In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation and Applications.
Cambridge University Press, 2003.

2

Basic Description Logics

Franz Baader
Institut für Theoretische Informatik, Fakultät Informatik, TU Dresden, Germany
http://wwwtcs.inf.tu-dresden.de/~baader/

Werner Nutt
Department of Computing and Electrical Engineering, Heriot-Watt University, Edinburgh, U.K.
http://www.cee.hw.ac.uk/~nutt/

Abstract

This chapter provides an introduction to Description Logics as a formal language for representing knowledge and reasoning about it. It first gives a short overview of the ideas underlying Description Logics. Then it introduces syntax and semantics, covering the basic constructors that are used in systems or have been introduced in the literature, and the way these constructors can be used to build knowledge bases. Finally, it defines the typical inference problems, shows how they are interrelated, and describes different approaches for effectively solving these problems. Some of the topics that are only briefly mentioned in this chapter will be treated in more detail in subsequent chapters.

2.1 Introduction

As sketched in the previous chapter, Description Logics (DLs) is the most recent name[1] for a family of knowledge representation (KR) formalisms that represent the knowledge of an application domain (the "world") by first defining the relevant concepts of the domain (its terminology), and then using these concepts to specify properties of objects and individuals occurring in the domain (the world description). As the name Description *Logics* indicates, one of the characteristics of these languages is that, unlike some of their predecessors, they are equipped with a formal, logic-based semantics. Another distinguished feature is the emphasis on reasoning as a central service: reasoning allows one to infer implicitly represented knowledge from the knowledge that is explicitly contained in the knowledge base. Description Logics support inference patterns that occur in many applications of intelligent information processing systems, and which are also used by humans to structure and understand the world: classification of concepts and individuals. Classification

[1] Previously used names are terminological knowledge representation languages, concept languages, term subsumption languages, and KL-ONE-based knowledge representation languages.

of concepts determines subconcept/superconcept relationships (called subsumption relationships in DL) between the concepts of a given terminology, and thus allows one to structure the terminology in the form of a subsumption hierarchy. This hierarchy provides useful information on the connection between different concepts, and it can be used to speed-up other inference services. Classification of individuals (or objects) determines whether a given individual is always an instance of a certain concept (i.e., whether this instance relationship is implied by the description of the individual and the definition of the concept). It thus provides useful information on the properties of an individual. Moreover, instance relationships may trigger the application of rules that insert additional facts into the knowledge base.

Because Description Logics are a KR formalism, and since in KR one usually assumes that a KR system should always answer the queries of a user in reasonable time, the reasoning procedures DL researchers are interested in are *decision procedures*, i.e., unlike, e.g., first-order theorem provers, these procedures should always terminate, both for positive and for negative answers. Since the guarantee of an answer in finite time need not imply that the answer is given in reasonable time, investigating the computational complexity of a given DL with decidable inference problems is an important issue. Decidability and complexity of the inference problems depend on the expressive power of the DL at hand. On the one hand, very expressive DLs are likely to have inference problems of high complexity, or they may even be undecidable. On the other hand, very weak DLs (with efficient reasoning procedures) may not be sufficiently expressive to represent the important concepts of a given application. As mentioned in the previous chapter, investigating this trade-off between the expressivity of DLs and the complexity of their reasoning problems has been one of the most important issues in DL research.

Description Logics are descended from so-called "structured inheritance networks" [Brachman, 1977; 1978], which were introduced to overcome the ambiguities of early semantic networks and frames, and which were first realized in the system KL-ONE [Brachman and Schmolze, 1985]. The following three ideas, first put forward in Brachman's work on structured inheritance networks, have largely shaped the subsequent development of DLs:

- The basic syntactic building blocks are atomic concepts (unary predicates), atomic roles (binary predicates), and individuals (constants).
- The expressive power of the language is restricted in that it uses a rather small set of (epistemologically adequate) constructors for building complex concepts and roles.
- Implicit knowledge about concepts and individuals can be inferred automatically with the help of inference procedures. In particular, subsumption relationships between concepts and instance relationships between individuals and concepts

play an important rôle: unlike IS-A links in Semantic Networks, which are explicitly introduced by the user, subsumption relationships and instance relationships are inferred from the definition of the concepts and the properties of the individuals.

After the first logic-based semantics for KL-ONE-like KR languages were proposed, the inference problems like subsumption could also be provided with a precise meaning, which led to the first formal investigations of the computational properties of such languages. It has turned out that the languages used in early DL systems were too expressive, which led to undecidability of the subsumption problem [Schmidt-Schauß, 1989; Patel-Schneider, 1989]. The first worst-case complexity results [Levesque and Brachman, 1987; Nebel, 1988] showed that the subsumption problem is intractable (i.e., not polynomially solvable) even for very inexpressive languages. As mentioned in the previous chapter, this work was the starting point of a thorough investigation of the worst-case complexity of reasoning in KL-ONE-like KR languages (see Chapter 3 for details).

Later on it has turned out, however, that intractability of reasoning (in the sense of being non-polynomial in the worst case) does not prevent a DL from being useful in practice, provided that sophisticated optimization techniques are used when implementing a system based on such a DL (see Chapter 9). When implementing a DL system, the efficient implementation of the basic reasoning algorithms is not the only issue, though. On the one hand, the derived system services (such as classification, i.e., constructing the subsumption hierarchy between all concepts defined in a terminology) must be optimized as well [Baader *et al.*, 1994]. On the other hand, one needs a good user and application programming interface (see Chapter 7 for more details). Most implemented DL systems provide for a rule language, which can be seen as a very simple, but effective, application programming mechanism (see Subsection 2.2.5 for details).

Section 2.2 introduces the basic formalism of Description Logics. By way of a prototypical example, it first introduces the formalism for describing concepts (i.e., the description language), and then defines the terminological (TBox) and the assertional (ABox) formalisms. Next, it introduces the basic reasoning problems and shows how they are related to each other. Finally, it defines the rule language that is available in many of the implemented DL systems.

Section 2.3 describes algorithms for solving the basic reasoning problems in DLs. After shortly sketching structural subsumption algorithms, it concentrates on tableau-based algorithms. Finally, it comments on the problem of reasoning w.r.t. terminologies.

Finally, Section 2.4 describes some additional language constructors that are not included in the prototypical family of description languages introduced in Sec-

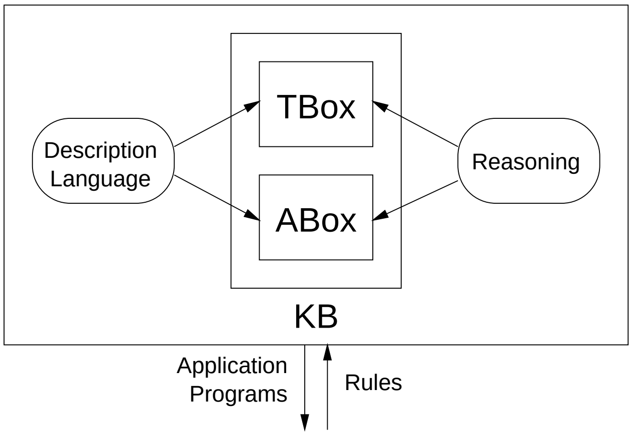

Fig. 2.1. Architecture of a knowledge representation system based on Description Logics.

tion 2.2, but have been considered in the literature and are available in some DL systems.

2.2 Definition of the basic formalism

A KR system based on Description Logics provides facilities to set up knowledge bases, to reason about their content, and to manipulate them. Figure 2.1 sketches the architecture of such a system (see Chapter 8 for more information on DL systems).

A knowledge base (KB) comprises two components, the TBox and the ABox. The TBox introduces the *terminology*, i.e., the vocabulary of an application domain, while the ABox contains *assertions* about named individuals in terms of this vocabulary.

The vocabulary consists of *concepts*, which denote sets of individuals, and *roles*, which denote binary relationships between individuals. In addition to atomic concepts and roles (concept and role names), all DL systems allow their users to build complex descriptions of concepts and roles. The TBox can be used to assign names to complex descriptions. The language for building descriptions is a characteristic of each DL system, and different systems are distinguished by their description languages. The description language has a model-theoretic semantics. Thus, statements in the TBox and in the ABox can be identified with formulae in first-order logic or, in some cases, a slight extension of it.

A DL system not only stores terminologies and assertions, but also offers services that *reason* about them. Typical reasoning tasks for a terminology are to determine whether a description is *satisfiable* (i.e., non-contradictory), or whether one description is more general than another one, that is, whether the first *subsumes* the second. Important problems for an ABox are to find out whether its set of assertions is *consistent,* that is, whether it has a model, and whether the assertions in the ABox entail that a particular individual is an *instance* of a given concept description. Satisfiability checks of descriptions and consistency checks of sets of assertions are useful to determine whether a knowledge base is meaningful at all. With subsumption tests, one can organize the concepts of a terminology into a hierarchy according to their generality. A concept description can also be conceived as a query, describing a set of objects one is interested in. Thus, with instance tests, one can retrieve the individuals that satisfy the query.

In any application, a KR system is embedded into a larger environment. Other components interact with the KR component by querying the knowledge base and by modifying it, that is, by adding and retracting concepts, roles, and assertions. A restricted mechanism to add assertions are rules. Rules are an extension of the logical core formalism, which can still be interpreted logically. However, many systems, in addition to providing an application programming interface that consists of functions with a well-defined logical semantics, provide an escape hatch by which application programs can operate on the KB in arbitrary ways.

2.2.1 Description languages

Elementary descriptions are *atomic concepts* and *atomic roles.* Complex descriptions can be built from them inductively with *concept constructors.* In abstract notation, we use the letters A and B for atomic concepts, the letter R for atomic roles, and the letters C and D for concept descriptions. Description languages are distinguished by the constructors they provide. In the sequel we shall discuss various languages from the family of $\mathcal{AL}$*-languages.* The language $\mathcal{AL}$ (= attributive language) has been introduced in [Schmidt-Schauß and Smolka, 1991] as a minimal language that is of practical interest. The other languages of this family are extensions of $\mathcal{AL}$.

2.2.1.1 The basic description language $\mathcal{AL}$

Concept descriptions in $\mathcal{AL}$ are formed according to the following syntax rule:

$$
\begin{array}{lcll}
C, D & \longrightarrow & A \mid & \text{(atomic concept)} \\
 & & \top \mid & \text{(universal concept)} \\
 & & \bot \mid & \text{(bottom concept)} \\
 & & \neg A \mid & \text{(atomic negation)} \\
 & & C \sqcap D \mid & \text{(intersection)}
\end{array}
$$

$$\begin{array}{ll} \forall R.C \mid & \text{(value restriction)} \\ \exists R.\top & \text{(limited existential quantification).} \end{array}$$

Note that, in $\mathcal{AL}$, negation can only be applied to atomic concepts, and only the top concept is allowed in the scope of an existential quantification over a role. For historical reasons, the sublanguage of $\mathcal{AL}$ obtained by disallowing atomic negation is called $\mathcal{FL}^-$ and the sublanguage of $\mathcal{FL}^-$ obtained by disallowing limited existential quantification is called $\mathcal{FL}_0$.

To give examples of what can be expressed in $\mathcal{AL}$, we suppose that Person and Female are atomic concepts. Then $\mathsf{Person} \sqcap \mathsf{Female}$ and $\mathsf{Person} \sqcap \neg\mathsf{Female}$ are $\mathcal{AL}$-concepts describing, intuitively, those persons that are female, and those that are not female. If, in addition, we suppose that $\mathsf{hasChild}$ is an atomic role, we can form the concepts $\mathsf{Person} \sqcap \exists\mathsf{hasChild}.\top$ and $\mathsf{Person} \sqcap \forall\mathsf{hasChild}.\mathsf{Female}$, denoting those persons that have a child, and those persons all of whose children are female. Using the bottom concept, we can also describe those persons without a child by the concept $\mathsf{Person} \sqcap \forall\mathsf{hasChild}.\bot$.

In order to define a formal semantics of $\mathcal{AL}$-concepts, we consider *interpretations* $\mathcal{I}$ that consist of a non-empty set $\Delta^\mathcal{I}$ (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept A a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and to every atomic role R a binary relation $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The interpretation function is extended to concept descriptions by the following inductive definitions:

$$\begin{array}{rcl} \top^\mathcal{I} & = & \Delta^\mathcal{I} \\ \bot^\mathcal{I} & = & \emptyset \\ (\neg A)^\mathcal{I} & = & \Delta^\mathcal{I} \setminus A^\mathcal{I} \\ (C \sqcap D)^\mathcal{I} & = & C^\mathcal{I} \cap D^\mathcal{I} \\ (\forall R.C)^\mathcal{I} & = & \{a \in \Delta^\mathcal{I} \mid \forall b.\ (a,b) \in R^\mathcal{I} \to b \in C^\mathcal{I}\} \\ (\exists R.\top)^\mathcal{I} & = & \{a \in \Delta^\mathcal{I} \mid \exists b.\ (a,b) \in R^\mathcal{I}\}. \end{array}$$

We say that two concepts C, D are *equivalent*, and write $C \equiv D$, if $C^\mathcal{I} = D^\mathcal{I}$ for all interpretations $\mathcal{I}$. For instance, going back to the definition of the semantics of concepts, one easily verifies that $\forall\mathsf{hasChild}.\mathsf{Female} \sqcap \forall\mathsf{hasChild}.\mathsf{Student}$ and $\forall\mathsf{hasChild}.(\mathsf{Female} \sqcap \mathsf{Student})$ are equivalent.

2.2.1.2 The family of $\mathcal{AL}$-languages

We obtain more expressive languages if we add further constructors to $\mathcal{AL}$. The *union* of concepts (indicated by the letter $\mathcal{U}$) is written as $C \sqcup D$, and interpreted as

$$(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}.$$

Full existential quantification (indicated by the letter $\mathcal{E}$) is written as $\exists R.C$, and interpreted as

$$(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b.\ (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}.$$

Note that $\exists R.C$ differs from $\exists R.\top$ in that arbitrary concepts are allowed to occur in the scope of the existential quantifier.

Number restrictions (indicated by the letter $\mathcal{N}$) are written as $\geqslant n\, R$(at-least restriction) and as $\leqslant n\, R$ (at-most restriction), where n ranges over the nonnegative integers. They are interpreted as

$$(\geqslant n\, R)^{\mathcal{I}} = \Big\{a \in \Delta^{\mathcal{I}} \,\Big|\, |\{b \mid (a, b) \in R^{\mathcal{I}}\}| \geq n\Big\},$$

and

$$(\leqslant n\, R)^{\mathcal{I}} = \Big\{a \in \Delta^{\mathcal{I}} \,\Big|\, |\{b \mid (a, b) \in R^{\mathcal{I}}\}| \leq n\Big\},$$

respectively, where "$|\cdot|$" denotes the cardinality of a set. From a semantic viewpoint, the coding of numbers in number restrictions is immaterial. However, for the complexity analysis of inferences it can matter whether a number n is represented in binary (or decimal) notation or by a string of length n, since binary (decimal) notation allows for a more compact representation.

The *negation* of arbitrary concepts (indicated by the letter $\mathcal{C}$, for "complement") is written as $\neg C$, and interpreted as

$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}.$$

With the additional constructors, we can, for example, describe those persons that have either not more than one child or at least three children, one of which is female:

$$\mathsf{Person} \sqcap (\leqslant 1\, \mathsf{hasChild} \sqcup (\geqslant 3\, \mathsf{hasChild} \sqcap \exists \mathsf{hasChild.Female})).$$

Extending $\mathcal{AL}$ by any subset of the above constructors yields a particular $\mathcal{AL}$-language. We name each $\mathcal{AL}$-language by a string of the form

$$\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{N}][\mathcal{C}],$$

where a letter in the name stands for the presence of the corresponding constructor. For instance, $\mathcal{ALEN}$ is the extension of $\mathcal{AL}$ by full existential quantification and number restrictions (see the appendix on DL terminology for how to extend this naming scheme to more expressive DLs).

From the semantic point of view, not all these languages are distinct, however. The semantics enforces the equivalences $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$ and $\exists R.C \equiv \neg\forall R.\neg C$. Hence, union and full existential quantification can be expressed using negation. Conversely, the combination of union and full existential quantification gives us the possibility to express negation of concepts (through their equivalent negation normal form, see Section 2.3.2). Therefore, we assume w.l.o.g. that union and full existential quantification are available in every language that contains negation, and vice versa. It follows that (modulo the equivalences mentioned above), all $\mathcal{AL}$-languages can be written using the letters $\mathcal{U}$, $\mathcal{E}$, $\mathcal{N}$ only. It is not hard to see that the eight languages obtained this way are indeed pairwise non-equivalent. In the sequel, we shall not distinguish between an $\mathcal{AL}$-language with negation and its counterpart that has union and full existential quantification instead. In the same vein, we shall use the letter $\mathcal{C}$ instead of the letters $\mathcal{UE}$ in language names. For instance, we shall write $\mathcal{ALC}$ instead of $\mathcal{ALUE}$ and $\mathcal{ALCN}$ instead of $\mathcal{ALUEN}$.

