy}(\neg\omega) &= \neg\Psi_{xy}(\omega) \\
\Psi_{xy}(\omega_1 \sqcap \omega_2) &= \Psi_{xy}(\omega_1) \wedge \Psi_{xy}(\omega_2) \\
\Psi_{xy}(\omega_1 \sqcup \omega_2) &= \Psi_{xy}(\omega_1) \vee \Psi_{xy}(\omega_2)
\end{aligned}$$

and set $\Psi_x(\bowtie n\ \omega\ C) = \exists^{\bowtie n} x.\Psi_{xy}(\omega) \wedge \Psi_y(C)$.

This translation is obviously polynomial and satisfies, for every interpretation $\mathcal{I}$ and concept C,

$$C^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \mathcal{I} \models \Psi_x(C)(a)\}.$$

Hence, a concept C is satisfiable iff $\exists^{\geq 1} x.\Psi_x(C)$ is satisfiable. CBoxes can be reduced to C^2 as shown in Figure 5.1. This yields the desired reductions. ■

Since $\mathcal{ALCQI}$ is a subset of $\mathcal{ALCQI}b$, which, in turn, is a subset of $\mathcal{ALCQIB}$, the following is a simple corollary of Theorem 5.19 and Theorem 5.29:

Corollary 5.30
Satisfiability of $\mathcal{ALCQI}b$- or $\mathcal{ALCQIB}$-CBoxes is NExpTime*-hard. The problems are* NExpTime*-complete if unary coding of numbers in the input is assumed.*

Proof. The lower bound is immediate from Corollary 5.19 because the set of $\mathcal{ALCQI}$-concepts is strictly included in the set of $\mathcal{ALCQI}b$- and $\mathcal{ALCQIB}$-concepts. In the case of unary coding of numbers in the input, the upper bound follows from Lemma 5.29 and Fact 5.9. ■

Similarly, the results for reasoning with nominals also transfer from Corollary 5.21.

Corollary 5.31
Satisfiability of $\mathcal{ALCQIbO}$- and $\mathcal{ALCQIBO}$-concepts is NExpTime*-hard. The problems are* NExpTime*-complete if unary coding of numbers in the input is assumed.*

So, in the presence of cardinality restrictions or nominals, reasoning with $\mathcal{ALCQIB}$ is not harder than reasoning with $\mathcal{ALCQIb}$. Without cardinality restrictions or nominals, though, reasoning with $\mathcal{ALCQIb}$ is less complex (ExpTime-complete, Theorem 4.42) than reasoning for $\mathcal{ALCQIB}$. The reason for this is that $\mathcal{ALCQIB}$ can easily mimic cardinality restrictions (and hence nominals) using a fresh role:

Lemma 5.32
CBox satisfiability for $\mathcal{ALCQ}$ and $\mathcal{ALCQI}$ is polynomially reducible to concept satisfiability of $\mathcal{ALCQB}$ and $\mathcal{ALCQIB}$ respectively.

Proof. Let $\mathcal{C}$ be a $\mathcal{ALCQ}(\mathcal{I})\mathcal{B}$-CBox and R a role that does not occur in $\mathcal{C}$. We transform $\mathcal{C}$ into a $\mathcal{ALCQ}(\mathcal{I})\mathcal{B}$ concept $C_\mathcal{C}$ by setting

$$C_\mathcal{C} = (\leq 0 \; \neg R \; \top) \sqcap \prod_{i=1}^{k} (\bowtie_i n_i \; R \; C_i).$$

Claim 5.33 $C_\mathcal{C}$ is satisfiable iff $\mathcal{C}$ is satisfiable.

Let $\mathcal{I}$ be a model for $\mathcal{C}$. We define a model $\mathcal{I}'$ of $C_\mathcal{C}$ by setting $R^{\mathcal{I}'} := \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ and preserving the interpretation of all other names. Since R does not occur in $\mathcal{C}$, $C_i^\mathcal{I} = C_i^{\mathcal{I}'}$ holds for every i. Since R is interpreted by the universal relation, $(\leq 0 \; \neg R \; \top)^{\mathcal{I}'} = \Delta^{\mathcal{I}'}$ holds. Also, again since $R^{\mathcal{I}'}$ is the universal relation, for every $x \in \Delta^{\mathcal{I}'}$, $\{y \mid (x,y) \in R^{\mathcal{I}'} \text{ and } y \in C^{\mathcal{I}'}\} = C^{\mathcal{I}'}$. Thus, if $\mathcal{I} \models (\bowtie_i n_i \; C_i)$, then $(\bowtie n_i \; R \; C_i)^{\mathcal{I}'} = \Delta^{\mathcal{I}'}$. Hence, from $\mathcal{I} \models \mathcal{C}$ is follows that $C_\mathcal{C}^{\mathcal{I}'} = \Delta^{\mathcal{I}'}$, which proves its satisfiability.

For the converse direction, if $C_\mathcal{C}$ is satisfiable with $x \in C_\mathcal{C}^\mathcal{I}$ for an interpretation $\mathcal{I}$, then, since $x \in (\leq 0 \; \neg R_i \; \top)^\mathcal{I}$, $\{y \mid (x,y) \in R^\mathcal{I}\} = \Delta^\mathcal{I}$ must hold and hence $\{y \mid (x,y) \in R^\mathcal{I} \text{ and } y \in C^\mathcal{I}\} = C^\mathcal{I}$. It immediately follows that $\mathcal{I} \models \mathcal{C}$.

Obviously, the size of $C_\mathcal{C}$ is linear in the size of $\mathcal{C}$, which proves this lemma. ∎

Corollary 5.34
Concept satisfiability for $\mathcal{ALCQB}$ and $\mathcal{ALCQIB}$ is NExpTime*-hard. The problems are* NExpTime*-complete if unary coding of numbers in the input is assumed.*

Proof. Concept satisfiability for $\mathcal{ALCQIB}$ is NExpTime-hard by Lemma 5.32 and Theorem 5.19, it can be decided in NExpTime by Lemma 5.29 and Fact 5.9.

For $\mathcal{ALCQB}$ the situation is slightly more complicated because Lemma 5.32 yields NExpTime-hardness only for binary coding of numbers. Yet, Lutz and Sattler (2000) show that concept satisfiability even for $\mathcal{ALCB}$, i.e., $\mathcal{ALC}$ extended with Boolean role expressions, is NExpTime-hard, which yields the lower bound also for the case of unary coding of numbers. The matching upper bound (in the case of unary coding) again follows from Lemma 5.29 and Fact 5.9. ∎

Of course, (Lutz & Sattler, 2000) yields the lower bound also for $\mathcal{ALCQIB}$. Since the connection between reasoning with cardinality restrictions and full Boolean role expression established in Lemma 5.32 is interesting in itself and yields, as a simple corollary, the result for $\mathcal{ALCQIB}$, we include this alternative proof of this fact in this thesis.

Chapter 6

Transitive Roles and Role Hierarchies

This chapter explores reasoning with Description Logics that allow for transitive roles. Transitive roles play an important rôle in knowledge representation because, as argued by Sattler (2000), transitive roles in combination with role hierarchies are adequate to represent aggregated objects, which occur in many application areas of knowledge representation, like configuration (Wache & Kamp, 1996; Sattler, 1996b; McGuinness & Wright, 1998), ontologies (Rector & Horrocks, 1997), or data modelling (Calvanese, Lenzerini, & Nardi, 1998; Calvanese, De Giacomo, Lenzerini, Nardi, & Rosati, 1998; Calvanese, De Giacomo, & Rosati, 1999; Franconi, Baader, Sattler, & Vassiliadis, 2000).

Baader (1991) and Schild (1991) were the first to study the *transitive closure* of roles in DLs that extend $\mathcal{ALC}$, and they both developed DLs that are notational variants of PDL (Fischer & Ladner, 1979). Due to the expressive power of the transitive closure, these logics allow for the internalisation of terminological axioms (Baader, 1991; Schild, 1991; Baader et al., 1993) and hence reasoning for these logics is at least ExpTime-hard. Sattler (1996a) studies a number of DLs with transitive constructs and identifies the DL $\mathcal{S}$,[1] i.e., $\mathcal{ALC}$ extended with *transitive roles*, as an extension of $\mathcal{ALC}$ that still permits a PSpace reasoning procedure.

Horrocks and Sattler (1998) study $\mathcal{SI}$, the extension of $\mathcal{S}$ with inverse roles, and develop a tableau based reasoning procedure. While they conjecture that concept satisfiability and subsumption can be decided for $\mathcal{SI}$ in PSpace, their algorithm only yields an NExpTime upper bound. We verify their conjecture by refining their tableau algorithm so that it decides concept satisfiability (and hence subsumption) in PSpace. A comparable approach is used by Spaan (1993b) to show that satisfiability of the modal logic $\mathsf{K4}_t$—corresponding to $\mathcal{SI}$ with only a single, transitive role—can be decided in PSpace.

Subsequently $\mathcal{SI}$ is extended with role hierarchies (Horrocks & Gough, 1997) and qualifying number restrictions, which yields the DL $\mathcal{SHIQ}$. The expressive power of $\mathcal{SHIQ}$ is particularly well suited to capture many properties of aggregated objects (Sattler, 2000) and has applications in the area of conceptual data models (Calvanese, Lenzerini, & Nardi, 1994; Franconi & Ng, 2000) and query optimization (Horrocks, Sattler, Tessaris, & Tobies,

[1]Previously, this logic has been called $\mathcal{ALC}_{R^+}$. Here, we use $\mathcal{S}$ instead because of a vague correspondence of $\mathcal{ALC}_{R^+}$ with the modal logic $\mathsf{S4}$.

2000). Furthermore, there exists the OIL approach (Fensel, Horrocks, van Harmelen, Decker, Erdmann, & Klein, 2000) to add $\mathcal{SHIQ}$-based inference capabilities to the semantic web (Berners-Lee, 1999). These applications have only become feasible due to the availability of the highly optimized reasoner iFaCT (Horrocks, 1999) for $\mathcal{SHIQ}$.

We determine the worst-case complexity of reasoning with $\mathcal{SHIQ}$ as ExpTime-complete, even if binary coding of numbers in the number restrictions is used. This result relies on a reduction of $\mathcal{SHIQ}$ to $\mathcal{ALCQI}b$ with TBoxes, a problem we already know how to solve in ExpTime (Theorem 4.38). Using the same reduction we show that reasoning for $\mathcal{SHIQO}$, i.e., the extension of $\mathcal{SHIQ}$ with nominals, is NExpTime-complete (in the case of unary coding of numbers).

As the upper ExpTime-bound for $\mathcal{SHIQ}$ relies on a highly inefficient automata construction, Section 6.3 extends the tableau algorithm for $\mathcal{SHIF}$ (Horrocks & Sattler, 1999) to deal with full qualifying number restrictions. While this algorithm does not meet the worst-case complexity of the problem (a naive implementation of the tableau algorithm would run in 2-NExpTime), it is amenable to optimizations and forms the basis of the highly optimised DL system iFaCT (Horrocks, 1999). See Section 3.1 for a discussion of the different reasoning paradigms and issues of practicability of algorithms.

6.1 Transitive and Inverse Roles: $\mathcal{SI}$

In this section we study the complexity of reasoning with the DL $\mathcal{SI}$, an extension of the DL $\mathcal{ALC}$ with *transitive roles* and *inverse roles*:

Definition 6.1 (Syntax and Semantics of $\mathcal{SI}$)
Let NC *be a set of atomic* concept names, NR *a set of atomic* role names, *and* $\mathsf{NR}^+ \subseteq \mathsf{NR}$ *a set of* transitive role names. *With* $\overline{\mathsf{NR}} := \mathsf{NR} \cup \{R^{-1} \mid R \in \mathsf{NR}\}$ *we denote the set of* $\mathcal{SI}$-roles. *The set of* $\mathcal{SI}$-concepts *is built inductively from* NC *and* $\overline{\mathsf{NR}}$ *using the following grammar, where* $A \in \mathsf{NC}$ *and* $R \in \overline{\mathsf{NR}}$*:*

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

The semantics of $\mathcal{SI}$-concepts is defined similarly to the semantics of $\mathcal{ALC}$-concepts w.r.t. an interpretation $\mathcal{I}$, where, for an inverse role $R^{-1} \in \overline{\mathsf{NR}}$, we set $(R^{-1})^{\mathcal{I}} = \{(y,x) \mid (x,y) \in R^{\mathcal{I}}\}$. Moreover, we only consider those interpretations that interpret transitive roles $R \in \mathsf{NR}^+$ by transitive relations. An $\mathcal{SI}$-concept C is satisfiable *iff there exists an interpretation $\mathcal{I}$ such that, for every $R \in \mathsf{NR}^+$, $R^{\mathcal{I}}$ is transitive, and $C^{\mathcal{I}} \neq \emptyset$. Subsumption is defined as usual, again with the restriction to interpretations that interpret transitive roles with transitive relations.*

With $\mathcal{S}$ we denote the fragment of $\mathcal{SI}$ that does not contain any inverse roles. ◇

In order to make the following considerations easier, we introduce two functions on roles:

1. The inverse relation on roles is symmetric, and to avoid considering roles such as R^{-1-1}, we define a function Inv which returns the inverse of a role. More precisely, $\mathsf{Inv}(R) = R^{-1}$ if R is a role name, and $\mathsf{Inv}(R) = S$ if $R = S^{-1}$.

2. Obviously, the interpretation $R^{\mathcal{I}}$ of a role R is transitive if and only if the interpretation of $\mathsf{Inv}(R)$ is transitive. However, this may be required by either R or $\mathsf{Inv}(R)$ being in NR^+. We therefore define a function Trans, which is true iff R must be interpreted with a transitive relation—regardless of whether it is a role name or the inverse of a role name. More precisely, $\mathsf{Trans}(R) = \mathtt{true}$ iff $R \in \mathsf{NR}^+$ or $\mathsf{Inv}(R) \in \mathsf{NR}^+$.

6.1.1 The $\mathcal{SI}$-algorithm

We will now describe a tableau algorithm that decides satisfiability of $\mathcal{SI}$-concepts in PSpace, thus proving PSpace-completeness of $\mathcal{SI}$-satisfiability. Like other tableau algorithms, the $\mathcal{SI}$-algorithm tries to prove the satisfiability of a concept C by constructing a model for C. The model is represented by a so-called *completion tree*, a tree some of whose nodes correspond to individuals in the model, each node being labelled with two sets of $\mathcal{SI}$-concepts. When testing the satisfiability of an $\mathcal{SI}$-concept C, these sets are restricted to subsets of $sub(C)$, where $sub(C)$ is the set of subconcepts of C, which is defined in the obvious way. Before we formally present the algorithm, we first discuss some problems that need to be overcome when trying to develop an $\mathcal{SI}$-algorithm that can be implemented to run in PSpace.

Dealing with transitive roles in tableau algorithms requires extra considerations because transitivity of a role is, generally speaking, a *global* constraint whereas the expansion rules and clash conditions of the tableau algorithms that we have studied so far are of a more *local* nature. They only take into account a single node of the constraint system or at most a node and its direct neighbours. Also, many of our previous considerations relied on the fact that satisfiable concepts have tree models, which, in the presence of transitive roles is no longer the case. To circumvent these problems, we use the solution that has already been used, e.g., by Halpern and Moses (1992) to deal with the modal logic $\mathsf{S4}$, which possesses a reflexive and transitive accessibility relation. Instead of directly dealing with models and transitive relations, we use abstractions of models—so called *tableaux*—that disregard transitivity of roles and have the form of a tree. This is done in a way that allows to recover a model of the input concept by transitively closing those role relations that are explicitly asserted in the tableau. To prove satisfiability of the input concept, the $\mathcal{SI}$-algorithm now tries to build a tableau instead of trying to construct a model. Apart from this difference, the $\mathcal{SI}$-algorithm is very similar to the tableau algorithms we have encountered so far: starting from an initial constraint system it employs completion rules until the constraint system is complete, in which case the existence of a tableau is evident, or until an obvious contradiction indicates an unsuccessful run of the (non-deterministic) algorithm.

While it would be possible to maintain the use of ABoxes to capture the constraint system that we will encounter during our discussion of DLs with transitive roles, it is much more convenient to emphasise the view of constraint systems as node- and edge-labelled trees, so this view will prevail in the remainder of this chapter.

6.1.2 Blocking

Sattler (1996a) shows that concept satisfiability for $\mathcal{S}$ can be determined in polynomial space using an adaptation of the techniques employed by Halpern and Moses (1992) to decide satisfiability for the modal logic $\mathsf{S4}$. To understand why these techniques cannot be extended easily to deal with inverse roles, as we have done in Chapter 4 when generalizing from $\mathcal{ALCQ}$ to $\mathcal{ALCQI}b$, we have to discuss the role of *blocking*.

The key difference between the algorithms from the previous chapters and the $\mathcal{SI}$-algorithm lies in the way universal restrictions are propagated through the constraint system: whenever $\forall R.C$ with $\mathsf{Trans}(R)$ appears in the label of a node x and x has an R-neighbour y, then not only C is asserted for y, but also $\forall R.C$. This makes sure that C is successively asserted for every node reachable from x via a *chain* of R-edges. These are exactly the nodes that are reachable from x with a single R-step once R has been transitively closed; exactly these nodes must satisfy C in order for $\forall R.C$ to hold at x.

Previously the termination of the tableau algorithms relied on the fact that the nesting of universal and existential restrictions strictly decreases along a path in the tableau. When dealing with transitive roles in the described manner, this is no longer guaranteed. For example, consider a node x labelled $\{C, \exists R.C, \forall R.(\exists R.C)\}$, where R is a transitive role. The described approach would cause the new node y that is created due for the $\exists R.C$ constraint to receive a label identical to the label of x. Thus, the expansion process could be repeated indefinitely.

The way we deal with this problem is by *blocking*: halting the expansion process when a cycle is detected (Baader, 1991; Buchheit et al., 1993; Halpern & Moses, 1992; Sattler, 1996a; Baader et al., 1996; Horrocks & Sattler, 1999). For logics without inverse roles, the general procedure is to check the constraints asserted for each new node y, and if they are a *subset* of the constraints for an existing node x, then no further expansion of y is performed: x is said to block y. The resulting constraint system corresponds to a cyclic model in which y is identified with x.[2] The validity of the cyclic model is an easy consequence of the fact that each $\exists R.D$ constraint for y must also be satisfied by x because the constraints for x are a superset of the constraints for y. Termination is now guaranteed by the fact that all constraints for individuals in the constraint system are ultimately derived from the decomposition of the input concept C, so every set of constraints for an individual must be a subset of the subconcepts of C, and a blocking situation must therefore occur within a finite number of expansion steps.

[2]For logics with a transitive closure operator it is necessary to check the validity of the cyclic model created by blocking (Baader, 1991), but for logics that only support transitive roles the cyclic model is always valid (Sattler, 1996a).

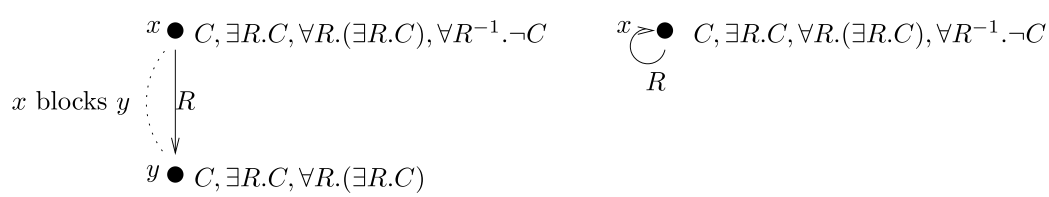
Dynamic Blocking

Blocking is more problematic when inverse roles are added to the logic, and a key feature of the algorithms presented in (Horrocks & Sattler, 1999) was the introduction of a *dynamic blocking* strategy. It uses label equality instead of the subset condition, and it allows blocks to be established, broken, and re-established.

Label inclusion as a blocking criterion is no longer correct in the presence of inverse roles because roles are now bi-directional, and hence universal restrictions at the blocking node can conflict with the constraints for the predecessor of the blocked node.

Taking the above example of a node labelled $\{C, \exists R.C, \forall R.(\exists R.C)\}$, if the successor of this node were blocked by a node whose label additionally included $\forall R^{-1}.\neg C$, then the cyclic model would clearly be invalid. This is shown in Figure 6.1, where x blocks its R-successor y (if subset-blocking is assumed) and hence in the induced model (shown on the right), there exists an R-cycle from x to x. Hence, C and $\forall R^{-1}.\neg C$, which have both been asserted for x, now stand in a conflict.

Figure 6.1 An invalid cyclic model



In (Horrocks & Sattler, 1999), this problem was overcome by allowing a node x to be blocked by one of its ancestors if and only if they were labelled with the same set of concepts.

Another difficulty introduced by inverse roles is the fact that it is no longer possible to establish a block on a once-and-for-all basis when a new node is added to the tree. This is because further expansion in other parts of the tree could lead to the labels of the blocking and/or blocked node being extended and the block being invalidated. Consider the example sketched in Figure 6.2, which shows parts of a tableau that was generated for the concept

$$A \sqcap \exists S.(\exists R.\top \sqcap \exists P.\top \sqcap \forall R.C \sqcap \forall P.(\exists R.\top) \sqcap \forall P.(\forall R.C) \sqcap \forall P.(\exists P.\top)),$$

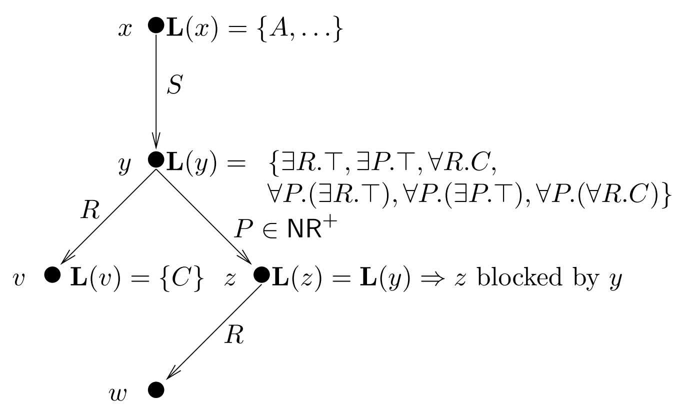
where C represents the concept

$$\forall R^{-1}.(\forall P^{-1}.(\forall S^{-1}.\neg A)).$$

This concept is clearly not satisfiable: w has to be an instance of C, which implies that x is an instance of $\neg A$. This is inconsistent with x being an instance of A.

Since P is a transitive role, all universal value restrictions over P are propagated from y to z, hence y and z are labelled with the same constraints and hence z is blocked by y.

Figure 6.2 A tableau where dynamic blocking is crucial



If the blocking of z was not subsequently broken when $\forall P^{-1}.(\forall S^{-1}.\neg A)$ is added to $\mathbf{L}(y)$ from $C \in \mathbf{L}(v)$, then $\neg A$ would never be added to $\mathbf{L}(x)$ and the unsatisfiability would not be detected.

As well as allowing blocks to be broken, it is also necessary to continue with some expansion of blocked nodes, because $\forall R.C$ concepts in their labels could affect other parts of the tree. Again, let us consider the example in Figure 6.2. After the blocking of z is broken and $\forall P^{-1}.(\forall S^{-1}.\neg A)$ is added to $\mathbf{L}(z)$ from $C \in \mathbf{L}(w)$, z is again blocked by y. However, the universal value restriction $\forall P^{-1}.(\forall S^{-1}.\neg A) \in \mathbf{L}(z)$ has to be expanded in order to detect the unsatisfiability.

These problems are overcome by using *dynamic* blocking: using label equality as blocking criterion and allowing blocks to be dynamically established and broken as the expansion progresses, and continuing to expand $\forall R.C$ concepts in the labels of blocked nodes.

Refined blocking

As mentioned before, in (Horrocks & Sattler, 1999), blocking of nodes is based on label equality. This leads to major problems when trying to establish a polynomial bound on the length of paths in the completion tree. If a node can only be blocked by an ancestor when the labels coincide, then there could potentially be exponentially many ancestors in a path before blocking actually occurs. Due to the non-deterministic nature of the expansion rules, these subsets might actually be generated; the algorithm would then need to store the node labels of a path of exponential length, thus consuming exponential space.

This problem is already present when one tries to implement a tableau algorithm for the logic $\mathcal{ALC}_{R^+}$ (Sattler, 1996a), where the non-deterministic nature of the expansion rules for disjunction might lead to the generation of a chain of exponential size before blocking occurs. Consider, for example, the concept

$$
\begin{aligned}
C &= \exists R.D \sqcap \forall R.(\exists R.D)\\
D &= (A_1 \sqcup B_1) \sqcap (A_2 \sqcup B_2) \sqcap \cdots \sqcap (A_n \sqcup B_n)
\end{aligned}
$$

where R is a transitive role. The concept C causes the generation of a chain of R-successors for all of which D is asserted. There are 2^n possible ways of expanding D because for every disjunctive concept $A_i \sqcup B_i$ the $\rightarrow_\sqcup$-rule can choose to add A_i or B_i. The completion tree for D is only complete once one node of this path is blocked and all unblocked nodes (including the blocking node) are fully expanded. For $\mathcal{ALC}_{R^+}$, a polynomial bound on the length of paths is obtained by applying a simple strategy: a new successor is only generated when no other rule can be applied, and propositional expansion of concepts only takes place if universal restrictions have been exhaustively been dealt with. Once a node is blocked, it is not necessary to perform its propositional expansion because it has already been ensured at the blocked node that such an expansion is possible without causing a clash.

However, in the presence of inverse roles, this strategy is no longer possible. Indeed, the expansion rules for $\mathcal{SI}$ as they have been presented in (Horrocks & Sattler, 1998) based on set equality might lead to a tableau with paths of exponential length for C—even though C does not contain any inverse roles. This is due to the fact that blocking is established on the basis of label equality. Since the label of the blocked and blocking node must be equal, this implies that, since the label of the blocking node must be fully expanded, this also must hold for the label of the blocked. Since there are 2^n possibilities for such an expansion, it might indeed take a path of $2^n + 1$ nodes before such a situation necessarily occurs and the completion tree is complete.

In order to obtain a tableau algorithm that circumvents this problem and guarantees blocking after a polynomial number of steps, we will keep the information that is relevant for blocking separated from the "irrelevant" information (due to propositional expansion) in a way which allows for a simple and comprehensible tableau algorithm. In the following, we will explain this "separation" idea in more detail.

Figure 6.3 Refined blocking

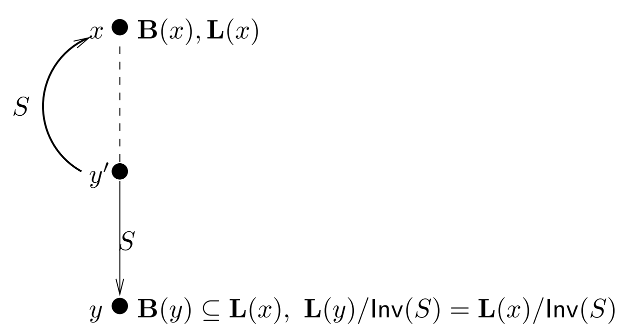


Figure 6.3 shows a blocking situation. Assume node y to be blocked by node x. When generating a model from this tree, the blocked node y will be omitted and y' will get x as an S-successor, which is indicated by the backward arrow. On the one hand, this construction yields a new S-successor x of y', a situation which is taken care of by the subset blocking used in the normal $\mathcal{ALC}_{R^+}$ tableau algorithms. On the other hand, x receives a new S^{-1}-successor y'. Now blocking has to make sure that, if x must satisfy a concept of the form

$\forall S^{-1}.D$, then D (and $\forall S^{-1}.D$ if S is a transitive role) is satisfied by y'.

This was dealt with by equality blocking in (Horrocks & Sattler, 1999). In the following algorithm it will be dealt with using two labels per node and a modified blocking condition that takes these two labels into account. In addition to the label **L**, each node now has a second label **B**, where the latter is always a subset of the former. The label **L** contains complete information, whereas **B** contains only information relevant to blocking. Propositional consequences of concepts in **L** and concepts being propagated "upwards" in the tree are stored in **L** only, as they are not important for blocking as long as they are not universal restrictions that state requirements on the predecessor in the completion tree. The modified blocking condition now looks as follows. For a node y to be blocked by a node x we require that

- the label $\mathbf{B}(y)$ of the blocked node y is contained in the label $\mathbf{L}(x)$ of the blocking node x. Expansions of disjunctions are only stored in **L** and thus cannot prevent a node from being blocked.
- if y is reachable from its predecessor in the completion tree via the role S, then the universal restrictions along $\mathsf{Inv}(S)$ asserted for y are the same as those asserted for x. This takes care of the fact that the predecessor y' of the blocked node y becomes a new $\mathsf{Inv}(S)$-successor of the blocking node x.

Summing up, we build a completion tree in a way that, for all nodes x,

- we have $\mathbf{B}(x) \subseteq \mathbf{L}(x)$,
- $\mathbf{B}(x)$ contains only concepts which move *down* the tree,
- $\mathbf{L}(x)$ contains, additionally, all concepts which move *up* the tree, and
- expansion of disjunctions and conjunctions only affects $\mathbf{L}(x)$.

6.1.3 A Tableau Algorithm for $\mathcal{SI}$

We now present a tableau algorithm derived from the one presented in (Horrocks & Sattler, 1999). We shape the rules in a way that allows for the separation of the concepts which are relevant for the two parts of the blocking condition. For ease of construction, we assume all concepts to be in *negation normal form* (NNF), that is, negation occurs only in front of concept names. Any $\mathcal{SI}$-concept can easily be transformed into an equivalent one in NNF in the same way as this can be done for $\mathcal{ALC}$-concepts (Definition 3.1).

The soundness and completeness of the algorithm will be proved by showing that it creates a *tableau* for C. In contrast to the approach we have taken in the previous chapters, where a constraint system stood in direct correspondence to a model, here we introduce tableaux as intermediate structures that encapsule the transition from the *syntactic object* of a completion tree to the *semantical* object of a model and takes care of the transitive roles on that way. This makes it possible for the algorithm to operate on trees even though $\mathcal{SI}$ does not have a genuine tree model property due to its transitive roles.

Definition 6.2 (A Tableau for $\mathcal{SI}$)
If C is an $\mathcal{SI}$-concept in NNF and $\overline{\mathsf{NR}}_C$ is the set of roles occurring in C together with their inverses, a tableau *$\mathcal{T}$ for C is a triple $(\mathcal{S}, \mathcal{L}, \mathcal{E})$ such that $\mathcal{S}$ is a non-empty set, $\mathcal{L} : \mathcal{S} \to 2^{sub(C)}$ maps each element of $\mathcal{S}$ to a subset of $sub(C)$, and $\mathcal{E} : \overline{\mathsf{NR}}_C \to 2^{\mathcal{S}\times\mathcal{S}}$ maps each role in $\overline{\mathsf{NR}}_C$ to a set of pairs of individuals. In addition, the following conditions must be satisfied:*

(T1) There is an $s \in \mathcal{S}$ with $C \in \mathcal{L}(s)$, and

for all $s, t \in \mathcal{S}$, $A, C_1, C_2, D \in sub(C)$, and $R \in \overline{\mathsf{NR}}_C$,

(T2) if $A \in \mathcal{L}(s)$, then $\neg A \notin \mathcal{L}(s)$, for $A \in \mathsf{NC}$,

(T3) if $C_1 \sqcap C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ and $C_2 \in \mathcal{L}(s)$,

(T4) if $C_1 \sqcup C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ or $C_2 \in \mathcal{L}(s)$,

(T5) if $\exists R.D \in \mathcal{L}(s)$, then there is some $t \in \mathcal{S}$ such that $(s,t) \in \mathcal{E}(R)$ and $D \in \mathcal{L}(t)$,

(T6) if $\forall R.D \in \mathcal{L}(s)$ and $(s,t) \in \mathcal{E}(R)$, then $D \in \mathcal{L}(t)$,

(T7) if $\forall R.D \in \mathcal{L}(s)$, $(s,t) \in \mathcal{E}(R)$ and $\mathsf{Trans}(R)$, then $\forall R.D \in \mathcal{L}(t)$, and

(T8) $(s,t) \in \mathcal{E}(R)$ iff $(t,s) \in \mathcal{E}(\mathsf{Inv}(R))$. $\diamond$

A tableau $\mathcal{T}$ for a concept C is a "syntactic witness" for the satisfiability of C:

Lemma 6.3
An $\mathcal{SI}$-concept C is satisfiable iff there exists a tableau for C.

Proof. For the *if*-direction, if $\mathcal{T} = (\mathcal{S}, \mathcal{L}, \mathcal{E})$ is a tableau for C with $C \in \mathcal{L}(s_0)$, a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of C can be defined as follows:

$$\begin{array}{rcl}
\Delta^{\mathcal{I}} & = & \mathcal{S}, \\
A^{\mathcal{I}} & = & \{s \mid A \in \mathcal{L}(s)\} \quad \text{for all concept names A in } sub(C), \\
R^{\mathcal{I}} & = & \left\{ \begin{array}{ll} \mathcal{E}(R)^+ & \text{if } \mathsf{Trans}(R) \\ \mathcal{E}(R) & \text{otherwise,} \end{array} \right.
\end{array}$$

where $\mathcal{E}(R)^+$ denotes the transitive closure of $\mathcal{E}(R)$. Transitive roles are interpreted by transitive relations by definition. By induction on the structure of concepts, we show that, if $D \in \mathcal{L}(s)$, then $s \in D^{\mathcal{I}}$. This implies $C^{\mathcal{I}} \neq \emptyset$ because $s_0 \in C^{\mathcal{I}}$. Let $D \in \mathcal{L}(s)$:

1. If $D = A \in \mathsf{NC}$ is a concept name, then $s \in D^{\mathcal{I}}$ by definition.

2. If $D = \neg A$ for $A \in \mathsf{NC}$ then $A \notin \mathcal{L}(s)$ (due to (T2)), so $s \in \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} = D^{\mathcal{I}}$.

3. If $D = (C_1 \sqcap C_2)$, then, due to (T3), $C_1 \in \mathcal{L}(s)$ and $C_2 \in \mathcal{L}(s)$, and hence, by induction, $s \in C_1^{\mathcal{I}}$ and $s \in C_2^{\mathcal{I}}$. Thus, $s \in (C_1 \sqcap C_2)^{\mathcal{I}}$.

4. The case $D = (C_1 \sqcup C_2)$ is analogous to the previous one.

5. If $D = \exists R.E$, then, due to (T5), there is some $t \in \mathcal{S}$ such that $(s,t) \in \mathcal{E}(R)$ and $E \in \mathcal{L}(t)$. By definition of $\mathcal{I}$, $(s,t) \in R^{\mathcal{I}}$ holds as follows. It is immediate, if $R \in \mathsf{NR}$. If $R = S^{-1}$ for $S \in \mathsf{NR}$, then $(s,t) \in \mathcal{E}(R)$ implies $(t,s) \in \mathcal{E}(S)$ by (T8). Hence, $(t,s) \in S^{\mathcal{I}}$ and $(s,t) \in R^{\mathcal{I}}$ holds. By induction, $t \in E^{\mathcal{I}}$ and hence $s \in (\exists R.E)^{\mathcal{I}}$.

