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

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 $\perp$ denotes the concept with empty extension (the empty class):

$$\begin{aligned} \mathtt{Parent} &= \mathtt{Human} \sqcap \exists \mathtt{has\_child}.\mathtt{Human} \sqcap \forall \mathtt{has\_child}.\mathtt{Human} \\ \mathtt{Husband} &= \mathtt{Male} \sqcap \exists \mathtt{married\_to}.\mathtt{Human} \\ \mathtt{Human} &= \mathtt{Male} \sqcup \mathtt{Female} \\ \mathtt{Husband} &\sqsubseteq \forall \mathtt{married\_to}.\mathtt{Female} \\ \mathtt{Male} \sqcap \mathtt{Female} &= \perp \end{aligned}$$

  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 $\exists$has_child.$\exists$married_to.Female) and is an instance of $\neg$Husband because the TBox axiom Male $\sqcap$ 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

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

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

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

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 $\texttt{has\_offspring}^{-1}$, 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 $\exists\texttt{has\_child}^{-1}.\texttt{MARY}$. 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 $\mathcal{ALCQIb}$, 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 $\mathcal{ALCQIb}$, 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 $\mathcal{ALCQIb}$.

- 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 $D \sqsubseteq C$.* $\diamond$

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 $C \to D$ for $\neg C \sqcup D$ and $C \leftrightarrow D$ for $C \to D \sqcap D \to C$.

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* $x \neq y$ *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* $x \neq y$ *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* $\mathcal{I} \models \mathcal{A}$. $\diamond$

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* $\mathcal{I} \models \mathcal{K}$. $\diamond$

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 ($\neg$) 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:*

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

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. ◇

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:

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		$\mathcal{A} \rightarrow_{\forall} \mathcal{A} \cup \{y : D\}$

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*

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

◇

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*

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

$\diamond$

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 $x \neq y$.

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, $x \in (\forall R.D)^{\mathcal{I}_\mathcal{A}}$. $\blacksquare$

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:

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

Since every deterministic TM is a non-deterministic TM, $\textsc{Time}(f(n)) \subseteq \text{N}\textsc{Time}(f(n))$ and $\textsc{Space}(f(n)) \subseteq \text{N}\textsc{Space}(f(n))$ hold trivially for an arbitrary function f. Also, $\textsc{Time}(f(n)) \subseteq \textsc{Space}(f(n))$ and $\text{N}\textsc{Time}(f(n)) \subseteq \text{N}\textsc{Space}(f(n))$ 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.

Figure 3.2 Some complexity classes

$$\begin{aligned}
\text{P}\textsc{Space} &= \bigcup_{k\in\mathbb{N}} \textsc{Space}(n^k)\\
\text{NP}\textsc{Space} &= \bigcup_{k\in\mathbb{N}} \text{N}\textsc{Space}(n^k)\\
\textsc{ExpTime} &= \bigcup_{k\in\mathbb{N}} \textsc{Time}(2^{n^k})\\
\text{N}\textsc{ExpTime} &= \bigcup_{k\in\mathbb{N}} \text{N}\textsc{Space}(2^{n^k})\\
2\text{-}\textsc{ExpTime} &= \bigcup_{k\in\mathbb{N}} \textsc{Time}(2^{2^{n^k}})\\
2\text{-N}\textsc{ExpTime} &= \bigcup_{k\in\mathbb{N}} \text{N}\textsc{Time}(2^{2^{n^k}})
\end{aligned}$$

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

$$\text{P}\textsc{Space} = \text{NP}\textsc{Space} \subseteq \textsc{ExpTime} \subseteq \text{N}\textsc{ExpTime} \subseteq 2\text{-}\textsc{ExpTime} \subseteq 2\text{-N}\textsc{ExpTime},$$

where the fact that $\text{P}\textsc{Space} = \text{NP}\textsc{Space}$ 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 $L' \in \mathcal{C}$, 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 $|X|$. $\diamond$

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 $\mathcal{ALC}$.

```
ALC-SAT(C) := sat(x_0, {x_0 : C})
sat(x, A):
    while (the →⊓- or the →⊔-rule can be applied) and (A is clash-free) do
        apply the →⊓- or the →⊔-rule to A.
    od
    if A contains a clash then return "not satisfiable".
    E := {x : ∃R.D | x : ∃R.D ∈ A}
    while E ≠ ∅ do
        pick an arbitrary x : ∃R.D ∈ E
        A_new := {(x, y) : R, y : D} where y is a fresh individual
        while (the →∀-rule can be applied to A ∪ A_new) do
            apply the →∀-rule and add the new constraints to A_new
        od
        if A ∪ A_new contains a clash then return "not satisfiable".
        if sat(y, A ∪ A_new) = "not satisfiable" then return "not satisfiable"
        E := E \ {x : ∃R.D | y : D ∈ A_new}
        discard 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.*