2.2.1.3 Description languages as fragments of predicate logic

The semantics of concepts identifies description languages as fragments of first-order predicate logic. Since an interpretation $\mathcal{I}$ respectively assigns to every atomic concept and role a unary and binary relation over $\Delta^{\mathcal{I}}$, we can view atomic concepts and roles as unary and binary predicates. Then, any concept C can be translated effectively into a predicate logic formula $\phi_C(x)$ with one free variable x such that for every interpretation $\mathcal{I}$ the set of elements of $\Delta^{\mathcal{I}}$ satisfying $\phi_C(x)$ is exactly $C^{\mathcal{I}}$: An atomic concept A is translated into the formula $A(x)$; the constructors intersection, union, and negation are translated into logical conjunction, disjunction, and negation, respectively; if C is already translated into $\phi_C(x)$ and R is an atomic role, then value restriction and existential quantification are captured by the formulae

$$\begin{aligned}
\phi_{\exists R.C}(y) &= \exists x.\, R(y,x) \wedge \phi_C(x) \\
\phi_{\forall R.C}(y) &= \forall x.\, R(y,x) \rightarrow \phi_C(x),
\end{aligned}$$

where y is a new variable; number restrictions are expressed by the formulae

$$\begin{aligned}
\phi_{\geqslant n\, R}(x) &= \exists y_1,\ldots,y_n.\ R(x,y_1) \wedge \cdots \wedge R(x,y_n) \wedge \bigwedge_{i<j} y_i \neq y_j \\
\phi_{\leqslant n\, R}(x) &= \forall y_1,\ldots,y_{n+1}.\ R(x,y_1) \wedge \cdots \wedge R(x,y_{n+1}) \rightarrow \bigvee_{i<j} y_i = y_j.
\end{aligned}$$

Note that the equality predicate "=" is needed to express number restrictions, while concepts without number restrictions can be translated into equality-free formulae.

One may argue that, since concepts can be translated into predicate logic, there is no need for a special syntax. However, the above translations show that, in particular for number restrictions, the variable free syntax of description logics is much more concise. As can be seen from Section 2.3, it also lends itself easily to the development of algorithms.

A more detailed analysis of the connection between fragments of first-order predicate logic and DLs can be found in Chapter 4.

2.2.2 Terminologies

We have seen how we can form complex descriptions of concepts to describe classes of objects. Now, we introduce *terminological axioms*, which make statements about how concepts or roles are related to each other. Then we single out *definitions* as specific axioms and identify *terminologies* as sets of definitions by which we can introduce atomic concepts as abbreviations or *names* for complex concepts. If the definitions in a terminology contain cycles, we may have to adopt *fixpoint semantics* to make them unequivocal. We discuss for which types of terminologies fixpoint models exist.

2.2.2.1 Terminological axioms

In the most general case, *terminological axioms* have the form

$$C \sqsubseteq D \quad (R \sqsubseteq S) \qquad \text{or} \qquad C \equiv D \quad (R \equiv S),$$

where C, D are concepts (and R, S are roles). Axioms of the first kind are called *inclusions*, while axioms of the second kind are called *equalities*. To simplify the exposition, we deal in the following only with axioms involving concepts.

The semantics of axioms is defined as one would expect. An interpretation $\mathcal{I}$ *satisfies* an inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies an equality $C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. If $\mathcal{T}$ is a set of axioms, then $\mathcal{I}$ satisfies $\mathcal{T}$ iff $\mathcal{I}$ satisfies each element of $\mathcal{T}$. If $\mathcal{I}$ satisfies an axiom (resp. a set of axioms), then we say that it is a *model* of this axiom (resp. set of axioms). Two axioms or two sets of axioms are *equivalent* if they have the same models.

2.2.2.2 Definitions

An equality whose left-hand side is an atomic concept is a *definition*. Definitions are used to introduce *symbolic names* for complex descriptions. For instance, by the axiom

$$\mathsf{Mother} \equiv \mathsf{Woman} \sqcap \exists \mathsf{hasChild}.\mathsf{Person}$$

we associate to the description on the right-hand side the name Mother. Symbolic names may be used as abbreviations in other descriptions. If, for example, we have defined Father analogously to Mother, we can define Parent as

$$\mathsf{Parent} \equiv \mathsf{Mother} \sqcup \mathsf{Father}.$$

A set of definitions should be unequivocal. We call a finite set of definitions $\mathcal{T}$ a

$$\begin{array}{rcl}
\mathsf{Woman} & \equiv & \mathsf{Person} \sqcap \mathsf{Female} \\
\mathsf{Man} & \equiv & \mathsf{Person} \sqcap \neg\mathsf{Woman} \\
\mathsf{Mother} & \equiv & \mathsf{Woman} \sqcap \exists\mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Father} & \equiv & \mathsf{Man} \sqcap \exists\mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Parent} & \equiv & \mathsf{Father} \sqcup \mathsf{Mother} \\
\mathsf{Grandmother} & \equiv & \mathsf{Mother} \sqcap \exists\mathsf{hasChild}.\mathsf{Parent} \\
\mathsf{MotherWithManyChildren} & \equiv & \mathsf{Mother} \sqcap \geqslant 3\, \mathsf{hasChild} \\
\mathsf{MotherWithoutDaughter} & \equiv & \mathsf{Mother} \sqcap \forall\mathsf{hasChild}.\neg\mathsf{Woman} \\
\mathsf{Wife} & \equiv & \mathsf{Woman} \sqcap \exists\mathsf{hasHusband}.\mathsf{Man}
\end{array}$$

Fig. 2.2. A terminology (TBox) with concepts about family relationships.

terminology or *TBox* if no symbolic name is defined more than once, that is, if for every atomic concept A there is at most one axiom in $\mathcal{T}$ whose left-hand side is A. Figure 2.2 shows a terminology with concepts concerned with family relationships.

Suppose, $\mathcal{T}$ is a terminology. We divide the atomic concepts occurring in $\mathcal{T}$ into two sets, the *name symbols* $\mathcal{N}_\mathcal{T}$ that occur on the left-hand side of some axiom and the *base symbols* $\mathcal{B}_\mathcal{T}$ that occur only on the right-hand side of axioms. Name symbols are often called *defined* concepts and base symbols *primitive* concepts[1]. We expect that the terminology *defines* the name symbols in terms of the base symbols, which now we make more precise.

A *base interpretation* for $\mathcal{T}$ is an interpretation that interprets only the base symbols. Let $\mathcal{J}$ be such a base interpretation. An interpretation $\mathcal{I}$ that interprets also the name symbols is an *extension* of $\mathcal{J}$ if it has the same domain as $\mathcal{J}$, i.e., $\Delta^\mathcal{I} = \Delta^\mathcal{J}$, and if it agrees with $\mathcal{J}$ for the base symbols. We say that $\mathcal{T}$ is *definitorial* if every base interpretation has exactly one extension that is a model of $\mathcal{T}$. In other words, if we know what the base symbols stand for, and $\mathcal{T}$ is definitorial, then the meaning of the name symbols is completely determined. Obviously, if a terminology is definitorial, then every equivalent terminology is also definitorial.

The question whether a terminology is definitorial or not is related to the question whether or not its definitions are cyclic. For instance, the terminology that consists of the the single axiom

$$\mathsf{Human}' \equiv \mathsf{Animal} \sqcap \forall\mathsf{hasParent}.\mathsf{Human}' \qquad (2.1)$$

contains a cycle, which in this special case is very simple. In general, we define cycles in a terminology $\mathcal{T}$ as follows. Let A, B be atomic concepts occurring in $\mathcal{T}$. We say that A *directly uses* B in $\mathcal{T}$ if B appears on the right-hand side of the

[1] Note that some papers use the notion "primitive concept" with a different meaning; e.g., synonymous to what we call atomic concepts, or to denote the (atomic) left-hand sides of concept inclusions.

$$
\begin{array}{rcl}
\mathsf{Woman} & \equiv & \mathsf{Person} \sqcap \mathsf{Female} \\
\mathsf{Man} & \equiv & \mathsf{Person} \sqcap \neg(\mathsf{Person} \sqcap \mathsf{Female}) \\
\mathsf{Mother} & \equiv & (\mathsf{Person} \sqcap \mathsf{Female}) \sqcap \exists \mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Father} & \equiv & (\mathsf{Person} \sqcap \neg(\mathsf{Person} \sqcap \mathsf{Female})) \sqcap \exists \mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Parent} & \equiv & ((\mathsf{Person} \sqcap \neg(\mathsf{Person} \sqcap \mathsf{Female})) \sqcap \exists \mathsf{hasChild}.\mathsf{Person}) \\
& & \sqcup ((\mathsf{Person} \sqcap \mathsf{Female}) \sqcap \exists \mathsf{hasChild}.\mathsf{Person}) \\
\mathsf{Grandmother} & \equiv & ((\mathsf{Person} \sqcap \mathsf{Female}) \sqcap \exists \mathsf{hasChild}.\mathsf{Person}) \\
& & \sqcap \exists \mathsf{hasChild}.(((\mathsf{Person} \sqcap \neg(\mathsf{Person} \sqcap \mathsf{Female})) \\
& & \qquad \sqcap \exists \mathsf{hasChild}.\mathsf{Person}) \\
& & \qquad \sqcup ((\mathsf{Person} \sqcap \mathsf{Female}) \\
& & \qquad\quad \sqcap \exists \mathsf{hasChild}.\mathsf{Person})) \\
\mathsf{MotherWithManyChildren} & \equiv & ((\mathsf{Person} \sqcap \mathsf{Female}) \sqcap \exists \mathsf{hasChild}.\mathsf{Person}) \sqcap \geqslant 3\, \mathsf{hasChild} \\
\mathsf{MotherWithoutDaughter} & \equiv & ((\mathsf{Person} \sqcap \mathsf{Female}) \sqcap \exists \mathsf{hasChild}.\mathsf{Person}) \\
& & \sqcap \forall \mathsf{hasChild}.(\neg(\mathsf{Person} \sqcap \mathsf{Female})) \\
\mathsf{Wife} & \equiv & (\mathsf{Person} \sqcap \mathsf{Female}) \\
& & \sqcap \exists \mathsf{hasHusband}.(\mathsf{Person} \sqcap \neg(\mathsf{Person} \sqcap \mathsf{Female}))
\end{array}
$$

Fig. 2.3. The expansion of the Family TBox in Figure 2.2.

definition of A, and we call *uses* the transitive closure of the relation *directly uses*. Then $\mathcal{T}$ contains a *cycle* iff there exists an atomic concept in $\mathcal{T}$ that uses itself. Otherwise, $\mathcal{T}$ is called *acyclic*.

Unique extensions need not exist if a terminology contains cycles. Consider, for instance, the terminology that contains only Axiom (2.1). Here, Human' is a name symbol and Animal and $\mathsf{hasParent}$ are base symbols. For an interpretation where $\mathsf{hasParent}$ relates every animal to its progenitors, many extensions are possible to interpret Human' in a such a way that the axiom is satisfied: Human' can, among others, be interpreted as the set of all animals, as some species, or any other set of animals with the property that for each animal it contains also its progenitors.

In contrast, if a terminology $\mathcal{T}$ is acyclic, then it is definitorial. The reason is that we can expand through an iterative process the definitions in $\mathcal{T}$ by replacing each occurrence of a name on the right-hand side of a definition with the concepts that it stands for. Since there is no cycle in the set of definitions, the process eventually stops and we end up with a terminology $\mathcal{T}'$ consisting solely of definitions of the form $A \equiv C'$, where C' contains only base symbols and no name symbols. We call $\mathcal{T}'$ the *expansion* of $\mathcal{T}$. Note that the size of the expansion can be exponential in the size of the original terminology [Nebel, 1990]. The Family TBox in Figure 2.2 is acyclic. Therefore, we can compute the expansion, which is shown in Figure 2.3.

Proposition 2.1 *Let $\mathcal{T}$ be a acyclic terminology and $\mathcal{T}'$ be its expansion. Then*

(i) *$\mathcal{T}$ and $\mathcal{T}'$ have the same name and base symbols;*

(ii) *$\mathcal{T}$ and $\mathcal{T}'$ are equivalent;*

(iii) *both, $\mathcal{T}$ and $\mathcal{T}'$, are definitorial.*

Proof Let $\mathcal{T}_1$ be a terminology. Suppose $A \equiv C$ and $B \equiv D$ are definitions in $\mathcal{T}_1$ such that B occurs in C. Let C' be the concept obtained from C by replacing each occurrence of B in C with D, and let $\mathcal{T}_2$ be the terminology obtained from $\mathcal{T}_1$ by replacing the definition $A \equiv C$ with $A \equiv C'$. Then both terminologies have the same name and base symbols. Moreover, since $\mathcal{T}_2$ has been obtained from $\mathcal{T}_1$ by replacing equals by equals, both terminologies have the same models. Since $\mathcal{T}'$ is obtained from $\mathcal{T}$ by a sequence of replacement steps like the ones above, this proves claims (i) and (ii).

Suppose now that $\mathcal{J}$ is an interpretation of the base symbols. We extend it to an interpretation $\mathcal{I}$ that covers also the name symbols by setting $A^{\mathcal{I}} = C'^{\mathcal{J}}$, if $A \equiv C'$ is the definition of A in $\mathcal{T}'$. Clearly, $\mathcal{I}$ is a model of $\mathcal{T}'$, and it is the only extension of $\mathcal{J}$ that is a model of $\mathcal{T}'$. This shows that $\mathcal{T}'$ is definitorial. Moreover, $\mathcal{T}$ is definitorial as well, since it is equivalent to $\mathcal{T}'$. $\square$

It is characteristic for acyclic terminologies, in a sense to be made more precise, to uniquely define the name symbols in terms of the base symbols.

Of course, there are also terminologies *with* cycles that are definitorial. Consider for instance the one consisting of the axiom

$$A \equiv \forall R.B \sqcup \exists R.(A \sqcap \neg A), \tag{2.2}$$

which has a cycle. However, since $\exists R.(A \sqcap \neg A)$ is equivalent to the bottom concept, Axiom (2.2) is equivalent to the acyclic axiom

$$A \equiv \forall R.B. \tag{2.3}$$

This example is typical for the general situation.

Theorem 2.2 *Every definitorial $\mathcal{ALC}$-terminology is equivalent to an acyclic terminology.*

The theorem is a reformulation of Beth's Definability Theorem [Gabbay, 1972] for the modal propositional logic $\mathbf{K}_n$, which, as shown by Schild [1991], is a notational variant of $\mathcal{ALC}$.

2.2.2.3 Fixpoint semantics for terminological cycles

Under the semantics we have studied so far, which is essentially the semantics of first-order logic, terminologies have definitorial impact only if they are essentially acyclic. Following Nebel [1991], we shall call this semantics *descriptive* semantics to distinguish it from the fixpoint semantics introduced below. Fixpoint semantics are motivated by the fact that there are situations where intuitively cyclic definitions are meaningful and the intuition can be captured by least or greatest fixpoint semantics.

Example 2.3 Suppose that we want to specify the concept of a "man who has only male offspring," for short Momo. In particular, such a man is a Mos, that is, a "man who has only sons." A Mos can be defined without cycles as

$$\mathsf{Mos} \quad \equiv \quad \mathsf{Man} \sqcap \forall \mathsf{hasChild}.\mathsf{Man}.$$

For a Momo, however, we want to make a statement about the fillers of the transitive closure of the role $\mathsf{hasChild}$. Here a recursive definition of Momo seems to be natural. A man having only male offspring is himself a man, and all his children are men having only male offspring:

$$\mathsf{Momo} \quad \equiv \quad \mathsf{Man} \sqcap \forall \mathsf{hasChild}.\mathsf{Momo}. \tag{2.4}$$

In order to achieve the desired meaning, we have to interpret this definition under an appropriate fixpoint semantics. We shall show below that greatest fixpoint semantics captures our intuition here. ∎

Cycles also appear when we want to model recursive structures, e.g., binary trees.[1]

Example 2.4 We suppose that there is a set of objects that are Trees and a binary relation $\mathsf{has\text{-}branch}$ between objects that leads from a tree to its subtrees. Then the binary trees are the trees with at most two subtrees that are themselves binary trees:

$$\mathsf{BinaryTree} \quad \equiv \quad \mathsf{Tree} \sqcap \leqslant 2\, \mathsf{has\text{-}branch} \sqcap \forall \mathsf{has\text{-}branch}.\mathsf{BinaryTree}.$$

As with the definition of Momo, a fixpoint semantics will yield the desired meaning. However, for this example we have to use least fixpoint semantics. ∎

We now give a formal definition of fixpoint semantics. In a terminology $\mathcal{T}$, every name symbol A occurs exactly once as the left-hand side of an axiom $A \equiv C$. Therefore, we can view $\mathcal{T}$ as a mapping that associates to a name symbol A the concept description $\mathcal{T}(A) = C$. With this notation, an interpretation $\mathcal{I}$ is a model of $\mathcal{T}$ if, and only if, $A^{\mathcal{I}} = (\mathcal{T}(A))^{\mathcal{I}}$. This characterization has the flavour of a fixpoint equation. We exploit this similarity to introduce a family of mappings such that an interpretation is a model of $\mathcal{T}$ iff it is a fixpoint of such a mapping.

Let $\mathcal{T}$ be a terminology, and let $\mathcal{J}$ be a fixed base interpretation of $\mathcal{T}$. By $\mathit{Ext}_{\mathcal{J}}$ we denote the set of all extensions of $\mathcal{J}$. Let $\mathcal{T}_{\mathcal{J}}\colon \mathit{Ext}_{\mathcal{J}} \to \mathit{Ext}_{\mathcal{J}}$ be the mapping

[1] The following example is taken from [Nebel, 1991].

that maps the extension $\mathcal{I}$ to the extension $\mathcal{T}_{\mathcal{J}}(\mathcal{I})$ defined by $A^{\mathcal{T}_{\mathcal{J}}(\mathcal{I})} = (\mathcal{T}(A))^{\mathcal{I}}$ for each name symbol A.

Now, $\mathcal{I}$ is a fixpoint of $\mathcal{T}_{\mathcal{J}}$ iff $\mathcal{I} = \mathcal{T}_{\mathcal{J}}(\mathcal{I})$, i.e., iff $A^{\mathcal{I}} = A^{\mathcal{T}_{\mathcal{J}}(\mathcal{I})}$ for all name symbols. This means that, for every definition $A \equiv C$ in $\mathcal{T}$, we have $A^{\mathcal{I}} = A^{\mathcal{T}_{\mathcal{J}}(\mathcal{I})} = (\mathcal{T}(A))^{\mathcal{I}} = C^{\mathcal{I}}$, which means that $\mathcal{I}$ is a model of $\mathcal{T}$. This proves the following result.