6. If $D = (\forall R.E)$ and $(s,t) \in R^{\mathcal{I}}$, then either

   (a) $(s,t) \in \mathcal{E}(R)$ and $E \in \mathcal{L}(t)$ (due to (T6)), or

   (b) $(s,t) \notin \mathcal{E}(R)$. Due to (T8), this can only be the case if R is transitive and there exists a path of length $n \geq 1$ such that $(s,s_1), (s_1,s_2), \ldots, (s_n,t) \in \mathcal{E}(R)$. Due to (T7), $\forall R.E \in \mathcal{L}(s_i)$ for all $1 \leqslant i \leqslant n$, and we have $E \in \mathcal{L}(t)$, again due to (T6).

   In both cases, by induction $t \in E^{\mathcal{I}}$ holds, and hence $s \in (\forall R.E)^{\mathcal{I}}$.

For the converse direction, if $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of C, then a tableau $\mathcal{T} = (\mathcal{S}, \mathcal{L}, \mathcal{E})$ for C can be defined by:

$$\begin{aligned} \mathcal{S} &= \Delta^{\mathcal{I}}, \\ \mathcal{E}(R) &= R^{\mathcal{I}}, \\ \mathcal{L}(s) &= \{D \in \mathit{sub}(D) \mid s \in D^{\mathcal{I}}\}. \end{aligned}$$

It remains to demonstrate that $\mathcal{T}$ is a tableau for C:

1. $\mathcal{T}$ satisfies (T1) – (T6) and (T8) as a direct consequence of the semantics of $\mathcal{SI}$ concepts and of inverse roles.

2. If $s \in (\forall R.D)^{\mathcal{I}}$, $(s,t) \in R^{\mathcal{I}}$ and $\mathsf{Trans}(R)$, then $t \in (\forall R.D)^{\mathcal{I}}$ unless there is some u such that $(t,u) \in R^{\mathcal{I}}$ and $u \notin D^{\mathcal{I}}$. However, if $(s,t) \in R^{\mathcal{I}}$, $(t,u) \in R^{\mathcal{I}}$ and $\mathsf{Trans}(R)$, then $(s,u) \in R^{\mathcal{I}}$, which would imply $s \notin (\forall R.D)^{\mathcal{I}}$. T therefore satisfies (T7). ∎

6.1.4 Constructing an $\mathcal{SI}$ Tableau

From Lemma 6.3, it follows that an algorithm that constructs a tableau for an $\mathcal{SI}$-concept C can be used as a decision procedure for the satisfiability of C. Such an algorithm will now be described.

Like the tableau algorithms that we have studied so far, the algorithm for $\mathcal{SI}$ works by manipulating a constraint system. In the presence of blocking, and especially in the case of the refined blocking we are using for $\mathcal{SI}$, it is more convenient to emphasise the graph structure of the constraint system and deal with an edge- and node-labelled graph instead of an ABox. In case of the $\mathcal{SI}$-algorithm, the constraint system has the form of a *completion tree*.

Algorithm 6.4 (The $\mathcal{SI}$-algorithm)
Let C be an $\mathcal{SI}$-concept in NNF to be tested for satisfiability and $\overline{\mathsf{NR}}_C$ the set of roles that occur in C together with their inverse. A completion tree $\mathbf{T} = (\mathbf{V}, \mathbf{E}, \mathbf{L}, \mathbf{B})$ *is a labelled tree in which each node $x \in \mathbf{V}$ is labelled with two subsets $\mathbf{L}(x)$ and $\mathbf{B}(x)$ of $sub(C)$. Furthermore, each edge $(x, y) \in \mathbf{E}$ in the tree is labelled $\mathbf{L}(x, y) = R$ for some (possibly inverse) role $R \in \overline{\mathsf{NR}}_C$. Nodes and edges are added when expanding $\exists R.D$ and $\exists R^{-1}.D$ constraints; they correspond to relationships between pairs of individuals and are always directed from the root node to the leaf nodes. The algorithm expands the tree by extending $\mathbf{L}(x)$ (and possibly $\mathbf{B}(x)$) for some node x, or by adding new leaf nodes.*

A completion tree $\mathbf{T}$ is said to contain a clash *if, for a node x in $\mathbf{T}$, it holds that there is a concept name A such that $\{A, \neg A\} \subseteq \mathbf{L}(x)$.*

If nodes x and y are connected by an edge $(x, y) \in \mathbf{E}$, then y is called a successor *of x and x is called a* predecessor *of y. If $\mathbf{L}(x, y) = R$, then y is called an R-*successor *of x and x is called an $\mathsf{Inv}(R)$-*predecessor *of y.*

Ancestor *is the transitive closure of* predecessor *and* descendant *is the transitive closure of* successor. *A node y is called an R-*neighbour *of a node x if either y is an R-successor of x or y is an R-predecessor of x.*

To define the blocking condition we need the following auxiliary definition. For a (possibly inverse) role $S \in \overline{\mathsf{NR}}_C$, we define the set $\mathbf{L}(y)/S$ by

$$\mathbf{L}(y)/S = \{\forall S.D \in \mathbf{L}(y)\}.$$

A node y is blocked *if for some ancestor x of y, x is blocked or*

$$\mathbf{B}(y) \subseteq \mathbf{L}(x) \quad \textit{and} \quad \mathbf{L}(y)/\mathsf{Inv}(S) = \mathbf{L}(x)/\mathsf{Inv}(S)$$

for the unique predecessor y' of y in the completion tree, $\mathbf{L}(y', y) = S$ holds.

The algorithm initializes a tree $\mathbf{T}$ to contain a single node x_0, called the root *node, with $\mathbf{L}(x_0) = \mathbf{B}(x_0) = \{C\}$. $\mathbf{T}$ is then expanded by repeatedly applying the rules from Figure 6.4.*

The $\rightarrow_\exists$-rule is called generating; *all other rules are called* non-generating.

The completion tree is complete *if, for some node x, $\mathbf{L}(x)$ contains a clash or if none of the expansion rules is applicable. If the expansion rules can be applied in such a way that they yield a complete, clash-free completion tree, then the algorithm returns "C is satisfiable"; otherwise, the algorithm returns "C is not satisfiable".* ◇

Like for all other tableau algorithms studied in this thesis, it turns out (see the proof of Lemma 6.8) that the choice of which rule to apply where and when is don't-care non-deterministic—no choice can prohibit the discovery of a complete and clash-free completion tree for a satisfiable concept. On the other hand, as before, the choice of the $\rightarrow_\sqcup$-rule is don't-know non-deterministic—only certain choices will lead to the discovery of a complete and clash-free completion tree for a satisfiable concept. For an implementation this means that an arbitrary strategy that selects which rule to apply where will yield a complete implementation but exhaustive search is required to consider the different choices of the

Figure 6.4 Tableau expansion rules for $\mathcal{SI}$

$\rightarrow_{\sqcap}$: **if** 1. $C_1 \sqcap C_2 \in \mathbf{L}(x)$ and
2. $\{C_1, C_2\} \not\subseteq \mathbf{L}(x)$
then $\mathbf{L}(x) \rightarrow_{\sqcap} \mathbf{L}(x) \cup \{C_1, C_2\}$

$\rightarrow_{\sqcup}$: **if** 1. $C_1 \sqcup C_2 \in \mathbf{L}(x)$ and
2. $\{C_1, C_2\} \cap \mathbf{L}(x) = \emptyset$
then $\mathbf{L}(x) \rightarrow_{\sqcup} \mathbf{L}(x) \cup \{E\}$ for some $E \in \{C_1, C_2\}$

$\rightarrow_{\forall}$: **if** 1. $\forall R.D \in \mathbf{L}(x)$ and
2. there is an R-successor y of x with $D \notin \mathbf{B}(y)$
then $\mathbf{L}(y) \rightarrow_{\forall} \mathbf{L}(y) \cup \{D\}$ and $\mathbf{B}(y) \rightarrow_{\forall} \mathbf{B}(y) \cup \{D\}$ **or**
2'. there is an R-predecessor y of x with $D \notin \mathbf{L}(y)$
then $\mathbf{L}(y) \rightarrow_{\forall} \mathbf{L}(y) \cup \{D\}$ and delete *all* descendants of y.

$\rightarrow_{\forall_+}$: **if** 1. $\forall R.D \in \mathbf{L}(x)$ and $\mathsf{Trans}(R)$ and
2. there is an R-successor y of x with $\forall R.D \notin \mathbf{B}(y)$
then $\mathbf{L}(y) \rightarrow_{\forall_+} \mathbf{L}(y) \cup \{\forall R.D\}$ and $\mathbf{B}(y) \rightarrow_{\forall_+} \mathbf{B}(y) \cup \{\forall R.D\}$ **or**
2'. there is an R-predecessor y of x with $\forall R.D \notin \mathbf{L}(y)$
then $\mathbf{L}(y) \rightarrow_{\forall_+} \mathbf{L}(y) \cup \{\forall R.D\}$ and delete *all* descendants of y.

$\rightarrow_{\exists}$: **if** 1. $\exists R.D \in \mathbf{L}(x)$, x is not blocked and no non-generating rule is applicable to x and any of its ancestors, and
2. x has no R-neighbour y with $D \in \mathbf{B}(y)$
then create a new node y with $\mathbf{L}(x, y) = R$ and $\mathbf{L}(y) = \mathbf{B}(y) = \{D\}$

$\rightarrow_{\sqcup}$-rule. A similar situation exist in case of the $\mathcal{SHIQ}$-algorithm in Section 6.3, where it follows from the proof of Lemma 6.36 that choice of which rule to apply where is don't-care non-deterministic.

Note that in the definition of successor and predecessor, the tree structure is reflected. If y is an R-successor of x than this implies that y is successor of x in the completion tree and it is *not* the case that x is an $\mathsf{Inv}(R)$-predecessor of y. Successor and predecessor always refer to the relative position of nodes in the completion tree. This is necessary because, in the construction of a tableau from a complete and clash-free completion tree, the edges pointing to blocked successors will be redirected to the respective blocking nodes, which makes the relative position of nodes in the completion tree significant.

We are aiming for a PSpace-decision procedure, so, like the$\mathcal{ALCQI}b$-algorithm (Algorithm 4.21), the $\rightarrow_{\forall}$- and $\rightarrow_{\forall_+}$-rules delete parts of the completion tree whenever information is propagated upward in the completion tree to make tracing possible.

Correctness

As before, correctness of the algorithm will be demonstrated by proving that, for an $\mathcal{SI}$-concept C, it always terminates and that it returns "satisfiable" if and only if C is satisfiable. To prove this, we follow a slightly different approach than the one that is indicated by Theorem 3.6. The reason for this is that it is unclear how to deal with blocked nodes when trying to define a suitable notion of satisfiability for a completion tree. We will come back to this topic.

Before we start proving the properties we need to establish correctness of the $\mathcal{SI}$-algorithm, let us state an obvious property of the $\mathcal{SI}$-algorithm:

Lemma 6.5
Let $\mathbf{T}$ be a completion tree generated by the $\mathcal{SI}$-algorithm. Then, for every node x of $\mathbf{T}$, $\mathbf{B}(x) \subseteq \mathbf{L}(x)$.

Proof. Obviously, $\mathbf{B}(x_0) \subseteq \mathbf{L}(x_0)$ holds for the only node x_0 of the initial tree. Subsequently, whenever a concept D is added to $\mathbf{B}(x)$ by an application of one of the rules, then it is always also added to $\mathbf{L}(x)$. ∎

We first show termination of the algorithm:

Lemma 6.6 (Termination)
For each $\mathcal{SI}$-concept C, the tableau algorithm terminates.

Proof. Let $m = \sharp sub(C)$. Obviously, m is linear in the length of C. Termination is a consequence of the following properties of the expansion rules:

1. The expansion rules never remove concepts from node labels.

2. Successors are only generated for concepts of the form $\exists R.D$ and, for any node, each of these concepts triggers the generation of at most one successor. Since $sub(C)$ contains at most m concepts of the form $\exists R.D$, the out-degree of the tree is bounded by m.

3. Nodes are labelled with nonempty subsets of $sub(C)$. If a path p is of length $> 2^{2m}$, then there are 2 nodes x, y on p, with $\mathbf{L}(x) = \mathbf{L}(y)$ and $\mathbf{B}(x) = \mathbf{B}(y)$, and blocking occurs. Since a path on which nodes are blocked cannot become longer, paths are of length at most $2^{2m} + 1$.

An infinite run of the completion algorithm can thus only occur due to an infinite number of deletions of nodes of the tree. That this can never happen can be shown in exactly the same way this has been done in the proof of Lemma 4.22. ∎

Lemma 6.7 (Soundness)
If the $\mathcal{SI}$-algorithm generates a complete and clash-free completion tree for a concept C, then C has a tableau.

Proof. Let $\mathbf{T} = (\mathbf{V}, \mathbf{E}, \mathbf{L}, \mathbf{B})$ be the complete and clash-free completion tree constructed by the tableau algorithm for C. A tableau $\mathcal{T} = (\mathcal{S}, \mathcal{L}, \mathcal{E})$ can be defined by

$$\begin{aligned}
\mathcal{S} &= \{x \mid x \text{ is a node in } \mathbf{T}\text{, and } x \text{ is not blocked}\},\\
\mathcal{L} &= \mathbf{L}|_{\mathcal{S}},\\
\mathcal{E}(R) &= \{(x,y) \in \mathcal{S} \times \mathcal{S} \mid \begin{array}[t]{l} 1.\, y \text{ is an } R\text{-neighbour of } x \quad \textit{or}\\ 2.\, \exists z.\mathbf{L}(x,z) = R \text{ and } y \text{ blocks } z \quad \textit{or}\\ 3.\, \exists z.\mathbf{L}(y,z) = \mathsf{Inv}(R) \text{ and } x \text{ blocks } z\}. \end{array}
\end{aligned}$$

We now show that $\mathcal{T}$ is a tableau for C. By definition of $\mathcal{T}$, we have $\mathbf{L} = \mathcal{L}$ for all $x \in \mathcal{S}$, so it is sufficient to establish the required properties for the function $\mathbf{L}$.

- $C \in \mathbf{L}(x_0)$ for the root x_0 of $\mathbf{T}$ and, since x_0 has no predecessors, it cannot be blocked. Hence $C \in \mathbf{L}(s)$ for some $s \in \mathcal{S}$ and **(T1)** holds.
- **(T2)** is satisfied because $\mathbf{T}$ is clash-free.
- **(T3)** and **(T4)** are satisfied because neither the $\rightarrow_\sqcap$-rule nor the $\rightarrow_\sqcup$-rule apply to any $x \in \mathcal{S}$. Hence $\mathbf{L}(x)$ satisfies the required properties.
- **(T5)** is satisfied because, for all $x \in \mathbf{S}$, if $\exists R.D \in \mathbf{L}(x)$, then the $\rightarrow_\exists$-rule ensures that there is either:
    1. an R-predecessor y with $D \in \mathbf{B}(y) \subseteq \mathbf{L}(y)$ (see Lemma 6.5). Because y is a predecessor of x (which is an unblocked node), it cannot be blocked, so $y \in \mathcal{S}$ and $(x,y) \in \mathcal{E}(R)$.
    2. an R-successor y with $D \in \mathbf{B}(y) \subseteq \mathbf{L}(y)$ (again, see Lemma 6.5). If y is not blocked, then $y \in \mathcal{S}$ and $(x,y) \in \mathcal{E}(R)$. Otherwise, y is blocked by some z with $\mathbf{B}(y) \subseteq \mathbf{L}(z)$. Hence $D \in \mathbf{L}(z)$, $z \in \mathcal{S}$ and $(x,z) \in \mathcal{E}(R)$.
- To show that **(T6)** is satisfied for all $x \in \mathcal{S}$, if $\forall R.D \in \mathbf{L}(x)$ and $(x,y) \in \mathcal{E}(R)$, we have to consider three possible cases:
    1. y is an R-neighbour of x. The $\rightarrow_\forall$-rule guarantees $D \in \mathbf{L}(y)$.
    2. $\mathbf{L}(x,z) = R$, y blocks z. Then by the $\rightarrow_\forall$-rule we have $D \in \mathbf{B}(z)$ and, by the definition of blocking, $\mathbf{B}(z) \subseteq \mathbf{L}(y)$. Hence $D \in \mathbf{L}(y)$.
    3. $\mathbf{L}(y,z) = \mathsf{Inv}(R)$, x blocks z. From the definition of blocking we have that $\mathbf{L}(z)/R = \mathbf{L}(x)/R$. Hence $\forall R.D \in \mathbf{L}(z)$ and the $\rightarrow_\forall$-rule guarantees $D \in \mathbf{L}(y)$.

  In all three cases, $D \in \mathbf{L}(y)$ holds.

- For **(T7)**, let $x \in \mathcal{S}$ with $\forall R.D \in \mathbf{L}(x)$, $(x, y) \in \mathcal{E}(R)$, and $\mathsf{Trans}(R)$. There are three possible cases:

  1. y is an R-neighbour of x. The $\rightarrow_{\forall_+}$-rule guarantees $\forall R.D \in \mathbf{L}(y)$.
  2. $\mathbf{L}(x, z) = R$, y blocks z. Then, by the $\rightarrow_{\forall_+}$-rule, we have $\forall R.D \in \mathbf{B}(z)$ and, by the definition of blocking, $\mathbf{B}(z) \subseteq \mathbf{L}(y)$. Hence $\forall R.D \in \mathbf{L}(y)$.
  3. $\mathbf{L}(y, z) = \mathsf{Inv}(R)$, x blocks z. From the definition of blocking, we have that $\mathbf{L}(z)/R = \mathbf{L}(x)/R$. Hence $\forall R.D \in \mathbf{L}(z)$ and the $\rightarrow_{\forall_+}$-rule guarantees $\forall R.D \in \mathbf{L}(y)$.

- **(T8)** is satisfied because, for each $(x, y) \in \mathcal{E}(R)$, either:

  1. x is an R-neighbour of y, so y is an $\mathsf{Inv}(R)$-neighbour of x and $(y, x) \in \mathcal{E}(\mathsf{Inv}(R))$.
  2. $\mathbf{L}(x, z) = R$ and y blocks z, so $\mathbf{L}(x, z) = \mathsf{Inv}(\mathsf{Inv}(R))$ and $(y, x) \in \mathcal{E}(\mathsf{Inv}(R))$.
  3. $\mathbf{L}(y, z) = \mathsf{Inv}(R)$ and x blocks z, so $(y, x) \in \mathcal{E}(\mathsf{Inv}(R))$. ■

We have already mentioned that it is problematic to define a suitable notion of satisfiability for $\mathcal{SI}$-completion trees (as it would be required by Theorem 3.6) due to blocking. As one can see, the blocked nodes of the completion tree do not play a rôle when defining the tableau, so (hidden) inconsistencies in the labels of indirectly blocked nodes should not prevent a complete and clash-free tree from being satisfiable. On the other hand, due to dynamic blocking, blocked nodes may become unblocked during a run of the algorithm, in which case inconsistencies in these nodes may prevent the discovery of a complete and clash-free tree. Consequently, for the completeness proof, we require all nodes, also the blocked ones, to be free from inconsistencies. Since we have not found a way to uniformly combine these two different approaches into a single notion of satisfiability for completion trees, we give a proof for the correctness of the $\mathcal{SI}$-algorithm that does not rely on Theorem 3.6.

Lemma 6.8
Let C be an $\mathcal{SI}$-concept in NNF. If C has a tableau, then the expansion rules can be applied in such a way that the tableau algorithm yields a complete and clash-free completion tree for C.

Proof. Let $\mathcal{T} = (\mathcal{S}, \mathcal{L}, \mathcal{E})$ be a tableau for C. Using $\mathcal{T}$, we guide the application of the non-deterministic $\rightarrow_\sqcup$-rule such that the algorithm yields a completion tree $\mathbf{T}$ that is both complete and clash-free. The algorithm starts with the initial tree $\mathbf{T}$ consisting of a single node x_0, the root, with $\mathbf{B}(x_0) = \mathbf{L}(x_0) = \{D\}$.

$\mathcal{T}$ is a tableau, hence there is some $s_0 \in \mathcal{S}$ with $D \in \mathcal{L}(s_0)$. When applying the expansion rules to $\mathbf{T}$, the application of the non-deterministic $\rightarrow_\sqcup$-rule is guided by the labelling in the tableau $\mathcal{T}$. We will expand $\mathbf{T}$ in such a way that the following invariant holds: there exists a function π that maps the nodes of $\mathbf{T}$ to elements of $\mathcal{S}$ such that

$$\begin{array}{l} \mathbf{L}(x) \subseteq \mathcal{L}(\pi(x)) \text{ holds for all nodes } x \text{ of } \mathbf{T}, \text{ and} \\ \text{if } \mathbf{L}(x,y) = R \text{ then } (\pi(x), \pi(y)) \in \mathcal{E}(R) \text{ for all nodes } x, y \text{ in } \mathbf{T}. \end{array} \qquad (*)$$

Claim 6.9 If $(*)$ holds for a completion tree $\mathbf{T}$ and a rule is applicable to $\mathbf{T}$, then it can be applied in a way that maintains $(*)$.

We have to distinguish the different rules:

- If the $\rightarrow_\sqcap$**-rule** can be applied to x in $\mathbf{T}$ with $D = C_1 \sqcap C_2 \in \mathbf{L}(x) \subseteq \mathcal{L}(\pi(x))$, then C_1, C_2 are added to $\mathbf{L}(x)$. Since $\mathcal{T}$ is a tableau, $\{C_1, C_2\} \subseteq \mathcal{L}(\pi(x))$, and hence the $\rightarrow_\sqcap$-rule preserves $\mathbf{L}(x) \subseteq \mathcal{L}(\pi(x))$.

- If the $\rightarrow_\sqcup$**-rule** can be applied to x in $\mathbf{T}$ with $D = C_1 \sqcup C_2 \in \mathbf{L}(x) \subseteq \mathcal{L}(\pi(x))$, then there is an $E \in \{C_1, C_2\}$ such that $E \in \mathcal{L}(\pi(x))$, and the $\rightarrow_\sqcup$-rule can add E to $\mathbf{L}(x)$. Hence the $\rightarrow_\sqcup$-rule can be applied in a way that preserves $\mathbf{L}(x) \subseteq \mathcal{L}(\pi(x))$.

- If the $\rightarrow_\exists$**-rule** can be applied to x in $\mathbf{T}$ with $D = \exists R.E \in \mathbf{L}(x) \subseteq \mathcal{L}(\pi(x))$, then $D \in \mathcal{L}(\pi(x))$ and there is some $t \in \mathcal{S}$ with $(\pi(x), t) \in \mathcal{E}(R)$ and $E \in \mathbf{L}(t)$. The $\rightarrow_\exists$-rule creates a new successor y of x and we extend π by setting $\pi' := \pi[y \mapsto t]$, i.e., π' is the extension of π that maps y to t. It is easy to see that the extended completion tree together with the function π' satisfy $(*)$.

- If the $\rightarrow_\forall$**-rule** can be applied to x in $\mathbf{T}$ with $D = \forall R.E \in \mathbf{L}(x) \subseteq \mathcal{L}(\pi(x))$ and y is an R-neighbour of x, then $(\pi(x), \pi(y)) \in \mathcal{E}(R)$, and thus $E \in \mathbf{L}(\pi(y))$. The $\rightarrow_\forall$-rule adds E to $\mathbf{L}(y)$ and thus preserves $\mathbf{L}(x) \subseteq \mathcal{L}(\pi(x))$. The deletion of nodes can never violate $(*)$.

- If the $\rightarrow_{\forall_+}$**-rule** can be applied to x in $\mathbf{T}$ with $D = \forall R.E \in \mathbf{L}(x) \subseteq \mathcal{L}(\pi(x))$, $\mathsf{Trans}(R)$, and y an R-neighbour of x, then $(\pi(x), \pi(y)) \in \mathcal{E}(R)$, and thus $\forall R.E \in \mathcal{L}(\pi(y))$. The $\rightarrow_{\forall_+}$-rule adds $\forall R.E$ to $\mathbf{L}(y)$ and thus preserves $\mathbf{L}(y) \subseteq \mathcal{L}(\pi(y))$. The deletion of nodes can never violate $(*)$.

From this claim, the lemma can be derived as follows. It is obvious that the initial tree satisfies $(*)$: since $\mathcal{T}$ is a tableau for C, there exists an element $s_0 \in \mathcal{S}$ with $C \in \mathcal{L}(s_0)$ and hence the function π that maps x_0 to s_0 satisfies the required properties. The claim states that, whenever a rule is applicable, it can be applied in a way that preserves $(*)$. Obviously, no completion tree that satisfies $(*)$ contains a clash as this would contradict (T2). Moreover, from Lemma 6.6, we have that the expansion process terminates and thus must eventually yield a complete and clash-free completion tree. ■

Theorem 6.10
The $\mathcal{SI}$-algorithm is a non-deterministic decision procedure for satisfiability and subsumption of $\mathcal{SI}$-concepts.

Proof. Theorem 6.10 is an immediate consequence of Lemma 6.3, 6.6, 6.7, and 6.8. Moreover, since $\mathcal{SI}$ is closed under negation, subsumption of concepts $C \sqsubseteq D$ can be reduced to the (un-)satisfiability of $C \sqcap \neg D$. ■

6.1.5 Complexity

In Lemma 6.6 we have seen that the depths of a completion tree generated by the $\mathcal{SI}$-algorithm is bounded exponentially in the size of the input concept. To show that the algorithm can indeed be implemented to run in polynomial space, we need to carry out a closer analysis of the length of paths in a completion tree.

In Lemma 6.11 and 6.12, we establish a polynomial bound on the length of paths in the completion tree in a manner similar to that used for the modal logic $\mathsf{S4}$ and $\mathcal{ALC}_{R^+}$ in (Halpern & Moses, 1992; Sattler, 1996a). It then remains to show that such a tree can be constructed using only polynomial space.

Lemma 6.11
Let C be an $\mathcal{SI}$-concept and $m = \sharp sub(C)$, $n > m^3$, and $R \in \overline{\mathsf{NR}}_C$ be a role with $\mathsf{Trans}(R)$. Let $x_1, \ldots, x_n$ be successive nodes of a completion tree generated for C by the $\mathcal{SI}$-algorithm with $\mathbf{L}(x_i, x_{i+1}) = R$ for $1 \leq i < n$. If the $\rightarrow_\forall$- or the $\rightarrow_{\forall_+}$-rules cannot be applied to these nodes, then there is a blocked node x_i among them.

Proof. For each node x of the completion tree, $\mathbf{B}(x)$ only contains two kinds of concepts: the concept that triggered the generation of the node x, denoted by C_x, and concepts which were propagated *down* the completion tree by the first alternative of the $\rightarrow_\forall$- or $\rightarrow_{\forall_+}$-rules. Moreover, $\mathbf{B}(x) \subseteq \mathbf{L}(x)$ holds for any node in the completion tree.

Firstly, consider the elements of $\mathbf{B}(x_i)$ for $i \geq 1$. Let C_{x_i} denote the concept that caused the generation of the node x_i. Then $\mathbf{B}(x_i) - \{C_{x_i}\}$ contains only concepts which have been inserted using the $\rightarrow_\forall$- or the $\rightarrow_{\forall_+}$-rule. Let $D \in \mathbf{B}(x_i) - \{C_{x_i}\}$. Then either $\forall R.D \in \mathbf{L}(x_{i-1})$ and the $\rightarrow_{\forall_+}$-rule makes sure that $\forall R.D \in \mathbf{B}(x_i)$, or D is already of the from $\forall R.D'$ and has been inserted into $\mathbf{B}(x_i)$ by an application of the $\rightarrow_{\forall_+}$-rule to x_{i-1}. In both cases, it follows that the $\rightarrow_\forall$- or the $\rightarrow_{\forall_+}$-rule yield $D \in \mathbf{B}(x_{i+1})$. Hence we have

$$\mathbf{B}(x_i) - \{C_{x_i}\} \subseteq \mathbf{B}(x_{i+1}) \text{ for all } 1 \leq i < n,$$

and, since we have m choices for C_{x_i},

$$\sharp\{\mathbf{B}(x_i) \mid 1 \leq i \leq n\} \leq m^2.$$

Secondly, consider $\mathbf{L}(x_i)/\mathsf{Inv}(R)$. Again, the $\rightarrow_\forall$- and the $\rightarrow_{\forall_+}$-rules yield

$$\mathbf{L}(x_i)/\mathsf{Inv}(R) \subseteq \mathbf{L}(x_{i-1})/\mathsf{Inv}(R) \text{ for all } 1 < i \leq n,$$

which implies

$$\sharp\{\mathbf{L}(x_i)/\mathsf{Inv}(R) \mid 1 \leq i \leq n\} \leq m.$$

Summing up, within $m^3 + 1$ nodes, there must be at least two nodes x_j, x_k which satisfy

$$\mathbf{B}(x_j) = \mathbf{B}(x_k) \quad \text{and} \quad \mathbf{L}(x_j)/\mathsf{Inv}(R) = \mathbf{L}(x_k)/\mathsf{Inv}(R).$$

This implies that one of these nodes is blocked by the other. ∎

We will now use this lemma to give a polynomial bound on the length of paths in a completion tree generated by the completion rules.

Lemma 6.12
The paths of a completion tree generated by the $\mathcal{SI}$-algorithm for a concept C have a length of at most m^4 where $m = \sharp sub(C)$.

Proof. Let $\mathbf{T}$ be a completion tree generated for C by the $\mathcal{SI}$-algorithm. For every node x of $\mathbf{T}$ we define $\ell(x) = \max\{|D| \mid D \in \mathbf{L}(x)\}$. If x is a predecessor of y in $\mathbf{T}$, then this implies $\ell(x) \geq \ell(y)$. If not $\mathsf{Trans}(R)$ and $\mathbf{L}(x, y) = R$, then this implies $\ell(x) > \ell(y)$. Furthermore, for $R_1 \neq R_2$ (but possibly $R_1 = \mathsf{Inv}(R_2)$), $\mathbf{L}(x, y) = R_1$ and $\mathbf{L}(y, z) = R_2$ implies $\ell(x) > \ell(z)$.

The only way that the maximal length of concepts does not decrease is along a pure R-path with $\mathsf{Trans}(R)$. However, the $\rightarrow_\forall$- and the $\rightarrow_{\forall_+}$-rule must be applied before the $\rightarrow_\exists$-rule may generate a new successor. Together with Lemma 6.11, this guarantees that these pure R-paths have a length of at most m^3.

Summing up, we can have a path of length at most m^3 before decreasing the maximal length of the concept in the node labels (or blocking occurs), which can happen at most m times and thus yields an upper bound of m^4 on the length of paths in a completion tree. ∎

Note that the extra condition for the $\rightarrow_\exists$-rule, which delays its application until no other rules are applicable, is necessary to prevent the generation of paths of exponential length. Consider the following example for some R with $\mathsf{Trans}(R)$:

$$\begin{aligned} C &= \exists R.D \sqcap \forall R.(\exists R.D) \sqcap \forall R^{-1}.A_0 \\ D &= (\forall R^{-1}.A_1 \sqcup \forall R^{-1}.B_1) \sqcap \cdots \sqcap (\forall R^{-1}.A_n \sqcup \forall R^{-1}.B_n) \end{aligned}$$

When started with a root node x_0 labelled $\mathbf{B}(x_0) = \mathbf{L}(x_0) = \{C\}$, the tableau algorithm generates a successor node x_1 with

$$\mathbf{B}(x_1) = \{D, \exists R.D, \forall R.(\exists R.D)\}$$

which, in turn, is capable of generating a further successor x_2 with $\mathbf{B}(x_2) = \mathbf{B}(x_1)$. Without blocking, this would lead to an infinite path in the completion tree. Obviously, for x_1 and x_2, the first part of the blocking condition is satisfied since $\mathbf{B}(x_2) \subseteq \mathbf{B}(x_1)$. However, the second condition causes a problem since, in this example, we can generate 2^n different sets of universal restrictions along R^{-1} for each node. If we can apply the $\rightarrow_\exists$-rule freely, then the algorithm might generate all of these 2^n nodes to find out that (after finally applying the $\rightarrow_{\forall_+}$-rule that causes propagation of the concepts of the form $\forall R^{-1}.E$ upward in the tree) within the first $n+1$ nodes on this path there is a blocked one.

Lemma 6.13
The $\mathcal{SI}$-algorithm can be implemented in PSPACE.

Proof. Let C be the $\mathcal{SI}$-concept to be tested for satisfiability. We can assume C to be in NNF because every $\mathcal{SI}$-concept can be turned into NNF in linear time.

Let $m = \sharp sub(C)$. For each node x of the completion tree, the labels $\mathbf{L}(x)$ and $\mathbf{B}(x)$ can be stored using m bits for each set. Starting from the initial tree consisting of only a single node x_0 with $\mathbf{L}(x_0) = \mathbf{B}(x_0) = \{D\}$, the expansion rules, as given in Figure 6.4, are applied. If a clash is generated, then the algorithm fails and returns "C is unsatisfiable". Otherwise, the completion tree is generated in a depth-first way: the algorithm keeps track of exactly one path of the completion tree by memorizing, for each node x, which of the $\exists R.D$-concepts in $\mathbf{L}(x)$ successors have yet to be generated. This can be done using additional m bits for each node. The "deletion" of all successors in the $\rightarrow_\forall$- or the $\rightarrow_{\forall_+}$-rule of a node x is then simply realized by setting all these additional bits to "has yet to be generated". There are three possible results of an investigation of a child of x:

- A clash is detected. This stops the algorithm with "C is unsatisfiable".
- The $\rightarrow_\forall$- or the $\rightarrow_{\forall_+}$-rule leads to an increase of $\mathbf{L}(x)$. This causes reconsideration of all children of x, re-using the space used for former children of x.
- Neither of these first two cases happens. We can then forget about this subtree and start the investigation of another child of x. If all children have been investigated, we consider x's predecessor.

Proceeding like this, the algorithm can be implemented using $2m + m$ bits for each node, where the $2m$ bits are used to store the two labels of the node, while m bits are used to keep track of the successors already generated. Since we reuse the memory for the successors, we only have to store one path of the completion tree at a time. From Lemma 6.12, the length of this path is bounded by m^4. Summing up, we can test for the existence of a completion tree using $\mathcal{O}(m^5)$ bits.

Unfortunately, due to the $\rightarrow_\sqcup$-rule, the $\mathcal{SI}$-algorithm is a non-deterministic algorithm. However, Savitch's theorem (Savitch, 1970) tells us that there is a deterministic implementation of this algorithm using at most $\mathcal{O}(m^{10})$ bits, which is still a polynomial bound. ■

Since $\mathcal{ALC}$ is a fragment of $\mathcal{SI}$, satisfiability of $\mathcal{SI}$-concepts is PSPACE-hard, which yields:

Theorem 6.14
Satisfiability and subsumption of $\mathcal{SI}$-concepts is PSPACE*-complete.*

There is an immediate optimization of the algorithm which has been omitted for the sake of the clarity of the presentation. We have only disallowed the application of the $\rightarrow_\exists$-rule to a blocked node, which is sufficient to guarantee the termination of the algorithm. It is also possible to disallow the application of more rules to a blocked node without violating the soundness or the completeness of the algorithm, if the notion of blocking is slightly adapted. It then becomes necessary to distinguish directly and indirectly blocked nodes. More details can be found in (Horrocks & Sattler, 1998). The technique presented there will stop the expansion of a blocked node earlier during the runtime of the algorithm and hence will save some work.