Proposition 2.5 *Let $\mathcal{T}$ be a terminology, $\mathcal{I}$ be an interpretation, and $\mathcal{J}$ be the restriction of $\mathcal{I}$ to the base symbols of $\mathcal{T}$. Then $\mathcal{I}$ is a model of $\mathcal{T}$ if, and only if, $\mathcal{I}$ is a fixpoint of $\mathcal{T}_{\mathcal{J}}$.*

According to the preceding proposition, a terminology $\mathcal{T}$ is definitorial iff every base interpretation $\mathcal{J}$ has a unique extension that is a fixpoint of $\mathcal{T}_{\mathcal{J}}$.

Example 2.6 To get a feel for why cyclic terminologies are not definitorial, we discuss as an example the terminology $\mathcal{T}^{\mathsf{Momo}}$ that consists only of Axiom (2.4). Consider the base interpretation $\mathcal{J}$ defined by

$$\begin{array}{rcl} \Delta^{\mathcal{J}} & = & \{\mathit{Charles}_1,\, \mathit{Charles}_2, \ldots\} \cup \{\mathit{James}_1, \ldots, \mathit{James}_{\mathit{Last}}\}, \\ \mathsf{Man}^{\mathcal{J}} & = & \Delta^{\mathcal{J}}, \\ \mathsf{hasChild}^{\mathcal{J}} & = & \{(\mathit{Charles}_i, \mathit{Charles}_{(i+1)}) \mid i \geq 1\} \cup \\ & & \{(\mathit{James}_i, \mathit{James}_{(i+1)}) \mid 1 \leq i < \mathit{Last}\}. \end{array}$$

This means that the *Charles* dynasty does not die out, whereas there is a last member of the *James* dynasty.

We want to identify the fixpoints of $\mathcal{T}_{\mathcal{J}}^{\mathsf{Momo}}$. Note that an individual without children, i.e., without fillers of $\mathsf{hasChild}$, is always in the interpretation of $\forall\mathsf{hasChild}.\mathsf{Momo}$, no matter how Momo is interpreted. Therefore, if $\mathcal{I}$ is a fixpoint extension of $\mathcal{J}$, then $\mathit{James}_{\mathit{Last}}$ is in $(\forall\mathsf{hasChild}.\mathsf{Momo})^{\mathcal{I}}$, and thus in $\mathsf{Momo}^{\mathcal{I}}$. We conclude that every *James* is a Momo. Let $\mathcal{I}_1$ be the extension of $\mathcal{J}$ such that $\mathsf{Momo}^{\mathcal{I}_1}$ comprises exactly the *James* dynasty. Then it is easy to check that $\mathcal{I}_1$ is a fixpoint. If, in addition to the *James* dynasty, also some *Charles* is a Momo, then all the members of the *Charles* dynasty before and after him must belong to the concept Momo. One can easily check that the extension $\mathcal{I}_2$ that interprets Momo as the entire domain is also a fixpoint, and that there is no other fixpoint. ∎

In order to give definitorial impact to a cyclic terminology $\mathcal{T}$, we must single out a particular fixpoint of the mapping $\mathcal{T}_{\mathcal{J}}$ if there are more than one. To this end, we define a partial ordering "$\preceq$" on the extensions of $\mathcal{J}$. We say that $\mathcal{I} \preceq \mathcal{I}'$ if $A^{\mathcal{I}} \subseteq A^{\mathcal{I}'}$ for every name symbol in $\mathcal{T}$. In the above example, Momo is the only name symbol. Since $\mathsf{Momo}^{\mathcal{I}_1} \subseteq \mathsf{Momo}^{\mathcal{I}_2}$, we have $\mathcal{I}_1 \preceq \mathcal{I}_2$.

A fixpoint $\mathcal{I}$ of $\mathcal{T}_{\mathcal{J}}$ is the *least fixpoint* (lfp) if $\mathcal{I} \preceq \mathcal{I}'$ for every other fixpoint $\mathcal{I}'$. We say that $\mathcal{I}$ is a *least fixpoint model* of $\mathcal{T}$ if $\mathcal{I}$ is the least fixpoint of $\mathcal{T}_{\mathcal{J}}$. for some base interpretation $\mathcal{J}$. Under *least fixpoint semantics* we only admit the least fixpoint models of $\mathcal{T}$ as intended interpretations. Greatest fixpoints (gfp), greatest fixpoint models, and greatest fixpoint semantics are defined analogously. In the Momo example, $\mathcal{I}_1$ is the least and $\mathcal{I}_2$ the greatest fixpoint of $\mathcal{T}_{\mathcal{J}}$.

2.2.2.4 Existence of fixpoint models

Least and greatest fixpoint models need not exist for every terminology.

Example 2.7 As a simple example, consider the axiom

$$A \equiv \neg A. \tag{2.5}$$

If $\mathcal{I}$ is a model of this axiom, then $A^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$, which implies $\Delta^{\mathcal{I}} = \emptyset$, an absurdity.

A terminology containing Axiom (2.5) thus does not have any models, and therefore also no gfp (lfp) models.

There are also cases where models (i.e., fixpoints) exist, but there is neither a least one nor a greatest one. As an example, consider the terminology $\mathcal{T}$ with the single axiom

$$A \equiv \forall R.\neg A. \tag{2.6}$$

Let $\mathcal{J}$ be the base interpretation with $\Delta^{\mathcal{J}} = \{a, b\}$ and $R^{\mathcal{J}} = \{(a, b), (b, a)\}$. Then there are two fixpoint extensions $\mathcal{I}_1$, $\mathcal{I}_2$, defined by $A^{\mathcal{I}_1} = \{a\}$ and $A^{\mathcal{I}_2} = \{b\}$. However, they are not comparable with respect to "$\preceq$". ∎

In order to identify terminologies with the property that for every base interpretation there exists a least and a greatest fixpoint extension, we draw upon results from lattice theory. Recall that a lattice is *complete* if every family of elements has a least upper bound.

On $Ext_{\mathcal{J}}$ we have introduced the partial ordering "$\preceq$". For a family of interpretations $(\mathcal{I}_i)_{i\in I}$ in $Ext_{\mathcal{J}}$ we define $\mathcal{I}_0 = \bigsqcup_{i\in I} \mathcal{I}_i$ as the pointwise union of the $\mathcal{I}_i$s, that is, for every name symbol A we have $A^{\mathcal{I}_0} = \bigcup_{i\in I} A^{\mathcal{I}_i}$. Then $\mathcal{I}_0$ is the least upper bound of the $\mathcal{I}_i$s, which shows that $(Ext_{\mathcal{J}}, \preceq)$ is a complete lattice.

A function $f\colon L \to L$ on a lattice $(L, \preceq)$ is *monotone* if $f(x) \preceq f(y)$ whenever $x \preceq y$. Tarski's Fixpoint Theorem [Tarski, 1955] says that for a monotone function on a complete lattice the set of fixpoints is nonempty and forms itself a complete lattice. In particular, there is a least and a greatest fixpoint.

We define that a terminology $\mathcal{T}$ is *monotone* if the mapping $\mathcal{T}_{\mathcal{J}}$ is monotone for all base interpretations $\mathcal{J}$. By Tarski's theorem, such terminologies have greatest and least fixpoints. However, to apply the theorem, we must be able to recognize monotone terminologies. A simple syntactic criterion is the following. We call a terminology *negation free* if no negation occurs in it. By an induction over the depth of concept descriptions one can check that every negation free $\mathcal{ALCN}$-terminology is monotone.

Proposition 2.8 *If $\mathcal{T}$ is a negation free terminology and $\mathcal{J}$ a base interpretation, then there exist extensions of $\mathcal{J}$ that are a lfp-model and a gfp-model of $\mathcal{T}$, respectively.*

Negation free terminologies are not the most general class of terminologies having least and greatest fixpoints. We have seen in Proposition 2.1 that acyclic terminologies are definitorial and thus for a given base interpretation admit only a single extension that is a model, which then is both a least and a greatest fixpoint model.

We obtain a more refined criterion for the existence of least and greatest fixpoints if we pay attention to the interplay between cycles and negation. To this end, we associate to a terminology $\mathcal{T}$ a *dependency* graph $G_{\mathcal{T}}$, whose nodes are the name symbols in $\mathcal{T}$. If $\mathcal{T}$ contains the axiom $A \equiv C$, then for every occurrence of the name symbol A' in C, there is an arc from A to A' in $G_{\mathcal{T}}$. Arcs are labeled as positive and negative. The arc from A to A' is positive if A' occurs in C in the scope of an even number of negations, and it is negative if A' occurs in the scope of an odd number of negations. A sequence of nodes $A_1, \ldots, A_n$ is a *path* if there is an arc in $G_{\mathcal{T}}$ from A_i to A_{i+1} for all $i = 1, \ldots, n-1$. A path is a cycle if $A_1 = A_n$.

Proposition 2.9 *Let $\mathcal{T}$ be a terminology such that each cycle in $G_{\mathcal{T}}$ contains an even number of negative arcs. Then $\mathcal{T}$ is monotone.*

We call a terminology satisfying the precondition of this proposition *syntactically monotone.*

2.2.2.5 Terminologies with inclusion axioms

For certain concepts we may be unable to define them completely. In this case, we can still state necessary conditions for concept membership using an inclusion. We call an inclusion whose left-hand side is atomic a *specialization.*

For example, if a (male) knowledge engineer thinks that the definition of "woman" in our example TBox (Figure 2.2) is not satisfactory, but if he also feels that he is not able to define the concept "woman" in all detail, he can require that every woman is a person with the specialization

$$\mathsf{Woman} \sqsubseteq \mathsf{Person}. \tag{2.7}$$

If we allow also specializations in a terminology, then the terminology loses its definitorial impact, even if it is acyclic. A set of axioms $\mathcal{T}$ is a *generalized terminology* if the left-hand side of each axiom is an atomic concept and for every atomic concept there is at most one axiom where it occurs on the left-hand side.

We shall transform a generalized terminology $\mathcal{T}$ into a regular terminology $\bar{\mathcal{T}}$, containing definitions only, such that $\bar{\mathcal{T}}$ is equivalent to $\mathcal{T}$ in a sense that will be specified below. We obtain $\bar{\mathcal{T}}$ from $\mathcal{T}$ by choosing for every specialization $A \sqsubseteq C$ in $\mathcal{T}$ a new base symbol $\bar{A}$ and by replacing the specialization $A \sqsubseteq C$ with the definition $A \equiv \bar{A} \sqcap C$. The terminology $\bar{\mathcal{T}}$ is the *normalization* of $\mathcal{T}$.

If a TBox contains the specialization (2.7), then the normalization contains the definition

$$\mathsf{Woman} \equiv \overline{\mathsf{Woman}} \sqcap \mathsf{Person}.$$

Intuitively, the additional base symbol $\overline{\mathsf{Woman}}$ stands for the qualities that distinguish a woman among persons. Thus, normalization results in a TBox with a definition for Woman that is similar to the one in the Family TBox.

Proposition 2.10 *Let $\mathcal{T}$ be a generalized terminology and $\bar{\mathcal{T}}$ its normalization.*

- *Every model of $\bar{\mathcal{T}}$ is a model of $\mathcal{T}$.*
- *For every model $\mathcal{I}$ of $\mathcal{T}$ there is a model $\bar{\mathcal{I}}$ of $\bar{\mathcal{T}}$ that has the same domain as $\mathcal{I}$ and agrees with $\mathcal{I}$ on the atomic concepts and roles in $\mathcal{T}$.*

Proof The first claim holds because a model $\bar{\mathcal{I}}$ of $\bar{\mathcal{T}}$ satisfies $A^{\bar{\mathcal{I}}} = (\bar{A} \sqcap C)^{\bar{\mathcal{I}}} = \bar{A}^{\bar{\mathcal{I}}} \cap C^{\bar{\mathcal{I}}}$, which implies $A^{\bar{\mathcal{I}}} \subseteq C^{\bar{\mathcal{I}}}$. Conversely, if $\mathcal{I}$ is a model of $\mathcal{T}$, then the extension $\bar{\mathcal{I}}$ of $\mathcal{I}$, defined by $\bar{A}^{\bar{\mathcal{I}}} = A^{\mathcal{I}}$, is a model of $\bar{\mathcal{T}}$, because $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ implies $A^{\mathcal{I}} = A^{\mathcal{I}} \cap C^{\mathcal{I}} = \bar{A}^{\bar{\mathcal{I}}} \cap C^{\bar{\mathcal{I}}}$, and therefore $\bar{\mathcal{I}}$ satisfies $A \equiv \bar{A} \sqcap C$. □

Thus, in theory, inclusion axioms do not add to the expressivity of terminologies. However, in practice, they are a convenient means to introduce terms into a terminology that cannot be defined completely.

2.2.3 World descriptions

The second component of a knowledge base, in addition to the terminology or TBox, is the *world description* or *ABox*.

2.2.3.1 Assertions about individuals

In the ABox, one describes a specific state of affairs of an application domain in terms of concepts and roles. Some of the concept and role atoms in the ABox may be defined names of the TBox. In the ABox, one introduces individuals, by giving them names, and one asserts properties of these individuals. We denote individual

MotherWithoutDaughter(MARY)	Father(PETER)
hasChild(MARY, PETER)	hasChild(PETER, HARRY)
hasChild(MARY, PAUL)	

Fig. 2.4. A world description (ABox).

names as a, b, c. Using concepts C and roles R, one can make assertions of the following two kinds in an ABox:

$$C(a), \qquad R(b,c).$$

By the first kind, called *concept assertions*, one states that a belongs to (the interpretation of) C, by the second kind, called *role assertions*, one states that c is a filler of the role R for b. For instance, if PETER, PAUL, and MARY are individual names, then Father(PETER) means that Peter is a father, and hasChild(MARY, PAUL) means that Paul is a child of Mary. An *ABox*, denoted as $\mathcal{A}$, is a finite set of such assertions. Figure 2.4 shows an example of an ABox.

In a simplified view, an ABox can be seen as an instance of a relational database with only unary or binary relations. However, contrary to the "closed-world semantics" of classical databases, the semantics of ABoxes is an "open-world semantics," since normally knowledge representation systems are applied in situations where one cannot assume that the knowledge in the KB is complete.[1] Moreover, the TBox imposes semantic relationships between the concepts and roles in the ABox that do not have counterparts in database semantics.

We give a semantics to ABoxes by extending interpretations to individual names. From now on, an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ not only maps atomic concepts and roles to sets and relations, but in addition maps each individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. We assume that distinct individual names denote distinct objects. Therefore, this mapping has to respect the *unique name assumption* (UNA), that is, if a, b are distinct names, then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. The interpretation $\mathcal{I}$ *satisfies* the concept assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and it *satisfies* the role assertion $R(a,b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An interpretation *satisfies* the ABox $\mathcal{A}$ if it satisfies each assertion in $\mathcal{A}$. In this case we say that $\mathcal{I}$ is a *model* of the assertion or of the ABox. Finally, $\mathcal{I}$ *satisfies* an assertion α or an ABox $\mathcal{A}$ *with respect to* a TBox $\mathcal{T}$ if in addition to being a model of α or of $\mathcal{A}$, it is a model of $\mathcal{T}$. Thus, a model of $\mathcal{A}$ and $\mathcal{T}$ is an abstraction of a concrete world where the concepts are interpreted as subsets of the domain as required by the TBox and where the membership of the individuals to concepts and their relationships with one another in terms of roles respect the assertions in the ABox.

[1] We discuss implications of this difference in semantics in Section 2.2.4.4.

2.2.3.2 Individual names in the description language

Sometimes, it is convenient to allow *individual names* (also called *nominals*) not only in the ABox, but also in the description language. Some concept constructors employing individuals occur in systems and have been investigated in the literature. The most basic one is the "set" (or *one-of*) constructor, written

$$\{a_1, \ldots, a_n\},$$

where $a_1, \ldots, a_n$ are individual names. As one would expect, such a set concept is interpreted as

$$\{a_1, \ldots, a_n\}^{\mathcal{I}} = \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}. \tag{2.8}$$

With sets in the description language one can for instance define the concept of permanent members of the UN security council as $\{\mathsf{CHINA}, \mathsf{FRANCE}, \mathsf{RUSSIA}, \mathsf{UK}, \mathsf{USA}\}$.

In a language with the union constructor "$\sqcup$", a constructor $\{a\}$ for singleton sets alone adds sufficient expressiveness to describe arbitrary finite sets since, according to the semantics of the set constructor in Equation (2.8), the concepts $\{a_1, \ldots, a_n\}$ and $\{a_1\} \sqcup \cdots \sqcup \{a_n\}$ are equivalent.

Another constructor involving individual names is the "fills" constructor

$$R : a,$$

for a role R. The semantics of this constructor is defined as

$$(R : a)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid (d, a^{\mathcal{I}}) \in R^{\mathcal{I}}\}, \tag{2.9}$$

that is, $R : a$ stands for the set of those objects that have a as a filler of the role R. To a description language with singleton sets and full existential quantification, "fills" does not add anything new, since Equation (2.9) implies that $R : a$ and $\exists R.\{a\}$ are equivalent.

We note, finally, that "fills" allows one to express role assertions through concept assertions: an interpretation satisfies $R(a, b)$ iff it satisfies $(\exists R.\{b\})(a)$.