6.2 Adding Role Hierarchies and Qualifying Number Restrictions: $\mathcal{SHIQ}$

In this section, we study aspects of reasoning with the DL $\mathcal{SHIQ}$, i.e., $\mathcal{SI}$ extended with qualifying number restrictions and role hierarchies. Qualifying number restrictions have already been introduced in Chapter 4 and require no further discussion. Role hierarchies (1997), which have already been present in early DL systems like BACK (Quantz & Kindermann, 1990) , allow to express inclusion relationships between roles. For example, role hierarchies can be used to state that the role `has_child` is a sub-role of `has_offspring`. This makes it possible to infer that the child of someone whose offsprings are all rich must also be rich.

The combination of role hierarchies with transitive roles is particularly interesting because it allows to capture various aspects of part-whole relations (Sattler, 2000). It is also interesting because it is sufficiently expressive to *internalise* general TBoxes (Baader, 1991; Schild, 1991; Baader et al., 1993), i.e., it allows for a reduction of concept satisfiability w.r.t. general TBoxes to pure concept satisfiability—always an indication for high expressive power of a Description Logic.

After defining syntax and semantics of $\mathcal{SHIQ}$, we show how internalisation of general axioms can be accomplished. We then determine the worst-case complexity of satisfiability for $\mathcal{SHIQ}$-concepts as ExpTime-complete even if numbers in the input are in binary coding. This is achieved by a reduction from $\mathcal{SHIQ}$ to $\mathcal{ALCQIb}$, where role conjunction and general TBoxes are used to simulate role hierarchies and transitive roles.

While this reduction helps to determine the exact worst-case complexity or the problem, one cannot expect to obtain an efficient algorithm from it. The reason for this is that it relies on the highly inefficient automata construction used to prove Theorem 4.38. To overcome this problem, we present a tableau algorithm that decides satisfiability of $\mathcal{SHIQ}$ concepts. In the worst case, this algorithm runs in 2-NExpTime. Yet, it is amenable to optimizations and is the basis of the highly optimized DL system iFaCT (Horrocks, 1999), a offspring of the FaCT system (Horrocks, 1998), which exhibits good performance in system comparisons (Massacci & Donini, 2000; Horrocks, 2000).

6.2.1 Syntax and Semantics

Definition 6.15 (Syntax and Semantics of $\mathcal{SHIQ}$)
Let NR *be a set of atomic* role names, *and* $\mathsf{NR}^+ \subseteq \mathsf{NR}$ *a set of* transitive role names. *The set of $\mathcal{SHIQ}$-roles is defined as the set of $\mathcal{SI}$-roles by* $\overline{\mathsf{NR}} := \mathsf{NR} \cup \{R^{-1} \mid R \in \mathsf{NR}\}$. *A role* $R \in \overline{\mathsf{NR}}$ *is called* transitive *iff* $R \in \mathsf{NR}^+$ *or* $\mathsf{Inv}(R) \in \mathsf{NR}^+$.

A role inclusion axiom *is of the form* $R \sqsubseteq S$, *for two $\mathcal{SHIQ}$-roles R and S. A set of role inclusion axioms $\mathcal{R}$ is called* a role hierarchy. *We define* $\sqsubseteq^*$ *as the transitive-reflexive of the relation* $\mathcal{R} \cup \{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. *If* $R \sqsubseteq^* S$ *then R is called* a sub-role *of S and S is called* a super-role *of R (w.r.t. $\mathcal{R}$).*

A role R is called simple *with respect to $\mathcal{R}$ iff R does not have a transitive sub-role.*

Let NC *be a set of* concept names. *The set of $\mathcal{SHIQ}$*-concepts *is built inductively from these using the following grammar, where* $A \in \mathsf{NC}$, $n \in \mathbb{N}$, $R \in \overline{\mathsf{NR}}$ *is an arbitrary role, and* $S \in \overline{\mathsf{NR}}$ *is a simple role:*

$$C ::= A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall R.C \mid \exists R.C \mid ({\geq} n\ S\ C) \mid ({\leq} n\ S\ C).$$

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *consists of a non-empty set* $\Delta^{\mathcal{I}}$ *and a valuation* $\cdot^{\mathcal{I}}$ *that maps every concept name A to a subset* $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ *and every role name R to a binary relation* $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ *with the additional property that every transitive role name* $R \in \mathsf{NR}^+$ *is interpreted by a transitive relation.*

Such an interpretation is inductively extended to arbitrary $\mathcal{SHIQ}$-concepts in the usual way. (See Definition 4.17).

An interpretation $\mathcal{I}$ satisfies a role hierarchy *$\mathcal{R}$ iff* $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ *for each* $R \sqsubseteq S \in \mathcal{R}$*; we denote this fact by* $\mathcal{I} \models \mathcal{R}$ *and say that $\mathcal{I}$ is a model of $\mathcal{R}$.*

A concept C is satisfiable *with respect to a role hierarchy $\mathcal{R}$ iff there is some interpretation $\mathcal{I}$ such that* $\mathcal{I} \models \mathcal{R}$ *and* $C^{\mathcal{I}} \neq \emptyset$. *Such an interpretation is called* a model of C *w.r.t. $\mathcal{R}$. A concept D* subsumes *a concept C w.r.t. $\mathcal{R}$ (written* $C \sqsubseteq_{\mathcal{R}} D$*) iff* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ *holds for each model $\mathcal{I}$ of $\mathcal{R}$. For an interpretation $\mathcal{I}$, an individual* $x \in \Delta^{\mathcal{I}}$ *is called* an instance *of a concept C iff* $x \in C^{\mathcal{I}}$.

Satisfiability of concepts w.r.t. TBoxes and role hierarchies is defined in the usual way.

◇

As shown in (Horrocks et al., 1999), the restriction of qualifying number restrictions to simple roles is necessary to maintain decidability of $\mathcal{SHIQ}$. Without this restriction, it is possible to reduce an undecidable tiling problem (Berger, 1966) to concept satisfiability.

Internalization of TBoxes

An evidence of $\mathcal{SHIQ}$'s high expressivity is the fact that it allows for the *internalization* of TBoxes using a "universal" role U, that is, a transitive super-role of all relevant roles.

Lemma 6.16
Let C be a $\mathcal{SHIQ}$-concept, $\mathcal{R}$ a role hierarchy, and $\mathcal{T}$ a $\mathcal{SHIQ}$-TBox. Define

$$C_{\mathcal{T}} := \bigsqcap_{C_i \sqsubseteq D_i \in \mathcal{T}} \neg C_i \sqcup D_i,$$

and let $U \in \mathsf{NR}^+$ be a transitive role that does not occur in $\mathcal{T}, C$, or $\mathcal{R}$. We set

$$\mathcal{R}_U := \mathcal{R} \cup \{R \sqsubseteq U, \mathsf{Inv}(R) \sqsubseteq U \mid R \text{ occurs in } \mathcal{T}, C, \text{ or } \mathcal{R}\}.$$

Then C is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{R}$ iff $C \sqcap C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}}$ is satisfiable w.r.t. $\mathcal{R}_U$.

Note that augmenting $\mathcal{R}$ to obtain $\mathcal{R}_U$ in this manner does not turn simple roles into non-simple roles. The proof of Lemma 6.16 is similar to the ones that can be found in (Schild, 1991; Baader, 1991). Most importantly, it must be shown that, (a) if a $\mathcal{SHIQ}$-concept C is satisfiable with respect to a TBox $\mathcal{T}$ and a role hierarchy $\mathcal{R}$, then $C, \mathcal{T}$ have a *connected* model, i.e., a model where all elements are connected by roles occurring in C or $\mathcal{T}$, and (b) if y is reachable from x via a role path (possibly involving inverse roles), then $(x, y) \in U^{\mathcal{I}}$. These are easy consequences of the semantics of $\mathcal{SHIQ}$ and the definition of $\mathcal{R}_U$. As a corollary, we get:

Theorem 6.17
Satisfiability and subsumption of $\mathcal{SHIQ}$-concepts w.r.t. general TBoxes and role hierarchies are polynomially reducible to (un)satisfiability of $\mathcal{SHIQ}$-concepts w.r.t. role hierarchies.

Cycle-free Role Hierarchies

In what we have said so far, a role hierarchy $\mathcal{R}$ may contain a *cycle*, i.e., there may be roles $R, S \in \mathsf{NR}$ with $R \neq S$, $S \sqsubseteq^* R$, and $R \sqsubseteq^* S$. Such cycles would add extra difficulties to the following considerations. The next lemma shows that, w.l.o.g., we only need to consider role hierarchies that are *cycle-free*.

Lemma 6.18
Let C be a $\mathcal{SHIQ}$-concept and $\mathcal{R}$ a role hierarchy. There exists a $\mathcal{SHIQ}$-concept C' and role hierarchy $\mathcal{R}'$ polynomially computable from $C, \mathcal{R}$ such that $\mathcal{R}'$ is cycle free and C is satisfiable w.r.t. $\mathcal{R}$ iff C' is satisfiable w.r.t. $\mathcal{R}'$.

Proof. The set $\mathcal{R}$ can be viewed as a directed graph $G = (V, E)$ with vertices $V = \{R \mid R \text{ occurs in } \mathcal{R}\ \}$ and $E = \{(S, R) \mid S \sqsubseteq R \in \mathcal{R}\}$. The strongly connected components of G can be calculated in quadratic time. For every non-trivial strongly connected component $\{R_1, \dots R_k\}$, select an arbitrary $S \in \{R_1, \dots, R_k\}$ such that $S \in \mathsf{NR}^+$ if $\{R_1, \dots, R_k\} \cap \mathsf{NR}^+ \neq \emptyset$. For every $1 \leq i \leq k$, replace R_k in C and $\mathcal{R}$ by S. The results of this replacement are called C' and $\mathcal{R}'$, respectively. It is obvious that these can be obtained from $C, \mathcal{R}$ in polynomial time and that $(\mathcal{R}')^+$ is cycle-free.

For every $\mathcal{I}$ with $\mathcal{I} \models \mathcal{R}$, it easy to see that, for every strongly connected component $\{R_1, \dots, R_k\}$ of G, $R_i^{\mathcal{I}} = R_j^{\mathcal{I}}$ holds for every $1 \leq i, j \leq k$ and, if $\{R_1, \dots, R_k\} \cap \mathsf{NR}^+ \neq \emptyset$, then $R_i^{\mathcal{I}}$ is transitive for every $1 \leq i \leq k$. Hence, C is satisfiable w.r.t. $\mathcal{R}$ iff C' is satisfiable w.r.t. $\mathcal{R}'$ ∎

Thus, from now on, we only consider cycle-free role hierarchies.

6.2.2 The Complexity of Reasoning with $\mathcal{SHIQ}$

So far, the exact complexity of reasoning with $\mathcal{SHIQ}$ has been an open problem. It was clear that it is ExpTime-hard as a corollary of Theorem 3.18 and this also holds for pure concept satisfiability by Theorem 6.17. Following De Giacomo's ExpTime-completeness result for the DL $\mathcal{CIQ}$ (1995), it has been conjectured that the problem can be solved in ExpTime. Yet, the results from (De Giacomo, 1995) are valid for unary coding of numbers only and do not easily transfer to $\mathcal{SHIQ}$ because of the presence of role hierarchies. Here, we verify the conjecture by giving a polynomial reduction of $\mathcal{SHIQ}$-satisfiability to satisfiability of $\mathcal{ALCQIb}$-concepts w.r.t. general TBoxes. In Theorem 4.38, we have already shown that the latter problem can be solved in ExpTime, also for the case of binary coding of numbers in the input. Our reduction combines two techniques:

1. To deal with a role hierarchy $\mathcal{R}$, we replace every role R by the role conjunction
$$R^{\uparrow} = \bigsqcap_{R \sqsubseteq^* S} S.$$
Note that, since $R \sqsubseteq^* R$, R occurs in $R^{\uparrow}$. This usage of role conjunction to express role hierarchies is common knowledge in the DL community but, to the best of our knowledge, there exists no publication that explicitly mentions it.

2. For transitive roles, we shift the technique employed in the $\mathcal{SI}$-algorithm to deal with transitive roles into a set of TBox axioms. For $\mathcal{SI}$, transitive roles were dealt with by explicitly propagating assertions of the form $\forall R.D$ to all R-successors of a node x using the $\rightarrow_{\forall_+}$-rule. This makes it possible to turn an $\mathcal{SI}$-tableau into a model (which must interpret transitive roles with transitive relations) by transitively closing the role relations explicitly asserted in the tableau. Here, we achieve the same effect using a set of TBox axioms. A similar idea can be found in (de Nivelle, 2000).

The usage of $R^{\uparrow}$ to capture the role hierarchies is motivated by the following observations.

Lemma 6.19
Let $\mathcal{R}$ be a role hierarchy. If $S \sqsubseteq^* R$ then $(S^{\uparrow})^{\mathcal{I}} \subseteq (R^{\uparrow})^{\mathcal{I}}$ for every interpretation $\mathcal{I}$. Also, for every interpretation $\mathcal{I}$ with $\mathcal{I} \models \mathcal{R}$, $(R^{\uparrow})^{\mathcal{I}} = R^{\mathcal{I}}$

Proof. If $S \sqsubseteq^* R$ then $\{S' \mid S \sqsubseteq^* S'\} \supseteq \{S' \mid R \sqsubseteq^* S'\}$ and hence $(S^{\uparrow})^{\mathcal{I}} \subseteq (R^{\uparrow})^{\mathcal{I}}$. If $\mathcal{I} \models \mathcal{R}$ then $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ for every S with $R \sqsubseteq^* S$. Hence $(R^{\uparrow})^{\mathcal{I}} = R^{\mathcal{I}}$. ∎

The reduction that captures the transitive roles involves concepts from the following set:

Definition 6.20
For a $\mathcal{SHIQ}$ concept C and a role hierarchy $\mathcal{R}$, the set $clos(C, \mathcal{R})$ is the smallest set X of $\mathcal{SHIQ}$-concepts that satisfies the following conditions:

- $C \in X$,
- *X is closed under sub-concepts and $\sim$,[3] and*
- *if $\forall R.D \in X$, $T \sqsubseteq^* R$, and $\mathsf{Trans}(T)$, then $\forall T.D \in X$.* $\diamond$

It is easy to see that, for a concept C and a role hierarchy $\mathcal{R}$, the set $clos(C, \mathcal{R})$ is "small":

Lemma 6.21
For a $\mathcal{SHIQ}$-concept C in NNF and a role hierarchy $\mathcal{R}$, $\sharp clos(C, \mathcal{R}) = \mathcal{O}(|C| \times |\mathcal{R}|)$.

Proof. Like in the proof of Lemma 4.9, it is easy to see that the smallest set X' that contains C and is closed under sub-concepts and $\sim$ contains $\mathcal{O}(|C|)$ concepts. For $\mathcal{SHIQ}$, we additionally have to add concepts $\forall T.D$ to X' if $\forall R.D \in X'$ with $T \sqsubseteq^* R$ and $\mathsf{Trans}(T)$, and again close X' under sub-concepts and $\sim$ to obtain $clos(C, \mathcal{R})$.. This may yield at most two concepts for every concept in X' and every role T that occurs in $\mathcal{R}$ because, for every such concept $\forall R.D \in X'$, it suffices to add $\forall T.D$ and $\exists T.{\sim}D$. Since it is a sub-concept of $allRD$, D does not need to be reconsidered in the closure process. ∎

We now formally introduce the employed reduction from $\mathcal{SHIQ}$ to $\mathcal{ALCQI}b$.

Definition 6.22
Let C be a $\mathcal{SHIQ}$-concept in NNF and $\mathcal{R}$ a role hierarchy. For every concept $\forall R.D \in clos(C)$ let $X_{R,D} \in \mathsf{NC}$ a be unique concept name that does not occur in C. We define the function $\cdot^{tr}$ inductively on the structure of concepts (in NNF) by setting

$$
\begin{aligned}
A^{tr} &= A \textit{ for all } A \in \mathsf{NC}\\
(\neg A)^{tr} &= \neg A \textit{ for all } A \in \mathsf{NC}\\
(C_1 \sqcap C_2)^{tr} &= C_1^{tr} \sqcap C_2^{tr}\\
(C_1 \sqcup C_2)^{tr} &= C_1^{tr} \sqcup C_2^{tr}\\
({\bowtie} n\ R\ D)^{tr} &= ({\bowtie} n\ R^{\uparrow}\ D^{tr})\\
(\forall R.D)^{tr} &= X_{R,D}\\
(\exists R.D)^{tr} &= \neg X_{R,\sim D}
\end{aligned}
$$

[3]Like in Chapter 4, with $\sim D$ we denote NNF($\neg D$).

The TBox $\mathcal{T}_C$ is defined by

$$\mathcal{T}_C = \{X_{R,D} \doteq \forall R^{\uparrow}.D^{tr} \mid \forall R.D \in clos(C,\mathcal{R})\} \cup \\ \Big\{X_{R,D} \sqsubseteq \bigsqcap_{T \sqsubseteq^* R, \mathsf{Trans}(T)} \forall T^{\uparrow}.X_{T,D} \;\Big|\; \forall R.D \in clos(C,\mathcal{R})\Big\}$$

$\diamond$

Lemma 6.23
Let C be a $\mathcal{SHIQ}$-concept in NNF, $\mathcal{R}$ a role hierarchy, and $\cdot^{tr}$ and $\mathcal{T}_C$ as defined in Definition 6.22. C is satisfiable w.r.t. $\mathcal{R}$ iff the $\mathcal{ALCQIb}$-concept C^{tr} is satisfiable w.r.t. $\mathcal{T}_C$.

Proof. For the *only-if*-direction, let C be a $\mathcal{SHIQ}$-concept and $\mathcal{R}$ a set of role axioms. Assume that $\mathcal{I}$ is a ($\mathcal{SHIQ}$-)model of C w.r.t. $\mathcal{R}$. Let $\mathcal{X} = \{X_{R,D} \mid \forall R.D \in clos(C,\mathcal{R})\}$ bet the set of freshly introduced concept names from Definition 6.22.

We will construct an $\mathcal{ALCQIb}$-model $\mathcal{I}'$ for C^{tr} and $\mathcal{T}_C$ from $\mathcal{I}$ by setting

$$(X_{R,D})^{\mathcal{I}'} = (\forall R.D)^{\mathcal{I}}$$

for every $X_{R,D} \in \mathcal{X}$, and maintaining the interpretation of all other concept and role names.

Claim 6.24 For all $D \in clos(C,\mathcal{R})$, $(D^{tr})^{\mathcal{I}'} = D^{\mathcal{I}}$.

This claim is proved by induction on the structure of concepts. For the base case $D = A \in \mathsf{NC} \setminus \mathcal{X}$, $A^{tr} = A$ holds, and hence $(A^{tr})^{\mathcal{I}'} = A^{\mathcal{I}}$. For all other cases, except for $D = \forall R.E$ and $D = \exists R.E$, the claim follows immediately by induction because $R^{\mathcal{I}} = (R^{\uparrow})^{\mathcal{I}} = (R^{\uparrow})^{\mathcal{I}'}$ for every $R \in \mathsf{NR}$, since $\mathcal{I} \models \mathcal{R}$ and because of Lemma 6.19.

For $D = \forall R.E$, $D^{tr} = X_{R,E}$ and by construction of $\mathcal{I}'$, $(X_{R,E})^{\mathcal{I}'} = (\forall R.E)^{\mathcal{I}}$. For $D = \exists R.E$, $D^{tr} = \neg X_{R,\sim E}$ and

$$(D^{tr})^{\mathcal{I}'} = \Delta^{\mathcal{I}'} \setminus (X_{R,\sim E})^{\mathcal{I}'} = \Delta^{\mathcal{I}} \setminus (\forall R.{\sim}E)^{\mathcal{I}} = (\exists R.E)^{\mathcal{I}},$$

which finishes the proof of the claim.

In particular, since $C^{\mathcal{I}} \neq \emptyset$, also $(C^{tr})^{\mathcal{I}'} \neq \emptyset$. It remains to show that $\mathcal{I}' \models \mathcal{T}_C$ holds. For the first set of axioms, this holds because

$$(X_{R,D})^{\mathcal{I}'} = (\forall R.D)^{\mathcal{I}} = (\forall R^{\uparrow}.D^{tr})^{\mathcal{I}'},$$

since $R^{\mathcal{I}} = (R^{\uparrow})^{\mathcal{I}'}$ because of Lemma 6.19, and $D^{\mathcal{I}} = (D^{tr})^{\mathcal{I}'}$ due to Claim 6.24.

For the second set of axioms, let T be a role with $T \sqsubseteq^* R$ and $\mathsf{Trans}(T)$. Then

$$(X_{R,D})^{\mathcal{I}'} \subseteq (\forall T^{\uparrow}.X_{T,D})^{\mathcal{I}'},$$

unless there is an $x \in (X_{R,D})^{\mathcal{I}'}$ and an $y \in \Delta^{\mathcal{I}'}$ with $(x,y) \in (T^{\uparrow})^{\mathcal{I}'} = T^{\mathcal{I}}$ and $y \notin (X_{T,D})^{\mathcal{I}'} = (\forall T.D)^{\mathcal{I}}$. This implies the existence of an element $z \in \Delta^{\mathcal{I}}$ with $(y,z) \in T^{\mathcal{I}}$

and $z \notin D^{\mathcal{I}}$. Since $(x, y) \in T^{\mathcal{I}}$ and $(y, z) \in T^{\mathcal{I}}$, transitivity of $T^{\mathcal{I}}$ implies $(x, z) \in T^{\mathcal{I}} \subseteq R^{\mathcal{I}} = (R^{\uparrow})^{\mathcal{I}'}$. Thus $(x, z) \in (R^{\uparrow})^{\mathcal{I}'}$ and $z \notin (D^{tr})^{\mathcal{I}'}$ because $D^{\mathcal{I}} = (D^{tr})^{\mathcal{I}'}$. This implies $x \notin (\forall R^{\uparrow}.D^{tr})^{\mathcal{I}'}$, which is a contradiction because $(\forall R^{\uparrow}.D^{tr})^{\mathcal{I}'} = (\forall R.D)^{\mathcal{I}} = (X_{R,D})^{\mathcal{I}'}$ and $x \in (X_{R,D})^{\mathcal{I}'}$. Summing up, $(X_{R,D})^{\mathcal{I}'}$ is contained in the interpretation of every conjunct that appears on the right-hand side of the axioms, and hence

$$(X_{R,D})^{\mathcal{I}'} \subseteq \Big(\prod_{T \sqsubseteq^* R, \mathsf{Trans}(T)} \forall T^{\uparrow}.X_{T,D} \Big)^{\mathcal{I}'}.$$

This holds for every $X_{R,D} \in \mathcal{X}$ and hence $\mathcal{I}' \models \mathcal{T}_C$. Thus, we have shown that C^{tr} is satisfiable w.r.t. $\mathcal{T}_C$.

For the *if*-direction, let $\mathcal{I}$ be an interpretation with $(C^{tr})^{\mathcal{I}} \neq \emptyset$ and $\mathcal{I} \models \mathcal{T}_C$. From $\mathcal{I}$ we construct an interpretation $\mathcal{I}'$ such that $C^{\mathcal{I}'} \neq \emptyset$ and $\mathcal{I}' \models \mathcal{R}$. To achieve the latter, we define $R^{\mathcal{I}'}$ as follows:

$$R^{\mathcal{I}'} = \begin{cases} ((R^{\uparrow})^{\mathcal{I}})^+ & \text{if } \mathsf{Trans}(R), \\ (R^{\uparrow})^{\mathcal{I}} \cup \bigcup\limits_{S \sqsubseteq^* R, S \neq R} S^{\mathcal{I}'} & \text{otherwise.} \end{cases}$$

Since $\mathcal{R}$ is cycle-free, $R^{\mathcal{I}'}$ is well-defined for every R and it is obvious that, for every R with $\mathsf{Trans}(R)$, $R^{\mathcal{I}'}$ is transitive. First, we check that $\mathcal{I}'$ indeed satisfies $\mathcal{R}$.

Claim 6.25 $\mathcal{I}' \models S \sqsubseteq R$ for every $S \sqsubseteq^* R$.

If $\neg\mathsf{Trans}(R)$, this is immediate from the construction. If $\mathsf{Trans}(R)$, then the proof is more complicated. It is by induction on the number $\|S\| = \sharp\{S' \mid S' \sqsubseteq^* S, S' \neq S\}$, where the case for $\mathsf{Trans}(S)$ does not make use of the induction hypothesis.

- If $\|S\| = 0$ and $\neg\mathsf{Trans}(S)$, then $S^{\mathcal{I}'} = (S^{\uparrow})^{\mathcal{I}} \subseteq (R^{\uparrow})^{\mathcal{I}} \subseteq ((R^{\uparrow})^{\mathcal{I}})^+$ due to Lemma 6.19 because $S \sqsubseteq^* R$.
- If $\|S\| = n \geq 0$ and $\mathsf{Trans}(S)$, then $(S^{\uparrow})^{\mathcal{I}} \subseteq (R^{\uparrow})^{\mathcal{I}}$ since $S \sqsubseteq^* R$, and hence $S^{\mathcal{I}'} = ((S^{\uparrow})^{\mathcal{I}})^+ \subseteq ((R^{\uparrow})^{\mathcal{I}})^+ = R^{\mathcal{I}'}$.
- If $\|S\| = n > 0$ and $\neg\mathsf{Trans}(S)$, then $S^{\mathcal{I}'} = (S^{\uparrow})^{\mathcal{I}} \cup \bigcup_{S' \sqsubseteq^* S, S' \neq S} S'^{\mathcal{I}'}$. For every $S' \sqsubseteq^* S$ with $S' \neq S$, $\|S'\| < \|S\|$ because $\mathcal{R}$ is cycle-free. Since $S' \sqsubseteq^* R$ holds by the definition of $\sqsubseteq^*$, induction yields $(S')^{\mathcal{I}'} \subseteq R^{\mathcal{I}'}$. Also, since $S \sqsubseteq^* R$, $(S^{\uparrow})^{\mathcal{I}} \subseteq (R^{\uparrow})^{\mathcal{I}}$ and hence $S^{\mathcal{I}'} \subseteq R^{\mathcal{I}'}$.

Claim 6.26 If $(x, y) \in R^{\mathcal{I}'}$, then $(x, y) \in (R^{\uparrow})^{\mathcal{I}}$ or there exists a role $T \sqsubseteq^* R$ with $\mathsf{Trans}(T)$ and a path $x_0, \ldots, x_k$ such that $k > 1, x = x_0, y = x_k$, and $(x_i, x_{i+1}) \in (T^{\uparrow})^{\mathcal{I}}$ for $0 \leq i < k$.

Again, then proof is by induction on $\| \cdot \|$ and, if $\mathsf{Trans}(R)$ holds, then we do not need to make use of the induction hypothesis.

- If $\|R\| = 0$ and $\neg\mathsf{Trans}(R)$, then $R^{\mathcal{I}'} = (R^{\uparrow})^{\mathcal{I}}$ and thus $(x, y) \in R^{\mathcal{I}'}$ implies $(x, y) \in (R^{\uparrow})^{\mathcal{I}}$.

- If $\|R\| = n \geq 0$ and $\mathsf{Trans}(R)$, then $R^{\mathcal{I}'} = ((R^{\uparrow})^{\mathcal{I}})^{+}$. If $(x, y) \in (R^{\uparrow})^{\mathcal{I}}$, then we are done. Otherwise, there exists a path $x_0, \ldots, x_k$ with $k > 1, x = x_0, y = y_k$, and $(x_i, x_{i+1}) \in (R^{\uparrow})^{\mathcal{I}}$. Also, by definition, $R \sqsubseteq^* R$.

- If $\|R\| = n > 0$ and $\neg\mathsf{Trans}(R)$, then either $(x, y) \in (R^{\uparrow})^{\mathcal{I}}$ or $(x, y) \in S^{\mathcal{I}'}$ for some $S \sqsubseteq^* R$ with $S \neq R$. In the latter case, $\|S\| < \|R\|$ and, by induction, either $(x, y) \in (S^{\uparrow})^{\mathcal{I}} \subseteq (R^{\uparrow})^{\mathcal{I}}$, or there exists a role $T \sqsubseteq^* S$ with $\mathsf{Trans}(T)$ and a path $x_0, \ldots, x_k$ with $k > 1, x = x_0, y = x_k$, and $(x_i, x_{i+1}) \in (T^{\uparrow})^{\mathcal{I}}$ for $0 \leq i < k$. Since $T \sqsubseteq^* S \sqsubseteq^* R$, also $T \sqsubseteq^* R$.

Claim 6.27 For a simple role R, $R^{\mathcal{I}'} = (R^{\uparrow})^{\mathcal{I}}$.

The proof is by induction on the number of sub-roles of R. If R is simple and has no sub-roles, then $R^{\mathcal{I}'} = (R^{\uparrow})^{\mathcal{I}}$ holds by definition of $\mathcal{I}'$. If R is simple, then every role S with $S \sqsubseteq^* R$ and $S \neq R$ has less sub-roles than R because $\mathcal{R}$ is cycle free, and must be simple because otherwise R would not be simple. Hence, the induction hypothesis is applicable to each such S, which yields $S^{\mathcal{I}'} = (S^{\uparrow})^{\mathcal{I}}$. Also, since $S \sqsubseteq^* R$, $S^{\mathcal{I}'} = (S^{\uparrow})^{\mathcal{I}} \subseteq (R^{\uparrow})^{\mathcal{I}}$ holds by Lemma 6.19 and hence $R^{\mathcal{I}} = (R^{\uparrow})^{\mathcal{I}'}$.

Claim 6.28 $D^{\mathcal{I}'} = (D^{tr})^{\mathcal{I}}$ for every $D \in clos(C, \mathcal{R})$.

The proof is by induction on the value $\lceil \cdot \rceil$ of concepts in $clos(C, \mathcal{R})$, where the function $\lceil \cdot \rceil$ is defined by

$$\lceil D \rceil = \begin{cases} 2 \times \|D\| + 1 & \text{if } D = \exists R.E \\ 2 \times \|D\| & \text{otherwise} \end{cases}$$

where the definition of the norm $\| \cdot \|$ of a $\mathcal{SHIQ}$-concept is similar to the definition for $\mathcal{ALCQ}$ extended to universal and existential restrictions:

$$\begin{array}{llllll} \|A\| & := & \|\neg A\| & := & 0 & \text{for } A \in \mathsf{NC} \\ \|C_1 \sqcap C_2\| & := & \|C_1 \sqcup C_2\| & := & 1 + \|C_1\| + \|C_2\| & \\ \|\forall R.C\| & := & \|\exists R.C\| & := & 1 + \|C\| & \\ \|(\bowtie\ n\ S\ C)\| & & & := & 1 + \|C\| & \end{array}$$

The purpose of this (seemingly rather strange) definition of $\lceil \cdot \rceil$ is to reduce the case for an existential restriction to its dual universal restriction. Except for existential, universal, and number restrictions, all cases are straightforward.

- If $D = \forall R.E$, then $D^{tr} = X_{R,E}$. If $x \notin (X_{R,E})^{\mathcal{I}}$ then, since $\mathcal{I} \models X_{R,E} = \forall R^{\uparrow}.E^{tr}$, also $x \notin \forall R^{\uparrow}.E^{tr}$. By induction, $(E^{tr})^{\mathcal{I}} = E^{\mathcal{I}'}$ and $(R^{\uparrow})^{\mathcal{I}} \subseteq R^{\mathcal{I}'}$, and hence $x \notin (\forall R.E)^{\mathcal{I}'}$.

  If $x \in (X_{R,E})^{\mathcal{I}}$ and $(x, y) \in R^{\mathcal{I}'}$, then, by Claim 6.25, there are two possibilities.

  - If $(x, y) \in (R^{\uparrow})^{\mathcal{I}}$, then $y \in (E^{tr})^{\mathcal{I}}$ holds because $\mathcal{I} \models X_{R,E} = \forall R^{\uparrow}.E^{tr}$. By induction, $(E^{tr})^{\mathcal{I}} = E^{\mathcal{I}'}$, and hence $y \in E^{\mathcal{I}'}$.

– There is a role $T \sqsubseteq^* R$ with $\mathsf{Trans}(T)$ and a path $x_0, \ldots, x_k$ with $k > 1, x = x_0, y = x_k$, and $(x_i, x_{i+1}) \in (T^\uparrow)^\mathcal{I}$ for $0 \leq i < k$. Since $\mathcal{I} \models X_{R,E} \sqsubseteq \forall T^\uparrow.X_{T,E}$ and $\mathcal{I} \models X_{T,E} = \forall T^\uparrow.X_{T,E}$, we have $x_i \in (X_{T,E})^\mathcal{I}$ for every $1 \leq i \leq k$, and $x_{k-1} \in (X_{T,E})^\mathcal{I}$ in particular. Since $\mathcal{I} \models X_{T,E} \doteq \forall T^\uparrow.E^{tr}$, it follows that $y = x_k \in (E^{tr})^\mathcal{I}$ and, by induction, $y \in E^{\mathcal{I}'}$.

In any case, we have shown that $y \in E^{\mathcal{I}'}$ and, since y has been chosen arbitrarily with $(x, y) \in R^{\mathcal{I}'}$, $x \in (\forall R.E)^{\mathcal{I}'}$ holds.

- If $D = \exists R.E$, then $D^{tr} = \neg X_{R,\sim E}$ and

$$D^{\mathcal{I}'} = \Delta^{\mathcal{I}'} \setminus (\forall R.{\sim}E)^{\mathcal{I}'} = \Delta^\mathcal{I} \setminus (X_{R,\sim E})^\mathcal{I} = (\neg X_{R,\sim E})^\mathcal{I},$$

where $(\forall R.{\sim}E)^{\mathcal{I}'} = (X_{R,\sim E})^\mathcal{I}$ follows by induction since $\lceil \forall R.{\sim}E \rceil < \lceil \exists R.E \rceil$.

- If $D = (\bowtie n\ R\ E)$, then $D^{tr} = (\bowtie n\ R^\uparrow\ E^{tr})$ and R is simple. By Claim 6.27, $R^{\mathcal{I}'} = (R^\uparrow)^\mathcal{I}$ holds. Also, by induction, $(E^{tr})^\mathcal{I} = E^{\mathcal{I}'}$ and hence $D^{\mathcal{I}'} = (D^{tr})^\mathcal{I}$.

This finishes the proof of Claim 6.28, which yields $C^{\mathcal{I}'} = (C^{tr})^\mathcal{I} \neq \emptyset$. Since we have already shown that $\mathcal{I}' \models \mathcal{R}$, we have proved satisfiability of C w.r.t. $\mathcal{R}$. ∎

Since the reduction from Definition 6.22 is obviously polynomial in $|C|$ and $|\mathcal{R}|$, Lemma 6.23 together with Theorem 4.38 and Theorem 6.17 yield the following corollary.

Corollary 6.29
The following problems are ExpTime*-complete even in the case of binary coding of numbers in the input:*

- *Satisfiability and subsumption of $\mathcal{SHIQ}$-concepts w.r.t. role hierarchies.*
- *Satisfiability and subsumption of $\mathcal{SHIQ}$-concepts w.r.t. general TBoxes and role hierarchies.*

Obviously, the reduction from Definition 6.22 works also for $\mathcal{SHIQ}$-ABoxes and so, from Theorem 4.42, we get that also $\mathcal{SHIQ}$-knowledge bases can be handled in ExpTime.

Corollary 6.30
Knowledge base satisfiability and instance checking for $\mathcal{SHIQ}$ are ExpTime*-complete, even in the case of binary coding of numbers in the input.*

Finally, it is easy to see how to extend the reduction from Definition 6.22 to $\mathcal{SHIQO}$, the extension of $\mathcal{SHIQ}$ with nominals. Simply set $i^{tr} = i$ for every individual $i \in \mathsf{NI}$. Since $\mathcal{SHIQO}$ strictly contains $\mathcal{ALCQIO}$, we get the following.

Corollary 6.31
Concept satisfiability, satisfiability w.r.t. general TBoxes, and knowledge base satisfiability for $\mathcal{SHIQO}$ are NExpTime*-hard. The problems are* NExpTime*-complete if unary coding of numbers in the input is assumed.*

Proof. The lower bound is immediate from Corollary 5.27, since $\mathcal{ALCQIO}$ is strictly contained in $\mathcal{SHIQO}$. For the upper bound, the reduction from Definition 6.22 extended to $\mathcal{SHIQO}$ by setting $i^{tr} = i$ for every individual $i \in \mathsf{NI}$ yields a reduction from $\mathcal{SHIQO}$ to $\mathcal{ALCQIO}$, for which the corresponding problems are solvable in NExpTime if unary coding of numbers in the input is assumed. ∎

6.3 Practical Reasoning for $\mathcal{SHIQ}$

The previous ExpTime-completeness results for $\mathcal{SHIQ}$ rely on the highly inefficient automata construction of Definition 4.34 used to prove Theorem 4.38 and, in the case of knowledge base reasoning, also on the wasteful pre-completion technique used to prove Theorem 4.42. Thus, we cannot expect to obtain an implementation from these algorithms that exhibits acceptable runtimes even on relatively "easy" instances. This, of course, is a prerequisite for using $\mathcal{SHIQ}$ in real-world applications.

For less expressive DLs, some of the implementations of reasoners that perform fastest in system comparisons (Massacci & Donini, 2000) are based on tableau calculi similar to the ones we have already studied in this thesis. Among them are FaCT (Horrocks, 1998) for the DL $\mathcal{SHF}$, RACE (Haarslev & Möller, 1999) for $\mathcal{SHN}$, and DLP (Patel-Schneider, 2000) for an extension of $\mathcal{ALC}_{\text{reg}}$ with number restrictions. The efficiency of these implementations is due to a number of optimizations (Baader, Franconi, Hollunder, Nebel, & Profitlich, 1994; Horrocks, 1997; Horrocks & Patel-Schneider, 1999; Horrocks & Tobies, 2000; Haarslev & Möller, 2000c) for which tableau algorithms proved to be amenable.

To make these optimizations applicable and to allow for an easy extension of existing implementations to $\mathcal{SHIQ}$, we develop a tableau algorithm that decides concept satisfiability for $\mathcal{SHIQ}$. By Theorem 6.17, such an algorithm can also be used to decide concept satisfiability w.r.t. general TBoxes. This algorithm can be seen as the culmination point of the development of tableau-based decision procedures for more and more expressive DLs. To mention only the more recent ones: Sattler (1996a) describes an algorithm for $\mathcal{S}$ that is subsequently extended to deal with role hierarchies ($\mathcal{SH}$) by Horrocks (1998). Haarslev and Möller (2000a) add number restrictions ($\mathcal{SHN}$) while Horrocks and Sattler (1999) add inverse roles and functional restrictions ($\mathcal{SHIF}$). Here, we extend the latter algorithm to deal with qualifying number restrictions to obtain a tableau based decision procedure for $\mathcal{SHIQ}$.

Many techniques required for this extension are already present in the $\mathcal{SHIF}$-algorithm (Horrocks & Sattler, 1999) and in the $\mathcal{ALCQ}$-algorithm presented in Chapter 4. In addition to these techniques, we develop a novel way to construct a model from a completion tree to prove soundness of the $\mathcal{SHIQ}$-algorithm. This is necessary because $\mathcal{SHIQ}$ no longer has the finite model property.

6.3.1 A $\mathcal{SHIQ}$-Tableau

For the tableau algorithm, it will be helpful to have a syntactic satisfiability criterion for satisfiability that deals with the extra complexity caused by transitive roles, similar to the tableau for $\mathcal{SI}$ defined in Definition 6.2. The $\mathcal{SHIQ}$-algorithm will then search for $\mathcal{SHIQ}$-tableaux rather than for models. Like for $\mathcal{SI}$, elements of a $\mathcal{SHIQ}$-tableau are labelled with sets of "relevant" concepts. Due to the presence of qualifying number restrictions, not only the sub-concepts of the input concepts are of relevance but also their negations (in NNF). Also, propagation of universal restrictions along transitive roles is slightly more complicated in the presence of a role hierarchy compared to the case of $\mathcal{SI}$, and may involve universal restrictions that are not present as sub-concepts of the input concept. Hence, elements of the tableau are labelled not only with sub-concepts of the input concept but rather from the larger set $clos(C, \mathcal{R})$ that is defined in Definition 6.20.

Based on this set, the definition of a tableau for $\mathcal{SHIQ}$ is now similar to the one for $\mathcal{SI}$ in Definition 6.2.

Definition 6.32 (A Tableau for $\mathcal{SHIQ}$)
Let C be a $\mathcal{SHIQ}$-concept in NNF, $\mathcal{R}$ a role hierarchy, and $\overline{\mathsf{NR}}_{C,\mathcal{R}}$ the set of roles occurring in $C, \mathcal{R}$, together with their inverses. A tableau *$\mathcal{T}$ for C w.r.t. $\mathcal{R}$ is a triple $(\mathcal{S}, \mathcal{L}, \mathcal{E})$ such that $\mathcal{S}$ is a non-empty set, $\mathcal{L} : \mathcal{S} \to 2^{clos(C,\mathcal{R})}$ maps each element to a subset of $clos(C, \mathcal{R})$, $\mathcal{E} : \overline{\mathsf{NR}}_{C,\mathcal{R}} \to 2^{\mathcal{S}\times\mathcal{S}}$ maps each role in $\overline{\mathsf{NR}}_{C,\mathcal{R}}$ to a set of pairs of individuals, and the following conditions are satisfied:*

(T1) There is an $s \in \mathcal{S}$ with $C \in \mathcal{L}(s)$, and

for all $s, t \in \mathcal{S}$, $A, C_1, C_2, D \in clos(C, \mathcal{R})$, and $R, S \in \overline{\mathsf{NR}}_{C,\mathcal{R}}$,

(T2) if $A \in \mathcal{L}(s)$, then $\neg A \notin \mathcal{L}(s)$ for $A \in \mathsf{NC}$,

(T3) if $C_1 \sqcap C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ and $C_2 \in \mathcal{L}(s)$,

(T4) if $C_1 \sqcup C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ or $C_2 \in \mathcal{L}(s)$,

(T5) if $\exists R.D \in \mathcal{L}(s)$, then there is some $t \in \mathcal{S}$ such that $(s, t) \in \mathcal{E}(R)$ and $D \in \mathcal{L}(t)$,

(T6) if $\forall R.D \in \mathcal{L}(s)$ and $(s, t) \in \mathcal{E}(R)$, then $D \in \mathcal{L}(t)$,

(T7) if $\forall R.D \in \mathcal{L}(s)$, $(s, t) \in \mathcal{E}(T)$ for some $T \sqsubseteq^* R$ with $\mathsf{Trans}(T)$, then $\forall T.D \in \mathcal{L}(t)$,

(T8) $(s, t) \in \mathcal{E}(R)$ iff $(t, s) \in \mathcal{E}(\mathsf{Inv}(R))$.

(T9) if $(s, t) \in \mathcal{E}(S)$ and $S \sqsubseteq^* R$, then $(s, t) \in \mathcal{E}(R)$,

(T10) if $(\geq n\ S\ D) \in \mathcal{L}(s)$, then $\sharp S^{\mathcal{T}}(s, D) \geq n$,

(T11) if $(\leq n\ S\ D) \in \mathcal{L}(s)$, then $\sharp S^{\mathcal{T}}(s, D) \leq n$,

(T12) if $(\bowtie n\ S\ D) \in \mathcal{L}(s)$ and $(s, t) \in \mathcal{E}(S)$, then $D \in \mathcal{L}(t)$ or $\sim D \in \mathcal{L}(t)$,

where we use $\bowtie$ as a placeholder for both $\leq$ and $\geq$ and we define

$$S^{\mathcal{T}}(s, D) := \{t \in \mathcal{S} \mid (s,t) \in \mathcal{E}(S) \text{ and } D \in \mathcal{L}(t)\}.$$

The existence of a tableau is a necessary and sufficient criterion for satisfiability:

Lemma 6.33
A $\mathcal{SHIQ}$-concept C is satisfiable with respect to a role hierarchy $\mathcal{R}$ iff there exists a tableau for C with respect to $\mathcal{R}$.

Proof. For the *if*-direction, the construction of a model of C from a tableau for C is similar to the one presented in the proof of Lemma 6.3 where the interpretation of the roles is defined in the same manner as it was done in the proof of Lemma 6.23. To be more precise, if $\mathcal{T} = (\mathcal{S}, \mathcal{L}, \mathcal{E})$ is a tableau for C w.r.t. $\mathcal{R}$ and $C \in \mathcal{L}(s_0)$, a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of C can be defined as follows:

$$\begin{array}{rcl}
\Delta^{\mathcal{I}} & = & \mathcal{S} \\
A^{\mathcal{I}} & = & \{s \mid A \in \mathcal{L}(s)\} \text{ for all concept names } A \text{ in } \mathit{clos}(C, \mathcal{R}) \\
R^{\mathcal{I}} & = & \begin{cases} \mathcal{E}(R)^+ & \text{if } \mathsf{Trans}(R) \\ \mathcal{E}(R) \cup \bigcup\limits_{S \sqsubseteq^* R, S \neq R} S^{\mathcal{I}} & \text{otherwise} \end{cases}
\end{array}$$

Like in the proof of Lemma 6.23, it is easy to see that $\mathcal{I} \models \mathcal{R}$ and that $(s,t) \in R^{\mathcal{I}}$ iff $(s,t) \in \mathcal{E}(R)$ or there exists a role $T \sqsubseteq^* R$ with $\mathsf{Trans}(T)$ and a path $s_0, \ldots, s_k$ such that $k > 1$, $s = s_0$, $t = s_k$, and $(s_i, s_{i+1}) \in \mathcal{E}(T)$ for $0 \leq i < k$. Moreover, if R is simple, then $R^{\mathcal{I}} = \mathcal{E}(R)$.

It remains to show that $C^{\mathcal{I}} \neq \emptyset$. This is done by proving that $D \in \mathcal{L}(s)$ implies $s \in D^{\mathcal{I}}$ for each $D \in \mathit{clos}(C, \mathcal{R})$ and $s \in \mathcal{S}$. Since $C \in \mathcal{L}(s_0)$, we then have $s_0 \in C^{\mathcal{I}}$ and hence $\mathcal{I}$ is a model of C. The proof is by induction on the *norm* $\| \cdot \|$ of concepts as defined in the proof of Lemma 6.23. The two base cases of the induction are $D = A$ or $D = \neg A$ for $A \in \mathsf{NC}$. If $A \in \mathcal{L}(s)$, then, by definition, $s \in A^{\mathcal{I}}$. If $\neg A \in \mathcal{L}(s)$, then, by (T2), $A \notin \mathcal{L}(s)$ and hence $s \notin A^{\mathcal{I}}$. For the induction step, we have to distinguish several cases:

- The cases $D = C_1 \sqcap C_2$, $D = C_1 \sqcup C_2$, and $D = \exists R.E$ are exactly as for $\mathcal{SI}$ in the proof of Lemma 6.3
- $D = \forall R.E$. Let $s \in \mathcal{S}$ with $D \in \mathcal{L}(s)$, let $t \in \mathcal{S}$ be an arbitrary individual such that $(s,t) \in R^{\mathcal{I}}$. There are two possibilities:
  - $(s,t) \in \mathcal{E}(R)$. Then (T6) implies $E \in \mathcal{L}(t)$ and, by induction, $t \in E^{\mathcal{I}}$.
  - $(s,t) \notin \mathcal{E}(R)$. Due to (T8), this can only be the case if there is a role $T \sqsubseteq^* R$ with $\mathsf{Trans}(T)$ and a path $(s, s_1), (s_1, s_2), \ldots, (s_{k-1}, t) \in \mathcal{E}(T)$ with $k > 1$. Then (T7) implies $\forall T.E \in \mathcal{L}(s_i)$ for all $1 \leq i \leq k-1$ and particularly $\forall T.E \in \mathcal{L}(s_{k-1})$. Due to (T6), $E \in \mathcal{L}(t)$ also holds. Again, by induction, this implies $t \in E^{\mathcal{I}}$.

In both cases, we have $t \in E^{\mathcal{I}}$ and, since t has been chosen arbitrarily, $s \in D^{\mathcal{I}}$ holds.

- $D = (\geq n\ S\ E)$. For an s with $D \in \mathcal{L}(s)$, we have $\sharp S^{\mathcal{T}}(s, E) \geq n$ by (T10). Hence there are n individuals $t_1, \ldots, t_n$ such that $t_i \neq t_j$ for $i \neq j$, $(s, t_i) \in \mathcal{E}(S)$, and $E \in \mathcal{L}(t_i)$ for all i. By induction, we have $t_i \in E^{\mathcal{I}}$ and, since $\mathcal{E}(S) \subseteq S^{\mathcal{I}}$, also $s \in D^{\mathcal{I}}$.

- $D = (\leq n\ S\ E)$. For this case, it is crucial that S is a simple role because this implies $S^{\mathcal{I}} = \mathcal{E}(S)$. Let s be an individual with $D \in \mathcal{L}(s)$. Due to (T12), we have $E \in \mathcal{L}(t)$ or ${\sim}E \in \mathcal{L}(t)$ for each t with $(s, t) \in \mathcal{E}(S)$. Moreover, $\sharp S^{\mathcal{T}}(s, E) \leq n$ holds due to (T11). We show that $\sharp S^{\mathcal{I}}(s, E) \leq \sharp S^{\mathcal{T}}(s, E)$: assume $\sharp S^{\mathcal{I}}(s, E) > \sharp S^{\mathcal{T}}(s, E)$. This implies the existence of some t with $(s, t) \in S^{\mathcal{I}}$ and $t \in E^{\mathcal{I}}$ but $E \notin \mathcal{L}(t)$ (because $S^{\mathcal{I}} = \mathcal{E}(S)$). By (T12), this implies ${\sim}E \in \mathcal{L}(t)$, which, by induction, yields $t \in ({\sim}E)^{\mathcal{I}}$, in contradiction to $t \in E^{\mathcal{I}}$.

For the *only-if*-direction, we have to show that satisfiability of C w.r.t. $\mathcal{R}$ implies the existence of a tableau T for C w.r.t. $\mathcal{R}$.

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of C with $\mathcal{I} \models \mathcal{R}$. A tableau $\mathcal{T} = (\mathcal{S}, \mathcal{L}, \mathcal{E})$ for C can be defined by:

$$\begin{aligned} \mathcal{S} &= \Delta^{\mathcal{I}}, \\ \mathcal{E}(R) &= R^{\mathcal{I}}, \\ \mathcal{L}(s) &= \{D \in \mathit{clos}(C, \mathcal{R}) \mid s \in D^{\mathcal{I}}\}. \end{aligned}$$

It remains to demonstrate that $\mathcal{T}$ is a tableau for D:

- Except for (T7) and (T9), all conditions are satisfied as a direct consequence of the definition of the semantics of $\mathcal{SHIQ}$-concepts.

- For (T7), if $s \in (\forall R.D)^{\mathcal{I}}$ and $(s, t) \in T^{\mathcal{I}}$ for T with $\mathsf{Trans}(T)$ and $T \sqsubseteq^* R$, then $t \in (\forall T.D)^{\mathcal{I}}$ unless there is some u such that $(t, u) \in T^{\mathcal{I}}$ and $u \notin D^{\mathcal{I}}$. In this case, since $(s, t) \in T^{\mathcal{I}}$, $(t, u) \in T^{\mathcal{I}}$, and $\mathsf{Trans}(T)$, it holds that $(s, u) \in T^{\mathcal{I}}$. Hence $(s, u) \in R^{\mathcal{I}}$ and $s \notin (\forall R.D)^{\mathcal{I}}$—in contradiction to the assumption. $\mathcal{T}$ therefore satisfies (T7).

- Condition (T9) is satisfied because $\mathcal{I} \models \mathcal{R}$ and set-inclusion is a transitive property.

■

6.3.2 A Tableau Algorithm for $\mathcal{SHIQ}$

In the following, we present an algorithm that, given a $\mathcal{SHIQ}$-concept C and a role hierarchy $\mathcal{R}$, decides the existence of a tableau for C w.r.t. $\mathcal{R}$. As before, we assume that $\mathcal{R}$ is cycle-free. Like the $\mathcal{SI}$-algorithm, the $\mathcal{SHIQ}$-algorithm works on a finite completion tree, and employs a blocking technique to guarantee termination.

Figure 6.5 A tableau where pair-wise blocking is crucial

$$
\begin{array}{ll}
x & \mathbf{L}(x) = \{\neg A, (\leq 1\ F), \exists F^{-}.D, \forall R^{-}.(\exists F^{-}.D)\} \\
& \downarrow F^{-} \\
y & \mathbf{L}(y) = \{D, \exists F^{-}.D, \forall R^{-}.(\exists F^{-}.D), A, (\leq 1\ F), \exists F.\neg A\} \\
& \downarrow F^{-} \\
z & \mathbf{L}(z) = \{D, \exists F^{-}.D, \forall R^{-}.(\exists F^{-}.D), A, (\leq 1\ F), \exists F.\neg A\}
\end{array}
$$

Pair-wise Blocking

From the fact that $\mathcal{SHIQ}$ no longer has the finite model property, it is immediately clear that the tableau construction we have employed for $\mathcal{SI}$ will not work without modification for $\mathcal{SHIQ}$, as this technique always resulted in finite tableaux and hence in finite models.

Horrocks and Sattler (1999) show that for the fragment $\mathcal{SHIF}$ of $\mathcal{SHIQ}$, dynamic blocking no longer is sufficient and describe the *pair-wise blocking* technique that can successfully be applied also for $\mathcal{SHIQ}$: if a path contains two pairs of successive nodes that have pair-wise identical labels and whose connecting edges have identical labels, then the path beyond the second pair is no longer expanded—it is blocked. Blocked paths are then "unraveled" to construct an infinite tableau from a finite completion tree. The identical labels make sure that copies of the blocking node and its descendants can be substituted for the blocked node and its respective descendants. Note the similarity between this pair-wise blocking condition and the condition imposed by the combination of the blocking condition and the cut rule by De Giacomo and Massacci (2001) for CPDL.

Figure 6.5 shows that pair-wise blocking is crucial in order to ensure that the algorithm discovers the unsatisfiability of the concept

$$\neg A \sqcap (\leq 1\ F) \sqcap \exists F^{-1}.D \sqcap \forall R^{-1}.(\exists F^{-1}.D),$$

where $\mathsf{Trans}(R)$, $F \sqsubseteq R$, and D represents the concept

$$A \sqcap (\leq 1\ F) \sqcap \exists F.\neg A.$$

Using dynamic blocking, z would be blocked by y. The resulting tree cannot represent a cyclic model in which y is related to itself by an F^{-1} role as this would conflict with $(\leq 1\ F) \in \mathbf{L}(y)$. The tree must therefore represent the infinite model generated by recursively replacing each occurrence of z with a copy of the tree rooted at y. However, this also does not lead to a valid model, since, if z is substituted by a copy of y, then the constraint $\exists F.\neg A \in \mathbf{L}(y)$, which was satisfied because of $\neg A \in \mathbf{L}(x)$, is no longer satisfied in its new location.

When pair-wise blocking is used, z is no longer blocked by y as the labels of their predecessors (y and x respectively) are not equal, and the algorithm continues to expand $\mathbf{L}(z)$. The expansion of $\exists F.\neg A \in \mathbf{L}(z)$ calls for the existence of a node whose label includes

$\neg A$ and that is connected to z by an F-labelled edge. Because of $(\leq 1\ F) \in \mathbf{L}(z)$, this node must be y, and this results in a contradiction as both A and $\neg A$ will be in $\mathbf{L}(y)$.

To extend the $\mathcal{SHIF}$-algorithm from (Horrocks & Sattler, 1999) to $\mathcal{SHIQ}$, we add rules that deal with qualifying number restrictions, similar to the ones used by the standard algorithm for $\mathcal{ALCQ}$ (Algorithm 4.4), namely a $\rightarrow_{\geq}$-rule that introduces new successor nodes to satisfy $\geq$-restrictions, a $\rightarrow_{\leq}$-rule that identifies nodes as required by $\leq$-restrictions, and a $\rightarrow_{\text{choose}}$-rule that makes sure that all relevant concepts at a node are either positively or negatively asserted for that node (refer to (T12)).

In order to guarantee the termination of the algorithm, we have to make sure that the $\rightarrow_{\geq}$- and $\rightarrow_{\leq}$-rules cannot be applied in a way that would yield an infinite sequence of rule applications, generating and identifying successors indefinitely. This is enforced by recording in a relation "$\neq$" which nodes have been introduced by an application of the $\rightarrow_{\geq}$-rule and by prohibiting the identification of these nodes by the $\rightarrow_{\leq}$-rule.

Algorithm 6.34 (The $\mathcal{SHIQ}$-algorithm)
Let C be a $\mathcal{SHIQ}$-concept in NNF to be tested for satisfiability w.r.t. a role hierarchy $\mathcal{R}$ and $\overline{\mathsf{NR}}_{C,\mathcal{R}}$ the set of roles that occur in C and $\mathcal{R}$ together with their inverses. A completion tree $\mathbf{T} = (\mathbf{V}, \mathbf{E}, \mathbf{L})$ *is a labelled tree in which each node $x \in \mathbf{V}$ is labelled with a set $\mathbf{L}(x) \subseteq clos(C, \mathcal{R})$ and each edge $(x, y) \in E$ is labelled with a set $\mathbf{L}(x, y) \subset \overline{\mathsf{NR}}_{C,\mathcal{R}}$. The algorithm expands the tree by extending $\mathbf{L}(x)$ for some node x, or by adding new leaf nodes. Additionally, we keep track of inequalities between nodes of the tree with a symmetric binary relation $\neq$ between nodes in $\mathbf{V}$.*

*Given a completion tree, a node y is called an R-*successor *of a node x if y is a successor of x and $S \in \mathbf{L}(x, y)$ for some S with $S \sqsubseteq^* R$; y is called an R-*neighbour *of x if it is an R-successor of x, or if x is an $\mathsf{Inv}(R)$-successor of y.*

For a role R, a concept D, and a node $x \in \mathbf{V}$, we define $R^{\mathbf{T}}(x, D)$ by

$$R^{\mathbf{T}}(x, D) = \{y \mid y \text{ is an } R\text{-neighbour of } x \text{ and } D \in \mathbf{L}(y)\}.$$

A node x is directly blocked *if none of its ancestors is blocked, and it has ancestors x', y, and y', such that*

1. *x is a successor of x' and y is a successor of y', and*

2. *$\mathbf{L}(x) = \mathbf{L}(y)$ and $\mathbf{L}(x') = \mathbf{L}(y')$* and

3. *$\mathbf{L}(x', x) = \mathbf{L}(y', y)$.*

In this case we will say that y blocks x.

A node is indirectly blocked *if its predecessor is directly or indirectly blocked, and in order to avoid wasted expansion after an application of the $\rightarrow_{\leq}$-rule, a node y will also be taken to be indirectly blocked if y is a successor of a node x and $\mathbf{L}(x, y) = \emptyset$.*

For a node x, $\mathbf{L}(x)$ contains a clash *if, for some concept name $A \in \mathsf{NC}$, $\{A, \neg A\} \subseteq \mathbf{L}(x)$, or if, for a some concept D, some role S, and some $n \in \mathbb{N}$: $(\leq n\ S\ D) \in \mathbf{L}(x)$ and there* *are $n+1$ nodes $y_0, \ldots, y_n$ such that $D \in \mathbf{L}(y_i)$, y_i is an S-neighbour of x, and $y_i \neq y_j$ for all $0 \leq i < j \leq n$.*

The algorithm initializes the tree $\mathbf{T}$ to contain a single node x_0, called the root *node, with $\mathbf{L}(x_0) = \{C\}$. The inequality relation $\neq$ is initialized with the empty set. $\mathbf{T}$ is then expanded by repeatedly applying the rules from Figure 6.6.*

The completion tree is complete if, for some node x, $\mathbf{L}(x)$ contains a clash, or if none of the rules is applicable. If, for an input concept C, the expansion rules can be applied in such a way that they yield a complete, clash-free completion tree, then the algorithm returns "C is satisfiable", and "C is unsatisfiable" otherwise. $\diamond$

For a discussion of the different kinds on non-determinism present in the $\mathcal{SHIQ}$-algorithm, compare below Algorithm 6.4.

Like for $\mathcal{SI}$, the definition of *successor* and *predecessor* reflects the relative position of two nodes in the completion tree: if x is an R-successor of y then this implies that $(x, y) \in \mathbf{E}$ and it is not the case that y is an $\mathsf{Inv}(R)$-successor of x. It is necessary to make this pedantic distinction because when we construct a tableau from a complete and clash-free completion tree in the proof of Lemma 6.38, a blocked successor is replaced by a copy of the sub-completion tree consisting of the respective blocking node and its descendants. This makes the distinction between an R-successors and an $\mathsf{Inv}(R)$- predecessors significant has has to be reflected in the completion rules.

Note that the definition of blocking is recursive because the status of a node depends, among other things, on the status of its predecessor. Since the dependency is on the predecessor and the ancestors, one can determine the status every node starting at the root, which has no predecessor or ancestor and hence is never blocked. Once the blocking status of a node has been determined, one can then determine the status of its successors.

Since we only block along a path in the completion tree, for every directly blocked node there is a uniquely determined blocking node. Assume there would be a directly blocked node x and two distinct unblocked nodes y_1, y_2 blocking x. Since both y_1 and y_2 must be ancestors of x, w.o.l.g., y_1 is an ancestor of y_2. Yet, this implies that y_1 directly blocks y_2 and x cannot be directly blocked because it has the blocked ancestor y_2, a contradiction.

Before we prove the correctness of the $\mathcal{SHIQ}$-algorithm, we discuss the intuition behind the expansion rules and their correspondence to the constructors of $\mathcal{SHIQ}$. Roughly speaking,[4] the completion tree is a partial description of a model whose individuals correspond to nodes and whose interpretation of concept and role names is determined by the node and edge labels. Since the completion tree is a tree, this would not yield a correct interpretation of transitive roles, and thus the interpretation of transitive roles is built via the transitive closure of the relations induced by the corresponding edge labels.

The $\rightarrow_\sqcap$-, $\rightarrow_\sqcup$-, $\rightarrow_\exists$-, and $\rightarrow_\forall$-rules are the standard tableau rules for $\mathcal{ALC}$ from Algorithm 3.2, with the exception that we limit the applicability of the $\rightarrow_\forall$- and $\rightarrow_{\forall_+}$-rule to those nodes that are not blocked or directly blocked. The $\rightarrow_{\forall_+}$-rule is similar to the

[4] For the following considerations, we employ a simpler view of the correspondence between completion trees and models, and do not bother with the unraveling construction mentioned above.

Figure 6.6 Tableau expansion rules for $\mathcal{SHIQ}$

$\rightarrow_\sqcap$: **if** 1. $C_1 \sqcap C_2 \in \mathbf{L}(x)$ and
2. $\{C_1, C_2\} \not\subseteq \mathbf{L}(x)$
then $\mathbf{L}(x) \rightarrow_\sqcap \mathbf{L}(x) \cup \{C_1, C_2\}$

$\rightarrow_\sqcup$ **if** 1. $C_1 \sqcup C_2 \in \mathbf{L}(x)$ and
2. $\{C_1, C_2\} \cap \mathbf{L}(x) = \emptyset$
then $\mathbf{L}(x) \rightarrow_\sqcup \mathbf{L}(x) \cup \{E\}$ for some $E \in \{C_1, C_2\}$

$\rightarrow_\forall$: **if** 1. $\forall R.D \in \mathbf{L}(x)$, x is not indirectly blocked, and
2. there is an R-neighbour y of x with $D \notin \mathbf{L}(y)$
then $\mathbf{L}(y) \rightarrow_\forall \mathbf{L}(y) \cup \{D\}$

$\rightarrow_{\forall_+}$: **if** 1. $\forall R.D \in \mathbf{L}(x)$, x is not indirectly blocked, and
2. there is some T with $\mathsf{Trans}(T)$ and $T \sqsubseteq^* R$
3. there is a T-neighbour y of x with $\forall T.D \notin \mathbf{L}(y)$
then $\mathbf{L}(y) \rightarrow_{\forall_+} \mathbf{L}(y) \cup \{\forall T.D\}$

$\rightarrow_\exists$: **if** 1. $\exists R.D \in \mathbf{L}(x)$, x is not blocked
2. x has no R-neighbour y with $D \in \mathbf{L}(y)$
then create a new successor y of x with $\mathbf{L}(x, y) = \{R\}$ and $\mathbf{L}(y) = \{D\}$

$\rightarrow_{\text{choose}}$: **if** 1. $(\bowtie\ n\ S\ D) \in \mathbf{L}(x)$, x is not indirectly blocked, and
2. there is an S-neighbour y of x with $\{D, {\sim}D\} \cap \mathbf{L}(y) = \emptyset$
then $\mathbf{L}(y) \rightarrow_{\text{choose}} \mathbf{L}(y) \cup \{E\}$ for some $E \in \{D, {\sim}D\}$

$\rightarrow_\geq$: **if** 1. $(\geq n\ S\ D) \in \mathbf{L}(x)$, x is not blocked, and
2. there are not n S-neighbours $y_1, \ldots, y_n$ of x with
$D \in \mathbf{L}(y_i)$ and $y_i \neq y_j$ for $1 \leq i < j \leq n$
then create n new successors $y_1, \ldots, y_n$ of x with $\mathbf{L}(x, y_i) = \{S\}$,
$\mathbf{L}(y_i) = \{D\}$, and $y_i \neq y_j$ for $1 \leq i < j \leq n$.

$\rightarrow_\leq$: **if** 1. $(\leq n\ S\ D) \in \mathbf{L}(x)$ with $n \geq 1$, x is not indirectly blocked, and
2. $\sharp S^{\mathbf{T}}(x, D) > n$ and there are two S-neighbours y, z of x with
$D \in \mathbf{L}(y), D \in \mathbf{L}(z)$, y is a successor of x, and not $y \neq z$
then 1. $\mathbf{L}(z) \rightarrow_\leq \mathbf{L}(z) \cup \mathbf{L}(y)$ and
2. if z is a predecessor of x
then $\mathbf{L}(z, x) \rightarrow_\leq \mathbf{L}(z, x) \cup \mathsf{Inv}(\mathbf{L}(x, y))$
else $\mathbf{L}(x, z) \rightarrow_\leq \mathbf{L}(x, z) \cup \mathbf{L}(x, y)$
3. $\mathbf{L}(x, y) \rightarrow_\leq \emptyset$
4. Set $u \neq z$ for all u with $u \neq y$

$\rightarrow_{\forall_+}$-rule for $\mathcal{SI}$ without refined blocking extended to deal with role-hierarchies as follows. Assume a situation that satisfies the precondition of the $\rightarrow_{\forall_+}$-rule, i.e., $\forall R.D \in \mathbf{L}(x)$, and there is a T-neighbour y of x with $\mathsf{Trans}(T)$, $T \sqsubseteq^* R$, and $\forall R.D \notin \mathbf{L}(y)$. If y has a

T-neighbour z, then, due to the transitivity of T, x and z are also related via T. Since $T \sqsubseteq^* R$, it is also an R-neighbour of x and hence must satisfy D. This is ensured by adding $\forall T.D$ to $\mathbf{L}(y)$, which, in turn, causes D to be added to $\mathbf{L}(z)$.

The rules dealing with qualifying number restrictions work similarly to the rules of the standard algorithm for $\mathcal{ALCQ}$ (Algorithm 4.4). For a concept $(\geq n\ S\ D) \in \mathbf{L}(x)$, the $\rightarrow_{\geq}$-rule generates n S-successors $y_1, \ldots, y_n$ of x with $D \in \mathbf{L}(y_i)$. To prevent the $\rightarrow_{\leq}$-rule from indentifying these new nodes, it also sets $y_i \neq y_j$ for each $1 \leq i < j \leq n$. Conversely, if $(\leq n\ S\ D) \in \mathbf{L}(x)$ and x has more than n S-neighbours that are labelled with D, then the $\rightarrow_{\leq}$-rule chooses two of them, say y, z, that are not required to be distinct by $\neq$ and merges them, together with the edges connecting them with x. The algorithm constructs a completion *tree* so at least one of y, z must be a successor of x. Let this be y. If z is a predecessor of x in the completion tree, then it is necessary that we join y onto z and not vice vesus because otherwise x would become detached in the completion tree.

The definition of a clash takes care of the situation where the $\neq$ relation makes it impossible to merge any two S-neighbours of x, while the $\rightarrow_{\text{choose}}$-rule ensures that all S-neighbours of x are labelled with either D or $\sim D$. The relation $\neq$ is used to prevent infinite sequences of rule applications for contradicting number restrictions of the form $(\geq n\ S\ D)$ and $(\leq m\ S\ D)$, with $n > m$.

Labeling edges with sets of roles allows a single node to be both an S- and R-successor of x even if S and R are not comparable with respect to $\sqsubseteq^*$. An example for a concept that enforces such a situation is $(\geq 2\ S_1\ A) \sqcap (\geq 2\ S_2\ A) \sqcap (\leq 3\ R\ A)$ with $S_i \sqsubseteq R$, which enforces a successor reachable both via S_1 and S_2.

We will now prove correctness of the tableau algorithm in a manner similar to the one for the $\mathcal{SI}$-algorithm.

Termination

Like for $\mathcal{SI}$, termination of the algorithm is ensured by blocking, which prevents the creation of unbounded paths in the completion tree.

Lemma 6.35 (Termination)
For each $\mathcal{SHIQ}$-concept C and role hierarchy $\mathcal{R}$, the tableau algorithm terminates.

Proof. Let $m = \sharp clos(C, \mathcal{R})$, $k = \sharp \overline{\mathsf{NR}}_{C,\mathcal{R}}$, and n_{max} the maximum n that occurs in a concept of the form $(\bowtie\ n\ S\ D) \in clos(C, \mathcal{R})$. Termination is a consequence of the following properties of the expansion rules:

- The expansion rules never remove nodes from the tree or concepts from node labels. Edge labels can only be changed by the $\rightarrow_{\leq}$-rule which either expands them or sets them to $\emptyset$; in the latter case, the node below the $\emptyset$-labelled edge is blocked and this block is never broken.

- Each successor of a node x is the result of the application of the $\rightarrow_{\exists}$-rule or the $\rightarrow_{\geq}$-rule to x. (Note that the $\rightarrow_{\leq}$-rule, does not move nodes in the tree.) For a node x, each concept in $\mathbf{L}(x)$ can trigger the generation of successors at most once.