2.2.4 Inferences

A knowledge representation system based on DLs is able to perform specific kinds of reasoning. As said before, the purpose of a knowledge representation system goes beyond storing concept definitions and assertions. A knowledge base—comprising TBox and ABox—has a semantics that makes it equivalent to a set of axioms in first-order predicate logic. Thus, like any other set of axioms, it contains implicit knowledge that can be made explicit through inferences. For example, from the TBox in Figure 2.2 and the ABox in Figure 2.4 one can conclude that Mary is a grandmother, although this knowledge is not explicitly stated as an assertion.

The different kinds of reasoning performed by a DL system (see Chapter 8) are defined as logical inferences. In the following, we shall discuss these inferences, first for concepts, then for TBoxes and ABoxes, and finally for TBoxes and ABoxes together. It will turn out that there is one main inference problem, namely the consistency check for ABoxes, to which all other inferences can be reduced.

2.2.4.1 Reasoning tasks for concepts

When a knowledge engineer models a domain, she constructs a terminology, say $\mathcal{T}$, by defining new concepts, possibly in terms of others that have been defined before. During this process, it is important to find out whether a newly defined concept makes sense or whether it is contradictory. From a logical point of view, a concept makes sense for us if there is some interpretation that satisfies the axioms of $\mathcal{T}$ (that is, a model of $\mathcal{T}$) such that the concept denotes a nonempty set in that interpretation. A concept with this property is said to be *satisfiable* with respect to $\mathcal{T}$ and *unsatisfiable* otherwise.

Checking satisfiability of concepts is a key inference. As we shall see, a number of other important inferences for concepts can be reduced to the (un)satisfiability. For instance, in order to check whether a domain model is correct, or to optimize queries that are formulated as concepts, we may want to know whether some concept is more general than another one: this is the *subsumption problem.* A concept C *is subsumed* by a concept D if in every model of $\mathcal{T}$ the set denoted by C is a subset of the set denoted by D. Algorithms that check subsumption are also employed to organize the concepts of a TBox in a taxonomy according to their generality. Further interesting relationships between concepts are *equivalence* and *disjointness.*

These properties are formally defined as follows. Let $\mathcal{T}$ be a TBox.

Satisfiability: A concept C is *satisfiable* with respect to $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty. In this case we say also that $\mathcal{I}$ is a *model* of C.

Subsumption: A concept C is *subsumed* by a concept D with respect to $\mathcal{T}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. In this case we write $C \sqsubseteq_{\mathcal{T}} D$ or $\mathcal{T} \models C \sqsubseteq D$.

Equivalence: Two concepts C and D are *equivalent* with respect to $\mathcal{T}$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. In this case we write $C \equiv_{\mathcal{T}} D$ or $\mathcal{T} \models C \equiv D$.

Disjointness: Two concepts C and D are *disjoint* with respect to $\mathcal{T}$ if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$.

If the TBox $\mathcal{T}$ is clear from the context, we sometimes drop the qualification "with respect to $\mathcal{T}$."

We also drop the qualification in the special case where the TBox is empty, and we simply write $\models C \sqsubseteq D$ if C is subsumed by D, and $\models C \equiv D$ if C and D are equivalent.

Example 2.11 With respect to the TBox in Figure 2.2, Person subsumes Woman, both Woman and Parent subsume Mother, and Mother subsumes Grandmother. Moreover, Woman and Man, and Father and Mother are disjoint. The subsumption relationships follow from the definitions because of the semantics of "$\sqcap$" and "$\sqcup$". That Man is disjoint from Woman is due to the fact that Man is subsumed by the negation of Woman. ■

Traditionally, the basic reasoning mechanism provided by DL systems checked the subsumption of concepts. This, in fact, is sufficient to implement also the other inferences, as can be seen by the following reductions.

Proposition 2.12 (Reduction to Subsumption) *For concepts C, D we have*

(i) *C is unsatisfiable $\Leftrightarrow$ C is subsumed by $\bot$;*
(ii) *C and D are equivalent $\Leftrightarrow$ C is subsumed by D and D is subsumed by C;*
(iii) *C and D are disjoint $\Leftrightarrow$ $C \sqcap D$ is subsumed by $\bot$.*

The statements also hold with respect to a TBox.

All description languages implemented in actual DL systems provide the intersection operator "$\sqcap$" and almost all of them contain an unsatisfiable concept. Thus, most DL systems that can check subsumption can perform all four inferences defined above.

If, in addition to intersection, a system allows one also to form the negation of a description, one can reduce subsumption, equivalence, and disjointness of concepts to the satisfiability problem (see also Smolka [1988]).

Proposition 2.13 (Reduction to Unsatisfiability) *For concepts C, D we have*

(i) *C is subsumed by D $\Leftrightarrow$ $C \sqcap \neg D$ is unsatisfiable;*
(ii) *C and D are equivalent $\Leftrightarrow$ both $(C \sqcap \neg D)$ and $(\neg C \sqcap D)$ are unsatisfiable;*
(iii) *C and D are disjoint $\Leftrightarrow$ $C \sqcap D$ is unsatisfiable.*

The statements also hold with respect to a TBox.

The reduction of subsumption can easily be understood if one recalls that, for sets M, N, we have $M \subseteq N$ iff $M \setminus N = \emptyset$. The reduction of equivalence is correct because C and D are equivalent if, and only if, C is subsumed by D and D is subsumed by C. Finally, the reduction of disjointness is just a rephrasing of the definition.

Because of the above proposition, in order to obtain decision procedures for any of the four inferences we have discussed, it is sufficient to develop algorithms that decide the satisfiability of concepts, provided the language for which we can decide satisfiability supports conjunction as well as negation of arbitrary concepts.

In fact, this observation motivated researchers to study description languages in which, for every concept, one can also form the negation of that concept [Smolka, 1988; Schmidt-Schauß and Smolka, 1991; Donini *et al.*, 1991; 1997]. The approach to consider satisfiability checking as the principal inference gave rise to a new kind of algorithms for reasoning in DLs, which can be understood as specialized tableaux calculi (see Section 2.3 in this chapter and Chapter 3). Also, the most recent generation of DL systems, like Kris [Baader and Hollunder, 1991], Crack [Bresciani *et al.*, 1995], FaCT [Horrocks, 1998], Dlp [Patel-Schneider, 1999], and Race [Haarslev and Möller, 2001b], are based on satisfiability checking, and a considerable amount of research work is spent on the development of efficient implementation techniques for this approach [Baader *et al.*, 1994; Horrocks, 1998; Horrocks and Patel-Schneider, 1999; Haarslev and Möller, 2001a].

In an $\mathcal{AL}$-language without full negation, subsumption and equivalence cannot be reduced to unsatisfiability in the simple way shown in Proposition 2.13 and therefore these inferences may be of different complexity.

As seen in Proposition 2.12, from the viewpoint of worst-case complexity, subsumption is the most general inference for any $\mathcal{AL}$-language. The next proposition shows that unsatisfiability is a special case of each of the other problems.

Proposition 2.14 (Reducing Unsatisfiability) *Let C be a concept. Then the following are equivalent:*

(i) *C is unsatisfiable;*
(ii) *C is subsumed by $\bot$;*
(iii) *C and $\bot$ are equivalent;*
(iv) *C and $\top$ are disjoint.*

The statements also hold with respect to a TBox.

From Propositions 2.12 and 2.14 we see that, in order to obtain upper and lower complexity bounds for inferences on concepts in $\mathcal{AL}$-languages, it suffices to assess lower bounds for unsatisfiability and upper bounds for subsumption. More precisely, for each $\mathcal{AL}$-language, an upper bound for the complexity of the subsumption problem is also an upper bound for the complexity of the unsatifiability, the equivalence, and the disjointness problem. Moreover, a lower bound for the complexity of the unsatifiability problem is also a lower bound for the complexity of the subsumption, the equivalence, and the disjointness problem.

2.2.4.2 Eliminating the TBox

In applications, concepts usually come in the context of a TBox. However, for developing reasoning procedures it is conceptually easier to abstract from the TBox or, what amounts to the same, to assume that it is empty.

We show that, if $\mathcal{T}$ is an acyclic TBox, we can always reduce reasoning problems with respect to $\mathcal{T}$ to problems with respect to the empty TBox. As we have seen in Proposition 2.1, $\mathcal{T}$ is equivalent to its expansion $\mathcal{T}'$. Recall that in the expansion every definition is of the form $A \equiv D$ such that D contains only base symbols, but no name symbols. Now, for each concept C we define the *expansion of C with respect to $\mathcal{T}$* as the concept C' that is obtained from C by replacing each occurrence of a name symbol A in C by the concept D, where $A \equiv D$ is the definition of A in $\mathcal{T}'$, the expansion of $\mathcal{T}$.

For example, we obtain the expansion of the concept

$$\mathsf{Woman} \sqcap \mathsf{Man} \tag{2.10}$$

with respect to the TBox in Figure 2.2 by considering the expanded TBox in Figure 2.3, and replacing Woman and Man with the right-hand sides of their definitions in this expansion. This results in the concept

$$\mathsf{Person} \sqcap \mathsf{Female} \sqcap \mathsf{Person} \sqcap \neg(\mathsf{Person} \sqcap \mathsf{Female}). \tag{2.11}$$

We can readily deduce a number of facts about expansions. Since the expansion C' is obtained from C by replacing names with descriptions in such a way that both are interpreted in the same way in any model of $\mathcal{T}$, it follows that

- $C \equiv_{\mathcal{T}} C'$.

Hence, C is satisfiable w.r.t. $\mathcal{T}$ iff C' is satisfiable w.r.t. $\mathcal{T}$. However, C' contains no defined names, and thus C' is satisfiable w.r.t. $\mathcal{T}$ iff it is satisfiable. This yields that

- *C is satisfiable w.r.t. $\mathcal{T}$ iff C' is satisfiable.*

If D is another concept, then we have also $D \equiv_{\mathcal{T}} D'$. Thus, $C \sqsubseteq_{\mathcal{T}} D$ iff $C' \sqsubseteq_{\mathcal{T}} D'$, and $C \equiv_{\mathcal{T}} D$ iff $C' \equiv_{\mathcal{T}} D'$. Again, since C' and D' contain only base symbols, this implies

- $\mathcal{T} \models C \sqsubseteq D$ *iff* $\models C' \sqsubseteq D'$;
- $\mathcal{T} \models C \equiv D$ *iff* $\models C' \equiv D'$.

With similar arguments we can show that

- *C and D are disjoint w.r.t. $\mathcal{T}$ iff C' and D' are disjoint.*

Summing up, expanding concepts with respect to an acyclic TBox allows one to get rid of the TBox in reasoning problems. Going back to our example from above, this means that, in order to verify whether Man and Woman are disjoint with respect to the Family TBox, which amounts to checking whether $\mathsf{Man} \sqcap \mathsf{Woman}$ is unsatisfiable, it suffices to check that the concept (2.11) is unsatisfiable.

Expanding concepts may be computationally costly, since in the worst case the size of $\mathcal{T}'$ is exponential in the size of $\mathcal{T}$, and therefore C' may be larger than C by a factor that is exponential in the size of $\mathcal{T}$. A complexity analysis of the difficulty of reasoning with respect to TBoxes shows that the expansion of definitions is a source of complexity that cannot always be avoided (see Subsection 2.3.3 of this chapter and Chapter 3).

2.2.4.3 Reasoning tasks for ABoxes

After a knowledge engineer has designed a terminology and has used the reasoning services of her DL system to check that all concepts are satisfiable and that the the expected subsumption relationships hold, the ABox can be filled with assertions about individuals. We recall that an ABox contains two kinds of assertions, concept assertions of the form $C(a)$ and role assertions of the form $R(a, b)$. Of course, the representation of such knowledge has to be consistent, because otherwise—from the viewpoint of logic—one could draw arbitrary conclusions from it. If, for example, the ABox contains the assertions Mother(MARY) and Father(MARY), the system should be able to find out that, together with the Family TBox, these statements are inconsistent.

In terms of our model theoretic semantics we can easily give a formal definition of consistency. An ABox $\mathcal{A}$ is *consistent with respect to a TBox* $\mathcal{T}$, if there is an interpretation that is a model of both $\mathcal{A}$ and $\mathcal{T}$. We simply say that $\mathcal{A}$ is *consistent* if it is consistent with respect to the empty TBox.

For example, the set of assertions $\{\mathsf{Mother(MARY)},\ \mathsf{Father(MARY)}\}$ is consistent (with respect to the empty TBox), because without any further restrictions on the interpretation of Mother and Father, the two concepts can be interpreted in such a way that they have a common element. However, the assertions are not consistent with respect to the Family TBox, since in every model of it, Mother and Father are interpreted as disjoint sets.

Similarly as for concepts, checking the consistency of an ABox with respect to an acyclic TBox can be reduced to checking an expanded ABox. We define the *expansion* of $\mathcal{A}$ with respect to $\mathcal{T}$ as the ABox $\mathcal{A}'$ that is obtained from $\mathcal{A}$ by replacing each concept assertion $C(a)$ in $\mathcal{A}$ with the assertion $C'(a)$, where C' is the expansion of C with respect to $\mathcal{T}$.[1] In every model of $\mathcal{T}$, a concept C and its

[1] We expand only concept assertions because the description language considered until now does not provide constructors for role descriptions and therefore we have not considered TBoxes with role definitions.

expansion C' are interpreted in the same way. Therefore, $\mathcal{A}'$ is consistent w.r.t. $\mathcal{T}$ iff $\mathcal{A}$ is so. However, since $\mathcal{A}'$ does not contain a name symbol defined in $\mathcal{T}$, it is consistent w.r.t. $\mathcal{T}$ iff it is consistent. We conclude:

- *$\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ iff its expansion $\mathcal{A}'$ is consistent.*

A technique to check the consistency of $\mathcal{ALCN}$-ABoxes is discussed in Section 2.3.2.

Other inferences that we are going to introduce can also be defined with respect to a TBox or for an ABox alone. As in the case of consistency, reasoning tasks for ABoxes with respect to acyclic TBoxes can be reduced to reasoning on expanded ABoxes. For the sake of simplicity, we shall give only definitions of inferences with ABoxes alone, and leave it to the reader to formulate the appropriate generalization to inferences with respect to TBoxes and to verify that they can be reduced to inferences about expansions, provided the TBox is acyclic.

Over an ABox $\mathcal{A}$, one can pose queries about the relationships between concepts, roles and individuals. The prototypical ABox inference on which such queries are based is the *instance check*, or the check whether an assertion is entailed by an ABox. We say that an assertion α is *entailed* by $\mathcal{A}$ and we write $\mathcal{A} \models \alpha$, if every interpretation that satisfies $\mathcal{A}$, that is, every model of $\mathcal{A}$, also satisfies α. If α is a role assertion, the instance check is easy, since our description language does not contain constructors to form complex roles. If α is of the form $C(a)$, we can reduce the instance check to the consistency problem for ABoxes because there is the following connection:

- *$\mathcal{A} \models C(a)$ iff $\mathcal{A} \cup \{\neg C(a)\}$ is inconsistent.*

Also reasoning about concepts can be reduced to consistency checking. We have seen in Proposition 2.13 that the important reasoning problems for concepts can be reduced to the one to decide whether a concept is (un)satisfiable. Similarly, concept satisfiability can be reduced to ABox consistency because for every concept C we have

- *C is satisfiable iff $\{C(a)\}$ is consistent,*

where a is an arbitrarily chosen individual name. Conversely, Schaerf has shown that ABox consistency can be reduced to concept satisfiability in languages with the "set" and the "fills" constructor [Schaerf, 1994]. If these constructors are not available, however, then instance checking may be harder than the satisfiability and the subsumption problem [Donini *et al.*, 1994].

For applications, usually more complex inferences than consistency and instance

If the description language is richer, and TBoxes contain also role definitions, then they clearly have to be taken into account in the definition of expansions.

checking are required. If we consider a knowledge base as a means to store information about individuals, we may want to know all individuals that are instances of a given concept description C, that is, we use the description language to formulate queries. In our example, we may want to know from the system all parents that have at least two children—for instance, because they are entitled to a specific family tax break. The *retrieval problem* is, given an ABox $\mathcal{A}$ and a concept C, to find all individuals a such that $\mathcal{A} \models C(a)$. A non-optimized algorithm for a retrieval query can be realized by testing for each individual occurring in the ABox whether it is an instance of the query concept C.

The dual inference to retrieval is the *realization problem*: given an individual a and a set of concepts, find the *most specific concepts* C from the set such that $\mathcal{A} \models C(a)$. Here, the most specific concepts are those that are minimal with respect to the subsumption ordering $\sqsubseteq$. Realization can, for instance, be used in systems that generate natural language if terms are indexed by concepts and if a term as precise as possible is to be found for an object occurring in a discourse.

2.2.4.4 Closed- vs. open-world semantics

Often, an analogy is established between databases on the one hand and DL knowledge bases on the other hand (see also Chapter 16). The schema of a database is compared to the TBox and the instance with the actual data is compared to the ABox. However, the semantics of ABoxes differs from the usual semantics of database instances. While a database instance represents exactly one interpretation, namely the one where classes and relations in the schema are interpreted by the objects and tuples in the instance, an ABox represents many different interpretations, namely all its models. As a consequence, absence of information in a database instance is interpreted as negative information, while absence of information in an ABox only indicates lack of knowledge.

For example, if the only assertion about Peter is $\mathsf{hasChild}(\mathsf{PETER}, \mathsf{HARRY})$, then in a database this is understood as a representation of the fact that Peter has only one child, Harry. In an ABox, the assertion only expresses that, in fact, Harry is a child of Peter. However, the ABox has several models, some in which Harry is the only child and others in which he has brothers or sisters. Consequently, even if one also knows (by an assertion) that Harry is male, one cannot deduce that all of Peter's children are male. The only way of stating in an ABox that Harry is the only child is by doing so explicitly, that is by adding the assertion $(\leqslant 1\, \mathsf{hasChild})(\mathsf{PETER})$. This means that, while the information in a database is always understood to be complete, the information in an ABox is in general viewed as being incomplete. The semantics of ABoxes is therefore sometimes characterized as an "open-world" semantics, while the traditional semantics of databases is characterized as a "closed-world" semantics.

hasChild(IOKASTE, OEDIPUS)	hasChild(IOKASTE, POLYNEIKES)
hasChild(OEDIPUS, POLYNEIKES)	hasChild(POLYNEIKES, THERSANDROS)
Patricide(OEDIPUS)	¬Patricide(THERSANDROS)

Fig. 2.5. The Oedipus ABox $\mathcal{A}_{oe}$.