For the $\rightarrow_\exists$-rule, if a successor y of x was generated for a concept $\exists R.D \in \mathbf{L}(x)$ and later $\mathbf{L}(x,y)$ is set to $\emptyset$ by the $\rightarrow_\leq$-rule, then there is some R-neighbour z of x with $D \in \mathbf{L}(z)$.

For the $\rightarrow_\geq$-rule, if $y_1, \ldots, y_n$ were generated by the $\rightarrow_\geq$-rule for $(\geq n\ S\ D) \in \mathbf{L}(x)$, then $y_i \not\doteq y_j$ holds for all $1 \leq i < j \leq n$. This implies that there are always n S-neighbours $y'_1, \ldots, y'_n$ of x with $D \in \mathbf{L}(y'_i)$ and $y'_i \not\doteq y'_j$ for all $1 \leq i < j \leq n$, since the $\rightarrow_\leq$-rule never merges two nodes y'_i, y'_j with $y'_i \not\doteq y'_j$ and, whenever an application of the $\rightarrow_\leq$-rule sets $\mathbf{L}(x, y'_i)$ to $\emptyset$, there is some S-neighbour z of x which "inherits" both D and all inequalities from y'_i.

Since $clos(C, \mathcal{R})$ contains a total of at most m concept of the form $\exists R.D$ and $(\geq n\ S\ D)$, the out-degree of the tree is bounded by $m \cdot n_{max}$.

- Nodes are labelled with non-empty subsets of $clos(C, \mathcal{R})$ and edges with subsets of $\overline{\mathsf{NR}}_{C,\mathcal{R}}$, so there are at most 2^{2mk} different possible labellings for a pair of nodes and an edge. Therefore, if a path is of length $> 2^{2mk}$, then, from the pair-wise blocking condition, there must be two nodes x, y on this path such that x is directly blocked by y.

  Since a path on which nodes are blocked cannot become longer, paths are of length at most 2^{2mk}. ∎

Completeness

To prove completeness of the $\mathcal{SHIQ}$-algorithm, we proceed as for the $\mathcal{SI}$-algorithm and guide the application of the non-deterministic $\rightarrow_\sqcup$-, $\rightarrow_{\text{choose}}$-, and $\rightarrow_\leq$-rule using a function that maps nodes of the completion tree to elements of a tableau.

Lemma 6.36
Let C be a $\mathcal{SHIQ}$-concept in NNF. If C has a tableau w.r.t. $\mathcal{R}$, then the expansion rules can be applied in such a way that the tableau algorithm yields a complete and clash-free completion tree for C w.r.t. $\mathcal{R}$.

Proof. Let $T = (\mathcal{S}, \mathcal{L}, \mathcal{E})$ be a tableau for C w.r.t. $\mathcal{R}$. We use this tableau to guide the application of the non-deterministic rules. To do this, we will inductively define a function π, mapping the nodes $\mathbf{V}$ of the tree $\mathbf{T}$ to $\mathcal{S}$ such that, for each $x, y \in \mathbf{V}$:

$$\left.\begin{array}{l} \mathbf{L}(x) \subseteq \mathcal{L}(\pi(x)) \\ \text{if } y \text{ is an } R\text{-neighbour of } x, \text{ then } (\pi(x), \pi(y)) \in \mathcal{E}(R) \\ x \not\doteq y \text{ implies } \pi(x) \neq \pi(y) \end{array}\right\} (*)$$

Claim 6.37 Let $\mathbf{T}$ be a completion-tree and π a function that satisfies $(*)$. If a rule is applicable to $\mathbf{T}$, then the rule is applicable to $\mathbf{T}$ in a way that yields a completion-tree $\mathbf{T}'$ and an extension of π that satisfies $(*)$.

Let $\mathbf{T}$ be a completion-tree and π a function that satisfies $(*)$. We have to consider the various rules.

- **For the $\rightarrow_\sqcap$-, $\rightarrow_\sqcup$-, and $\rightarrow_\exists$-rule**, this is analogous to the proof of Lemma 6.8 for $\mathcal{SI}$.

- **The $\rightarrow_\forall$-rule:** If $\forall R.D \in \mathbf{L}(x)$, then $\forall R.D \in \mathcal{L}(\pi(x))$, and if y is an R-neighbour of x, then also $(\pi(x), \pi(y)) \in \mathcal{E}(R)$ due to $(*)$. (T6) implies $D \in \mathbf{L}(\pi(y))$ and hence the $\rightarrow_\forall$-rule can be applied without violating $(*)$.

- **The $\rightarrow_{\forall_+}$-rule:** If $\forall R.D \in \mathbf{L}(x)$, then $\forall R.D \in \mathcal{L}(\pi(x))$, and if there is some $T \sqsubseteq^* R$ with $\mathsf{Trans}(T)$ and y is an T-neighbour of x, then also $(\pi(x), \pi(y)) \in \mathcal{E}(T)$ due to $(*)$. (T7) implies $\forall T.D \in \mathcal{L}(\pi(y))$ and hence the $\rightarrow_{\forall_+}$-rule can be applied without violating $(*)$.

- **The $\rightarrow_{\mathbf{choose}}$-rule:** If $(\bowtie n\ S\ D) \in \mathbf{L}(x)$, then $(\bowtie n\ S\ D) \in \mathcal{L}(\pi(x))$, and, if there is an S-neighbour y of x, then $(\pi(x), \pi(y)) \in \mathcal{E}(S)$ due to $(*)$. (T12) implies $\{D, \sim D\} \cap \mathcal{L}(\pi(y) \neq \emptyset$. Hence the $\rightarrow_{\text{choose}}$-rule can add an appropriate concept $E \in \{D, \sim D\}$ to $\mathbf{L}(x)$ such that $\mathbf{L}(y) \subseteq \mathcal{L}(\pi(y))$ holds.

- **The $\rightarrow_\geq$-rule:** If $(\geq n\ S\ D) \in \mathbf{L}(x)$, then $(\geq n\ S\ D) \in \mathcal{L}(\pi(x))$ and (T10) implies $\sharp S^{\mathcal{T}}(\pi(x), D) \geq n$. Hence there are elements $t_1, \ldots, t_n \in \mathcal{S}$ such that $(\pi(x), t_i) \in \mathcal{E}(S)$, $D \in \mathcal{L}(t_i)$, and $t_i \neq t_j$ for $1 \leq i < j \leq n$. The $\rightarrow_\geq$-rule generates n new nodes $y_1, \ldots, y_n$. By extending $\pi := \pi[y_1 \mapsto t_1, \cdots y_n \mapsto t_n]$, one obtains a function π' that satisfies $(*)$ for the extended tree.

- **The $\rightarrow_\leq$-rule:** If $(\leq n\ S\ D) \in \mathbf{L}(x)$, then $(\leq n\ S\ D) \in \mathcal{L}(\pi(x))$ and (T11) implies $\sharp S^{\mathcal{T}}(\pi(x), D) \leq n$. If the $\rightarrow_\leq$-rule is applicable, we have $\sharp S^{\mathbf{T}}(x, D) > n$, which implies that there are at least $n+1$ S-neighbours $y_0, \ldots, y_n$ of x such that $D \in \mathbf{L}(y_i)$. Thus, there must be two nodes $y, z \in \{y_0, \ldots, y_n\}$ such that $\pi(y) = \pi(z)$ (because otherwise $\sharp S^{\mathcal{T}}(\pi(x), D) > n$ would hold). Since $\pi(y) = \pi(z)$, we have that $y \not\doteq z$ cannot hold because of $(*)$, and y, z can be chosen such that y is a successor of x because x has at most one predecessor. Hence the $\rightarrow_\leq$-rule can be applied without violating $(*)$.

Why does this claim yield the completeness of the tableau algorithm? For the initial completion-tree consisting of a single node x_0 with $\mathbf{L}(x_0) = \{C\}$ and $\not\doteq\, = \emptyset$, the function $\pi = [x_0 \mapsto s_0]$ for some $s_0 \in \mathcal{S}$ with $C \in \mathbf{L}(s_0)$ satisfies $(*)$. Such an s_0 exists due to (T1). Whenever a rule is applicable to $\mathbf{T}$, it can be applied in a way that maintains $(*)$, and, since the algorithm terminates, we have that any sequence of rule applications must terminate. Property $(*)$ implies that any tree $\mathbf{T}$ generated by these rule-applications must be clash-free as there are only two possibilities for a clash, and it is easy to see that neither of these can hold in $\mathbf{T}$:

- $\mathbf{T}$ cannot contain a node x such that $\{A, \neg A\} \in \mathbf{L}(x)$ because $\mathbf{L}(x) \subseteq \mathcal{L}(\pi(x))$ and hence (T2) would be violated for $\pi(x)$.

- $\mathbf{T}$ cannot contain a node x with $(\leq n\ S\ D) \in \mathbf{L}(x)$ and $n+1$ S-neighbours $y_0, \dots y_n$ of x with $D \in \mathbf{L}(y_i)$ and $y_i \not\doteq y_j$ for $0 \leq i < j \leq n$ because $(\leq n\ S\ D) \in \mathcal{L}(\pi(x))$, and, since $y_i \not\doteq y_j$ implies $\pi(y_i) \neq \pi(y_j)$, $\sharp S^{\mathcal{T}}(\pi(x), D) > n$, in contradiction to (T11). ∎

Soundness

Due to the presence of qualifying number restrictions and the lack of the finite model property, the construction of a tableau from a complete and clash-free completion tree is much more involved than this has been the case for $\mathcal{SI}$ where, in case of a block, it was possible to generate a cyclic model. Here, the completion tree is unraveled into an infinite tree by successively substituting a blocked node by the subtree rooted at the blocking node. The presence of qualifying number restrictions makes it necessary, in case of a blocking situation, to record the pair of blocking and blocked node (see the case for (T10) in the proof below).

Lemma 6.38 (Soundness)
If the $\mathcal{SHIQ}$-algorithm generates a complete and clash-free completion tree for a concept C and a role hierarchy $\mathcal{R}$, then C has a tableau, w.r.t. $\mathcal{R}$.

Proof. Let $\mathbf{T} = (\mathbf{V}, \mathbf{E}, \mathbf{L})$ be a complete and clash-free completion tree. A *path* is a sequence of pairs of nodes of $\mathbf{T}$ of the form $p = [\frac{x_0}{x'_0}, \dots, \frac{x_n}{x'_n}]$. We define auxiliary functions $\mathsf{Tail}, \mathsf{Tail}'$ by setting, for such a path p, $\mathsf{Tail}(p) = x_n$ and $\mathsf{Tail}'(p) = x'_n$. With $[p|\frac{x_{n+1}}{x'_{n+1}}]$ we denote the path $[\frac{x_0}{x'_0}, \dots, \frac{x_n}{x'_n}, \frac{x_{n+1}}{x'_{n+1}}]$. The set $\mathsf{Paths}(\mathbf{T})$ is defined inductively as follows:

- For the root node x_0 of $\mathbf{T}$, $[\frac{x_0}{x_0}] \in \mathsf{Paths}(\mathbf{T})$, and
- For a path $p \in \mathsf{Paths}(\mathbf{T})$ and a node z in $\mathbf{T}$:
  - if z is a successor of $\mathsf{Tail}(p)$ and z is not blocked, then $[p|\frac{z}{z}] \in \mathsf{Paths}(\mathbf{T})$, or
  - if, for some node y in $\mathbf{T}$, y is a successor of $\mathsf{Tail}(p)$ and z blocks y, then $[p|\frac{z}{y}] \in \mathsf{Paths}(\mathbf{T})$.

Please note that, due to the construction of Paths, for $p \in \mathsf{Paths}(\mathbf{T})$ with $p = [p'|\frac{x}{x'}]$, we have that x is not blocked, x' is blocked iff $x \neq x'$, and x' is never indirectly blocked—it is either directly blocked or unblocked. Furthermore, the blocking condition implies $\mathbf{L}(x) = \mathbf{L}(x')$.

Now we can define a tableau $\mathcal{T} = (\mathcal{S}, \mathcal{L}, \mathcal{E})$ with:

$$
\begin{aligned}
\mathcal{S} &= \mathsf{Paths}(\mathbf{T}),\\
\mathcal{L}(p) &= \mathbf{L}(\mathsf{Tail}(p)),\\
\mathcal{E}(R) &= \{(p,q) \in \mathcal{S} \times \mathcal{S} \mid \begin{array}{l} \text{Either } q = [p|\frac{x}{x'}] \text{ and } x' \text{ is an } R\text{-successor of } \mathsf{Tail}(p),\\ \text{or } p = [q|\frac{x}{x'}] \text{ and } x' \text{ is an } \mathsf{Inv}(R)\text{-successor of } \mathsf{Tail}(q)\}. \end{array}
\end{aligned}
$$

Claim 6.39 $\mathcal{T}$ is a tableau for C w.r.t. $\mathcal{R}$.

We show that $\mathcal{T}$ satisfies all the properties from Definition 6.32.

- $C \in \mathcal{L}[\frac{x_0}{x_0}]$ because $C \in \mathbf{L}(x_0)$, hence **(T1)** holds.
- **(T2)** holds because $\mathbf{T}$ is clash-free.
- **(T3), (T4)** hold because $\mathbf{T}$ is complete.
- **(T5)**: assume $\exists R.D \in \mathcal{L}(p)$ and let $x = \mathsf{Tail}(p)$. In $\mathbf{T}$, there is an R-neighbour y of x with $D \in \mathbf{L}(y)$ because x is not blocked and the $\rightarrow_\exists$-rule is not applicable. There are two possibilities:
  - y is a successor of x in $\mathbf{T}$. If y is not blocked, then $q := [p|\frac{y}{y}] \in \mathcal{S}$ and $(p, q) \in \mathcal{E}(R)$ as well as $D \in \mathcal{L}(q)$. If y is blocked by some node z in $\mathbf{T}$, then $q := [p|\frac{z}{y}] \in \mathcal{S}$, $(p, q) \in \mathcal{E}(R)$ and, since $\mathcal{L}(q) = \mathbf{L}(z) = \mathbf{L}(y)$, $D \in \mathcal{L}(q)$.
  - y is a predecessor of x. Again, there are two possibilities:
    * p is of the form $p = [q|\frac{x}{x}]$ with $\mathsf{Tail}(q) = y$.
    * p is of the form $p = [q|\frac{x}{x'}]$ with $\mathsf{Tail}(q) = u \neq y$. Since x only has one predecessor in $\mathbf{T}$, the node u cannot be the predecessor of x. Then it must be the predecessor of x' in $\mathbf{T}$, $x' \neq x$, and x blocks x', all due to the construction of $\mathsf{Paths}(\mathbf{T})$. Together with the definition of the blocking condition, this implies $\mathbf{L}(u, x') = \mathbf{L}(y, x)$ as well as $\mathbf{L}(u) = \mathbf{L}(y)$, due to the pair-wise blocking condition.

  In both cases, $(p, q) \in \mathcal{E}(R)$ and $D \in \mathcal{L}(q)$.
- **(T6)**: assume $\forall R.D \in \mathcal{L}(p)$ and $(p, q) \in \mathcal{E}(R)$. If $q = [p|\frac{x}{x'}]$, then x' is an R-successor of $\mathsf{Tail}(p)$, and thus $D \in \mathbf{L}(x')$ because the $\rightarrow_\forall$-rule is not applicable. Since $\mathcal{L}(q) = \mathbf{L}(x) = \mathbf{L}(x')$, we have $D \in \mathcal{L}(q)$. If $p = [q|\frac{x}{x'}]$, then x' is an $\mathsf{Inv}(R)$-successor of $\mathsf{Tail}(q)$, $\mathsf{Tail}(q)$ an R-neighbour of x' and thus $D \in \mathcal{L}(q) = \mathbf{L}(\mathsf{Tail}(q))$ because x' is not indirectly blocked and the $\rightarrow_\forall$-rule is not applicable.
- **(T7)**: assume $\forall R.D \in \mathcal{L}(p)$ and $(p, q) \in \mathcal{E}(T)$ for some $T \sqsubseteq^* R$ with $\mathsf{Trans}(T)$. If $q = [p|\frac{x}{x'}]$, then x' is a T-successor of $\mathsf{Tail}(p)$ and thus $\forall T.D \in \mathbf{L}(x')$ because otherwise the $\rightarrow_{\forall_+}$-rule would be applicable. From $\mathcal{L}(q) = \mathbf{L}(x) = \mathbf{L}(x')$, it follows that $\forall T.D \in \mathcal{L}(q)$. If $p = [q|\frac{x}{x'}]$, then x' is an $\mathsf{Inv}(R)$-successor of $\mathsf{Tail}(q)$, and hence $\mathsf{Tail}(q)$ is a T-neighbour of x'. Because x' is not indirectly blocked, this implies $\forall T.D \in \mathcal{L}(q) = \mathbf{L}(\mathsf{Tail}(q))$.
- **(T12)**: assume $(\bowtie n\ S\ D) \in \mathcal{L}(p)$ and $(p, q) \in \mathcal{E}(S)$. If $q = [p|\frac{x}{x'}]$, then x' is an S-successor of $\mathsf{Tail}(p)$ and thus $\{D, \sim D\} \cap \mathbf{L}(x') \neq \emptyset$ because the $\rightarrow_{\text{choose}}$-rule is not applicable. Since $\mathcal{L}(q) = \mathbf{L}(x) = \mathbf{L}(x')$, we have $\{D, \sim D\} \cap \mathcal{L}(q) \neq \emptyset$. If $p = [q|\frac{x}{x'}]$, then x' is an $\mathsf{Inv}(S)$-successor of $\mathsf{Tail}(q)$, $\mathsf{Tail}(q)$ is an S-neighbour of x', and thus $\{D, \sim D\} \cap \mathcal{L}(q) = \mathbf{L}(\mathsf{Tail}(q)) \neq \emptyset$ because x' is not indirectly blocked and the $\rightarrow_{\text{choose}}$-rule is not applicable.

- **(T8)** is satisfied due to the symmetric definition of $\mathcal{E}$.
- **(T9)** is satisfied due to the definition of R-successors that takes into account the role hierarchy $\sqsubseteq^*$.
- **(T10)**: assume $(\geq n\ S\ D) \in \mathcal{L}(p)$. Completeness of $\mathbf{T}$ implies that there exist n distinct individuals $y_1, \ldots, y_n$ in $\mathbf{T}$ such that each y_i is an S-neighbour of $\mathsf{Tail}(p)$ and $D \in \mathbf{L}(y_i)$. We claim that, for each of these individuals, there is a path q_i such that $(p, q_i) \in \mathcal{E}(S)$, $D \in \mathcal{L}(q_i)$, and $q_i \neq q_j$ for all $1 \leq i < j \leq n$. Obviously, this implies $\sharp S^{\mathcal{T}}(p, D) \geq n$. For each y_i, there are three possibilities:
  - y_i is an S-successor of x and y_i is not blocked in $\mathbf{T}$. Then $q_i = [p|\frac{y_i}{y_i}]$ is a path with the desired properties.
  - y_i is an S-successor of x and y_i is blocked in $\mathbf{T}$ by some node z. Then $q_i = [p|\frac{z}{y_i}]$ is the path with the desired properties. Since the same z may block several of the y_js, it is indeed necessary to include the blocking nodes explicitly into the path construction to make these paths distinguishable.
  - x is an $\mathsf{Inv}(S)$-successor of y_i. Since $\mathbf{T}$ is a tree, there may be at most one such y_i. This implies that p is of the form $p = [q|\frac{x}{x'}]$ with $\mathsf{Tail}(q) = y_i$. The path q has the desired properties and, obviously, q is distinct from all other paths q_j.
- Assume **(T11)** is violated. Hence there is some $p \in \mathcal{S}$ with $(\leq n\ S\ D) \in \mathcal{L}(p)$ and $\sharp S^{\mathcal{T}}(p, D) > n$. We show that this implies $\sharp S^{\mathbf{T}}(\mathsf{Tail}(p), D) > n$, in contradiction to either clash-freeness or completeness of $\mathbf{T}$. Define $x = \mathsf{Tail}(p)$ and $\mathbf{P} = S^{\mathcal{T}}(p, D)$. Due to the assumption, we have $\sharp\mathbf{P} > n$. We distinguish two cases:
  - $\mathbf{P}$ contains only paths of the form $q = [p|\frac{y}{y'}]$. We claim that the function Tail' is injective on $\mathbf{P}$. Assume that there are two paths $q_1, q_2 \in \mathbf{P}$ with $q_1 \neq q_2$ and $\mathsf{Tail}'(q_1) = \mathsf{Tail}'(q_2) = y'$. Then q_1 is of the form $q_1 = [p|\frac{y_1}{y'}]$ and q_2 is of the form $q_2 = [p|\frac{y_2}{y'}]$ with $y_1 \neq y_2$. If y' is not blocked in $\mathbf{T}$, then $y_1 = y' = y_2$, contradicting $q_1 \neq q_2$. If y' is blocked in $\mathbf{T}$, then both y_1 and y_2 block y', which implies $y_1 = y_2$, again a contradiction.

    Since Tail' is injective on $\mathbf{P}$, it holds that $\sharp\mathbf{P} = \sharp\mathsf{Tail}'(\mathbf{P})$. Also for each $y' \in \mathsf{Tail}'(\mathbf{P})$, y' is an S-successor of x, and $D \in \mathbf{L}(y')$. This implies $\sharp S^{\mathbf{T}}(x, D) > n$.
  - $\mathbf{P}$ contains a path q where p is of the form $p = [q|\frac{x}{x'}]$. Obviously, $\mathbf{P}$ may only contain one such path. As in the previous case, Tail' is an injective function on the set $\mathbf{P}' := \mathbf{P} \setminus \{q\}$, each $y' \in \mathsf{Tail}'(\mathbf{P}')$ is an S-successor of x and $D \in \mathbf{L}(y')$ for each $y' \in \mathsf{Tail}'(\mathbf{P}')$. To show that indeed $\sharp S^{\mathbf{T}}(x, D) > n$ holds, we have to prove the existence of a further S-neighbour u of x with $D \in \mathbf{L}(u)$ and $u \notin \mathsf{Tail}'(\mathbf{P}')$. We distinguish two cases:
    * $x = x'$. Hence x is not blocked. This implies that x is an $\mathsf{Inv}(S)$-successor of z in $\mathbf{T}$. Since $\mathsf{Tail}'(\mathbf{P}')$ contains only successors of x, we have that $z \notin \mathsf{Tail}'(\mathbf{P}')$ and, by construction, z is an S-neighbour of x with $C \in \mathbf{L}(z)$.

* $x \neq x'$. This implies that x' is blocked in $\mathbf{T}$ by x and that x' is an $\mathsf{Inv}(S)$-successor of z in $\mathbf{T}$. The definition of pair-wise blocking implies that x is an $\mathsf{Inv}(S)$-successor of some node u in $\mathbf{T}$ with $\mathbf{L}(u) = \mathbf{L}(z)$. Again, since $\mathsf{Tail}'(\mathbf{P}')$ contains only successors of x, we have that $u \notin \mathsf{Tail}'(\mathbf{P}')$ and, by construction, u is an S-neighbour of x with $D \in \mathbf{L}(u)$. ■

By showing termination (Lemma 6.35), soundness (Lemma 6.36), and completeness (Lemma 6.38) of the $\mathcal{SHIQ}$-algorithm, we have established its correctness:

Theorem 6.40
The $\mathcal{SHIQ}$-algorithm is a non-deterministic decision procedure for satisfiability and subsumption of $\mathcal{SHIQ}$-concepts w.r.t. a role hierarchy.

Of course, due to Theorem 6.17, the tableau algorithm can also be used to decide satisfiability and subsumption of $\mathcal{SHIQ}$-concepts w.r.t. a general TBox. To apply the algorithm to $\mathcal{SHIQ}$-knowledge bases, one can either use a pre-completion approach similar to the one used to prove Theorem 4.42 (probably with catastrophic effects on the runtime of the algorithm), or one can integrate the ABox directly into the tableau algorithm. Horrocks, Sattler, and Tobies (2000b) present an algorithm that follows the latter approach.

We have already mentioned that we do not expect to obtain a worst-case optimal solution for $\mathcal{SHIQ}$-satisfiability from the tableau approach—such an algorithm has already been given in order to prove Theorem 6.29. Instead, Algorithm 6.34 is intended as a *practical* decision procedure that can be optimized so that it performs well for reasoning tasks occurring in applications. Nevertheless, it is interesting to know how far our tableau approach exceeds the worst-case complexity.

Lemma 6.41
The $\mathcal{SHIQ}$-algorithm runs in 2-NExpTime.

Proof. Let C be a $\mathcal{SHIQ}$-concept and $\mathcal{R}$ a role hierarchy. Let $m = \sharp clos(C, \mathcal{R})$, $k = \sharp\overline{\mathsf{NR}}_{C,R}$, and $n_{\max}$ be the maximum n that occurs in a qualifying number restriction in $clos(C, \mathcal{R})$. If we set $n = |C| + |\mathcal{R}|$, the it holds that $m = \mathcal{O}(|C| \cdot |\mathcal{R}|) = \mathcal{O}(n^2)$, $k = \mathcal{O}(|C| + |R|) = \mathcal{O}(n)$, and $n_{\max} = \mathcal{O}(2^{|C|}) = \mathcal{O}(2^n)$. In the proof of Lemma 6.35, we have shown that paths in a completion tree for C become no longer than 2^{2mk} and that the out-degree of a completion tree is bounded by $m \cdot n_{\max}$. Hence, the $\mathcal{SHIQ}$-algorithm will construct a tree with no more than

$$(m \cdot n_{\max})^{2^{2mk}} = \mathcal{O}((n^2 \cdot 2^n)^{2^{2n^3}}) = \mathcal{O}(2^{n \cdot 2^{2n^3}}) = \mathcal{O}(2^{2^{n^4}})$$

nodes. Each node of this tree is labelled with a subset of $clos(C, \mathcal{R})$ and each edge is labelled with a subset of $\overline{\mathsf{NR}}_{C,\mathcal{R}}$. Since every application of a rule either adds a node to the tree, a concept or role to one of the labels, or sets the label of an edge to $\emptyset$ (in which case the corresponding successor is blocked forever), the $\mathcal{SHIQ}$-algorithm runs in 2-NExpTime. ■

This seems to be a discouraging result: the tableau algorithm runs in 2-NExpTime while the worst-case complexity is only ExpTime. On the other hand, there exist DL systems like iFaCT (Horrocks, 1999), which is based on Algorithm 6.34, or RACE (Haarslev & Möller, 1999), which is based on a similar algorithm (Haarslev & Möller, 2000a). These systems show good performance in system comparisons (Massacci & Donini, 2000) and are successfully utilized in a number of applications (e.g., Haarslev & Möller, 2000b; Franconi & Ng, 2000). This can be explained by the fact that tableau algorithms seem to be particularly amenable to optimizations (Baader et al., 1994; Horrocks, 1997; Horrocks & Patel-Schneider, 1999; Horrocks & Tobies, 2000; Haarslev & Möller, 2000c). It is these optimizations that cause the good behaviour of implementations based on tableau algorithms like Algorithm 6.34.

Chapter 7

Guarded Fragments

The Guarded Fragment of first-order logic, introduced by Andréka, van Benthem, and Németi (1998), is a successful attempt to transfer many good properties of modal, temporal, and description logics to a large, naturally defined fragment of predicate logic. Among these are decidability, the finite model property, invariance under an appropriate variant of bisimulation, and other nice model theoretic properties (Andréka et al., 1998; Grädel, 1999b).

The Guarded Fragment (GF) is obtained from full first-order logic through relativization of quantifiers by so-called guard formulas. Every appearance of a quantifier in GF must be of the form

$$\exists \mathbf{y}(\alpha(\mathbf{x},\mathbf{y}) \land \phi(\mathbf{x},\mathbf{y})) \text{ or } \forall \mathbf{y}(\alpha(\mathbf{x},\mathbf{y}) \rightarrow \phi(\mathbf{x},\mathbf{y})),$$

where α is a positive atomic formula, the *guard*, that contains all free variables of ϕ. This generalises quantification in description, modal, and temporal logics, where quantification is restricted to those elements reachable via some accessibility relation. For example, in DLs, quantification occurs in the form of existential and universal restrictions like $\forall$`has_child.Rich`, which expresses that those individuals *reachable via the role* (guarded by) `has_child` must be rich.

By allowing for more general formulas as guards while preserving the idea of quantification only over elements that are *close together* in the model, one obtains generalisations of GF which are still well-behaved in the above sense. Most importantly, one can obtain the *loosely guarded fragment* (LGF) (van Benthem, 1997) and the *clique guarded fragment* (CGF) (Grädel, 1999a), for which decidability, invariance under clique guarded bisimulation, and some other properties have been shown in (Grädel, 1999a).

Guarded fragments have spawned considerable interest in the DL community, mainly for two reasons. On the one hand, many DLs can be embedded into suitable guarded fragments, which allows the transfer, e.g., of decidability results from guarded logics to DLs. Goncalves and Grädel (2000) prove decidability of the guarded fragment μACGFI, which, among other DLs, allows a simple embedding of $\mathcal{ALCQI}$ and $\mathcal{ALCQI}b$, proving the decidability of these logics. On the other hand, guarded fragments generalise DLs and add expressive power that is not present in classical DLs, but interesting for knowledge representation. For example, Lutz, Sattler, and Tobies (1999) present a restriction of GF that strictly contains $\mathcal{ALCI}$ and allows for n-ary relations instead of the binary roles of most DLs.

GF, LGF, and CGF are decidable and known to be 2-ExpTime complete, which is shown by Grädel (1999a, 1999b) using game and automata-based approaches. For these guarded fragments, the automata approach has the same problems as it has for modal and description logics: it is unclear how to turn the automata decision procedures into efficient implementations and a naive implementation has every-case exponential complexity, which makes it unusable for applications. So, while these approaches yield (worst-case) optimal complexity results for many logics, they appear to be unsuitable as a starting point for an efficient implementation. As we have seen, many decidability results for modal or description logics are based on tableau algorithms and some of the fastest implementations of modal satisfiability procedures are based on tableau calculi (Horrocks, 2000; Patel-Schneider, 2000). Unlike automata algorithms, the average-case behaviour in practice is so good that finding *really* hard problems to test these implementations has become a problem in itself (Horrocks et al., 2000). In this chapter, we generalise the principles of the tableau algorithms encountered in this thesis to develop a tableau algorithm for CGF.

Recall the conjecture by Vardi that the tree model property is the main reason for the decidability of many modal style logics (Vardi, 1996). As pointed out in (Grädel, 1999b), the generalised tree model property explains the similarly robust decidability of guarded logics, and can be seen as a strong indication that guarded logics are a generalisation of modal logics that retain the essence of modal logics. This becomes even more evident when regarding the respective fixed-point extensions (Grädel & Walukiewicz, 1999) and is the foundation of general decidability results for guarded logics via reduction to the modal μ-calculus and the monadic theory of countable trees ($S\omega S$) (Grädel, 2001). The generalised tree model property of CGF is also essential for our tableau algorithm. Indeed, as a corollary of the constructions used to show the soundness of our algorithm, we obtain an alternative proof for the fact that CGF has the generalised tree model property.

7.1 Syntax and Semantics

For the definitions of GF and LGF we refer the reader to (Grädel, 1999a). The *clique guarded fragment* CGF of first-order logic can be obtained in two equivalent ways, by either semantically or syntactically restricting the range of the first-order quantifiers. In the following we will use bold letters to refer to tuples of elements of the universe ($\mathbf{a}, \mathbf{b}, \dots$) resp. tuples of variables ($\mathbf{x}, \mathbf{y}, \dots$).

Definition 7.1 (Semantic CGF)
Let τ be a relational vocabulary. For a τ-structure $\mathfrak{A}$ with universe A, the Gaifman graph *of $\mathfrak{A}$ is defined as the undirected graph $G(\mathfrak{A}) = (A, E^{\mathfrak{A}})$ with*

$$E^{\mathfrak{A}} = \{(a, a') \ : \ a \neq a', \textit{there exists } R \in \tau \textit{ and } \mathbf{a} \in R^{\mathfrak{A}} \textit{ which contains both } a \textit{ and } a'\}.$$

Under clique guarded semantics *we understand the modification of standard first-order semantics, where, instead of ranging over all elements of the universe, a quantifier is restricted to elements that form a clique in the Gaifman graph, including the binding for the free variables of the matrix formula. More precisely, let $\mathfrak{A}$ be a τ-structure and ρ an environment mapping variables to elements of A. We define the model relation inductively over the structure of formulas as the usual* FO *semantics with the exception*

$$\begin{aligned} \mathfrak{A}, \rho \models \forall y.\phi(\mathbf{x}, y) \text{ iff, for all } a \in A, \text{ such that} \\ \rho(\mathbf{x}) \cup \{a\} \text{ forms a clique in } G(\mathfrak{A}), \\ \text{it is the case that } \mathfrak{A}, \rho[x \mapsto a] \models \phi \ , \end{aligned}$$

and a similar definition for the existential case. With CGF *we denote first-order logic restricted to clique guarded semantics.* ◇

Definition 7.2 (Syntactic CGF)
Let τ be a relational vocabulary. A formula α is a clique-formula *for a set $\mathbf{x} \subseteq \mathrm{free}(\alpha)$ if α is a (possibly empty if $\mathbf{x}$ contains only one variable) conjunction of atoms (excluding equality statements) such that each two distinct elements from $\mathbf{x}$ coexist in at least one atom, each atom contains at least an element from $\mathbf{x}$, and each element from $\mathrm{free}(\alpha) \setminus \mathbf{x}$ occurs exactly once in α. In the following, we will identify a clique-formula α with the set of its conjuncts.*

The syntactic CGF *is inductively defined as follows.*

1. *Every relational atomic formula $Rx_{i_1} \dots x_{i_m}$ or $x_i = x_j$ belongs to* CGF.
2. CGF *is closed under Boolean operations.*
3. *If $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are tuples of variables, $\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is a* clique-formula *for $\mathbf{x} \cup \mathbf{y}$ and $\phi(\mathbf{x}, \mathbf{y})$ is a formula in* CGF *such that $\mathrm{free}(\phi) \subseteq \mathbf{x} \cup \mathbf{y}$,*

$$\begin{array}{ll} \textit{then} & \exists \mathbf{yz}.(\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z}) \wedge \phi(\mathbf{x}, \mathbf{y})) \\ \textit{and} & \forall \mathbf{yz}.(\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z}) \rightarrow \phi(\mathbf{x}, \mathbf{y})) \end{array}$$

   belong to CGF.

We will use $(\exists \mathbf{yz}.\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z}))\phi(\mathbf{x}, \mathbf{y})$ and $(\forall \mathbf{yz}.\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z}))\phi(\mathbf{x}, \mathbf{y})$ as alternative notations for $\exists \mathbf{yz}.(\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z}) \wedge \phi(\mathbf{x}, \mathbf{y}))$ and $\forall \mathbf{yz}.(\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z}) \rightarrow \phi(\mathbf{x}, \mathbf{y}))$, respectively. A formula of the form $\forall \mathbf{yz}.(\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z}) \rightarrow \phi(\mathbf{x}, \mathbf{y}))$ is called universally quantified. ◇

The following Lemma can be shown by elementary formula manipulations that exploit that every $z \in \mathbf{z}$ occurs exactly once in α.

Lemma 7.3
Let $\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z})$ be a clique-formula for $\mathbf{x}, \mathbf{y}$. Then

$$\forall \mathbf{yz}.(\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z}) \rightarrow \phi(\mathbf{x}, \mathbf{y})) \equiv \ \forall \mathbf{y}.(\exists \mathbf{z}.\alpha(\mathbf{x}, \mathbf{y}, \mathbf{z}) \rightarrow \phi(\mathbf{x}, \mathbf{y})).$$

The use of the name CGF for both the semantic and the syntactic clique guarded fragment is justified by the following Lemma.

Lemma 7.4
Over any finite relational vocabulary the syntactic and semantic versions of the CGF *are equally expressive.*

Proof sketch: By some elementary equivalence transformations, every syntactically clique guarded formula can be brought into a form where switching from standard semantics to clique guarded semantics does not change its meaning. Conversely, for any finite signature there is a finite disjunction $\mathit{clique}(\mathbf{x}, y, \mathbf{z})$ of clique-formulas for $\mathbf{x}, y$ such that $\mathbf{a}, b$ form a clique in $G(\mathfrak{A})$ iff $\mathfrak{A} \models \exists \mathbf{z}.\mathit{clique}(\mathbf{a}, b, \mathbf{z})$. By guarding every quantifier with such a formula and applying some elementary formula transformations and Lemma 7.3, we get, for every FO formula ψ, a syntactically clique guarded formula that is equivalent to ψ under clique guarded semantics. If we fix a finite relational vocabulary, this transformation is polynomial in the number of variables of the formula, or, more precisely, the maximal number of free variables of all sub-formulas. ■

In the following we will only consider the syntactic variant of the clique guarded fragment.

Definition 7.5 (NNF, Closure, Width)
In the following, all formulas are assumed to be in negation normal form (NNF), where negation occurs only in front of atomic formulas. Every formula in CGF *can be transformed into NNF in linear time by pushing negation inwards using DeMorgan's law and the duality of the quantifiers.*

For a sentence $\psi \in$ CGF *in* NNF, *let* $\mathit{clos}(\psi)$ *be the smallest set that contains* ψ *and is closed under sub-formulas. Let* C *be a set of constants. With* $\mathit{clos}(\psi, C)$ *we denote the set*

$$\mathit{clos}(\psi, C) = \{\phi(\mathbf{a}) \ : \ \mathbf{a} \subseteq C, \phi(\mathbf{x}) \in \mathit{clos}(\psi)\}.$$

The width *of a formula* $\psi \in$ CGF *is defined by*

$$\mathrm{width}(\psi) := \max\{|\mathrm{free}(\phi)| \ : \ \phi \in \mathit{clos}(\psi)\}.$$

◇

7.2 Reasoning with Guarded Fragments

Many of the approaches for decision procedures for description and modal logics described in Section 3.1 have been successfully applied for CGF: Grädel (1999a) shows decidability of (a fixed point extension of) CGF using translation to the monadic second-order theory of countable trees $S\omega S$ (Rabin, 1969) and to the modal μ-calculus with backwards modalities (Vardi, 1998). Also in (Grädel, 1999a), it is shown that CGF is 2-ExpTime-complete and

ExpTime-complete for sentences of bounded width, where the upper bound is based on a reduction to emptiness of alternating two-way automata(Vardi, 1998). Resolution based decision procedures for guarded fragments are described in (Ganzinger & de Nivelle, 1999; de Nivelle & de Rijke, 2000) where the approach in (Ganzinger & de Nivelle, 1999) can be extended to CGF. Finally, a tableau decision procedure for a fragment of GF is given in (Marx, Schlobach, & Mikulás, 2000). Yet, to the best of our knowledge, there does not exist a tableau decision procedure that is capable of deciding the full GF, let alone CGF. In the following, we will supply such an algorithm. As it turns out, the algorithm as well as the proof of its correctness mainly employ ideas we have already encountered in the algorithms and proofs for DLs in this thesis—another indication for the modal nature of CGF, since it is amenable to the same techniques successfully used for description and modal logics.

Let us briefly recall the main "ingredients" of tableau algorithms for modal or description logics like the ones encountered in this thesis. Satisfiability of a concept C is decided by a syntactically guided search for a model for C. Models are usually represented by a graph in which the nodes correspond to elements and the edges correspond to the role relations in the model. Each node is labelled with a set of concepts that this node must satisfy, and new edges and nodes are created as required by existential restrictions. Since many modal and description logics have the tree model property, the graphs generated by these algorithms are trees, which allows for simpler algorithms and easier implementation and optimization of these algorithms. Indeed, some of the fastest implementations of modal or description logics satisfiability algorithms use tableau calculi (Horrocks, 2000; Patel-Schneider, 2000).

For many modal or description logics, e.g. K or $\mathcal{ALC}$, termination of these algorithms is due to the fact that the nesting of universal or existential restrictions of the concepts appearing at a node strictly decreases with every step from the root of the tree (e.g., compare Lemma 3.4). For other logics, e.g., $\mathsf{K4}$, K with the universal modality, or the DLs $\mathcal{SI}$ and $\mathcal{SHIQ}$, this is no longer true and termination has to be enforced by other means. One possibility for this is *blocking*, i.e., stopping the creation of new successor nodes below a node v if there already is an ancestor node w that is labelled with similar concepts as v (e.g., compare Lemma 6.6). Intuitively, in this case the model can fold back from the predecessor of v to w, creating a cycle. Unraveling of these cycles recovers an (infinite) tree model. Since the algorithms guarantee that the concepts occurring in the label of the nodes stem from a finite set (usually the sub-concepts of the input concept), every growing path will eventually contain a blocked node, preventing further growth of this path and (together with a bound on the degree of the tree) ensuring termination of the algorithm.

7.2.1 Tableau Reasoning for CGF

Our investigation of a tableau algorithm for CGF starts with the observation that CGF also has some kind of tree model property.

Definition 7.6
Let τ be a relational vocabulary. A τ-structure $\mathfrak{A}$ has tree width k *if $k \in \mathbb{N}$ is minimal with the following property.*

There exists a directed tree $T = (V, E)$ and a function $f : V \to 2^A$ such that

- *for every $v \in V$, $|f(v)| \leq k + 1$,*
- *for every $R \in \tau$ and $\mathbf{a} \in R^{\mathfrak{A}}$, there exists $v \in V$ with $\mathbf{a} \subseteq f(v)$, and*
- *for every $a \in A$, the set $V_a = \{v \in V \ : \ a \in f(v)\}$ induces a subtree of T.*

Every node v of T induces a substructure $\mathfrak{F}(v) \subseteq \mathfrak{A}$ of cardinality at most $k+1$. The tuple $\langle T, (\mathfrak{F}(v))_{v \in T}\rangle$ is called a tree decomposition *of $\mathfrak{A}$.*

A logic $\mathcal{L}$ has the generalised tree model property *if there exists a computable function t, assigning to every sentence $\psi \in \mathcal{L}$ a natural number $t(\psi)$ such that, if ψ is satisfiable, then ψ has a model of tree width at most $t(\psi)$.* $\diamond$

Fact 7.7 (Tree Model Property for CGF**)**
Every satisfiable sentence $\psi \in$ CGF of width k has a countable model of tree width at most $k - 1$.

This is a simple corollary of (Grädel, 1999a, , Theorem 4), where the same result is given for μCGF, that is CGF extended by a least fixed point operator.

Fact 7.7 is the starting point for our definition of a *completion tree* for a formula $\psi \in$ CGF. A node v of such a tree no longer stands for a single element of the model (as in the modal case), but rather for a substructure $\mathfrak{F}(v)$ of a tree decomposition of the model. To this purpose, we label every node v with a set $\mathbf{C}(v)$ of constants (the elements of the substructure) and a subset of $cl(\psi, \mathbf{C}(v))$, reflecting the formulas that must hold true for these elements.

To deal with *auxiliary elements*—elements helping to form a clique in $G(\mathfrak{A})$ that are not part of this clique themselves—we will use the auxiliary constant symbol $*$ as a placeholder for unspecified elements in atoms. The intention is to keep the number of constants at each node as small as possible. The $*$ will be used for the extra elements occurring in clique formulas that are not part of the clique itself.

The following definitions are useful when dealing with these generalised atoms.

Definition 7.8
Let K denote an infinite set of constants and $* \notin K$. For any set of constants $C \subseteq K$ we set $C^* = C \cup \{*\}$. We use $t_1, t_2, \ldots$ to range over elements of K^*. The relation $\geq^*$ is defined by

$$Rt_1 \ldots t_n \geq^* Rt'_1 \ldots t'_n \text{ iff for all } i \in \{1 \ldots n\} \text{ either } t_i = * \text{ or } t_i = t'_i.$$

For an atom β and a set of formulas Φ we define $\beta \in^* \Phi$ iff there is a $\beta' \in \Phi$ with $\beta \geq^* \beta'$.

For a set of constants $C \subseteq K$ and an atom $\beta = Rt_1 \ldots t_n$, we define

$$\beta \circledast C = Rt'_1 \ldots t'_n \text{ where } t'_i = \begin{cases} t_i & \text{if } t_i \in C, \\ * & \text{otherwise.} \end{cases}$$

$\diamond$

We use the notation $\mathbf{a}^*$ to indicate that the tuple $\mathbf{a}^*$ may contain $*$'s. Obviously, $\geq^*$ is transitive and reflexive, and $\beta \circledast C \geq^* \beta$ for all atoms β and sets of constants C.

While these are all syntactic notions, they have a semantic counterpart that clarifies the intuition of $*$ standing for an unspecified element. Let $\mathbf{a}'$ denote the tuple obtained from a tuple $\mathbf{a}^*$ by replacing every occurrence of $*$ in $\mathbf{a}^*$ with a distinct fresh variable, and let $\mathbf{z}$ be precisely the variables used in this replacement. For an atom β, we define

$$\mathfrak{A} \models \beta(\mathbf{a}^*) \text{ iff } \mathfrak{A} \models \exists \mathbf{z}.\beta(\mathbf{a}').$$

It is easy to see that

$$\begin{aligned} \beta(\mathbf{a}) \geq^* \beta(\mathbf{b}) \quad &\text{implies} \quad \beta(\mathbf{b}) \models \beta(\mathbf{a}), \text{ and} \\ \beta(\mathbf{a}) \in^* \Phi \quad &\text{implies} \quad \Phi \models \beta(\mathbf{a}) \end{aligned}$$

because, if $\mathbf{a} \geq^* \mathbf{b}$, then $\mathbf{b}$ is obtained from $\mathbf{a}$ by replacing some $*$ with constants, which provide witnesses for the existential quantifier.

We further write $\Phi|_C$ to denote the subset of Φ containing all formulas that only use constants in C.

Algorithm 7.9 (The CGF**-algorithm)**
Let $\psi \in$ CGF be a closed formula in NNF. *A* completion tree $\mathbf{T} = (\mathbf{V}, \mathbf{E}, \mathbf{C}, \Delta, \mathbf{N})$ *for ψ is a node labelled tree $(\mathbf{V}, \mathbf{E})$ with the labelling function $\mathbf{C}$ labelling each node $v \in \mathbf{V}$ with a subset of K, Δ labelling each node $v \in \mathbf{V}$ with a subset of $cl(\psi, \mathbf{C}(v)^*)$ where all formulas $\beta(\mathbf{x}, *, \ldots, *) \in \Delta(v)$ using $*$ are atoms (excluding equality statements), and the function $\mathbf{N} : \mathbf{V} \to \mathbb{N}$ mapping each node to a distinct natural number, with the additional property that, if v is an ancestor of w, then $\mathbf{N}(v) < \mathbf{N}(w)$.*

A constant $c \in K$ is called shared *between two nodes $v_1, v_2 \in \mathbf{V}$, if $c \in \mathbf{C}(v_1) \cap \mathbf{C}(v_2)$, and $c \in \mathbf{C}(w)$ for all nodes w on the (unique, undirected, possibly empty) shortest path connecting v_1 to v_2.*

A node $v \in \mathbf{V}$ is called directly blocked[1] *by a node $w \in \mathbf{V}$, if w is not blocked, $\mathbf{N}(w) < \mathbf{N}(v)$, and there is an injective mapping π from $\mathbf{C}(v)$ into $\mathbf{C}(w)$ such that, for all constants $c \in \mathbf{C}(v)$ that are shared between v and w, $\pi(c) = c$, and $\pi(\Delta(v)) = \Delta(w)|_{\pi(\mathbf{C}(v)^*)}$. Here and throughout this thesis we use the convention $\pi(*) = *$ for every function π that verifies a blocking.*

A node is called blocked *if it is directly blocked or if its predecessor is blocked.*

A completion tree $\mathbf{T}$ contains a clash *if there is a node $v \in \mathbf{V}$ such that*

[1] The definition of blocking is recursive. Like for the $\mathcal{SI}$- and the $\mathcal{SHIQ}$-algorithm, this does not cause any problems because the status of a node v only depends on its label and the status of nodes w with $\mathbf{N}(w) < \mathbf{N}(v)$. The recursion terminates at the root node, where the $\mathbf{N}$-value is minimal.

- *for a constant* $c \in \mathbf{C}(v)$, $c \neq c \in \Delta(v)$, *or*
- *there is an atomic formula* β *and a tuple* $\mathbf{a} \subseteq \mathbf{C}(v)$ *such that* $\{\beta(\mathbf{a}), \neg\beta(\mathbf{a})\} \subseteq \Delta(v)$.

Otherwise, $\mathbf{T}$ *is called* clash-free*. A completion tree* $\mathbf{T}$ *is called* complete *if none of the* completion rules *given in Figure 7.1 can be applied to* $\mathbf{T}$*. A complete and clash-free completion tree for* ψ *is called* a tableau *for* ψ*.*

To test ψ *for satisfiability, the tableau algorithm creates an initial tree with only a single node* v_0*,* $\Delta(v_0) = \{\psi\}$ *and* $\mathbf{C}(v_0) = \{a_0\}$ *for an arbitrary constant* a_0*. The rules from Figure 7.1 are applied until either a clash occurs, producing output "*ψ *is not satisfiable", or the tree is complete, in which case "*ψ *is satisfiable" is output.* ◇

The set $\mathbf{C}(v_0)$ is initialized with a non-empty set of constants to make sure that empty structures are excluded. For a discussion of the different kinds of non-determinism that occur in the CGF-algorithm, see below Lemma 7.12.

While our notion of tableaux has many similarities to the tableaux appearing in (Grädel & Walukiewicz, 1999), there are two important differences that make the version used here more suitable as basis for a tableau algorithm. We will see that every completion tree generated by the tableau algorithm is finite. Conversely, tableaux in (Grädel & Walukiewicz, 1999), in general, can be infinite. Also, in (Grädel & Walukiewicz, 1999) every node is labelled with a complete $(\psi, \mathbf{C}(v))$-type, i.e., every formula $\phi \in \mathit{clos}(\psi, \mathbf{C}(v))$ is explicitly asserted true of false at v. Conversely, a completion tree contains only assertions about relevant formulas. This implies a lower degree of non-determinism in the algorithm, which is important for an efficient implementation.

7.2.2 Correctness

The techniques used to establish correctness of the CGF-algorithm bear a strong resemblance to the techniques we have employed for the tableau algorithms for description logics in the previous chapters. Extra complexity is added by the fact that completion trees for CGFare more complex objects than the completion trees for Description Logics, mainly because each node now stands for a substructure rather than for a single element of the model.

Termination

The following technical lemma is a consequence of the completion rules and the blocking condition.

Lemma 7.10
Let $\psi \in \mathrm{CGF}$ *be a sentence in NNF with* $|\psi| = n$*,* $\mathrm{width}(\psi) = m$*, and* $\mathbf{T}$ *a completion tree generated for* ψ *by application of the rules in Figure 7.1. For every node* v *in* $\mathbf{T}$*,*

1. $|\mathbf{C}(v)| \leq m$,

2. $|\Delta(v)| \leq n \times (m+1)^m$, *and*

3. any $\ell > 2^{n \times (m+1)^m}$ *distinct nodes in* $\mathbf{T}$ *contain a blocked node.*

Proof. Nodes are only generated when initializing the tree (with a single constant) and by the $\rightarrow_\exists$-rule and no constants are added to a $\mathbf{C}(v)$ once v has been generated (but some may be removed by application of the $\rightarrow_=$-rule).

When triggered by the formula $(\exists \mathbf{yz}.\alpha(\mathbf{a},\mathbf{y},\mathbf{z}))\phi(\mathbf{a},\mathbf{y})$, the $\rightarrow_\exists$-rule initializes $\mathbf{C}(w)$ such that it contains $\mathbf{a}$ and another constant for every variable in $\mathbf{x}$ and $\mathbf{y}$. Hence,

$$|\mathbf{C}(w)| \leq |\mathbf{a} \cup \mathbf{y} \cup \mathbf{z}| \leq |\mathrm{free}(\alpha)| \leq \mathrm{width}(\psi).$$

The set $\Delta(v)$ is a subset of $cl(\psi, \mathbf{C}(v)^*)$, for which $|cl(\psi, \mathbf{C}(v))| \leq n \times (m+1)^m$ holds because there are at most n formulas in $cl(\psi)$, each of which has at most m free variables.

Figure 7.1 The completion rules for CGF

$\rightarrow_\wedge$:	**if**	1.	$\phi \wedge \theta \in \Delta(v)$
		2.	$\{\phi, \theta\} \not\subseteq \Delta(v)$
	then		$\Delta(v) \rightarrow_\wedge \Delta(v) \cup \{\phi, \theta\}$
$\rightarrow_\vee$:	**if**	1.	$\phi \vee \theta \in \Delta(v)$ and
		2.	$\{\phi, \theta\} \cap \Delta(v) = \emptyset$
	then		$\Delta(v) \rightarrow_\vee \Delta(v) \cup \{\chi\}$ for some $\chi \in \{\phi, \theta\}$
$\rightarrow_=$:	**if**		$a = b \in \Delta(v)$ and $a \neq b$
	then		for all w that share a with v, $\mathbf{C}(w) \rightarrow_= (\mathbf{C}(w) \setminus \{a\}) \cup \{b\}$
			and $\Delta(w) \rightarrow_= \Delta(w)[a \mapsto b]$
$\rightarrow_\forall$:	**if**	1.	$(\forall \mathbf{yz}.\alpha(\mathbf{a},\mathbf{y},\mathbf{z}))\phi(\mathbf{a},\mathbf{y}) \in \Delta(v)$, and
		2.	there exists $\mathbf{b} \subseteq \mathbf{C}(v)$ such that for all $\beta(\mathbf{x},\mathbf{y},\mathbf{z}) \in \alpha$, $\beta(\mathbf{a},\mathbf{b},*\cdots*) \in^* \Delta(v)$, and
		3.	$\phi(\mathbf{a},\mathbf{b}) \notin \Delta(v)$
	then		$\Delta(v) \rightarrow_\forall \Delta(v) \cup \{\phi(\mathbf{a},\mathbf{b})\}$
$\rightarrow_\exists$:	**if**	1.	$(\exists \mathbf{yz}.\alpha(\mathbf{a},\mathbf{y},\mathbf{z}))\phi(\mathbf{a},\mathbf{y}) \in \Delta(v)$, and
		2.	for every $\mathbf{b},\mathbf{c} \subseteq \mathbf{C}(v)$, $\{\alpha(\mathbf{a},\mathbf{b},\mathbf{c}), \phi(\mathbf{a},\mathbf{b})\} \not\subseteq \Delta(v)$, and
		3.	there is no child w of v with $\{\alpha(\mathbf{a},\mathbf{b},\mathbf{c}), \phi(\mathbf{a},\mathbf{b})\} \subseteq \Delta(w)$ for some $\mathbf{b},\mathbf{c} \subseteq \mathbf{C}(w)$, and
		4.	v is not blocked
	then		$\mathbf{V} \rightarrow_\exists \mathbf{V} \cup \{w\}$, $\mathbf{E} \rightarrow_\exists \mathbf{E} \cup \{(v,w)\}$ for a fresh node w
			let $\mathbf{b},\mathbf{c}$ be sequences of distinct and fresh constants that match
			the lengths of $\mathbf{y},\mathbf{z}$ and set
			$\mathbf{C}(w) = \mathbf{a} \cup \mathbf{b} \cup \mathbf{c}$, and
			$\Delta(w) = \{\alpha(\mathbf{a},\mathbf{b},\mathbf{c}), \phi(\mathbf{a},\mathbf{b})\}$, and
			$\mathbf{N}(w) = 1 + \max\{\mathbf{N}(v) \ : \ v \in \mathbf{V} \setminus \{w\}\}$
$\rightarrow_\updownarrow$:	**if**	1.	$\beta(\mathbf{a}^*) \in \Delta(v)$, β atomic, not an equality, and
		2.	w is a neighbour of v with $\mathbf{a}^* \cap \mathbf{C}(w) \neq \emptyset$, and
		3.	$\beta(\mathbf{a}^*) \circledast \mathbf{C}(w) \notin \Delta(w)$
	then		$\Delta(w) \rightarrow_\updownarrow \Delta(w) \cup \{\beta(\mathbf{a}^*) \circledast \mathbf{C}(w)\}$
$\rightarrow_{\updownarrow\forall}$:	**if**	1.	$\phi(\mathbf{a}) \in \Delta(v)$, $\phi(\mathbf{a})$ is universally. quantified, and
		2.	w is a neighbour of v with $\mathbf{a} \subseteq \mathbf{C}(w)$, and
		3.	$\phi(\mathbf{a}) \notin \Delta(w)$
	then		$\Delta(w) \rightarrow_{\updownarrow\forall} \Delta(w) \cup \{\phi(\mathbf{a})\}$

There are at most $(|\mathbf{C}(v)|+1)^m$ distinct sequences of length m with constants from $\mathbf{C}(v)^*$.

Let $v_1, \ldots, v_\ell$ be $\ell > 2^{n\times(m+1)^m}$ distinct nodes. For every v_i, we will construct an injective mapping $\pi_i : \mathbf{C}(v_i) \to \{1, \ldots m\}$ such that, if a constant a is shared between two nodes v_i, v_j, then $\pi_i(a) = \pi_j(a)$.

Let $u_1, \cdots, u_k$ denote the nodes of a subtree of $\mathbf{T}$ that contains every node v_i and that is rooted at u_1. By induction over the distance to u_1, we define an injective mapping $\nu_i : \mathbf{C}(u_i) \to \{1, \ldots, m\}$ for every $i \in \{1, \ldots, k\}$ as follows. For ν_1 we pick an arbitrary injective function from $\mathbf{C}(u_1)$ to $\{1, \ldots, m\}$. For a node u_i let u_j be the predecessor of u_i in $\mathbf{T}$ and ν_j the corresponding function, which has already been defined because u_j has a smaller distance to u_1 than u_i. For ν_i we choose an arbitrary injective function such that $\nu_i(a) = \nu_j(a)$ for all $a \in \mathbf{C}(u_i) \cap \mathbf{C}(u_j)$.

All mappings ν_i are injective. For any constant a the set $\mathbf{V}_a := \{v \in \mathbf{V} \mid a \in \mathbf{C}(v)\}$ induces a subtree of $\mathbf{T}$. If $u_i, u_j \in \mathbf{V}_a$ are neighbours, the definition above ensures $\nu_i(a) = \nu_j(a)$. By induction over the length of the shortest connecting path we obtain the same for arbitrary $u_i, u_j \in \mathbf{V}_a$.

For every node v_i there is a j_i such that $v_i = u_{j_i}$ and we set $\pi_i = \nu_{j_i}$. There are at most $2^{n\times(m+1)^m}$ distinct subsets of $cl(\psi, \{1, \ldots, m, *\})$. Hence, there must be two nodes v_i, v_j such that $\pi_i(\Delta(v_i)) = \pi_j(\Delta(v_j))$ and, w.l.o.g., $\mathbf{N}(v_i) < \mathbf{N}(v_j)$. This implies that v_j is blocked by v_i via $\pi := \pi_i^{-1} \circ \pi_j$. Note that for π to be well-defined, π_i must be injective. By construction, π preserves shared constants. Since $\pi_i(\Delta(v_i)) = \pi_j(\Delta(v_j))$, $\pi(\Delta(v_j)) = \Delta(v_i)|_{\pi(\mathbf{C}(v_j))}$ holds. ∎

Lemma 7.11 (Termination)
Let $\psi \in$ CGF be a sentence in NNF. Any sequence of rule applications of the tableau algorithm starting from the initial tree terminates.

Proof. For any completion tree $\mathbf{T}$ generated by the tableau algorithm, we define $\|\cdot\| : \mathbf{V} \mapsto \mathbb{N}^3$ by

$$\begin{aligned}\|v\| := (&|\mathbf{C}(v)|, \quad n \times (m+1)^m - |\Delta(v)|,\\ &|\{\phi \in \Delta(v) : \phi \text{ triggers the } \to_\exists\text{-rule for } v\}|).\end{aligned}$$

The lexicographic order $\prec$ on $\mathbb{N}^3$ is well-founded, i.e. it has no infinite decreasing chains. Any rule application decreases $\|v\|$ w.r.t. $\prec$ for at least one node v, and never increases $\|v\|$ w.r.t. $\prec$ for an existing node v. However it may create new successors, one at a time. Since $\prec$ is well-founded, there can only be a finite number of applications of rules to every node in $\mathbf{T}$ and hence a finite number of successors and an infinite sequence of rule applications would generate a tree of infinite depth.

Yet, as a corollary of Lemma 7.10, we have that the depth of $\mathbf{T}$ is bounded by $2^{n\times(m+1)^m}$. For assume that there is a path of length $> 2^{n\times(m+1)^m}$ in $\mathbf{T}$ with deepest node v. By the time v has been created (by an application of the $\to_\exists$-rule to its predecessor u), the path from the root of $\mathbf{T}$ to u contains at least $2^{n\times(m+1)^m}$ nodes, and hence a blocked node. This implies that u is blocked too, and the $\to_\exists$-rule cannot be applied to create v. ∎

Completeness

Lemma 7.12
Let $\psi \in$ CGF be a closed formula in NNF. If ψ is satisfiable, then there is a sequence of rule applications starting from the initial tree that yields a tableau.

Proof. Since ψ is satisfiable, there is a model $\mathfrak{A}$ of ψ. We will use $\mathfrak{A}$ to guide the application of the non-deterministic $\rightarrow_\vee$-rule. For this we incremently define a function $g : \bigcup\{\mathbf{C}(v) \mid v \in \mathbf{V}\} \rightarrow A$ such that for all $v \in \mathbf{V} \;:\; \mathfrak{A} \models g(\Delta(v))$. We refer to this property by (§).

The set $\Delta(v)$ can contain atomic formulas $\alpha(\mathbf{a}^*)$, where $*$ occurs at some positions of $\mathbf{a}^*$. The constant $*$ is not mapped to an element of A by g. We deal with this as described just after Definition 7.8 by setting

$$\mathfrak{A} \models g(\alpha(\mathbf{a}^*)) \;\text{ iff }\; \mathfrak{A} \models \exists \mathbf{z}.g(\alpha(\mathbf{a}')).$$

Claim 7.13 If, for a completion tree $\mathbf{T}$, there exists a function g, such that (§) holds and a rule is applicable to $\mathbf{T}$, then it can be applied in a way that maintains (§).

- For the $\rightarrow_\wedge$- and the $\rightarrow_\vee$-rule this is obvious.

- If the $\rightarrow_=$-rule is applicable to $v \in \mathbf{V}$ with $a = b \in \Delta(v)$, then, since $\mathfrak{A} \models g(a) = g(b)$, $g(a) = g(b)$ must hold. Hence, for every node w that shares a with v, $g(\Delta(w)) = g(\Delta(w)[a \mapsto b])$ and the rule can be applied without violating (§).

- If the $\rightarrow_\forall$-rule is applicable to $v \in \mathbf{V}$ with $(\forall \mathbf{yz}.\alpha(\mathbf{a},\mathbf{y},\mathbf{z}))\phi(\mathbf{a},\mathbf{y}) \in \Delta(v)$ and $\mathbf{b} \subseteq \mathbf{C}(v)$ with $\beta(\mathbf{a},\mathbf{b},*\cdots*) \in^* \Delta(v)$ for all atoms $\beta(\mathbf{x},\mathbf{y},\mathbf{z}) \in \alpha$, then, from the definition of $\in^*$, there is a tuple $\mathbf{c}^* \subseteq \mathbf{C}(v)^*$, such that $\beta(\mathbf{a},\mathbf{b},*\cdots*) \geq^* \beta(\mathbf{a},\mathbf{b},\mathbf{c}^*)$ and $\beta(\mathbf{a},\mathbf{b},\mathbf{c}^*) \in \Delta(v)$. From (§) we get that $\mathfrak{A} \models \exists \mathbf{z}.\beta(g(\mathbf{a}),g(\mathbf{b}),\mathbf{z})$ and since every $z \in \mathbf{z}$ appears exactly once in α, also $\mathfrak{A} \models \exists \mathbf{z}.\alpha(g(\mathbf{a}),g(\mathbf{b}),\mathbf{z})$. Hence, we have

  $$\{\mathfrak{A} \models \{\forall \mathbf{yz}.\alpha(g(\mathbf{a}),\mathbf{y},\mathbf{z})) \rightarrow \phi(g(\mathbf{a}),\mathbf{y}), \exists \mathbf{z}.\alpha(g(\mathbf{a}),g(\mathbf{b}),\mathbf{z})\},$$

  which, by Lemma 7.3, implies $\mathfrak{A} \models \phi(g(\mathbf{a}),g(\mathbf{b}))$ and hence $\phi(\mathbf{a},\mathbf{b})$ can be added to $\Delta(v)$ without violating (§).

- If the $\rightarrow_\exists$-rule is applicable to $v \in \mathbf{V}$ with $(\exists \mathbf{yz}.\alpha(\mathbf{a},\mathbf{y},\mathbf{z}))\phi(\mathbf{a},\mathbf{y})$, then this implies

  $$\mathfrak{A} \models g((\exists \mathbf{yz}.\alpha(\mathbf{a},\mathbf{y},\mathbf{z}))\phi(\mathbf{a},\mathbf{y})).$$

  Hence, there are sequences $\mathbf{b}', \mathbf{c}' \subseteq A$ such that $\mathfrak{A} \models \{\alpha(g(\mathbf{a}),\mathbf{b}',\mathbf{c}'), \phi(g(\mathbf{a}),\mathbf{b}')\}$. If we define g such that $g(\mathbf{b}) = \mathbf{b}'$ and $g(\mathbf{c}) = \mathbf{c}'$, then $\mathfrak{A} \models \{g(\alpha(\mathbf{a},\mathbf{b},\mathbf{c}), g(\phi(\mathbf{a},\mathbf{b}))\}$. Note, that this might involve setting $g(b_1) = g(b_2)$ for some $b_1, b_2 \in \mathbf{b}$. With this construction the resulting extended completion-tree $\mathbf{T}$ and extended function g again satisfy (§).

- If the $\rightarrow_{\updownarrow}$-rule is applicable to $v \in \mathbf{V}$ with $\beta(\mathbf{a}^*) \in \Delta(v)$ and a neighbour w with $\mathbf{a}^* \cap \mathbf{C}(w) \neq \emptyset$, then it adds $\beta(\mathbf{a}^*) \circledast \mathbf{C}(w)$ to $\Delta(w)$. From (§) we get that $\mathfrak{A} \models \beta(g(\mathbf{a}^*))$, and since $\beta(\mathbf{b}^*) := \beta(\mathbf{a}^*) \circledast \mathbf{C}(w) \geq^* \beta(\mathbf{a}^*)$, this implies $\mathfrak{A} \models \beta(g(\mathbf{b}^*))$. Hence, adding $\beta(\mathbf{a}^*) \circledast \mathbf{C}(w) = \beta(\mathbf{b}^*)$ to $\Delta(w)$ does not violate (§).
- If the $\rightarrow_{\updownarrow\forall}$-rule is applicable to a node $v \in \mathbf{V}$ with a universally quantified formula $\phi(\mathbf{a}) \in \Delta(v)$ and a neighbour w which shares $\mathbf{a}$ with v, (§) yields $\mathfrak{A} \models \phi(g(\mathbf{a}))$. Hence, adding $\phi(\mathbf{a})$ to $\Delta(w)$ does not violate (§).

Claim 7.14 A completion-tree $\mathbf{T}$ for which a function g exists such that (§) holds is clash free.

Assume that $\mathbf{T}$ contains a clash, namely, there is a node $v \in \mathbf{V}$ such that either $a \neq a \in \mathbf{V}(v)$—implying $\mathfrak{A} \models g(a) \neq g(a)$—, or that there is a sequence $\mathbf{a} \subseteq \mathbf{C}(v)$, and an atomic formula β such that $\{\beta(\mathbf{a}), \neg\beta(\mathbf{a})\} \subseteq \Delta(v)$. From (§), $\mathfrak{A} \models \{\beta(g(\mathbf{a})), \neg\beta(g(\mathbf{a}))\}$ would follow, also a contradiction.

These claims yield Lemma 7.12 as follows. Let $\mathbf{T}$ be a tableau for ψ. Since $\mathfrak{A} \models \psi$, (§) is satisfied for the initial tree together with the function g mapping a_0 to an arbitrary element of the universe of $\mathfrak{A}$. By Lemma 7.11, any sequence of applications is finite, and from Claim 7.13 we get that there is a sequence of rule-applications that maintains (§). By Claim 7.14, this sequence results in a tableau. This completes the proof of Lemma 7.12.

■

Lemma 7.12 involves two different kinds of non-determinism, namely, the choice which rule to apply to which constraint (as several rules might be applicable simultaneously), and which disjunct to choose in an application of the $\rightarrow_{\vee}$-rule. While the latter choice is *don't-know* non-deterministic, i.e., for a satisfiable formula only certain choices will lead to the discovery of a tableau, the former choice is *don't-care* non-deterministic. This means that arbitrary choices of which rule to apply next will lead to the discovery of a tableau for a satisfiable formula. For an implementation of the tableau algorithm this has the following consequences. Exhaustive search is necessary to deal with all possible expansions of the $\rightarrow_{\vee}$-rule, but arbitrary strategies of choosing which rule to apply next, and where to apply it, will lead to a correct implementation, although the efficiency of the implementation will strongly depend on a sophisticated strategy.

Soundness

In order to prove the correctness of the tableau algorithm we have to show that the existence of a tableau for ψ implies satisfiability of ψ. To this purpose, we will construct a model from a tableau. From the construction employed in the proof we obtain an alternative proof of Fact 7.7.

Lemma 7.15
Let $\psi \in \mathrm{CGF}[\tau]$ with $k = \mathrm{width}(\psi)$ and let $\mathbf{T}$ be a tableau for ψ generated by the tableau algorithm. Then ψ is satisfiable and has a model of tree width at most $k - 1$.

Proof. Let $\mathbf{T} = (\mathbf{V}, \mathbf{E}, \mathbf{C}, \Delta, \mathbf{N})$ a tableau for ψ. For every direct blocking situation we fix a mapping π verifying this blocking. Using an unraveling construction, we will construct a model $\mathfrak{A}$ for ψ of width at most $k-1$ from $\mathbf{T}$. First, we "unravel" blocking situations in $\mathbf{T}$ by successively replacing every blocked node with a copy of the subtree of $\mathbf{T}$ rooted at the blocking node. Formally, this is achieved by the following path construction. We define

$$\mathbf{V}_u = \{v \in \mathbf{V} \ : \ v \text{ is not blocked or directly blocked } \}.$$

Since from now on we only deal with nodes from $\mathbf{V}_u$, every blocking is direct and we will no longer explicitly mention this fact.