This view has consequences for the way queries are answered. Essentially, a query is a description of a class of objects. In our setting, we assume that queries are concept descriptions. A database (in the sense introduced above) is a listing of a single finite interpretation. A finite interpretation, say $\mathcal{I}$, could be written up as a set of assertions of the form $A(a)$ and $R(b,c)$, where A is an atomic concept and R an atomic role. Such a set looks syntactically like an ABox, but is not an ABox because of the difference in semantics. Answering a query, represented by a complex concept C, over that database amounts to computing $C^{\mathcal{I}}$ as it was defined in Section 2.2.1. From a logical point of view this means that query evaluation in a database is not logical reasoning, but finite model checking (i.e., evaluation of a formula in a fixed finite model).

Since an ABox represents possibly infinitely many interpretations, namely its models, query answering is more complex: it requires nontrivial reasoning. Here we are only concerned with semantical issues (algorithmic aspects will be treated in Section 2.3). To illustrate the difference between a semantics that identifies a database with a single model, and the open-world semantics of ABoxes, we discuss the so-called Oedipus example, which has stimulated a number of theoretical developments in DL research.

Example 2.15 The example is based on the Oedipus story from ancient Greek mythology. In a nutshell, the story recounts how Oedipus killed his father, married his mother Iokaste, and had children with her, among them Polyneikes. Finally, also Polyneikes had children, among them Thersandros.

We suppose the ABox $\mathcal{A}_{oe}$ in Figure 2.5 represents some rudimentary facts about these events. For the sake of the example, our ABox asserts that Oedipus is a patricide and that Thersandros is not, which is represented using the atomic concept Patricide.

Suppose now that we want to know from the ABox whether Iokaste has a child that is a patricide and that itself has a child that is not a patricide. This can be expressed as the entailment problem

$$\mathcal{A}_{oe} \quad \models \quad (\exists \mathsf{hasChild}.(\mathsf{Patricide} \sqcap \exists \mathsf{hasChild}.\neg \mathsf{Patricide}))(\mathsf{IOKASTE})\ ?$$

One may be tempted to reason as follows. Iokaste has two children in the ABox.

One, Oedipus, is a patricide. He has one child, Polyneikes. But nothing tells us that Polyneikes *is not* a patricide. So, Oedipus is not the child we are looking for. The other child is Polyneikes, but again, nothing tells us that Polyneikes *is* a patricide. So, Polyneikes is also not the child we are looking for. Based on this reasoning, one would claim that the assertion about Iokaste is not entailed.

However, the correct reasoning is different. All the models of $\mathcal{A}_{oe}$ can be divided into two classes, one in which Polyneikes is a patricide, and another one in which he is not. In a model of the first kind, Polyneikes is the child of Iokaste that is a patricide and has a child, namely Thersandros, that isn't. In a model of the second kind, Oedipus is the child of Iokaste that is a patricide and has a child, namely Polyneikes, that isn't. Thus, in all models Iokaste has a child that is a patricide and that itself has a child that is not a patricide (though this is not always the same child). This means that the assertion $(\exists \mathsf{hasChild}.(\mathsf{Patricide} \sqcap \exists \mathsf{hasChild}.\neg \mathsf{Patricide}))(\mathsf{IOKASTE})$ is indeed entailed by $\mathcal{A}_{oe}$. ■

As this example shows, open-world reasoning may require to make case analyses. As will be explained in more detail in Chapter 3, this is one of the reasons why inferences in DLs are often more complex than query answering in databases.

2.2.5 Rules

The knowledge bases we have discussed so far consist of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. We denote such a knowledge base as a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$.

In some DL systems, such as CLASSIC [Brachman *et al.*, 1991] or LOOM [MacGregor, 1991], in addition to terminologies and world descriptions, one can also use *rules* to express knowledge. The simplest variant of such rules are expressions of the form

$$C \Rightarrow D,$$

where C, D are concepts. The meaning of such a rule is "if an individual is proved to be an instance of C, then derive that it is also an instance of D." Such rules are often called *trigger rules.*

Operationally, the semantics of a finite set $\mathcal{R}$ of trigger rules can be described by a forward reasoning process. Starting with an initial knowledge base $\mathcal{K}$, a series of knowledge bases $\mathcal{K}^{(0)}$, $\mathcal{K}^{(1)}, \ldots$ is constructed, where $\mathcal{K}^{(0)} = \mathcal{K}$ and $\mathcal{K}^{(i+1)}$ is obtained from $\mathcal{K}^{(i)}$ by adding a new assertion $D(a)$ whenever $\mathcal{R}$ contains a rule $C \Rightarrow D$ such that $\mathcal{K}^{(i)} \models C(a)$ holds, but $\mathcal{K}^{(i)}$ does not contain $D(a)$. This process eventually halts because the initial knowledge base contains only finitely many individuals and there are only finitely many rules. Hence, there are only finitely many assertions $D(a)$ that can possibly be added. The result of the rule applications is a knowledge base $\mathcal{K}^{(n)}$ that has the same TBox as $\mathcal{K}^{(0)}$ and whose ABox is augmented by the membership assertions introduced by the rules. We call this final knowledge base the *procedural extension* of $\mathcal{K}$ and denote it as $\bar{\mathcal{K}}$. It is easy to see that this procedural extension is independent of the order of rule applications. Consequently, a set of trigger rules $\mathcal{R}$ uniquely specifies how to generate, for each knowledge base $\mathcal{K}$, an extended knowledge base $\bar{\mathcal{K}}$. The semantics of a knowledge base $\mathcal{K}$, augmented by a set of trigger rules, can thus be understood as the set of models of $\bar{\mathcal{K}}$.

This defines the semantics of trigger rules only operationally. It would be preferable to specify the semantics declaratively and then to prove that the extension computed with the trigger rules correctly represents this semantics. It might be tempting to use the declarative semantics of inclusion axioms as semantics for rules. However, this does not correctly reflect the operational semantics given above. An important difference between the trigger rule $C \Rightarrow D$ and the inclusion axiom $C \sqsubseteq D$ is that the trigger rule is not equivalent to its contrapositive $\neg D \Rightarrow \neg C$. In addition, when applying trigger rules one does not make a case analysis. For example, the inclusions $C \sqsubseteq D$ and $\neg C \sqsubseteq D$ imply that every object belongs to D, whereas none of the trigger rules $C \Rightarrow D$ and $\neg C \Rightarrow D$ applies to an individual a for which neither $C(a)$ nor $\neg C(a)$ can be proven.

In order to capture the meaning of trigger rules in a declarative way, we must augment description logics by an operator $\mathbf{K}$, which does not refer to objects in the domain, but to what the knowledge base knows about the domain. Therefore, $\mathbf{K}$ is an *epistemic operator*. More information on epistemic operators in DLs can be found in Chapter 6.

To introduce the $\mathbf{K}$-operator, we enrich both the syntax and the semantics of description languages. Originally, the $\mathbf{K}$-operator has been defined for $\mathcal{ALC}$ [Donini *et al.*, 1992; 1998]. In this subsection, we discuss only how to extend the basic language $\mathcal{AL}$. For other languages, one can proceed analogously (see also Chapter 6).

First, we add one case to the syntax rule in Section 2.2.1.1 that allows us to construct epistemic concepts:

$$C, D \quad \longrightarrow \quad \mathbf{K}C \qquad \text{(epistemic concept).}$$

Intuitively, the concept $\mathbf{K}C$ denotes those objects for which the knowledge base knows that they are instances of C.

Next, using $\mathbf{K}$, we translate trigger rules $C \Rightarrow D$ into inclusion axioms

$$\mathbf{K}C \sqsubseteq D. \tag{2.12}$$

Intuitively, the $\mathbf{K}$ operator in front of the concept C has the effect that the axiom is only applicable to individuals that appear in the ABox and for which ABox and TBox imply that they are instances of C. Such a restricted applicability prevents the inclusion axiom from influencing satisfiability or subsumption relationships between concepts. In the sequel, we will define a formal semantics for the operator $\mathbf{K}$ that has exactly this effect.

A *rule knowledge base* is a triple $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{R})$, where $\mathcal{T}$ is a TBox, $\mathcal{A}$ is an ABox, and $\mathcal{R}$ is a set of rules written as inclusion axioms of the form (2.12). The *procedural extension* of such a triple is the knowledge base $\bar{\mathcal{K}} = (\mathcal{T}, \bar{\mathcal{A}})$ that is obtained from $(\mathcal{T}, \mathcal{A})$ by applying the trigger rules as described above.

The semantics of epistemic inclusions will be defined in such a way that it applies only to individuals in the knowledge base that provably are instances of C, but not to arbitrary domain elements, which would be the case if we dropped $\mathbf{K}$. The semantics will go beyond first-order logic because we not only have to interpret concepts, roles and individuals, but also have to model the knowledge of a knowledge base. The fact that a knowledge base has knowledge about the domain can be understood in such a way that it considers only a subset $\mathcal{W}$ of the set of all interpretations as possible states of the world. Those individuals that are interpreted as elements of C under all interpretations in $\mathcal{W}$ are then "known" to be in C.

To make this formal, we modify the definition of ordinary (first-order) interpretations by assuming that:

(i) there is a fixed countably infinite set Δ that is the domain of every interpretation (Common Domain Assumption);
(ii) there is a mapping γ from the individuals to the domain elements that fixes the way individuals are interpreted (Rigid Term Assumption).

The Common Domain Assumption guarantees that all interpretations speak about the same domain. The Rigid Term Assumption allows us to identify each individual symbols with exactly one domain element. These assumptions do not essentially reduce the number of possible interpretations. As a consequence, properties like satisfiability and subsumption of concepts are the same independently of whether we define them with respect to arbitrary interpretations or those that satisfy the above assumptions.

Now, we define an *epistemic interpretation* as a pair $(\mathcal{I}, \mathcal{W})$, where $\mathcal{I}$ is a first-order interpretation and $\mathcal{W}$ is a set of first-order interpretations, all satisfying the above assumptions. Every epistemic interpretation gives rise to a unique mapping $\cdot^{\mathcal{I},\mathcal{W}}$ associating concepts and roles with subsets of Δ and $\Delta \times \Delta$, respectively. For $\top$, $\bot$, for atomic concepts, negated atomic concepts, and for atomic roles, $\cdot^{\mathcal{I},\mathcal{W}}$ agrees with $\cdot^{\mathcal{I}}$. For intersections, value restrictions, and existential quantifications, the definition is similar to the one of $\cdot^{\mathcal{I}}$:

$$
\begin{aligned}
(C \sqcap D)^{\mathcal{I},\mathcal{W}} &= C^{\mathcal{I},\mathcal{W}} \cap D^{\mathcal{I},\mathcal{W}} \\
(\forall R.C)^{\mathcal{I},\mathcal{W}} &= \{a \in \Delta \mid \forall b.\ (a,b) \in R^{\mathcal{I},\mathcal{W}} \to b \in C^{\mathcal{I},\mathcal{W}}\}
\end{aligned}
$$

$$(\exists R.\top)^{\mathcal{I},\mathcal{W}} \quad = \quad \{a \in \Delta \mid \exists b.\ (a,b) \in R^{\mathcal{I},\mathcal{W}}\}.$$

For other constructors, $\cdot^{\mathcal{I},\mathcal{W}}$ can be defined analogously. Note that for a concept C without an occurrence of $\mathbf{K}$, the sets $C^{\mathcal{I},\mathcal{W}}$ and $C^{\mathcal{I}}$ are identical. The set of interpretations $\mathcal{W}$ comes into play when we define the semantics of the epistemic operator:

$$(\mathbf{K}C)^{\mathcal{I},\mathcal{W}} \quad = \quad \bigcap_{\mathcal{J}\in\mathcal{W}} C^{\mathcal{J},\mathcal{W}}.$$

It would also be possible to allow the operator $\mathbf{K}$ to occur in front of roles and to define the semantics of role expressions of the form $\mathbf{K}R$ analogously. However, since epistemic roles are not needed to explain the semantics of rules, we restrict ourselves to epistemic concepts.

An epistemic interpretation $(\mathcal{I},\mathcal{W})$ *satisfies* an inclusion $C \sqsubseteq D$ if $C^{\mathcal{I},\mathcal{W}} \subseteq D^{\mathcal{I},\mathcal{W}}$, and an equality $C \equiv D$ if $C^{\mathcal{I},\mathcal{W}} = D^{\mathcal{I},\mathcal{W}}$. It satisfies an assertion $C(a)$ if $a^{\mathcal{I},\mathcal{W}} = \gamma(a) \in C^{\mathcal{I},\mathcal{W}}$, and an assertion $R(a,b)$ if $(a^{\mathcal{I},\mathcal{W}}, b^{\mathcal{I},\mathcal{W}}) = (\gamma(a),\gamma(b)) \in R^{\mathcal{I},\mathcal{W}}$. It satisfies a rule knowledge base $\mathcal{K} = (\mathcal{T},\mathcal{A},\mathcal{R})$ if it satisfies every axiom in $\mathcal{T}$, every assertion in $\mathcal{A}$, and every rule in $\mathcal{R}$.

An *epistemic model* for a rule knowledge base $\mathcal{K}$ is a *maximal* nonempty set $\mathcal{W}$ of first-order interpretations such that, for each $\mathcal{I} \in \mathcal{W}$, the epistemic interpretation $(\mathcal{I},\mathcal{W})$ satisfies $\mathcal{K}$.

Note that, if $(\mathcal{T},\mathcal{A})$ is first-order satisfiable, then the set of all first-order models of $(\mathcal{T},\mathcal{A})$ is the only epistemic model of the rule knowledge base $\mathcal{K} = (\mathcal{T},\mathcal{A},\emptyset)$, whose rule set is empty. A similar statement holds for arbitrary rule knowledge bases. One can show that, if $\mathcal{W}_1$ and $\mathcal{W}_2$ are epistemic models, then the union $\mathcal{W}_1 \cup \mathcal{W}_2$ is one, too, which implies $\mathcal{W}_1 = \mathcal{W}_2$ because of the maximality of epistemic models.

Proposition 2.16 *Let $\mathcal{K} = (\mathcal{T},\mathcal{A},\mathcal{R})$ be a rule knowledge base such that $(\mathcal{T},\mathcal{A})$ is first-order satisfiable. Then $\mathcal{K}$ has a unique epistemic model.*

Example 2.17 Let $\mathcal{R}$ consist of the rule

$$\mathbf{K}\mathsf{Student} \sqsubseteq \forall \mathsf{eats}.\mathsf{JunkFood}. \tag{2.13}$$

The rule states that "those individuals that are known to be students eat only junk food".

We consider the rule knowledge base $\mathcal{K}_1 = (\emptyset,\mathcal{A}_1,\mathcal{R})$, where

$$\mathcal{A}_1 = \{\mathsf{Student}(\mathsf{PETER})\}.$$

Let us determine the epistemic model $\mathcal{W}$ of $\mathcal{K}_1$. Every first-order interpretation $\mathcal{I} \in \mathcal{W}$ must satisfy $\mathcal{A}_1$. Therefore, in every such $\mathcal{I}$, we have that $\mathsf{Student}(\mathsf{PETER})$

is true, and thus Peter is *known* to be a student. Since $\mathcal{W}$ satisfies Rule (2.13), also the assertion $\forall\mathsf{eats}.\mathsf{JunkFood}(\mathsf{PETER})$ holds in every $\mathcal{I}$.

For any other domain element $a \in \Delta$, there is at least one interpretation in $\mathcal{W}$ where a is not a student. Thus, Peter is the only domain element to which the rule applies. Summing up, the epistemic model of $\mathcal{K}_1$ consists exactly of the first order models of $\mathcal{A}_1 \cup \{\forall\mathsf{eats}.\mathsf{JunkFood}(\mathsf{PETER})\}$.

Next we demonstrate with this example that the epistemic semantics for rules disallows for contrapositive reasoning. We consider the rule knowledge base $\mathcal{K}_2 = (\emptyset, \mathcal{A}_2, \mathcal{R})$, where

$$\mathcal{A}_2 = \{\neg\forall\mathsf{eats}.\mathsf{JunkFood}(\mathsf{PETER})\}.$$

In this case, $\neg\forall\mathsf{eats}.\mathsf{JunkFood}(\mathsf{PETER})$ is true in every first-order interpretation of the epistemic model $\mathcal{W}$. However, because of the maximality of $\mathcal{W}$, there is at least *one* interpretation in $\mathcal{W}$ in which Peter *is* a student and *another one* where Peter *is not* a student. Therefore, Peter is *not known* to be a student. Thus, the epistemic model of $\mathcal{K}_2$ consists exactly of the first order models of $\mathcal{A}_2$. The rule is satisfied because the antecedent is false. ■

Clearly, the procedural extension of a rule knowledge base $\mathcal{K}$ contains only assertions that must be satisfied by the epistemic model of $\mathcal{K}$. It can be shown that the assertions added to $\mathcal{K}$ by the rule applications are in fact, as stated in the following proposition, a first-order representation of the information that is implicit in the rules (see [Donini *et al.*, 1998] for a proof).