The set $\mathsf{Paths}(\mathbf{T})$ is inductively defined by[2]

- $[\frac{v_0}{v_0}] \in \mathsf{Paths}(\mathbf{T})$ for the root v_0 of $\mathbf{T}$,
- if $[\frac{v_1}{v_1'} \dots \frac{v_n}{v_n'}] \in \mathsf{Paths}(\mathbf{T})$, the node w is a successor of v_n and w is not blocked, then $[\frac{v_1}{v_1'} \dots \frac{v_n}{v_n'} \frac{w}{w}] \in \mathsf{Paths}(\mathbf{T})$,
- if $[\frac{v_1}{v_1'} \dots \frac{v_n}{v_n'}] \in \mathsf{Paths}(\mathbf{T})$, w is a successor of v_n blocked by the node $u \in \mathbf{V}$, then $[\frac{v_1}{v_1'} \dots \frac{v_n}{v_n'} \frac{u}{w}] \in \mathsf{Paths}(\mathbf{T})$.

The set $\mathsf{Paths}(\mathbf{T})$ forms a tree, with p' being a successor of p if p' is obtained from p by concatenating one element $\frac{u}{w}$ at the end. We define the auxiliary functions $\mathsf{Tail}, \mathsf{Tail}'$ by setting $\mathsf{Tail}(p) = v_n$ and $\mathsf{Tail}'(p) = v_n'$ for every path $p = [\frac{v_1}{v_1'} \dots \frac{v_n}{v_n'}]$.

Intuitively, for every node v of $\mathbf{T}$, the paths $p \in \mathsf{Paths}(\mathbf{T})$ with $v = \mathsf{Tail}(p)$ stand for distinct copies of v created by the unraveling. The universe of $\mathfrak{A}$ consists of (classes of) constants labelling nodes in $\mathbf{T}$ paired with the paths at whose Tail they appear to distinguish constants occurring at different copies of a node of $\mathbf{T}$. Formally, we define

$$\mathbf{C}(\mathbf{T}) = \{(a, p) \ : \ p \in \mathsf{Paths}(\mathbf{T}) \wedge a \in \mathbf{C}(\mathsf{Tail}(p))\}.$$

Constants appearing at consecutive nodes of $\mathbf{T}$ stand for the same element and the same holds for constants related by a mapping π verifying a block. Hence, to obtain the universe of $\mathfrak{A}$, we factorize $\mathbf{C}(\mathbf{T})$ as follows. Let $\sim$ be the smallest symmetric relation on $\mathbf{C}(\mathbf{T})$ satisfying

- $(a, p) \sim (a, q)$ if q is a successor of p in $\mathsf{Paths}(\mathbf{T})$, $\mathsf{Tail}'(q)$ is an unblocked successor of $\mathsf{Tail}(p)$, and $a \in \mathbf{C}(\mathsf{Tail}(p)) \cap \mathbf{C}(\mathsf{Tail}'(q))$,
- $(a, p) \sim (b, q)$ if q is a successor of p in $\mathsf{Paths}(\mathbf{T})$, $\mathsf{Tail}'(q)$ is a blocked successor of $\mathsf{Tail}(p)$, $a \in \mathbf{C}(\mathsf{Tail}(p)) \cap \mathbf{C}(\mathsf{Tail}'(q))$, and $\pi(a) = b$ for the function π that verifies that $\mathsf{Tail}'(q)$ is blocked by $\mathsf{Tail}(q)$.

[2]This complicated form of unraveling, where we record both blocked and blocking node is necessary because there might be a situation where two successors v_1, v_2 of a node are directly blocked by the same node w.

With $\approx$ we denote the reflexive, transitive closure of $\sim$ and with $[a,p]_\approx$ the class of (a,p), i.e., the set $\{(b,q) \in \mathbf{C}(\mathbf{T}) \mid (b,q) \approx (a,p)\}$. Since $(a,p) \sim (b,q)$ iff p,q are neighbours in $\mathsf{Paths}(\mathbf{T})$, for every (a,p), the set

$$\mathsf{Paths}([a,p]_\approx) := \{q \mid \exists b.(b,q) \in [a,p]_\approx\}$$

is a subtree of $\mathsf{Paths}(\mathbf{T})$.

The classes of $\mathbf{C}(\mathbf{T})/\approx$ will be the elements of the universe of $\mathfrak{A}$. First we need to prove some technicalities for this construction.

Claim 7.16 Let $p \in \mathsf{Paths}(\mathbf{T})$ and $a,b \in \mathbf{C}(\mathsf{Tail}(p))$. Then $(a,p) \approx (b,p)$ iff $a = b$.

Assume the claim does not hold and let $a \neq b$ with $(a,p) \approx (b,p)$. By definition of $\sim$, $(a,p) \not\sim (b,p)$ must hold. Hence, there must be a path $(c_1,p_1) \sim \cdots \sim (c_k,p_k)$ such that $a = c_1$, $b = c_k$, and $p = p_1 = p_k$. W.l.o.g., assume we have picked a,b,p such that this path has minimal length k. Such a minimal path must be of length $k = 3$, for if we assume a path of length $k > 3$, there must be $2 \leq i < j \leq k-1$ such that $p_i = p_j$, because the relation $\sim$ is defined along paths in the tree $\mathsf{Paths}(\mathbf{T})$. If $c_i = c_j$ then we can shorten the path between position i and j and obtain a shorter path. If $c_i \neq c_j$, then the path $(c_i,p_i) \sim \cdots \sim (c_j,p_j)$ is also a shorter path with the same properties. Hence, a minimal path must be of the form $(a,p) \sim (c,q) \sim (b,p)$. If $\mathsf{Tail}'(q)$ is not blocked, by the definition of $\sim$, $a = c = b$ must hold. Hence, since $a \neq b$, $\mathsf{Tail}'(q)$ must be blocked by $\mathsf{Tail}(q)$. From the definition of $\sim$ we have $a,b \in \mathbf{C}(\mathsf{Tail}'(q))$ and $\pi(a) = c = \pi(b)$ for the function π verifying that $\mathsf{Tail}'(q)$ is blocked by $\mathsf{Tail}(q)$. Since π must be injective, this is a contradiction.

Since the set $\mathsf{Paths}(\mathbf{T})$ is a tree, and as a consequence of Claim 7.16, we get the following.

Claim 7.17 Let $p,q \in \mathsf{Paths}(\mathbf{T})$ with $p = [\frac{v_1}{v_1'} \ldots \frac{v_n}{v_n'}]$, $q = [\frac{v_1}{v_1'} \ldots \frac{v_n}{v_n'} \frac{w}{w'}]$. If, for $a \in \mathbf{C}(v_n), b \in \mathbf{C}(w)$, $(a,p) \approx (b,q)$ then $(a,p) \sim (b,q)$.

If $(a,p) \approx (b,q)$ then there must be a path $(c_1,p_1) \sim \cdots \sim (c_k,p_k)$ such that $a = c_1$, $b = c_k$, $p = p_1$, and $q = p_k$. Since $\sim$ is only defined along paths in the tree $\mathsf{Paths}(\mathbf{T})$, there must be a step from p to q (or, dually, from q to p) in this path, more precisely, there must be an $i \in \{1, \ldots k-1\}$ such that $p_i = p$ and $p_{i+1} = q$ holds. Hence, we have the situation

$$(a,p) \approx (c_i,p) \sim (c_{i+1},q) \approx (b,q).$$

Claim 7.16 implies $a = c_i$ and $b = c_{i+1}$ and hence $(a,p) \sim (b,q)$.

Using Claim 7.17, we can show that the blocking condition and the $\rightarrow_{\updownarrow}$- and $\rightarrow_{\updownarrow\forall}$-rule work as desired.

Claim 7.18 Let $p,q \in \mathsf{Paths}(\mathbf{T})$, $\mathbf{a} \subseteq \mathbf{C}(\mathsf{Tail}(p)), \mathbf{b} \subseteq \mathbf{C}(\mathsf{Tail}(q))$, $\mathbf{a},\mathbf{b}$ non-empty tuples, and $(\mathbf{a},p) \approx (\mathbf{b},q)$.

- For every atom β, $\beta(\mathbf{a}, * \cdots *) \in^* \Delta(\mathsf{Tail}(p))$ iff $\beta(\mathbf{b}, * \cdots *) \in^* \Delta(\mathsf{Tail}(q))$.

- For every universally quantified ϕ, $\phi(\mathbf{a}) \in \Delta(\mathsf{Tail}(p))$ iff $\phi(\mathbf{b}) \in \Delta(\mathsf{Tail}(q))$.

Since both propositions are symmetric, we only need to prove one direction. If $(\mathbf{a}, p) \approx (\mathbf{b}, q)$ with $\mathbf{a} = a_1 a_2 \dots a_m$ and $\mathbf{b} = b_1 b_2 \dots b_m$, then

$$\{p, q\} \subseteq \bigcap_{i=1}^{m} \mathsf{Paths}([a_i, p]_\approx)$$

and, as an intersection of subtrees of $\mathsf{Paths}(\mathbf{T})$, $\bigcap_{i=1}^{m} \mathsf{Paths}([a_i, p]_\approx)$ is itself a subtree of $\mathsf{Paths}(\mathbf{T})$. Hence, in $\mathsf{Paths}(\mathbf{T})$ there is a path $p_1, \dots, p_k$ for which there exist tuples of constants $\mathbf{c}_1, \dots, \mathbf{c}_k$ with $(\mathbf{c}_1, p_1) \approx \cdots \approx (\mathbf{c}_k, p_k)$, $p = p_1$, $q = p_k$, $\mathbf{a} = \mathbf{c}_1$, and $\mathbf{b} = \mathbf{c}_k$. Since $\mathbf{a}, \mathbf{b}$ are non-empty, so are the $\mathbf{c}_i$. From Claim 7.17, we get that for any two neighbours p_i, p_{i+1} in $\mathsf{Paths}(\mathbf{T})$, $(\mathbf{c}_i, p_i) \approx (\mathbf{c}_{i+1}, p_{i+1})$ implies $(\mathbf{c}_i, p_i) \sim (\mathbf{c}_{i+1}, p_{i+1})$.

By two similar inductions on i with $1 \le i \le k$ we show that if $\beta(\mathbf{a}, * \cdots *) \in^* \Delta(\mathsf{Tail}(p))$ then $\beta(\mathbf{c}_i, * \cdots *) \in^* \Delta(\mathsf{Tail}(p_i))$ and if $\phi(\mathbf{a}) \in \Delta(\mathsf{Tail}(p))$ then $\phi(\mathbf{c}_i) \in \Delta(\mathsf{Tail}(\mathbf{c}_i))$.

For $i = 1$ in both cases nothing has to be shown. Now assume that the we have shown these properties up to i. W.l.o.g., assume p_{i+1} is a successor of p_i in the tree $\mathsf{Paths}(\mathbf{T})$. The other case is handled dually. There are two possibilities:

- $\mathsf{Tail}'(p_{i+1})$ is not blocked. Then $\mathsf{Tail}(p_{i+1}) = \mathsf{Tail}'(p_{i+1})$ and by the definition of $\sim$, $\mathsf{Tail}(p_{i+1})$ is a successor of $\mathsf{Tail}(p_i)$ in $\mathbf{T}$ and $\mathbf{c}_i = \mathbf{c}_{i+1}$ holds.

  If $\beta(\mathbf{a}, * \cdots *) \in^* \Delta(\mathsf{Tail}(p))$ then $\beta(\mathbf{c}_i, * \cdots *) \in^* \Delta(\mathsf{Tail}(p_i))$ holds by induction and due to the $\rightarrow_{\updownarrow}$-rule, this implies $\beta(\mathbf{c}_{i+1}, * \cdots *) \in^* \Delta(\mathsf{Tail}(p_{i+1}))$. The $\rightarrow_{\updownarrow}$-rule is applicable because, for the the non-empty tuple $\mathbf{c}_i$, $\mathbf{c}_i = \mathbf{c}_{i+1} \subseteq \mathbf{C}(\mathsf{Tail}(p_{i+1}))$ holds.

  If $\phi(\mathbf{a}) \in \Delta(\mathsf{Tail}(p))$ then by induction $\phi(\mathbf{c}_i) \in \Delta(\mathsf{Tail}(\mathbf{p}_i))$ and due to the $\rightarrow_{\updownarrow\forall}$-rule this implies $\phi(\mathbf{c}_{i+1}) \in \Delta(\mathsf{Tail}(p_{i+1}))$.

- $\mathsf{Tail}'(p_{i+1})$ is blocked by $\mathsf{Tail}(p_{i+1})$ (with function π) and $\mathsf{Tail}'(p_{i+1})$ is a successor of $\mathsf{Tail}(p_i)$ in $\mathbf{T}$. Then, by definition of $\sim$, we have $\mathbf{c}_{i+1} = \pi(\mathbf{c}_i)$ and $\mathbf{c}_i \subseteq \mathbf{C}(\mathsf{Tail}(p_i)) \cap \mathbf{C}(\mathsf{Tail}'(p_{i+1}))$.

  If $\beta(\mathbf{a}, * \cdots *) \in^* \Delta(\mathsf{Tail}(p))$ then $\beta(\mathbf{c}_i, * \cdots *) \in^* \Delta(\mathsf{Tail}(p_i))$ holds by induction and due to the $\rightarrow_{\updownarrow}$-rule this implies $\beta(\mathbf{c}_i, * \cdots *) \in^* \Delta(\mathsf{Tail}'(p_{i+1}))$. The $\rightarrow_{\updownarrow}$-rule is applicable because, for the non-empty tuple $\mathbf{c}_i$, $\mathbf{c}_i \subseteq \mathbf{C}(\mathsf{Tail}'(p_{i+1}))$ holds. The node $\mathsf{Tail}(p_{i+1})$ blocks $\mathsf{Tail}'(p_{i+1})$, which implies

  $$\pi(\beta(\mathbf{c}_i, * \cdots *)) = \beta(\mathbf{c}_{i+1}, * \cdots *) \in^* \Delta(\mathsf{Tail}(p_{i+1})).$$

  If $\phi(\mathbf{a}) \in \Delta(\mathsf{Tail}(p))$ then by induction $\phi(\mathbf{c}_i) \in \Delta(\mathsf{Tail}(p_i))$ and due to the $\rightarrow_{\updownarrow\forall}$-rule this implies $\phi(\mathbf{c}_i) \in \Delta(\mathsf{Tail}'(p_{i+1}))$. Since $\mathsf{Tail}(p_{i+1})$ blocks $\mathsf{Tail}'(p_{i+1})$, $\pi(\phi(\mathbf{c}_i)) = \phi(\mathbf{c}_{i+1}) \in \Delta(\mathsf{Tail}(p_{i+1}))$ holds.

We now define the structure $\mathfrak{A}$ over the universe $A = \mathbf{C}(\mathbf{T})/{\approx}$. For a relation $R \in \tau$ of arity m, $R^{\mathfrak{A}}$ is defined to be the set of tuples $([a_1, p_1]_\approx, \dots, [a_m, p_m]_\approx)$ for which there exists a path $p \in \mathsf{Paths}(\mathbf{T})$ and constants $c_1, \dots c_m$ such that $(c_i, p) \approx (a_i, p_i)$ for all $1 \leq i \leq m$, and $Rc_1 \dots c_m \in \Delta(\mathsf{Tail}(p))$.

It remains to show that this construction yields $\mathfrak{A} \models \psi$. This is a consequence of the following claim.

Claim 7.19 For every path $p \in \mathsf{Paths}(\mathbf{T})$ and $\mathbf{a} \subseteq \mathbf{C}(\mathsf{Tail}(p))$, if $\phi(\mathbf{a}) \in \Delta(\mathsf{Tail}(p))$, then $\mathfrak{A} \models \phi([\mathbf{a}, p]_\approx)$.

We show this claim by induction on the structure of ϕ. If $\phi(\mathbf{a}) = Ra_1 \dots a_m \in \Delta(\mathsf{Tail}(p))$, then the claim holds immediately by construction of $\mathfrak{A}$.

Assume $\phi(\mathbf{a}) = \neg R\mathbf{a} \in \Delta(\mathsf{Tail}(p))$, but $[\mathbf{a}, p]_\approx \in R^{\mathfrak{A}}$. Then, by the definition of $\mathfrak{A}$, there must be a path p' and constants $\mathbf{c}$ such that $(\mathbf{a}, p) \approx (\mathbf{c}, p')$ and $R\mathbf{c} \in \Delta(\mathsf{Tail}(p'))$. From Claim 7.18 we have that $(\mathbf{a}, p) \approx (\mathbf{c}, p')$ implies $R\mathbf{a} \in^* \Delta(\mathsf{Tail}(p))$ and, since $\mathbf{a}$ contains no occurrence of $*$, $R\mathbf{a} \in \Delta(\mathsf{Tail}(p))$. Hence $\mathbf{T}$ contains the clash $\{R\mathbf{a}, \neg R\mathbf{a}\} \subseteq \Delta(\mathsf{Tail}(p))$, a contradiction to the fact that $\mathbf{T}$ is clash-free. Thus, $[\mathbf{a}, p]_\approx \notin R^{\mathfrak{A}}$.

Assume $\phi(\mathbf{a}) = a \neq b \in \Delta(\mathsf{Tail}(p))$ but $[a, p]_\approx = [b, p]_\approx$. From Claim 7.16 we get that this implies $a = b$ and hence $\mathbf{T}$ contains the clash $a \neq a \in \Delta(\mathsf{Tail}(p))$. Again, this is a contradiction to the fact that $\mathbf{T}$ is clash-free and $[a, p]_\approx \neq [b, p]_\approx$ must hold.

For positive Boolean combinations the claim is immediate due to the $\rightarrow_\wedge$- and $\rightarrow_\vee$-rule.

Let $\phi(\mathbf{a}) = (\forall \mathbf{yz}.\alpha(\mathbf{a}, \mathbf{y}, \mathbf{z}))\chi(\mathbf{a}, \mathbf{y}) \in \Delta(\mathsf{Tail}(p))$ and $\mathbf{b}, \mathbf{p}, \mathbf{c}, \mathbf{q}$ arbitrarily chosen with

$$\mathfrak{A} \models \alpha([\mathbf{a}, p]_\approx, [\mathbf{b}, \mathbf{p}]_\approx, [\mathbf{c}, \mathbf{q}]_\approx). \tag{7.1}$$

We need to show that also $\mathfrak{A} \models \chi([\mathbf{a}, p]_\approx, [\mathbf{b}, \mathbf{p}]_\approx)$ holds. In order to bring completeness of $\mathbf{T}$ and the $\rightarrow_\forall$-rule into play, we show that information about the fact that (7.1) holds is present at a *single* node in $\mathbf{T}$ where it triggers the $\rightarrow_\forall$-rule. We rely on the fact that universal quantifiers must be guarded.

Every $y_i \in \mathbf{y}$ coexists with every other variable $y_j \in \mathbf{y}$ in at least one atom $\beta^{(i,j)} \in \alpha(\mathbf{a}, \mathbf{y}, \mathbf{z})$ and with every element $a_\ell \in \mathbf{a}$ in at least one atom $\gamma^{(i,\ell)} \in \alpha(\mathbf{a}, \mathbf{y}, \mathbf{z})$. For any two distinct variables y_i, y_j, $\mathfrak{A} \models \beta^{(i,j)}([\mathbf{a}, p]_\approx, [\mathbf{b}, \mathbf{p}]_\approx, [\mathbf{c}, \mathbf{q}]_\approx)$ holds and this can only be the case if there is a path $q^{(i,j)}$ and constants $d^{(i,j)}, e^{(i,j)}$ such that $(b_i, p_i) \approx (c^{(i,j)}, q^{(i,j)})$ and $(b_j, p_j) \approx (d^{(i,j)}, q^{(i,j)})$.

Similarly, for every element $[b_i, p_i]_\approx \in [\mathbf{b}, \mathbf{p}]_\approx$ and every element (a_ℓ, p) there exists a path $r^{(i,\ell)}$ and constants $f^{(i,\ell)}, g^{(i,\ell)}$ such that $(b_i, p_i) \approx (f^{(i,\ell)}, r^{(i,\ell)})$ and $(a_\ell, p) \approx (g^{(i,\ell)}, r^{(i,\ell)})$. For every i and ℓ, $\mathsf{Paths}([b_i, p_i]_\approx)$ and $\mathsf{Paths}([a_\ell, p]_\approx)$ are subtrees of $\mathsf{Paths}(\mathbf{T})$.

The tree $\mathsf{Paths}([b_i, p_i]_\approx)$ overlaps with the tree $\mathsf{Paths}([b_j, p_j]_\approx)$ at $q^{(i,j)}$ and with the tree $\mathsf{Paths}([a_\ell, p]_\approx)$ at $r^{(i,\ell)}$. From this it follows (Golumbic, 1980, Proposition 4.7) that there exists a common path

$$s \in \bigcap_i \mathsf{Paths}([b_i, p_i]_\approx) \cap \bigcap_\ell \mathsf{Paths}([a_\ell, p]_\approx).$$

Thus, there are tuples $\mathbf{a}'$, $\mathbf{b}'$ such that

$$(\mathbf{a}', s) \approx (\mathbf{a}, p) \text{ and } (\mathbf{b}', s) \approx (\mathbf{b}, \mathbf{p}). \tag{7.2}$$

We now show that the preconditions of the $\rightarrow_\forall$-rule are satisfied at $\mathsf{Tail}(s)$ for the formula $(\forall \mathbf{yz}.\alpha(\mathbf{a}', \mathbf{y}, \mathbf{z}))\chi(\mathbf{a}', \mathbf{y})$ and the tuple $\mathbf{b}'$. First, due to Claim 7.18, it holds that $(\forall \mathbf{yz}.\alpha(\mathbf{a}', \mathbf{y}, \mathbf{z}))\chi(\mathbf{a}', \mathbf{y}) \in \Delta(\mathsf{Tail}(s))$ because $(\mathbf{a}, p) \approx (\mathbf{a}', s)$ and $(\forall \mathbf{yz}.\alpha(\mathbf{a}, \mathbf{y}, \mathbf{z}))\chi(\mathbf{a}, \mathbf{y}) \in \Delta(\mathsf{Tail}(p))$.

For every $\beta(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \alpha(\mathbf{x}, \mathbf{y}, \mathbf{z})$, $\beta(\mathbf{a}', \mathbf{b}', * \cdots *) \in^* \Delta(\mathsf{Tail}(s))$ holds as follows: from (7.1, 7.2) we get

$$\mathfrak{A} \models \beta([\mathbf{a}', s]_\approx, [\mathbf{b}', s]_\approx, [\mathbf{c}, \mathbf{q}]_\approx).$$

Since β is an atom, this implies the existence of a path t and tuples $\mathbf{a}'', \mathbf{b}'', \mathbf{c}'$ with

$$\begin{array}{rc} & (\mathbf{a}', s) \approx (\mathbf{a}'', t),\ (\mathbf{b}', s) \approx (\mathbf{b}'', t),\ (\mathbf{c}, \mathbf{q}) \approx (\mathbf{c}', t) \\ \text{and} & \beta(\mathbf{a}'', \mathbf{b}'', \mathbf{c}') \in \Delta(\mathsf{Tail}(t)) \end{array} \tag{7.3}$$

Clearly, $\beta(\mathbf{a}'', \mathbf{b}'', * \cdots *) \geq^* \beta(\mathbf{a}'', \mathbf{b}'', \mathbf{c}')$ and, since $\beta(\mathbf{a}'', \mathbf{b}'', \mathbf{c}') \in \Delta(\mathsf{Tail}(t))$, it holds that $\beta(\mathbf{a}'', \mathbf{b}'', * \cdots *) \in^* \Delta(\mathsf{Tail}(t))$. Thus, by Claim 7.18 it holds that $\beta(\mathbf{a}', \mathbf{b}', * \cdots *) \in^* \Delta(\mathsf{Tail}(s))$.

Since this is true for every atom β, the preconditions of the $\rightarrow_\forall$-rule are satisfied and the completeness of $\mathbf{T}$ yields $\chi(\mathbf{a}', \mathbf{b}') \in \Delta(\mathsf{Tail}(s))$. By induction, $\mathfrak{A} \models \chi([\mathbf{a}', s]_\approx, [\mathbf{b}', s]_\approx)$ holds and together with (7.2) this implies $\mathfrak{A} \models \chi([\mathbf{a}, p]_\approx, [\mathbf{b}, \mathbf{p}]_\approx)$. Since $\mathbf{a}, \mathbf{p}, \mathbf{c}, \mathbf{q}$ have been chose arbitrarily, $\mathfrak{A} \models \phi([\mathbf{a}, p]_\approx)$ holds.

If $\phi(\mathbf{a}) = (\exists \mathbf{yz}.\alpha(\mathbf{a}, \mathbf{y}, \mathbf{z}))\chi(\mathbf{a}, \mathbf{y}) \in \Delta(\mathsf{Tail}(p))$, there are two possibilities:

- there are $\mathbf{b}, \mathbf{c} \subseteq \mathbf{C}(\mathsf{Tail}(p))$ with $\{\alpha(\mathbf{a}, \mathbf{b}, \mathbf{c}), \chi(\mathbf{a}, \mathbf{b})\} \subseteq \Delta(\mathsf{Tail}(p))$. Then, by induction, we have

  $$\mathfrak{A} \models \{\alpha([\mathbf{a}, p]_\approx, [\mathbf{b}, p]_\approx, [\mathbf{c}, p]_\approx), \chi([\mathbf{a}, p]_\approx, [\mathbf{b}, p]_\approx)\}$$

  and hence $\mathfrak{A} \models \phi([\mathbf{a}, p]_\approx)$.

- there are no such $\mathbf{b}, \mathbf{c} \subseteq \mathbf{C}(\mathsf{Tail}(p))$, then there is a successor w of $\mathsf{Tail}(p)$ and $\mathbf{b}, \mathbf{c} \subseteq \mathbf{C}(w)$ with $\{\alpha(\mathbf{a}, \mathbf{b}, \mathbf{c}), \chi(\mathbf{a}, \mathbf{b})\} \subseteq \Delta(w)$. The node w can be blocked or not.

  If w is not blocked, then $p' = [p, \frac{w}{w}] \in \mathsf{Paths}(\mathbf{T})$ and by induction

  $$\mathfrak{A} \models \{\alpha([\mathbf{a}, p']_\approx, [\mathbf{b}, p']_\approx, [\mathbf{c}, p']_\approx), \chi([\mathbf{a}, p']_\approx, [\mathbf{b}, p']_\approx)\}.$$

  From the definition of $\approx$ we have, $(\mathbf{a}, p') \approx (\mathbf{a}, p)$ and hence $\mathfrak{A} \models \phi([\mathbf{a}, p]_\approx)$.

  If w is blocked by a node u (with function π) then $p' = [p, \frac{u}{w}] \in \mathsf{Paths}(\mathbf{T})$. From the blocking condition, we have that u is unblocked and $\pi\{\alpha(\mathbf{a}, \mathbf{b}, \mathbf{c}), \chi(\mathbf{a}, \mathbf{b})\}) \subseteq \Delta(u)$. Hence, by induction

  $$\mathfrak{A} \models \{ \begin{array}{ll} \alpha([\pi(\mathbf{a}), p']_\approx, [\pi(\mathbf{b}), p']_\approx, [\pi(\mathbf{c}), p']_\approx), & \\ \chi([\pi(\mathbf{a}), p']_\approx, [\pi(\mathbf{b}), p']_\approx) & \}, \end{array}$$

  and, by definition of $\approx$, we have that $(\mathbf{a}, p) \approx (\pi(\mathbf{a}), p')$ and hence, $\mathfrak{A} \models \phi([\mathbf{a}, p]_\approx)$.

As a special instance of Claim 7.19 we get that $\mathfrak{A} \models \psi$. From Lemma 7.10, we get that, for every node $v \in \mathbf{V}$, $|\mathbf{C}(v)| \leq \mathrm{width}(\psi)$ and hence the tree $\mathsf{Paths}(\mathbf{T})$ together with the function $f : \mathsf{Paths}(\mathbf{T}) \rightarrow \mathbf{C}(\mathbf{T})/\approx$ with $f(p) = \mathbf{C}(\mathsf{Tail}(p))/\approx$ provides a tree decomposition of $\mathfrak{A}$ of width $\leq \mathrm{width}(\psi) - 1$. This completes the proof of Lemma 7.15. ∎

As an aside, together with Lemma 7.12, the construction used to prove Lemma 7.15 yields an alternative proof of Fact 7.7:

Corollary 7.20
CGF, *and hence also* LGF *and* GF *have the generalised tree model property.*

Proof. Let $\psi \in \mathrm{CGF}[\tau]$ be satisfiable. Then, from Lemma 7.12 we get that there is a tableau $\mathbf{T}$ for ψ. By Lemma 7.15, $\mathbf{T}$ induces a model for ψ of tree width at most $\mathrm{width}(\psi) - 1$. Note that we have never relied on Fact 7.7 to obtain any of the results in this thesis and hence have indeed given an alternative proof for the generalised tree model property of CGF. For LGF and GF, observe that the embedding of these logics into CGF may increase the width of the sentence but not by more than a recursive amount. ∎

Lemma 7.11, 7.12, and Lemma 7.15 yield correctness of the tableau algorithm for CGF.

Theorem 7.21
The tableau algorithm is a decision procedure for CGF*-satisfiability.*

An optimized implementation of this tableau algorithm is part of ongoing work. It will be interesting to see if the tableau algorithm is amenable to the same optimizations developed for modal or description logic tableau algorithms and how it performs in comparison with the resolution based approach from (Ganzinger & de Nivelle, 1999).

Chapter 8

Summary

The two major subjects of this thesis were (i) the worst-case complexity of reasoning with expressive description logics, particularly in the presence of counting operators; and (ii) the development of practical algorithms for description and guarded logics. This chapter summarizes and comments on the main results obtained on these topics.

Local Counting Qualifying number restrictions introduce a form of counting into DLs, which is *local* because only statements about the number of role successors of an individual are expressible. Until now, the impact of qualifying number restrictions on the complexity of reasoning was unknown, if binary coding of numbers in the input is assumed. In this thesis, we have shown that—in terms of worst-case complexity—qualifying number restrictions do not lead to a rise in complexity of the reasoning problems.

Like for $\mathcal{ALC}$, concept satisfiability for $\mathcal{ALCQ}$ is PSPACE-complete, even for the case of binary coding of numbers in the input (Theorem 4.6). The same applies to$\mathcal{ALCQI}b$ (Theorem 4.29), which extends $\mathcal{ALCQ}$ with inverse roles and safe role expressions, and is one of the most expressive DLs for which qualifying number restrictions have been studied.

For the case of the other inference problems, we have shown (Theorem 4.42) that knowledge base satisfiability for $\mathcal{ALCQI}b$ is EXPTIME-complete also in the case of binary coding of numbers in the input (Theorem 4.42), and hence again has the same complexity as the same problem for $\mathcal{ALC}$.

It is also possible to add qualifying number restrictions to DLs that allow for transitive roles and role hierarchies without an increase in worst-case complexity. Concept satisfiability (with or without general TBoxes) for $\mathcal{SHIQ}$ is EXPTIME-hard (Theorem 6.29), also if binary coding of numbers in the input is assumed. This matches the complexity of the same problems for the DL $\mathcal{SH}$, i.e., the fragment of $\mathcal{SHIQ}$ that does not allow for inverse roles or number restrictions. To maintain satisfiability of the decision problems for $\mathcal{SHIQ}$, it was necessary to allow qualifying number restrictions only over roles that are neither transitive nor have transitive sub-roles. In the absence of role hierarchies, the effect of number restrictions over transitive roles on complexity and decidability is open.

Global Counting If we additionally consider constructors that allow to express *global* counting statements like cardinality restrictions or nominals,[1] then this increases the complexity of the inference problems, independent on the coding of numbers in the input.

We have shown that knowledge base satisfiability becomes NExpTime-hard if cardinality restrictions are added to $\mathcal{ALCQ}$ (Theorem 5.20) or $\mathcal{ALCQI}$ (Theorem 5.19), while knowledge base satisfiability for $\mathcal{ALCQ}$ and $\mathcal{ALCQI}$ without cardinality restrictions is ExpTime-complete (as a Corollary of Theorem 4.42). The "gap" in the complexity is even wider if we consider nominals: the complexity of concept satisfiability rises from PSpace-complete to NExpTime-hard if nominals are added to $\mathcal{ALCQI}$ (Corollary 5.27).

A special case is the DL $\mathcal{ALCQIB}$, for which nominals or cardinality restrictions can be added without a change in the complexity of the inference problems. Yet, a closer look shows that cardinality restrictions (and hence nominals) can already be expressed by $\mathcal{ALCQIB}$-concepts (Lemma 5.32), which explains that they do not have an impact on the complexity.

Coding of Numbers One of the recurring themes of the thesis has been the impact that coding of numbers in the input has on the complexity of the reasoning problems. With respect to this topic, we have obtained only an incomplete picture.

All our results for local counting are independent on the coding of numbers and one of the main contributions of this thesis is the development of algorithms that deal with binary coding of numbers in the input without an additional exponential overhead.

For logics that allow for global counting, we obtain tight complexity result only if unary coding of numbers in the input is assumed. This is because the upper (NExpTime-) bounds rely on a reduction to C^2, the two-variable fragment of FOL with counting quantifiers, for which the exact complexity is also known only for unary case. Of course, reasoning does not become easier in the binary case, and so the lower bounds hold independently of the coding. For the upper bounds, we only know that all problems can be solved in 2-NExpTime but we do not have matching hardness-results. It is an interesting open question whether exponential blow-up is necessary or whether are algorithms that can deal with the binary case without an increase in complexity.

Until such algorithms are developed (or 2-NExpTime-hardness is proved), it is open if the complexity of these reasoning problems rises when switching from unary to binary coding of numbers in the input. The only case for which an increase in complexity is certain is $\mathcal{ALCQ}$ with cardinality restrictions, for which satisfiability is ExpTime-complete in the unary case (Corollary 5.8) and NExpTime-hard in the binary case (Theorem 5.20).

Practical Algorithms The practicality of inference algorithms, i.e., how easily they can be implemented and optimized and how they behave on "real world" instances, is important for their application in DL systems. In general, tableau algorithms have proven

[1] At first sight, it might look odd that we subsume nominals under global counting. Yet, the requirement that nominals must be interpreted by singletons can be seen as a form of global counting and, indeed, Lemma 5.5 exhibits a close connection between reasoning with nominals and with cardinality restrictions.

to be amenable to a number of powerful optimization techniques and are successfully employed in many DL systems. However, a tableau algorithm is not practical *just because* it is a tableau algorithm. One important criterion for the practicality of an algorithm seems to be the degree to which is depends on non-deterministic choices because, in a (necessarily deterministic) implementation, the different possibilities have to be searched exhaustively in order to obtain a complete algorithm. This search is what is responsible for most of the runtime of tableau algorithms and most of the aforementioned optimizations aim to reduce the size of the search space.