Proposition 2.18 *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{R})$ be a rule knowledge base. If $(\mathcal{T}, \mathcal{A})$ is first-order satisfiable, then the epistemic model of $\mathcal{K}$ consists precisely of the first-order models of the procedural extension $\bar{\mathcal{K}} = (\mathcal{T}, \bar{\mathcal{A}})$.*

2.3 Reasoning algorithms

In Section 2.2.4 we have seen that all the relevant inference problems can be reduced to the consistency problem for ABoxes, provided that the DL at hand allows for conjunction and negation. However, the description languages of all the early and also of some of the present day DL systems do not allow for negation. For such DLs, subsumption of concepts can usually be computed by so-called *structural subsumption algorithms*, i.e., algorithms that compare the syntactic structure of (possibly normalized) concept descriptions. In the first subsection, we will consider such algorithms in more detail. While they are usually very efficient, they are only complete for rather simple languages with little expressivity. In particular, DLs with (full) negation and disjunction cannot be handled by structural subsumption algorithms. For such languages, so-called *tableau-based algorithms* have turned out to be very useful. In the area of Description Logics, the first tableau-based algorithm was presented by Schmidt-Schauß and Smolka [1991] for satisfiability of $\mathcal{ALC}$-concepts. Since then, this approach has been employed to obtain sound and complete satisfiability (and thus also subsumption) algorithms for a great variety of DLs extending $\mathcal{ALC}$ (see, e.g., [Hollunder *et al.*, 1990; Hollunder and Baader, 1991; Donini *et al.*, 1997; Baader and Sattler, 1999] for languages with number restrictions; [Baader, 1991] for transitive closure of roles and [Sattler, 1996; Horrocks and Sattler, 1999] for transitive roles; and [Baader and Hanschke, 1991; Hanschke, 1992; Haarslev *et al.*, 1999] for constructors that allow to refer to concrete domains such as numbers). In addition, it has been extended to the consistency problem for ABoxes [Hollunder, 1990; Baader and Hollunder, 1991; Donini *et al.*, 1994; Haarslev and Möller, 2000], and to TBoxes allowing for general sets of inclusion axioms and more [Buchheit *et al.*, 1993; Baader *et al.*, 1996]. In the second subsection, we will first present a tableau-based satisfiability algorithm for $\mathcal{ALCN}$-concepts, then show how it can be extended to an algorithm for the consistency problem for ABoxes, and finally explain how general inclusion axioms can be taken into account. The third subsection is concerned with reasoning w.r.t. acyclic and cyclic terminologies.

Instead of designing new algorithms for reasoning in DLs, one can also try to reduce the problem to a known inference problem in logics (see also Chapter 4). For example, decidability of the inference problems for $\mathcal{ALC}$ and many other DLs can be obtained as a consequence of the known decidability result for the two variable fragment of first-order predicate logic. The language $\mathcal{L}^2$ consists of all formulae of first-order predicate logic that can be built with the help of predicate symbols (including equality) and constant symbols (but without function symbols) using only the variables x, y. Decidability of $\mathcal{L}^2$ has been shown in [Mortimer, 1975]. It is easy to see that, by appropriately re-using variable names, any concept description of the language $\mathcal{ALC}$ can be translated into an $\mathcal{L}^2$-formula with one free variable (see [Borgida, 1996] for details). A direct translation of the concept description $\forall R.(\exists R.A)$ yields the formula $\forall y.(R(x,y) \rightarrow (\exists z.(R(y,z) \wedge A(z))))$. Since the subformula $\exists z.(R(y,z) \wedge A(z))$ does not contain x, this variable can be re-used: renaming the bound variable z into x yields the equivalent formula $\forall y.(R(x,y) \rightarrow (\exists x.(R(y,x) \wedge A(x))))$, which uses only two variables. This connection between $\mathcal{ALC}$ and $\mathcal{L}^2$ shows that any extension of $\mathcal{ALC}$ by constructors that can be expressed with the help of only two variables yields a decidable DL. Number restrictions and composition of roles are examples of constructors that cannot be expressed within $\mathcal{L}^2$. Number restrictions can, however, be expressed in $\mathcal{C}^2$, the extension of $\mathcal{L}^2$ by counting quantifiers, which has recently been shown to be decidable [Grädel *et al.*, 1997; Pacholski *et al.*, 1997]. It should be noted, however, that the complexity of the decision procedures obtained this way is usually higher than necessary: for example, the satisfiability problem for $\mathcal{L}^2$ is NExpTime-complete, whereas satisfiability of $\mathcal{ALC}$-concept descriptions is "only" PSpace-complete.

Decision procedures with lower complexity can be obtained by using the connection between DLs and propositional modal logics. Schild [1991] was the first to observe that the language $\mathcal{ALC}$ is a syntactic variant of the propositional multi-modal logic **K**, and that the extension of $\mathcal{ALC}$ by transitive closure of roles [Baader, 1991] corresponds to Propositional Dynamic Logic (PDL). In particular, some of the algorithms used in propositional modal logics for deciding satisfiability are very similar to the tableau-based algorithms newly developed for DLs. This connection between DLs and modal logics has been used to transfer decidability results from modal logics to DLs [Schild, 1993; 1994; De Giacomo and Lenzerini, 1994a; 1994b] (see also Chapter 5). Instead of using tableau-based algorithms, decidability of certain propositional modal logics (and thus of the corresponding DLs), can also be shown by establishing the finite model property (see, e.g., [Fitting, 1993], Section 1.14) of the logic (i.e., showing that a formula/concept is satisfiable iff it is satisfiable in a finite interpretation) or by employing tree automata (see, e.g, [Vardi and Wolper, 1986]).

2.3.1 Structural subsumption algorithms

These algorithms usually proceed in two phases. First, the descriptions to be tested for subsumption are normalized, and then the syntactic structure of the normal forms is compared. For simplicity, we first explain the ideas underlying this approach for the small language $\mathcal{FL}_0$, which allows for conjunction ($C \sqcap D$) and value restrictions ($\forall R.C$). Subsequently, we show how the bottom concept ($\bot$), atomic negation ($\neg A$), and number restrictions ($\leqslant n\, R$ and $\geqslant n\, R$) can be handled. Evidently, $\mathcal{FL}_0$ and its extension by bottom and atomic negation are sublanguages of $\mathcal{AL}$, while adding number restrictions to the resulting language yields the DL $\mathcal{ALN}$.

An $\mathcal{FL}_0$-concept description is in *normal form* iff it is of the form

$$A_1 \sqcap \cdots \sqcap A_m \sqcap \forall R_1.C_1 \sqcap \cdots \sqcap \forall R_n.C_n,$$

where $A_1, \ldots, A_m$ are distinct concept names, $R_1, \ldots, R_n$ are distinct role names, and $C_1, \ldots, C_n$ are $\mathcal{FL}_0$-concept descriptions in normal form. It is easy to see that any description can be transformed into an equivalent one in normal form, using associativity, commutativity and idempotence of $\sqcap$, and the fact that the descriptions $\forall R.(C \sqcap D)$ and $(\forall R.C) \sqcap (\forall R.D)$ are equivalent.

Proposition 2.19 *Let*

$$A_1 \sqcap \cdots \sqcap A_m \sqcap \forall R_1.C_1 \sqcap \cdots \sqcap \forall R_n.C_n,$$

be the normal form of the $\mathcal{FL}_0$-concept description C, and

$$B_1 \sqcap \cdots \sqcap B_k \sqcap \forall S_1.D_1 \sqcap \cdots \sqcap \forall S_l.D_l,$$

the normal form of the $\mathcal{FL}_0$-concept description D. Then $C \sqsubseteq D$ iff the following two conditions hold:

(i) *for all $i, 1 \leq i \leq k$, there exists $j, 1 \leq j \leq m$ such that $B_i = A_j$.*
(ii) *For all $i, 1 \leq i \leq l$, there exists $j, 1 \leq j \leq n$ such that $S_i = R_j$ and $C_j \sqsubseteq D_i$.*

It is easy to see that this characterization of subsumption is sound (i.e., the "if" direction of the proposition holds) and complete (i.e., the "only-if" direction of the proposition holds as well). This characterization yields an obvious recursive algorithm for computing subsumption, which can easily be shown to be of polynomial time complexity [Levesque and Brachman, 1987].

If we extend $\mathcal{FL}_0$ by language constructors that can express unsatisfiable concepts, then we must, on the one hand, change the definition of the normal form. On the other hand, the structural comparison of the normal forms must take into account that an unsatisfiable concept is subsumed by every concept. The simplest DL where this occurs is $\mathcal{FL}_\perp$, the extension of $\mathcal{FL}_0$ by the bottom concept $\perp$.

An $\mathcal{FL}_\perp$-concept description is in *normal form* iff it is $\perp$ or of the form

$$A_1 \sqcap \cdots \sqcap A_m \sqcap \forall R_1.C_1 \sqcap \cdots \sqcap \forall R_n.C_n,$$

where $A_1, \ldots, A_m$ are distinct concept names different from $\perp$, $R_1, \ldots, R_n$ are distinct role names, and $C_1, \ldots, C_n$ are $\mathcal{FL}_\perp$-concept descriptions in normal form. Again, such a normal form can easily be computed. In principle, one just computes the $\mathcal{FL}_0$-normal form of the description (where $\perp$ is treated as an ordinary concept name): $B_1 \sqcap \cdots \sqcap B_k \sqcap \forall R_1.D_1 \sqcap \cdots \sqcap \forall R_n.D_n$. If one of the B_is is $\perp$, then replace the whole description by $\perp$. Otherwise, apply the same procedure recursively to the D_js. For example, the $\mathcal{FL}_0$-normal form of $\forall R.\forall R.B \sqcap A \sqcap \forall R.(A \sqcap \forall R.\perp)$ is

$$A \sqcap \forall R.(A \sqcap \forall R.(B \sqcap \perp)),$$

which yields the $\mathcal{FL}_\perp$-normal form

$$A \sqcap \forall R.(A \sqcap \forall R.\perp).$$

The structural subsumption algorithm for $\mathcal{FL}_\perp$ works just like the one for $\mathcal{FL}_0$, with the only difference that $\perp$ is subsumed by any description. For example, $\forall R.\forall R.B \sqcap A \sqcap \forall R.(A \sqcap \forall R.\perp) \sqsubseteq \forall R.\forall R.A \sqcap A \sqcap \forall R.A$ since the recursive comparison of their $\mathcal{FL}_\perp$-normal forms $A \sqcap \forall R.(A \sqcap \forall R.\perp)$ and $A \sqcap \forall R.(A \sqcap \forall R.A)$ finally leads to the comparison of $\perp$ and A.

The extension of $\mathcal{FL}_\perp$ by atomic negation (i.e., negation applied to concept names only) can be treated similarly. During the computation of the normal form, negated concept names are just treated like concept names. If, however, a name and its negation occur on the same level of the normal form, then $\bot$ is added, which can then be treated as described above. For example, $\forall R.\neg A \sqcap A \sqcap \forall R.(A \sqcap \forall R.B)$ is first transformed into $A \sqcap \forall R.(A \sqcap \neg A \sqcap \forall R.B)$, then into $A \sqcap \forall R.(\bot \sqcap A \sqcap \neg A \sqcap \forall R.B)$, and finally into $A \sqcap \forall R.\bot$. The structural comparison of the normal forms treats negated concept names just like concept names.

Finally, if we consider the language $\mathcal{ALN}$, the additional presence of number restrictions leads to a new type of conflict. On the one hand, as in the case of atomic negation, number restrictions may be conflicting with each other (e.g., $\geqslant 2\,R$ and $\leqslant 1\,R$). On the other hand, at-least restrictions $\geqslant n\,R$ for $n \geq 1$ are in conflict with value restrictions $\forall R.\bot$ that prohibit role successors. When computing the normal form, one can again treat number restrictions like concept names, and then take care of the new types of conflicts by introducing $\bot$ and using it for normalization as described above. During the structural comparison of normal forms, one must also take into account inherent subsumption relationships between number restrictions (e.g., $\geqslant n\,R \sqsubseteq \geqslant m\,R$ iff $n \geq m$). A more detailed description of a structural subsumption algorithm working on a graph-like data structure for a language extending $\mathcal{ALN}$ can be found in [Borgida and Patel-Schneider, 1994].

For larger DLs, structural subsumption algorithms usually fail to be complete. In particular, they cannot treat disjunction, full negation, and full existential restriction $\exists R.C$. For languages including these constructors, the tableau-approach to designing subsumption algorithms has turned out to be quite useful.

2.3.2 Tableau algorithms

Instead of directly testing subsumption of concept descriptions, these algorithms use negation to reduce subsumption to (un)satisfiability of concept descriptions: as we have seen in Subsection 2.2.4, $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable.

Before describing a tableau-based satisfiability algorithm for $\mathcal{ALCN}$ in more detail, we illustrate the underlying ideas by two simple examples. Let A, B be concept names, and let R be a role name.

As a first example, assume that we want to know whether $(\exists R.A) \sqcap (\exists R.B)$ is subsumed by $\exists R.(A \sqcap B)$. This means that we must check whether the concept description

$$C = (\exists R.A) \sqcap (\exists R.B) \sqcap \neg(\exists R.(A \sqcap B))$$

is unsatisfiable.

First, we push all negation signs as far as possible into the description, using de Morgan's rules and the usual rules for quantifiers. As a result, we obtain the description

$$C_0 = (\exists R.A) \sqcap (\exists R.B) \sqcap \forall R.(\neg A \sqcup \neg B),$$

which is in *negation normal form*, i.e., negation occurs only in front of concept names.

Then, we try to construct a finite interpretation $\mathcal{I}$ such that $C_0^{\mathcal{I}} \neq \emptyset$. This means that there must exist an individual in $\Delta^{\mathcal{I}}$ that is an element of $C_0^{\mathcal{I}}$.

The algorithm just generates such an individual, say b, and imposes the constraint $b \in C_0^{\mathcal{I}}$ on it. Since C_0 is the conjunction of three concept descriptions, this means that b must satisfy the following three constraints: $b \in (\exists R.A)^{\mathcal{I}}$, $b \in (\exists R.B)^{\mathcal{I}}$, and $b \in (\forall R.(\neg A \sqcup \neg B))^{\mathcal{I}}$.

From $b \in (\exists R.A)^{\mathcal{I}}$ we can deduce that there must exist an individual c such that $(b, c) \in R^{\mathcal{I}}$ and $c \in A^{\mathcal{I}}$. Analogously, $b \in (\exists R.B)^{\mathcal{I}}$ implies the existence of an individual d with $(b, d) \in R^{\mathcal{I}}$ and $d \in B^{\mathcal{I}}$. In this situation, one should not assume that $c = d$ since this would possibly impose too many constraints on the individuals newly introduced to satisfy the existential restrictions on b. Thus:

- *For any existential restriction the algorithm introduces a new individual as role filler, and this individual must satisfy the constraints expressed by the restriction.*

Since b must also satisfy the value restriction $\forall R.(\neg A \sqcup \neg B)$, and c, d were introduced as R-fillers of b, we obtain the additional constraints $c \in (\neg A \sqcup \neg B)^{\mathcal{I}}$ and $d \in (\neg A \sqcup \neg B)^{\mathcal{I}}$. Thus:

- *The algorithm uses value restrictions in interaction with already defined role relationships to impose new constraints on individuals.*

Now $c \in (\neg A \sqcup \neg B)^{\mathcal{I}}$ means that $c \in (\neg A)^{\mathcal{I}}$ or $c \in (\neg B)^{\mathcal{I}}$, and we must choose one of these possibilities. If we assume $c \in (\neg A)^{\mathcal{I}}$, this clashes with the other constraint $c \in A^{\mathcal{I}}$, which means that this search path leads to an obvious contradiction. Thus we must choose $c \in (\neg B)^{\mathcal{I}}$. Analogously, we must choose $d \in (\neg A)^{\mathcal{I}}$ in order to satisfy the constraint $d \in (\neg A \sqcup \neg B)^{\mathcal{I}}$ without creating a contradiction to $d \in B^{\mathcal{I}}$. Thus:

- *For disjunctive constraints, the algorithm tries both possibilities in successive attempts. It must backtrack if it reaches an obvious contradiction, i.e., if the same individual must satisfy constraints that are obviously conflicting.*

In the example, we have now satisfied all the constraints without encountering an obvious contradiction. This shows that C_0 is satisfiable, and thus $(\exists R.A) \sqcap (\exists R.B)$ is not subsumed by $\exists R.(A \sqcap B)$. The algorithm has generated an interpretation $\mathcal{I}$ as witness for this fact: $\Delta^{\mathcal{I}} = \{b, c, d\}$; $R^{\mathcal{I}} = \{(b, c), (b, d)\}$; $A^{\mathcal{I}} = \{c\}$ and $B^{\mathcal{I}} = \{d\}$.

For this interpretation, $b \in C_0^{\mathcal{I}}$. This means that $b \in ((\exists R.A) \sqcap (\exists R.B))^{\mathcal{I}}$, but $b \notin (\exists R.(A \sqcap B))^{\mathcal{I}}$.

In our second example, we add a number restriction to the first concept of the above example, i.e., we now want to know whether $(\exists R.A) \sqcap (\exists R.B) \sqcap \leqslant 1\, R$ is subsumed by $\exists R.(A \sqcap B)$. Intuitively, the answer should now be "yes" since $\leqslant 1\, R$ in the first concept ensures that the R-filler in A coincides with the R-filler in B, and thus there is an R-filler in $A \sqcap B$. The tableau-based satisfiability algorithm first proceeds as above, with the only difference that there is the additional constraint $b \in (\leqslant 1\, R)^{\mathcal{I}}$. In order to satisfy this constraint, the two R-fillers c, d of b must be identified with each other. Thus:

- *If an at-most number restriction is violated then the algorithm must identify different role fillers.*

In the example, the individual $c = d$ must belong to both $A^{\mathcal{I}}$ and $B^{\mathcal{I}}$, which together with $c = d \in (\neg A \sqcup \neg B)^{\mathcal{I}}$ always leads to a clash. Thus, the search for a counterexample to the subsumption relationship fails, and the algorithm concludes that $(\exists R.A) \sqcap (\exists R.B) \sqcap \leqslant 1\, R \sqsubseteq \exists R.(A \sqcap B)$.

2.3.2.1 A tableau-based satisfiability algorithm for $\mathcal{ALCN}$

Before we can describe the algorithm more formally, we need to introduce an appropriate data structure in which to represent constraints like "a belongs to (the interpretation of) C" and "b is an R-filler of a." The original paper by Schmidt-Schauß and Smolka [1991], and also many other papers on tableau algorithms for DLs, introduce the new notion of a constraint system for this purpose. However, if we look at the types of constraints that must be expressed, we see that they can actually be represented by ABox assertions. As we have seen in the second example above, the presence of at-most number restrictions may lead to the identification of different individual names. For this reason, we will not impose the unique name assumption (UNA) on the ABoxes considered by the algorithm. Instead, we allow for explicit *inequality assertions* of the form $x \neq y$ for individual names x, y, with the obvious semantics that an interpretation $\mathcal{I}$ satisfies $x \neq y$ iff $x^{\mathcal{I}} \neq y^{\mathcal{I}}$. These assertions are assumed to be symmetric, i.e., saying that $x \neq y$ belongs to an ABox $\mathcal{A}$ is the same as saying that $y \neq x$ belongs to $\mathcal{A}$.

Let C_0 by an $\mathcal{ALCN}$-concept in negation normal form. In order to test satisfiability of C_0, the algorithm starts with the ABox $\mathcal{A}_0 = \{C_0(x_0)\}$, and applies consistency preserving transformation rules (see Figure 2.6) to the ABox until no more rules apply. If the "complete" ABox obtained this way does not contain an obvious contradiction (called clash), then $\mathcal{A}_0$ is consistent (and thus C_0 is satisfiable), and inconsistent (unsatisfiable) otherwise. The transformation rules that handle disjunction and at-most restrictions are *non-deterministic* in the sense that a given

The $\rightarrow_\sqcap$-rule
Condition: $\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but it does not contain both $C_1(x)$ and $C_2(x)$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C_1(x), C_2(x)\}$.

The $\rightarrow_\sqcup$-rule
Condition: $\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}'' = \mathcal{A} \cup \{C_2(x)\}$.

The $\rightarrow_\exists$-rule
Condition: $\mathcal{A}$ contains $(\exists R.C)(x)$, but there is no individual name z such that $C(z)$ and $R(x,z)$ are in $\mathcal{A}$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C(y), R(x,y)\}$ where y is an individual name not occurring in $\mathcal{A}$.

The $\rightarrow_\forall$-rule
Condition: $\mathcal{A}$ contains $(\forall R.C)(x)$ and $R(x,y)$, but it does not contain $C(y)$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$.

The $\rightarrow_\geq$-rule
Condition: $\mathcal{A}$ contains $(\geqslant n\, R)(x)$, and there are no individual names $z_1, \dots, z_n$ such that $R(x, z_i)$ $(1 \leq i \leq n)$ and $z_i \neq z_j$ $(1 \leq i < j \leq n)$ are contained in $\mathcal{A}$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{R(x, y_i) \mid 1 \leq i \leq n\} \cup \{y_i \neq y_j \mid 1 \leq i < j \leq n\}$, where $y_1, \dots, y_n$ are distinct individual names not occurring in $\mathcal{A}$.

The $\rightarrow_\leq$-rule
Condition: $\mathcal{A}$ contains distinct individual names $y_1, \dots, y_{n+1}$ such that $(\leqslant n\, R)(x)$ and $R(x, y_1), \dots, R(x, y_{n+1})$ are in $\mathcal{A}$, and $y_i \neq y_j$ is not in $\mathcal{A}$ for some $i \neq j$.
Action: For each pair y_i, y_j such that $i > j$ and $y_i \neq y_j$ is not in $\mathcal{A}$, the ABox $\mathcal{A}_{i,j} = [y_i/y_j]\mathcal{A}$ is obtained from $\mathcal{A}$ by replacing each occurrence of y_i by y_j.

Fig. 2.6. Transformation rules of the satisfiability algorithm.

ABox is transformed into finitely many new ABoxes such that the original ABox is consistent iff *one of* the new ABoxes is so. For this reason we will consider finite sets of ABoxes $\mathcal{S} = \{\mathcal{A}_1, \dots, \mathcal{A}_k\}$ instead of single ABoxes. Such a set is *consistent* iff there is some i, $1 \leq i \leq k$, such that $\mathcal{A}_i$ is consistent. A rule of Figure 2.6 is applied to a given finite set of ABoxes $\mathcal{S}$ as follows: it takes an element $\mathcal{A}$ of $\mathcal{S}$, and replaces it by one ABox $\mathcal{A}'$, by two ABoxes $\mathcal{A}'$ and $\mathcal{A}''$, or by finitely many ABoxes $\mathcal{A}_{i,j}$.

The following lemma is an easy consequence of the definition of the transformation rules:

Lemma 2.20 (Soundness) *Assume that $\mathcal{S}'$ is obtained from the finite set of ABoxes $\mathcal{S}$ by application of a transformation rule. Then $\mathcal{S}$ is consistent iff $\mathcal{S}'$ is consistent.*

The second important property of the set of transformation rules is that the transformation process always terminates:

Lemma 2.21 (Termination) *Let C_0 be an $\mathcal{ALCN}$-concept description in negation normal form. There cannot be an infinite sequence of rule applications*

$$\{\{C_0(x_0)\}\} \to \mathcal{S}_1 \to \mathcal{S}_2 \to \cdots.$$

The main reasons for this lemma to hold are the following.[1]

Lemma 2.22 *Let $\mathcal{A}$ be an ABox contained in $\mathcal{S}_i$ for some $i \geq 1$.*

- *For every individual $x \neq x_0$ occurring in $\mathcal{A}$, there is a unique sequence $R_1, \ldots, R_\ell$ ($\ell \geq 1$) of role names and a unique sequence $x_1, \ldots, x_{\ell-1}$ of individual names such that $\{R_1(x_0, x_1), R_2(x_1, x_2), \ldots, R_\ell(x_{\ell-1}, x)\} \subseteq \mathcal{A}$. In this case, we say that x occurs on level ℓ in $\mathcal{A}$.*
- *If $C(x) \in \mathcal{A}$ for an individual name x on level ℓ, then the maximal role depth of C (i.e., the maximal nesting of constructors involving roles) is bounded by the maximal role depth of C_0 minus ℓ. Consequently, the level of any individual in $\mathcal{A}$ is bounded by the maximal role depth of C_0.*
- *If $C(x) \in \mathcal{A}$, then C is a subdescription of C_0. Consequently, the number of different concept assertions on x is bounded by the size of C_0.*
- *The number of different role successors of x in $\mathcal{A}$ (i.e., individuals y such that $R(x, y) \in \mathcal{A}$ for a role name R) is bounded by the sum of the numbers occurring in at-least restrictions in C_0 plus the number of different existential restrictions in C_0.*

Starting with $\{\{C_0(x_0)\}\}$, we thus obtain after a finite number of rule applications a set of ABoxes $\widehat{\mathcal{S}}$ to which no more rules apply. An ABox $\mathcal{A}$ is called *complete* iff none of the transformation rules applies to it. Consistency of a set of complete ABoxes can be decided by looking for obvious contradictions, called clashes. The ABox $\mathcal{A}$ contains a *clash* iff one of the following three situations occurs:

(i) $\{\bot(x)\} \subseteq \mathcal{A}$ for some individual name x;
(ii) $\{A(x), \neg A(x)\} \subseteq \mathcal{A}$ for some individual name x and some concept name A;
(iii) $\{(\leqslant n\, R)(x)\} \cup \{R(x, y_i) \mid 1 \leq i \leq n+1\} \cup \{y_i \neq y_j \mid 1 \leq i < j \leq n+1\} \subseteq \mathcal{A}$ for individual names $x, y_1, \ldots, y_{n+1}$, a nonnegative integer n, and a role name R.

Obviously, an ABox that contains a clash cannot be consistent. Hence, if all the ABoxes in $\widehat{\mathcal{S}}$ contain a clash, then $\widehat{\mathcal{S}}$ is inconsistent, and thus by the soundness lemma $\{C_0(x_0)\}$ is inconsistent as well. Consequently, C_0 is unsatisfiable. If, however, one of the complete ABoxes in $\widehat{\mathcal{S}}$ is clash-free, then $\widehat{\mathcal{S}}$ is consistent. By soundness of the rules, this implies consistency of $\{C_0(x_0)\}$, and thus satisfiability of C_0.

[1] A detailed proof of termination for a set of rules extending the one of Figure 2.6 can be found in [Baader and Sattler, 1999]. A termination proof for a slightly different set of rules has been given in [Donini *et al.*, 1997].

Lemma 2.23 (Completeness) *Any complete and clash-free ABox $\mathcal{A}$ has a model.*

This lemma can be proved by defining the *canonical interpretation* $\mathcal{I}_\mathcal{A}$ induced by $\mathcal{A}$:

(i) the domain $\Delta^{\mathcal{I}_\mathcal{A}}$ of $\mathcal{I}_\mathcal{A}$ consists of all the individual names occurring in $\mathcal{A}$;
(ii) for all atomic concepts A we define $A^{\mathcal{I}_\mathcal{A}} = \{x \mid A(x) \in \mathcal{A}\}$;
(iii) for all atomic roles R we define $R^{\mathcal{I}_\mathcal{A}} = \{(x, y) \mid R(x, y) \in \mathcal{A}\}$.

By definition, $\mathcal{I}_\mathcal{A}$ satisfies all the role assertions in $\mathcal{A}$. By induction on the structure of concept descriptions, it is easy to show that it satisfies the concept assertions as well. The inequality assertions are satisfied since $x \neq y \in \mathcal{A}$ only if x, y are different individual names.

The facts stated in Lemma 2.22 imply that the canonical interpretation has the shape of a finite tree whose depth is linearly bounded by the size of C_0 and whose branching factor is bounded by the sum of the numbers occurring in at-least restrictions in C_0 plus the number of different existential restrictions in C_0. Consequently, $\mathcal{ALCN}$ has the *finite tree model property*, i.e., any satisfiable concept C_0 is satisfiable in a finite interpretation $\mathcal{I}$ that has the shape of a tree whose root belongs to C_0.

To sum up, we have seen that the transformation rules of Figure 2.6 reduce satisfiability of an $\mathcal{ALCN}$-concept C_0 (in negation normal form) to consistency of a finite set $\widehat{\mathcal{S}}$ of complete ABoxes. In addition, consistency of $\widehat{\mathcal{S}}$ can be decided by looking for obvious contradictions (clashes).

Theorem 2.24 *It is decidable whether or not an $\mathcal{ALCN}$-concept is satisfiable.*

2.3.2.2 Complexity issues

The tableau-based satisfiability algorithm for $\mathcal{ALCN}$ presented above may need exponential time and space. In fact, the size of the canonical interpretation built by the algorithm may be exponential in the size of the concept description. For example, consider the descriptions C_n ($n \geq 1$), which are inductively defined as follows:

$$\begin{array}{rcl} C_1 & = & \exists R.A \sqcap \exists R.B, \\ C_{n+1} & = & \exists R.A \sqcap \exists R.B \sqcap \forall R.C_n. \end{array}$$

Obviously, the size of C_n grows linearly in n. However, given the input description C_n, the satisfiability algorithm introduced above generates a complete and clash-free ABox whose canonical model is the full binary tree of depth n, and thus consists of $2^{n+1} - 1$ individuals.

Nevertheless, the satisfiability algorithm can be modified such that it needs only polynomial space. The main reason is that different branches of the tree model to be generated by the algorithm can be investigated separately. Since the complexity class NPSPACE coincides with PSPACE [Savitch, 1970], it is sufficient to describe a non-deterministic algorithm using only polynomial space, i.e., for every non-deterministic rule we may simply assume that the algorithm chooses the correct alternative. In principle, the modified algorithm works as follows: it starts with $\{C_0(x_0)\}$ and

(i) applies the $\rightarrow_{\sqcap}$- and $\rightarrow_{\sqcup}$-rules as long as possible, and checks for clashes of the form $A(x_0), \neg A(x_0)$ and $\bot(x_0)$;
(ii) generates all the necessary direct successors of x_0 using the $\rightarrow_{\exists}$- and the $\rightarrow_{\geq}$-rule;
(iii) generates the necessary identifications of these direct successors using the $\rightarrow_{\leq}$-rule, and checks for clashes caused by at-most restrictions;
(iv) successively handles the successors in the same way.

Since after identification the remaining successors can be treated separately, the algorithm needs to store only one path of the tree model to be generated, together with the *direct* successors of the individuals on this path and the information which of these successors must be investigated next. We already know that the length of the path is linear in the size of the input description C_0. Thus, the only remaining obstacle on our way to a PSPACE-algorithm is the fact that the number of direct successors of an individual on the path also depends on the numbers in the at-least restrictions. If we assumed these numbers to be written in base 1 representation (where the size of the representation coincides with the number represented), this would not be a problem. However, for bases larger than 1 (e.g., numbers in decimal notation), the number represented may be exponential in the size of the representation. For example, the representation of $10^n - 1$ requires only n digits in base 10 representation. Thus, we cannot introduce all the successors required by at-least restrictions while only using polynomial space in the size of the concept description if the numbers in this description are written in decimal notation.

It turns out, however, that most of the successors required by the at-least restrictions need not be introduced at all. If an individual x obtains at least one R-successor due to the application of the $\rightarrow_{\exists}$-rule, then the $\rightarrow_{\geq}$-rule need not be applied to x for the role R. Otherwise, we simply introduce *one* R-successor as representative. In order to detect inconsistencies due to conflicting number restrictions, we need to add a *new type of clash*: $\{(\leqslant n\, R)(x), (\geqslant m\, R)(x)\} \subseteq \mathcal{A}$ for nonnegative integers $n < m$. The canonical interpretation obtained by this modified algorithm need not satisfy the at-least restrictions in C_0. However, it can easily by modified to an interpretation that does, by duplicating R-successors (more precisely, the whole subtrees starting at these successors).

Theorem 2.25 *Satisfiability of $\mathcal{ALCN}$-concept descriptions is* PSPACE*-complete.*

The above argument shows that the problem is in PSPACE. The hardness result follows from the fact that the satisfiability problem is already PSPACE-hard for the sublanguage $\mathcal{ALC}$, which can be shown by a reduction from validity of Quantified Boolean Formulae [Schmidt-Schauß and Smolka, 1991]. Since subsumption and satisfiability of $\mathcal{ALCN}$-concept descriptions can be reduced to each other in linear time, this also shows that subsumption of $\mathcal{ALCN}$-concept descriptions is PSPACE-complete.

2.3.2.3 Extension to the consistency problem for ABoxes

The tableau-based satisfiability algorithm described in Subsection 2.3.2.1 can easily be extended to an algorithm that decides consistency of $\mathcal{ALCN}$-ABoxes. Let $\mathcal{A}$ be an $\mathcal{ALCN}$-ABox such that (w.o.l.g.) all concept descriptions in $\mathcal{A}$ are in negation normal form. To test $\mathcal{A}$ for consistency, we first add inequality assertions $a \neq b$ for every pair of distinct individual names a, b occurring in $\mathcal{A}$.[1] Let $\mathcal{A}_0$ be the ABox obtained this way. The consistency algorithm applies the rules of Figure 2.6 to the singleton set $\{\mathcal{A}_0\}$.

Soundness and completeness of the rule set can be shown as before. Unfortunately, the algorithm need not terminate, unless one imposes a specific strategy on the order of rule applications. For example, consider the ABox

$$\mathcal{A}_0 = \{R(a,a),\ (\exists R.A)(a),\ (\leqslant 1\, R)(a),\ (\forall R.\exists R.A)(a)\}.$$

By applying the $\rightarrow_\exists$-rule to a, we can introduce a new R-successor x of a:

$$\mathcal{A}_1 = \mathcal{A}_0 \cup \{R(a,x),\ A(x)\}.$$

The $\rightarrow_\forall$-rule adds the assertion $(\exists R.A)(x)$, which triggers an application of the $\rightarrow_\exists$-rule to x. Thus, we obtain the new ABox

$$\mathcal{A}_2 = \mathcal{A}_1 \cup \{(\exists R.A)(x),\ R(x,y),\ A(y)\}.$$

Since a has two R-successors in $\mathcal{A}_2$, the $\rightarrow_\leq$-rule is applicable to a. By replacing every occurrence of x by a, we obtain the ABox

$$\mathcal{A}_3 = \mathcal{A}_0 \cup \{A(a),\ R(a,y),\ A(y)\}.$$

Except for the individual names (and the assertion $A(a)$, which is, however, irrelevant), $\mathcal{A}_3$ is identical to $\mathcal{A}_1$. For this reason, we can continue as above to obtain an infinite chain of rule applications.

We can easily regain termination by requiring that generating rules (i.e., the rules $\rightarrow_\exists$ and $\rightarrow_\geq$) may only be applied if none of the other rules is applicable. In the

[1] This takes care of the UNA.

above example, this strategy would prevent the application of the $\rightarrow_{\exists}$-rule to x in the ABox $\mathcal{A}_1 \cup \{(\exists R.A)(x)\}$ since the $\rightarrow_{\leq}$-rule is also applicable. After applying the $\rightarrow_{\leq}$-rule (which replaces x by a), the $\rightarrow_{\exists}$-rule is no longer applicable since a already has an R-successor that belongs to A.