So, one of the major design principles of the $\mathcal{SI}$-, $\mathcal{SHIQ}$-, and CGF-algorithm in this thesis was to avoid non-deterministic choices as much as possible. The $\mathcal{SHIQ}$-algorithm developed in this thesis (Algorithm 6.34), forms the basis of the highly optimized DL system iFaCT (Horrocks, 1999) that shows good performance in system comparisons (Horrocks, 2000) and is successfully applied in applications (see, e.g., Franconi & Ng, 2000). One problem of the $\mathcal{SHIQ}$-algorithm lies in the non-deterministic identification of nodes due to its $\rightarrow_{\leq}$-rule. The development of optimization techniques that specifically deal with this problem is part of ongoing work. Moreover, it will be interesting to extend the refined blocking strategy developed for the $\mathcal{SI}$-algorithm to $\mathcal{SHIQ}$ and implement it in the iFaCT system. We have claimed that the tableau algorithm for CGF (Algorithm 7.9) is useful as the basis of an efficient reasoner and the implementation of such a system is in progress. It will be interesting to see how this implementation performs in comparison with the existing decision procedures for guarded fragments based on refinements of general FOL theorem proving techniques.

In the $\mathcal{ALCQ}$-algorithm (Algorithm 3.2) and the $\mathcal{ALCQI}b$-algorithm (Algorithm 4.21), we have freely used non-determinism in order to obtain a space-efficient algorithm. This implies that they seem to be less suited for an implementation because of their highly non-deterministic $\rightarrow_{\geq}$-rule.

For the remaining algorithms developed in this thesis, it is at least questionable if they can serve as the basis of an efficient implementation. This is especially the case for the decision procedure used to prove ExpTime-completeness of concept satisfiability of $\mathcal{ALCQI}b$ w.r.t. general TBoxes (see Theorem 4.38), which is based on a highly inefficient automata-construction. It is even less likely that an efficient decision procedure can be obtained from our decision procedures for knowledge base satisfiability for $\mathcal{ALCQI}b$ (see Theorem 4.42) or from the worst-case optimal decision procedures for $\mathcal{SHIQ}$ (see Corollary 6.29 and Corollary 6.30) because these add a wasteful pre-completion technique and various translations on top of the already inefficient algorithm for $\mathcal{ALCQI}b$ with general TBoxes.

All decision procedures for the NExpTime-hard DLs presented in this thesis employ a reduction to C^2, for which the only known decision procedures work by model enumeration and so there exists no decision procedure for these logics that could be of practical use. This situation is particularly unsatisfactory for the DL $\mathcal{SHIQO}$, for which such a decision procedure would be of high interest due to $\mathcal{SHIQO}$'s rôle for inferences for the semantic web (Fensel et al., 2000; Horrocks & Sattler, 2001). Maybe the most intriguing question left open by this thesis is how practical decision procedures for NExpTime-complete modal and description logics can be developed.

Bibliography

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of Databases*. Addison Wesley.

Andréka, H., van Benthem, J., & Németi, I. (1998). Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, *27*, 217–274.

Areces, C. (2000). *Logic Engineering: The Case of Description and Hybrid Logics*. Ph.D. thesis, University of Amsterdam.

Areces, C., Blackburn, P., & Marx, M. (2000). Hybrid logics: Characterization, interpolation, and complexity. *Journal of Symbolic Logic*. To appear.

Areces, C. & de Rijke, M. (2000). Description and/or hybrid logics. In Wolter, F., Wansing, H., de Rijke, M., & Zakharyaschev, M. (Eds.), *Preliminary Proceedings of the Workshop Advances in Modal Logics (AiML2000)* Leipzig, Deutschland.

Areces, C., Gennari, R., Heguiabehere, J., & de Rijke, M. (2000). Tree-based heuristics in modal theorem proving. In Horn, W. (Ed.), *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI2000)* Berlin, Germany. IOS Press Amsterdam.

Areces, C., Blackburn, P., & Marx, M. (1999). A road-map on complexity for hybrid logics. In Flum, J. & Rodríguez-Artalejo, M. (Eds.), *Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL-99)*, No. 1683 in Lecture Notes in Computer Science, pp. 307–321 Madrid, Spain. Springer Verlag.

Baader, F. (1991). Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In J. Mylopoulos, R. R. (Ed.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 446–451 Sydney, Australia. Morgan Kaufman Publishers, Inc.

Baader, F., Buchheit, M., & Hollunder, B. (1996). Cardinality restrictions on concepts. *Artificial Intelligence*, *88*(1–2), 195–213.

Baader, F., Bürckert, H.-J., Nebel, B., Nutt, W., & Smolka, G. (1993). On the expressivity of feature logics with negation, functional uncertainty, and sort equations. *Journal of Logic, Language and Information*, *2*, 1–18.

Baader, F., Franconi, E., Hollunder, B., Nebel, B., & Profitlich, H. (1994). An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, *4*, 109–132.

Baader, F. & Hanschke, P. (1993). 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*, pp. 132–143 Bonn (Germany). Springer Verlag.

Baader, F. & Hollunder, B. (1991). KRIS: Knowledge representation and inference system, system description. *ACM SIGART Bulletin*, *2*, 8–14.

Baader, F., McGuiness, D., Nardi, D., & Patel-Schneider, P. (Eds.). (2001). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press. forthcoming.

Baader, F. & Sattler, U. (1999). Expressive number restrictions in description logics. *Journal for Logic and Computation*, *9*(3), 319–350.

Baader, F. & Sattler, U. (2000). An overview of tableau algorithms for description logics. *Studia Logica*. To appear.

Bechhofer, S. & Horrocks, I. (2000). Driving user interfaces from FaCT. In Baader, F. & Sattler, U. (Eds.), *Proceedings of the 2000 International Workshop on Description Logics (DL2000)*, Vol. 33 of *CEUR Workshop Proceedings*, pp. 45–54 RWTH Aachen, Germany. Available online from `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-33/`.

Berger, R. (1966). The undecidability of the dominoe problem. *Memoirs of the American Mathematical Society*, *66*, 1–17.

Berners-Lee, T. (1999). *Weaving the Web*. Harper, San Francisco.

Blackburn, P. & Seligman, J. (1995). Hybrid languages. *Journal of Logic, Language and Information*, *3*(4), 251–272.

Börger, E., Grädel, E., & Gurevich, Y. (1997). *The Classical Decision Problem*. Springer Verlag.

Borgida, A. (1995). Description logics in data management. *Transaction on Knowledge and Data Engineering*, *7*(5), 671–682.

Borgida, A. (1996). On the relative expressiveness of description logics and first order logics. *Artificial Intelligence*, *82*, 353–367.

Borgida, A. & Patel-Schneider, P. F. (1994). A semantics and complete algorithm for subsumption in the CLASSIC description logic. *Journal of Artificial Intelligence Research*, *1*, 277–308.

Brachman, R. J. & Levesque, H. J. (1984). The tractability of subsumption in frame-based description languages. In *Proceedings of the 4th National Conference on Artificial Intelligence (AAAI-84)*, pp. 34–37 Austin, TX, USA. AAAI Press.

Brachman, R. J. & Levesque, H. J. (Eds.). (1985). *Readings in Knowledge Representation*. Morgan Kaufman Publishers, Inc., Los Altos.

Brachman, R. J., McGuinness, D. L., Patel-Schneider, P. F., Resnick, L. A., & Borgida, A. (1991). Living with CLASSIC: When and how to use a KL-ONE-like language. In Sowa, J. F. (Ed.), *Principles of Semantic Networks*, pp. 401–456. Morgan Kaufman Publishers, Inc.

Brachman, R. J. & Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, *9*(2), 171–216.

Bresciani, P., Franconi, E., & Tessaris, S. (1995). Implementing and testing expressive description logics: Preliminary report. In Borgida, A., Lenzerini, M., Nardi, D., & Nebel, B. (Eds.), *Working Notes of the 1995 Description Logics Workshop*, Technical Report, RAP 07.95, Dipartimento di Informatica e Sistemistica, pp. 131–139 Rome (Italy).

Buchheit, M., Donini, F. M., & Schaerf, A. (1993). Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, *1*, 109–138.

Calvanese, D., De Giacomo, G., & Lenzerini, M. (1999). Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In Dean, T. (Ed.), *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 84–89 Stockholm, Sweden. Morgan Kaufman Publishers, Inc.

Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., & Rosati, R. (1998). Description logic framework for information integration. In Cohn, A. G., Schubert, L., & Shapiro, S. C. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 6th International Conference (KR'98)*, pp. 2–13 Trento, Italy. Morgan Kaufman Publishers, Inc.

Calvanese, D., De Giacomo, G., & Rosati, R. (1999). Data integration and reconciliation in data warehousing. *Network and Information Systems*, *4*(2), 413–432.

Calvanese, D., Lenzerini, M., & Nardi, D. (1994). A unified framework for class based representation formalism. In Doyle, J., Sandewall, E., & Torasso, P. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 4th International Conference (KR'94)*, pp. 109–120 Bonn, Germany. Morgan Kaufman Publishers, Inc.

Calvanese, D., Lenzerini, M., & Nardi, D. (1998). Description logics for conceptual data modeling. In Chomicki, J. & Saake, G. (Eds.), *Logics for Databases and Information Systems*, pp. 229–264. Kluwer Academic Publisher.

De Giacomo, G. (1995). *Decidability of Class-Based Knowledge Representation Formalisms*. Ph.D. thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza".

De Giacomo, G. & Lenzerini, M. (1994a). Boosting the correspondence between description logics and propositional dynamic logics. In Hayes-Roth, B. & Korf, R. E. (Eds.), *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pp. 205–212 Seattle, Washington, USA. AAAI Press.

De Giacomo, G. & Lenzerini, M. (1994b). A concept language with number restrictions and fixpoints, and its relationship with μ-calculus. In Cohn, A. G. (Ed.), *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI'94)* Amsterdam, The Netherlands. John Wiley and Sons.

De Giacomo, G. & Lenzerini, M. (1994c). Description logics with inverse roles, functional restrictions, and n-ary relations. In MacNish, C., Pearce, D., & Pereira, L. M. (Eds.), *1994 European Workshop on Logic in Artificial Intelligence (JELIA'94)*, No. 838 in Lecture Notes in Artificial Intelligence, pp. 332–346 York, UK. Springer Verlag.

De Giacomo, G. & Lenzerini, M. (1996). Tbox and abox reasoning in expressive description logics. In Aiello, L. C., Doyle, J., & Shapiro, S. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 5th International Conference (KR'96)*, pp. 316–327 Cambridge, Massachusetts, USA. Morgan Kaufman Publishers, Inc.

De Giacomo, G. & Massacci, F. (2001). Combining deduction and model checking into tableaux and algorithms for converse-PDL. *Information and Computation*. To appear.

de Nivelle, H. (2000). Translation of S4 and K5 into GF and 2VAR. Unpublished Manuscript.

de Nivelle, H. & de Rijke, M. (2000). Deciding the guarded fragments by resolution. *Journal for Logic and Computation*. To appear.

Donini, F. M., Hollunder, B., Lenzerini, M., Spaccamela, A. M., Nardi, D., & Nutt, W. (1992). The complexity of existential quantification in concept languages. *Artificial Intelligence*, *2–3*, 309–327.

Donini, F. M., Lenzerini, M., Nardi, D., & Nutt, W. (1991a). The complexity of concept languages. In Allen, J., Fikes, R., & Sandewall, E. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 2nd International Conference (KR'91)*, pp. 151–162 Cambridge, Massachusetts, USA. Morgan Kaufman Publishers, Inc.

Donini, F. M., Lenzerini, M., Nardi, D., & Nutt, W. (1991b). Tractable concept languages. In J. Mylopoulos, R. R. (Ed.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 458–463 Sydney, Australia. Morgan Kaufman Publishers, Inc.

Donini, F. M., Lenzerini, M., Nardi, D., & Nutt, W. (1997). The complexity of concept languages. *Information and Computation*, *134*, 1–58.

Donini, F. M. & Massacci, F. (2000). EXPTIME tableaux for ALC. *Artificial Intelligence*, *124*(1), 87–138.

Dowling, W. F. & Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, *1*(3), 267–28.

Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M., & Klein, M. (2000). OIL in a nutshell. In *Proceedings of the European Knowledge Acquisition Conference (EKAW-2000)*, No. 1937 in Lecture Notes in Artificial Intelligence, pp. 1–16. Springer Verlag.

Fischer, M. J. & Ladner, R. E. (1979). Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, *18*, 194–211.

Franconi, E. (1994). A treatment of plurals and plural quantification based on a theory of collections. *Minds and Machines*, *3*(4), 453–474.

Franconi, E., Baader, F., Sattler, U., & Vassiliadis, P. (2000). Multidimensional data models and aggregation. In Jarke, M., Lenzerini, M., Vassilious, Y., & Vassiliadis, P. (Eds.), *Fundamentals of Data Warehousing*, pp. 87–106. Springer-Verlag.

Franconi, E. & Ng, G. (2000). The i.com tool for intelligent conceptual modeling. In Bouzeghoub, M., Klusch, M., Nutt, W., & Sattler, U. (Eds.), *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases (KRDB'99)*, Vol. 29 of *CEUR Workshop Proceedings*, pp. 45–53 Berlin, Germany. RWTH Aachen. Available online from `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-29/`.

Ganzinger, H. & de Nivelle, H. (1999). A superposition decision procedure for the guarded fragment with equality. In *Proceedings of 14th IEEE Symposium on Logic in Computer Science (LICS'99)*, pp. 295–303 Trento, Italy. IEEE Computer Society Press.

Ganzinger, H., Meyer, C., & Veanes, M. (1999). The two-variable guarded fragment with transitive relations. In *Proceedings of 14th IEEE Symposium on Logic in Computer Science (LICS'99)*, pp. 25–34 Trento, Italy. IEEE Computer Society Press.

Gargov, G. & Goranko, V. (1993). Modal logic with names. *Journal of Philosophical Logic*, *22*, 607–636.

Gécseg, F. & Steinby, M. (1984). *Tree Automata*. Akadémiai Kiadó, Budapest.

Giunchiglia, F. & Sebastiani, R. (1996). A SAT-based decision procedure for ALC. In Aiello, L. C., Doyle, J., & Shapiro, S. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 5th International Conference (KR'96)* Cambridge, Massachusetts, USA. Morgan Kaufman Publishers, Inc.

Golumbic, M. (1980). *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, NY.

Goncalves, E. & Grädel, E. (2000). Decidability issues for action guarded logics. In Baader, F. & Sattler, U. (Eds.), *Proceedings of the 2000 International Workshop on Description Logics (DL2000)*, Vol. 33 of *CEUR Workshop Proceedings*, pp. 123–131 RWTH Aachen, Germany. Available online from `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-33/`.

Goré, R. (1998). Tableau methods for modal and temporal logics. In D'Agostino, M., Gabbay, D. M., Hähnle, R., & Posegga, J. (Eds.), *Handbook of Tableau Methods*. Kluwer, Dordrecht.

Grädel, E. (1999a). Decision procedures for guarded logics. In Ganzinger, H. (Ed.), *Automated Deduction - (CADE-16), 16th International Conference on Automated Deduction*, No. 1632 in Lecture Notes in Artificial Intelligence, pp. 31–51 Trento, Italy. Springer Verlag.

Grädel, E. (1999b). On the restraining power of guards. *Journal of Symbolic Logic*, *64*(4), 1719–1742.

Grädel, E. (1999c). Why are modal logics so robustly decidable?. *Bulletin of the European Association for Theoretical Computer Science*, *68*, 90–103.

Grädel, E. (2001). Guarded fixpoint logic and the monadic theory of trees. *Theoretical Computer Science*. To appear, available online from `http://www-mgi.informatik.rwth-aachen.de/Publications/pub/graedel/Gr-tc%s01.ps`.

Grädel, E., Otto, M., & Rosen, E. (1997). Two-variable logic with counting is decidable. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science (LICS'97)*, pp. 306–317 Warsaw, Poland. IEEE Computer Society Press.

Grädel, E. & Walukiewicz, I. (1999). Guarded fixed point logic. In *Proceedings of 14th IEEE Symposium on Logic in Computer Science (LICS'99)*, pp. 45–54 Trento, Italy. IEEE Computer Society Press.

Haarslev, V. & Möller, R. (1999). RACE system description. In Lambrix, P., Borgida, A., Lenzerini, M., Möller, R., & Patel-Schneider, P. (Eds.), *Proceedings of the 1999 International Workshop on Description Logics (DL;99)*, Vol. 22 of *CEUR Workshop*

Proceedings, pp. 130–132 Linkoeping, Sweden. Linköping University. Available online from `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-22/`.

Haarslev, V. & Möller, R. (2000a). Expressive Abox reasoning with number restrictions, role hierarchies, and transitively closed role. In Cohn, A. G., Giunchiglia, F., & Selman, B. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 7th International Conference (KR2000)*, pp. 273–284 Breckenridge, CO, USA. Morgan Kaufman Publishers, Inc.

Haarslev, V. & Möller, R. (2000b). High performance reasoning with very large knowledge bases. In Baader, F. & Sattler, U. (Eds.), *Proceedings of the 2000 International Workshop on Description Logics (DL2000)*, Vol. 33 of *CEUR Workshop Proceedings* RWTH Aachen, Germany. Available online from `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-33/`.

Haarslev, V. & Möller, R. (2000c). Optimizing Tbox and Abox reasoning with pseudo models. In Baader, F. & Sattler, U. (Eds.), *Proceedings of the 2000 International Workshop on Description Logics (DL2000)*, Vol. 33 of *CEUR Workshop Proceedings* RWTH Aachen, Germany. Available online from `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-33/`.

Halpern, J. Y. & Moses, Y. (1992). A guide to completeness and complexity for model logics of knowledge and belief. *Artificial Intelligence*, *54*(3), 319–379.

Hanschke, P. (1992). Specifying role interaction in concept languages. In Nebel, B., Rich, C., & Swartout, W. R. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 3rd International Conference (KR'92)*, pp. 318–329 Cambridge, Massachusetts, USA. Morgan Kaufman Publishers, Inc.

Hemaspaandra, E. (2000). Modal satisfiability is in deterministic linear space. In Clote, P. & Schwichtenberg, H. (Eds.), *Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL-2000)*, No. 1862 in Lecture Notes in Computer Science, pp. 332–342 Fischbachau, Germany. Springer Verlag.

Hirsch, C. & Tobies, S. (2000). A tableau algorithm for the clique guarded fragment. In F., W., Wansing, H., de Rijke, M., & Zakharyaschev, M. (Eds.), *Preliminary Proceedings of the Workshop Advances in Modal Logics (AiML2000)* Leipzig, Germany.

Hollunder, B. (1996). Consistency checking reduced to satisfiability of concepts in terminological systems. *Annals of Mathematics and Artificial Intelligence*, *18*(2–4), 133–157.

Hollunder, B. & Baader, F. (1991). Qualifying number restrictions in concept languages. In Allen, J., Fikes, R., & Sandewall, E. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 2nd International Conference (KR'91)*, pp. 335–346 Cambridge, Massachusetts, USA. Morgan Kaufman Publishers, Inc.

Hollunder, B., Nutt, W., & Schmidt-Schauß, M. (1990). Subsumption algorithms fo concept description languages. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI'90)*, pp. 348–353 Stockholm, Sweden.

Horrocks, I. (1997). *Optimising Tableaux Decision Procedures for Description Logics*. Ph.D. thesis, University of Manchester.

Horrocks, I. (1998). Using an expressive description logic: FaCT or fiction?. In Cohn, A. G., Schubert, L., & Shapiro, S. C. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 6th International Conference (KR'98)*, pp. 636–647 Trento, Italy. Morgan Kaufman Publishers, Inc.

Horrocks, I. (1999). FaCT and iFaCT. In Lambrix, P., Borgida, A., Lenzerini, M., Möller, R., & Patel-Schneider, P. (Eds.), *Proceedings of the 1999 International Workshop on Description Logics (DL;99)*, Vol. 22 of *CEUR Workshop Proceedings*, pp. 133–135 Linkoeping, Sweden. Linköping University. Available online from `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-22/`.

Horrocks, I. (2000). Benchmark analysis with FaCT. In Dyckhoff, R. (Ed.), *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, No. 1847 in Lecture Notes in Artificial Intelligence, pp. 62–66 St Andrews, Scotland, UK. Springer Verlag.

Horrocks, I. & Gough, G. (1997). Description logics with transitive roles. In Brachman, R. J., Donini, F. M., Franconi, E., Horrocks, I., Levy, A. Y., & Rousset, M.-C. (Eds.), *Proceedings of the 1996 International Workshop on Description Logics (DL'97)*, pp. 25–28. Université Paris-Sud, Centre d'Orsay, Laboratoire de Recherche en Informatique LRI. Available online from `http://www.lri.fr/~mcr/ps/dl97.html`.

Horrocks, I. & Patel-Schneider, P. F. (1999). Optimising description logic subsumption. *Journal for Logic and Computation*, *9*(3), 267–293.

Horrocks, I., Patel-Schneider, P. F., & Sebastiani, R. (2000). An analysis of empirical testing for modal decision procedures. *Logic Journal of the IGPL*, *8*(3), 293–323.

Horrocks, I. & Sattler, U. (1998). A description logic with transitive and converse roles and role hierarchies. In Franconi, E., Giacomo, G. D., MacGregor, R. M., Nutt, W., & Welty, C. A. (Eds.), *Proceedings of the 1998 International Workshop on Description Logics (DL'98)*, Vol. 11 of *CEUR Workshop Proceedings* IRST, Trento, Italy. Available online from `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-11/`.

Horrocks, I. & Sattler, U. (1999). A description logic with transitive and inverse roles and role hierarchies. *Journal for Logic and Computation*, *9*(3), 385–410.

Horrocks, I. & Sattler, U. (2001). Ontology reasoning in the SHOQ(D) description logic. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)* Seattle, USA. Morgan Kaufman Publishers, Inc.

Horrocks, I., Sattler, U., Tessaris, S., & Tobies, S. (2000). How to decide query containment under constraints using a description logic. In Voronkov, A. (Ed.), *Logic for Programming and Automated Reasoning, 7th International Conference (LPAR2000)*, Lecture Notes in Artificial Intelligence, pp. 308–325. Springer Verlag.

Horrocks, I., Sattler, U., & Tobies, S. (1999). Practical reasoning for expressive description logics. In Ganzinger, H., McAllester, D., & Voronkov, A. (Eds.), *Logic for Programming and Automated Reasoning, 6th International Conference (LPAR'99)*, No. 1705 in Lecture Notes in Artificial Intelligence, pp. 161–180 Tbilisi, Georgia. Springer Verlag.

Horrocks, I., Sattler, U., & Tobies, S. (2000a). Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, *8*(3), 239–264.

Horrocks, I., Sattler, U., & Tobies, S. (2000b). Reasoning with individuals for the description logic SHIQ. In McAllester, D. (Ed.), *Automated Deduction - (CADE-17), 17th International Conference on Automated Deduction*, No. 1831 in Lecture Notes in Artificial Intelligence, pp. 482–496 Pittsburgh, PA, USA. Springer Verlag.

Horrocks, I. & Tobies, S. (2000). Reasoning with axioms: Theory and practice. In Cohn, A. G., Giunchiglia, F., & Selman, B. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 7th International Conference (KR2000)*, pp. 285–296 Breckenridge, CO, USA. Morgan Kaufman Publishers, Inc.

Ladner, R. E. (1977). The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, *6*(3), 467–480.

Levesque, H. J. & Brachman, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, *3*, 78–93.

Lutz, C. & Sattler, U. (2000). The complexity of reasoning with boolean modal logic. In F., W., Wansing, H., de Rijke, M., & Zakharyaschev, M. (Eds.), *Preliminary Proceedings of the Workshop Advances in Modal Logics (AiML2000)* Leipzig, Germany.

Lutz, C., Sattler, U., & Tobies, S. (1999). A suggestion for an n-ary description logic. In Lambrix, P., Borgida, A., Lenzerini, M., Möller, R., & Patel-Schneider, P. (Eds.), *Proceedings of the 1999 International Workshop on Description Logics (DL;99)*, Vol. 22 of *CEUR Workshop Proceedings* Linkoeping, Sweden. Linköping University. Available online from `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-22/`.

Lutz, C. (1999). Complexity of terminological reasoning revisited. In Ganzinger, H., McAllester, D., & Voronkov, A. (Eds.), *Logic for Programming and Automated Reasoning, 6th International Conference (LPAR'99)*, No. 1705 in Lecture Notes in Artificial Intelligence, pp. 181–200 Tbilisi, Georgia. Springer Verlag.

MacGregor, R. (1991). Inside the LOOM description classifier. *ACM SIGART Bulletin*, *2*(3), 88–92.

Marx, M., Schlobach, S., & Mikulás, S. (2000). Labelled deduction for the guarded fragment. In Basin, D., D'Agostino, M., Gabbay, D. M., Matthews, S., & Viganò, L. (Eds.), *Labelled Deduction*. Kluwer Academic Publishers, Dordrecht.

Massacci, F. (1999). Design and results of the TABLEAUX-99 non-classical (modal) systems comparison. In Murray, N. (Ed.), *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'99)*, No. 1617 in Lecture Notes in Artificial Intelligence Saratoga Springs, NY, USA. Springer Verlag.

Massacci, F. & Donini, F. M. (2000). Design and results of TANCS-2000 non-classical (modal) system comparison. In Dyckhoff, R. (Ed.), *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, No. 1847 in Lecture Notes in Artificial Intelligence, pp. 52–56 St Andrews, Scotland, UK. Springer Verlag.

Mays, E., Dionne, R., & Weida, R. (1991). K-REP system overview. *ACM SIGART Bulletin*, *2*(3).

McGuinness, D. & Wright, J. (1998). Conceptual modelling for configuration: A description logic-based approach. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing Journal*, *12*, 333–344.

Minsky, M. (1981). A framework for representing knowledge. In Haugeland, J. (Ed.), *Mind Design*. The MIT Press. Republished in (Brachman & Levesque, 1985).

Nebel, B. (1990a). *Reasoning and Revision in Hybrid Representation Systems*. No. 422 in Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, Germany.

Nebel, B. (1990b). Terminological reasoning is inherently intractable. *Artificial Intelligence*, *43*(2), 235–249.

Nebel, B. (1988). Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, *34*(3), 371–383.

Pacholski, L., Szwast, W., & Tendera, L. (1997). Complexity of two-variable logic with counting. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science (LICS'97)*, pp. 318–327 Warsaw, Poland. IEEE Computer Society Press.

Papadimitriou, C. H. (1994). *Computational Complexity.* Addison Wesley Publishing Company.

Patel-Schneider, P. F. (2000). TANCS-2000 results for DLP. In Dyckhoff, R. (Ed.), *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, No. 1847 in Lecture Notes in Artificial Intelligence, pp. 72–76 St Andrews, Scotland, UK. Springer Verlag.

Patel-Schneider, P. F., McGuiness, D. L., Brachman, R. J., Resnick, L. A., & Borgida, A. (1991). The CLASSIC knowledge representation system: Guiding principles and implementation rational. *ACM SIGART Bulletin*, *2*(3), 108–113.

Pratt, V. R. (1979). Models of program logics. In *20th Annual Symposium on Foundations of Computer Science*, pp. 115–122 San Juan, Puerto Rico. IEEE Computer Society Press.

Quantz, J. & Kindermann, C. (1990). Implementation of the BACK-system version 4. Technical report TUB-FB13-KIT-78, Technical University of Berlin.

Quillian, M. R. (1967). Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science*, *12*, 410–430. Republished in (Brachman & Levesque, 1985).

Rabin, M. (1969). Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, *141*, 1–35.

Rector, A. & Horrocks, I. (1997). Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97).* AAAI Press, Menlo Park, California.

Sattler, U. (1996a). A concept language extended with different kinds of transitive roles. In Görz, G. & Hölldobler, S. (Eds.), *KI-96: Advances in Artificial Intelligence, 20th Annual German Conference on Artificial Intelligence*, No. 1137 in Lecture Notes in Artificial Intelligence. Springer Verlag.

Sattler, U. (1996b). Knowledge representation in process engineering. In *Proceedings of WRKP'96.* DFKI GmbH.

Sattler, U. (1998). *Terminological knowledge representation systems in a process engineering application.* Aachener Berichte zur Informatik, RWTH Aachen.

Sattler, U. (2000). Description logics for the representation of aggregated objects. In Horn, W. (Ed.), *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI2000)*, pp. 239–243 Berlin, Germany. IOS Press Amsterdam.

Savitch, W. J. (1970). Relationship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, *4*, 177–192.

Schaerf, A. (1994). Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, *13*(2), 141–176.

Schild, K. (1991). A correspondence theory for terminological logics: Preliminary report. In J. Mylopoulos, R. R. (Ed.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 466–471 Sydney, Australia. Morgan Kaufman Publishers, Inc.

Schild, K. (1994). Terminological cycles and the propositional μ-calculus. In Doyle, J., Sandewall, E., & Torasso, P. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 4th International Conference (KR'94)*, pp. 509–520 Bonn, Germany. Morgan Kaufman Publishers, Inc.

Schmidt, R. A. (1998). Resolution is a decision procedure for many propositional modal logics. In Kracht, M., de Rijke, M., Wansing, H., & Zakharyaschev, M. (Eds.), *Advances in Modal Logic, Volume 1*, Vol. 87 of *Lecture Notes*, pp. 189–208. CSLI Publications, Stanford.

Schmidt-Schauß, M. (1989). Subsumption in KL-ONE is undecidable. In Brachman, R. J., Levesque, H. J., & Reiter, R. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 1st International Conference (KR'89)*, pp. 421–431 Toronto, Canada. Morgan Kaufman Publishers, Inc.

Schmidt-Schauß, M. & Smolka, G. (1991). Attributive concept descriptions with complements. *Artificial Intelligence*, *48*, 1–26.

Schulz, S. & Hahn, U. (2000). Knowledge engineering by large-scale knowledge reuse—experience from the medical domain. In Cohn, A. G., Giunchiglia, F., & Selman, B. (Eds.), *Principles of Knowledge Representation and Reasoning, Proceedings of the 7th International Conference (KR2000)*, pp. 601–610 Breckenridge, CO, USA. Morgan Kaufman Publishers, Inc.

Spaan, E. (1993a). *Complexity of Modal Logics*. Ph.D. thesis, Universiteit van Amsterdam.

Spaan, E. (1993b). The complexity of propositional tense logics. In de Rijke, M. (Ed.), *Diamonds and Defaults*, pp. 287–307. Kluwer Academic Publishers, Dordrecht.

Stockmeyer, L. J. & Meyer, A. R. (1973). Word problems requiring exponential time. In *ACM Symposium on Theory of Computing (STOC '73)*, pp. 1–9 New York. ACM Press.

Thomas, W. (1992). Automata on infinite objects. In van Leeuwen, J. (Ed.), *Handbook of Theoretical Computer Science*, Vol. B. Elsevier and MIT Press.

Tobies, S. (1999a). A NExpTime-complete description logic strictly contained in C^2. In Flum, J. & Rodríguez-Artalejo, M. (Eds.), *Proceedings of the Annual Conference of* *the European Association for Computer Science Logic (CSL-99)*, No. 1683 in Lecture Notes in Computer Science, pp. 292–306 Madrid, Spain. Springer Verlag.

Tobies, S. (1999b). A PSpace algorithm for graded modal logic. In Ganzinger, H. (Ed.), *Automated Deduction - (CADE-16), 16th International Conference on Automated Deduction*, No. 1632 in Lecture Notes in Artificial Intelligence, pp. 52–66 Trento, Italy. Springer Verlag.

Tobies, S. (2000). The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *Journal of Artificial Intelligence Research*, *12*, 199–217.

Tobies, S. (2001). PSPACE reasoning for graded modal logics. *Journal for Logic and Computation*, *11*(1), 85–106.

van Benthem, J. (1997). Dynamic bits and pieces. ILLC research report, University of Amsterdam.

van der Hoek, W. & de Rijke, M. (1995). Counting objects. *Journal for Logic and Computation*, *5*(3), 325–345.

Vardi, M. Y. (1996). Why is modal logic so robustly decidable?. In Immerman, N. & Kolaitis, P. (Eds.), *Descriptive Complexity and Finite Models: Proceedings of a DIMACS Workshop, January 14-17, 1996*, No. 31 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 149–184. American Mathematical Society.

Vardi, M. Y. (1998). Reasoning about the past with two-way automata. In Larsen, K. G., Skyum, S., & Winskel, G. (Eds.), *Automata, Languages and Programming ICALP 98*, Vol. 1443 of *Lecture Notes in Computer Science*, pp. 628–641 Aalborg, Denmark. Springer Verlag.

Vardi, M. Y. & Wolper, P. (1986). Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, *32*, 183–221.

Vardi, M. & Wolper, P. (1994). Reasoning about infinite computations. *Information and Computation*, *115*, 1–37.

Voronkov, A. (1999). KK: a theorem prover for K. In Ganzinger, H. (Ed.), *Automated Deduction - (CADE-16), 16th International Conference on Automated Deduction*, No. 1632 in Lecture Notes in Artificial Intelligence, pp. 383–387 Trento, Italy. Springer Verlag.

Voronkov, A. (2000). How to optimize proof-search in modal logics: A new way of proving redundancy for sequent calculi. In *Proceedings of 15th IEEE Symposium on Logic in Computer Science (LICS2000)* Santa Barbara, USA. IEEE Computer Society Press.

Wache, H. & Kamp, G. (1996). Using description logic for configuration problems. In *Proceedings of WRKP'96*. DFKI GmbH.

Wang, H. (1963). Dominoes and the AEA case of the Decision Problem. *Bell Systems Technical Journal*, *40*, 1–41.

Woods, W. (1975). What's in a link: Foundations for semantic networks. In Bobrow, D. & Collins, A. (Eds.), *Representation and Understanding: Studies in Cognitive Science*, pp. 35–82. Academic Press.

Woods, W. & Schmolze, J. (1992). The KL-ONE family. *Computers and Mathematics with Applications, Special Issue on Semantic Networks in Artificial Intelligence, Part 1*, *23*(2–5), 133–178. Also available as Technical Report TR-20-90, Aiken Computation Laboratory, Harvard University.

List of Figures

Lebenslauf Stephan Tobies

Persönliche Daten

Geburtsdatum:	21. Main 1972 in Essen, Deutschland
Familienstand:	seit September 1997 verheiratet mit Antje Tobies, geb. Müller, zwei Töchter, Johanna und Annika
Nationalität:	deutsch
Anschrift:	Wiechertstraße 4, 45721 Haltern

Schulausbildung

- *August 1978 – Juli 1982:* Katholische Grundschule Erkelenz
- *August 1982 – Juni 1991:* Cusanus Gymnasium Erkelenz, Abschluß Abitur

Zivildienst

- *August 1991 – September 1992:* Internat Haus Overbach, Jülich–Barmen

Studium

- *Oktober 1992 – September 1994:* Grundstudium der Informatik (Nebenfach Mathematik) an der Rheinisch-Westfälischen Technischen Hochschule (RWTH) Aachen, Abschluß Vordiplom
- *Oktober 1994 – Februar 1998:* Hauptstudium der Informatik (Nebenfach Mathematik) an der RWTH Aachen, Abschluß Diplom

Promotionsstudium

- *März 1998 bis März 2001:* Promotionsstudium der Informatik an der RWTH Aachen

Beruflicher Werdegang

- *April 1998 bis März 2001:* Wissenschaftlicher Angestellter am Lehr- und Forschungsgebiet Theoretische Informatik der RWTH Aachen
- *seit April 2001:* Research Engineer am Nokia Research Center, Bochum

Auslandsaufenthalt

- *Oktober 1995 – Mai 1996:* Auslandsstudium an der University of Kent, Canterbury, Großbritannien

Haltern, 12. Juni 2001