Using a similar idea, one can reduce the consistency problem for $\mathcal{ALCN}$-ABoxes to satisfiability of $\mathcal{ALCN}$-concept descriptions [Hollunder, 1996]. In principle, this reduction works as follows: In a preprocessing step, one applies the transformation rules only to old individuals (i.e., individuals present in the original ABox). Subsequently, one can forget about the role assertions, i.e., for each individual name in the preprocessed ABox, the satisfiability algorithm is applied to the conjunction of its concept assertions (see [Hollunder, 1996] for details).

Theorem 2.26 *Consistency of $\mathcal{ALCN}$-ABoxes is* PSPACE*-complete.*

2.3.2.4 Extension to general inclusion axioms

In the above subsections, we have considered the satisfiability problem for concept descriptions and the consistency problem for ABoxes without an underlying TBox. In fact, for acyclic TBoxes one can simply expand the definitions (see Subsection 2.2.4). Expansion is, however, no longer possibly if one allows for general inclusion axioms of the form $C \sqsubseteq D$, where C and D may be complex descriptions. Instead of considering finitely many such axiom $C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n$, it is sufficient to consider the single axiom $\top \sqsubseteq \widehat{C}$, where

$$\widehat{C} = (\neg C_1 \sqcup D_1) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n).$$

The axiom $\top \sqsubseteq \widehat{C}$ simply says that any individual must belong to the concept $\widehat{C}$. The tableau algorithm introduced above can easily be modified such that it takes this axiom into account: all individuals (both the original individuals and the ones newly generated by the $\rightarrow_{\exists}$- and the $\rightarrow_{\geq}$-rule) are simply asserted to belong to $\widehat{C}$. However, this modification may obviously lead to nontermination of the algorithm. For example, consider what happens if this algorithm is applied to test consistency of the ABox $\mathcal{A}_0 = \{A(x_0), (\exists R.A)(x_0)\}$ w.r.t. the axiom $\top \sqsubseteq \exists R.A$: the algorithm generates an infinite sequence of ABoxes $\mathcal{A}_1, \mathcal{A}_2, \ldots$ and individuals $x_1, x_2, \ldots$ such that $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{R(x_i, x_{i+1}), A(x_{i+1}), (\exists R.A)(x_{i+1})\}$. Since all individuals x_i receive the same concept assertions as x_0, we may say that the algorithms has run into a cycle.

Termination can be regained by trying to detect such cyclic computations, and then blocking the application of generating rules: the application of the rules $\rightarrow_{\exists}$ and $\rightarrow_{\geq}$ to an individual x is *blocked* by an individual y in an ABox $\mathcal{A}$ iff $\{D \mid D(x) \in \mathcal{A}\} \subseteq \{D' \mid D'(y) \in \mathcal{A}\}$. The main idea underlying blocking is that the blocked individual x can use the role successors of y instead of generating new ones.

For example, instead of generating a new R-successor for x_1 in the above example, one can simply use the R-successor of x_0. This yields an interpretation $\mathcal{I}$ with $\Delta^{\mathcal{I}} = \{x_0, x_1\}$, $A^{\mathcal{I}} = \Delta^{\mathcal{I}}$, and $R^{\mathcal{I}} = \{(x_0, x_1), (x_1, x_1)\}$. Obviously, $\mathcal{I}$ is a model of $\mathcal{A}_0$ and of the axiom $\top \sqsubseteq \exists R.A$.

To avoid cyclic blocking (of x by y and vice versa), we consider an enumeration of all individual names, and define that an individual x may only be blocked by individuals y that occur before x in this enumeration. This, together with some other technical assumptions, makes sure that an algorithm using this notion of blocking is sound and complete as well as terminating (see [Buchheit *et al.*, 1993; Baader *et al.*, 1996] for details). Thus, consistency of $\mathcal{ALCN}$-ABoxes w.r.t. general inclusion axioms is decidable. It should be noted that the algorithm is no longer in PSPACE since it may generate role paths of exponential length before blocking occurs. In fact, even for the language $\mathcal{ALC}$, satisfiability w.r.t. a single general inclusion axiom is known to be EXPTIME-hard [Schild, 1994] (see also Chapter 3). The tableau-based algorithm sketched above is a NEXPTIME algorithm. However, using the translation technique mentioned at the beginning of this section, it can be shown [De Giacomo, 1995] that $\mathcal{ALCN}$-ABoxes and general inclusion axioms can be translated into PDL, for which satisfiability can be decided in exponential time. An EXPTIME tableau algorithm for $\mathcal{ALC}$ with general inclusion axiom was described by Donini and Massacci [2000].

Theorem 2.27 *Consistency of $\mathcal{ALCN}$-ABoxes w.r.t. general inclusion axioms is* EXPTIME-*complete.*

2.3.2.5 Extension to other language constructors

The tableau-based approach to designing concept satisfiability and ABox consistency algorithms can also be employed for languages with other concept and/or role constructors. In principle, each new constructor requires a new rule, and this rule can usually be obtained by simply considering the semantics of the constructor. Soundness of such a rule is often very easy to show. More problematic are completeness and termination since they must also take interactions between different rules into account. As we have seen above, termination can sometimes only be obtained if the application of rules is restricted by an appropriate strategy. Of course, one may only impose such a strategy if one can show that it does not destroy completeness.

2.3.3 Reasoning w.r.t. terminologies

Recall that terminologies (TBoxes) are sets of concept definitions (i.e., equalities of the form $A \equiv C$ where A is atomic) such that every atomic concept occurs at most once as a left-hand side. We will first comment briefly on the complexity of reasoning w.r.t. acyclic terminologies, and then consider in more detail reasoning w.r.t. cyclic terminologies.

2.3.3.1 Acyclic terminologies

As shown in Section 2.2.4, reasoning w.r.t. *acyclic* terminologies can be reduced to reasoning without terminologies by first expanding the TBox, and then replacing name symbols by their definitions in the terminology. Unfortunately, since the expanded TBox may be exponentially larger than the original one [Nebel, 1990], this increases the complexity of reasoning. Nebel [1990] also shows that this complexity can, in general, not be avoided: for the language $\mathcal{FL}_0$, subsumption between concept descriptions can be tested in polynomial time (see Section 2.3.1), whereas subsumption w.r.t. acyclic terminologies is coNP-complete (see also Section 2.3.3.2 below).

For more expressive languages, the presence of acyclic TBoxes may or may not increase the complexity of the subsumption problem. For example, subsumption of concept descriptions in the language $\mathcal{ALC}$ is PSPACE-complete, and so is subsumption w.r.t. acyclic terminologies [Lutz, 1999]. Of course, in order to obtain a PSPACE-algorithm for subsumption in $\mathcal{ALC}$ w.r.t. acyclic TBoxes, one cannot first expand the TBox completely since this might need exponential space. The main idea is that one uses a tableau-based algorithm like the one described in Section 2.3.2, with the difference that it receives concept descriptions containing name symbols as input. Expansion is then done on demand: if the tableau-based algorithm encounters an assertion of the form $A(x)$, where A is a name occurring on the left-hand side of a definition $A \equiv C$ in the TBox, then it adds the assertion $C(x)$. However, it does not further expand C at this stage. It is not hard to show that this really yields a PSPACE-algorithm for satisfiability (and thus also for subsumption) of concepts w.r.t. acyclic TBoxes in $\mathcal{ALC}$ [Lutz, 1999].

There are, however, extensions of $\mathcal{ALC}$ for which this technique no longer works. One such example is the language $\mathcal{ALCF}$, which extends $\mathcal{ALC}$ by functional roles as well as agreements and disagreements on chains of functional roles (see Section 2.4 below). Satisfiability of concepts is PSPACE-complete for this language [Hollunder and Nutt, 1990], but satisfiability of concepts w.r.t. acyclic terminologies is NEXPTIME-complete [Lutz, 1999].

2.3.3.2 Cyclic terminologies

For cyclic terminologies, expansion is no longer possible since it would not terminate. If we use descriptive semantics, then cyclic terminologies are a special case of terminologies with general inclusion axioms. Thus, the tableau-based algorithm for handling general inclusion axioms introduced in Subsection 2.3.2.4 can also be used for cyclic $\mathcal{ALCN}$-TBoxes with descriptive semantics. For cyclic $\mathcal{ALC}$-

TBoxes with fixpoint semantics, the connection between Description Logics and propositional modal logics turns out to be useful. In fact, syntactically monotone $\mathcal{ALC}$-TBoxes with least or greatest fixpoint semantics can be expressed within the propositional μ-calculus, which is an extension of the propositional multimodal logic $\mathbf{K_m}$ by fixpoint operators (see [Schild, 1994; De Giacomo and Lenzerini, 1994b; 1997] and Chapter 5 for details). Since reasoning w.r.t. general inclusion axioms in $\mathcal{ALC}$ and reasoning in the propositional μ-calculus are both ExpTime-complete, these reductions yield an ExpTime-upper bound for reasoning w.r.t. cyclic terminologies in sublanguages of $\mathcal{ALC}$.

For less expressive DLs, more efficient algorithms can, however, be obtained with the help of techniques based on finite automata. Following [Baader, 1996], we will sketch these techniques for the small language $\mathcal{FL}_0$. The results can, however, be extended to the language $\mathcal{ALN}$ [Küsters, 1998]. We will develop the results for $\mathcal{FL}_0$ in two steps, starting with an alternative characterization of subsumption between $\mathcal{FL}_0$-concept descriptions, and then extending this characterization to cyclic TBoxes with greatest fixpoint semantics. Baader [1996] also considers cyclic $\mathcal{FL}_0$-TBoxes with descriptive and with least fixpoint semantics. For these semantics, the characterization of subsumption is more involved; in particular, the characterization of subsumption w.r.t. descriptive semantics depends on finite automata working on infinite words, so-called Büchi automata. Acyclic TBoxes can be seen as a special case of cyclic TBoxes, where all three types of semantics coincide.

In Subsection 2.3.1, the equivalence $(\forall R.C) \sqcap (\forall R.D) \equiv \forall R.(C \sqcap D)$ was used as a rewrite rule from left to right in order to compute the *structural subsumption normal form* of $\mathcal{FL}_0$-concept descriptions. If we use this rule in the opposite direction, we obtain a different normal form, which we call *concept-centered normal form* since it groups the concept description w.r.t. concept names (and not w.r.t. role names, as the structural subsumption normal form does). Using this rule, any $\mathcal{FL}_0$-concept description can be transformed into an equivalent description that is a conjunction of descriptions of the form $\forall R_1. \cdots \forall R_m.A$ for $m \geq 0$ (not necessarily distinct) role names $R_1, \ldots, R_m$ and a concept name A. We abbreviate $\forall R_1. \cdots \forall R_m.A$ by $\forall R_1 \cdots R_m.A$, where $R_1 \cdots R_m$ is viewed as a word over the alphabet Σ of all role names. In addition, instead of $\forall w_1.A \sqcap \cdots \sqcap \forall w_\ell.A$ we write $\forall L.A$ where $L = \{w_1, \ldots, w_\ell\}$ is a finite set of words over Σ. The term $\forall \emptyset.A$ is considered to be equivalent to the top concept $\top$, which means that it can be added to a conjunction without changing the meaning of the concept. Using these abbreviations, any pair of $\mathcal{FL}_0$-concept descriptions C, D containing the concept names $A_1, \ldots, A_k$ can be rewritten as

$$C \equiv \forall U_1.A_1 \sqcap \cdots \sqcap \forall U_k.A_k \quad \text{and} \quad D \equiv \forall V_1.A_1 \sqcap \cdots \sqcap \forall V_k.A_k,$$

where U_i, V_i are finite sets of words over the alphabet of all role names. This normal

$$
\begin{aligned}
A &\equiv \forall R.A \sqcap \forall S.C \\
B &\equiv \forall R.\forall S.C \\
C &\equiv P \sqcap \forall S.C
\end{aligned}
$$

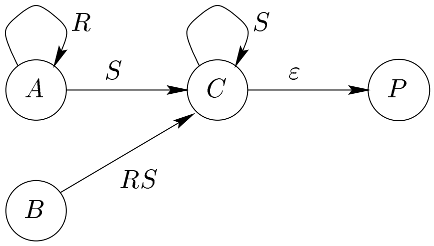

Fig. 2.7. A TBox and the corresponding automaton.

form provides us with the following *characterization of subsumption* of $\mathcal{FL}_0$-concept descriptions [Baader and Narendran, 1998]:

$$C \sqsubseteq D \quad \text{iff} \quad U_i \supseteq V_i \ \text{ for all } i, 1 \leq i \leq k.$$

Since the size of the concept-based normal forms is polynomial in the size of the original descriptions, and since the inclusion tests $U_i \supseteq V_i$ can also be realized in polynomial time, this yields a polynomial-time decision procedure for subsumption in $\mathcal{FL}_0$. In fact, as shown in [Baader *et al.*, 1998], the structural subsumption algorithm for $\mathcal{FL}_0$ can be seen as a special implementation of these inclusion tests.

This characterization of subsumption via inclusion of finite sets of words can be extended to cyclic TBoxes with greatest fixpoint semantics as follows. A given TBox $\mathcal{T}$ can be translated into a finite automaton[1] $\mathcal{A}_\mathcal{T}$ whose states are the concept names occurring in $\mathcal{T}$ and whose transitions are induced by the value restrictions occurring in $\mathcal{T}$ (see Figure 2.7 for an example and [Baader, 1996] for the formal definition).

For a name symbol A and a base symbol P in $\mathcal{T}$, the language $L_{\mathcal{A}_\mathcal{T}}(A, P)$ is the set of all words labeling paths in $\mathcal{A}_\mathcal{T}$ from A to P. The languages $L_{\mathcal{A}_\mathcal{T}}(A, P)$ represent all the value restrictions that must be satisfied by instances of the concept A. With this intuition in mind, the following *characterization of subsumption w.r.t. cyclic* $\mathcal{FL}_0$ *TBoxes with greatest fixpoint semantics* should not be surprising:

$$A \sqsubseteq_\mathcal{T} B \quad \text{iff} \quad L_{\mathcal{A}_\mathcal{T}}(A, P) \supseteq L_{\mathcal{A}_\mathcal{T}}(B, P) \ \text{ for all base symbols } P.$$

In the example of Fig. 2.7, we have $L_{\mathcal{A}_\mathcal{T}}(A, P) = R^*SS^* \supset RSS^* = L_{\mathcal{A}_\mathcal{T}}(B, P)$, and thus $A \sqsubseteq_\mathcal{T} B$, but not $B \sqsubseteq_\mathcal{T} A$.

Obviously, the languages $L_{\mathcal{A}_\mathcal{T}}(A, P)$ are regular, and any regular language can be obtained as such a language. Since inclusion of regular languages is a PSPACE-complete problem [Garey and Johnson, 1979], this shows that subsumption w.r.t. cyclic $\mathcal{FL}_0$-TBoxes with greatest fixpoint semantics is PSPACE-complete [Baader,

[1] Strictly speaking, we obtain a finite automaton with word transitions, i.e., transitions that may be labeled by a word over Σ rather than a letter of Σ.

1996]. For an acyclic terminology $\mathcal{T}$, the automaton $\mathcal{A}_\mathcal{T}$ is acyclic as well. Since inclusion of languages accepted by acyclic finite automata is coNP-complete, this proves Nebel's result that subsumption w.r.t. acyclic $\mathcal{FL}_0$-TBoxes is coNP-complete [Nebel, 1990].

2.4 Language extensions

In Section 2.2 we have introduced the language $\mathcal{ALCN}$ as a prototypical Description Logic. For many applications, the expressive power of $\mathcal{ALCN}$ is not sufficient. For this reason, various other language constructors have been introduced in the literature and are employed by systems. Roughly, these language extensions can be put into two categories, which (for lack of a better name) we will call "classical" and "nonclassical" extensions. Intuitively, a classical extension is one whose semantics can easily be defined within the model-theoretic framework introduced in Section 2.2, whereas defining the semantics of a nonclassical constructor is more problematic and requires an extension of the model-theoretic framework (such as the semantics of the epistemic operator $\mathbf{K}$ introduced in Section 2.2.5). In this section, we briefly introduce the most important classical extensions of Description Logics. Inference procedures for such expressive DLs are discussed in Chapter 5. Nonclassical extensions are the subject of Chapter 6.

In addition to constructors that can be used to build complex roles, we will introduce more expressive number restrictions, and constructors that allow one to express relationships between the role-filler sets of different (complex) roles.

2.4.1 Role constructors

Since roles are interpreted as binary relations, it is quite natural to employ the usual operations on binary relations (such as Boolean operators, composition, inverse, and transitive closure) as role forming constructors. Syntax and semantics of these constructors can be defined as follows:

Definition 2.28 (Role constructors) Every role name is a role description (atomic role), and if R, S are role descriptions, then $R \sqcap S$ (intersection), $R \sqcup S$ (union), $\neg R$ (complement), $R \circ S$ (composition), R^+ (transitive closure), R^- (inverse) are also role descriptions.

A given interpretation $\mathcal{I}$ is extended to (complex) role descriptions as follows:

(i) $(R \sqcap S)^\mathcal{I} = R^\mathcal{I} \cap S^\mathcal{I}$, $(R \sqcup S)^\mathcal{I} = R^\mathcal{I} \cup S^\mathcal{I}$, $(\neg R)^\mathcal{I} = \Delta^\mathcal{I} \times \Delta^\mathcal{I} \setminus R^\mathcal{I}$;
(ii) $(R \circ S)^\mathcal{I} = \{(a, c) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \mid \exists b.\ (a, b) \in R^\mathcal{I} \wedge (b, c) \in S^\mathcal{I}\}$;
(iii) $(R^+)^\mathcal{I} = \bigcup_{i \geq 1} (R^\mathcal{I})^i$, i.e., $(R^+)^\mathcal{I}$ is the transitive closure of $(R^\mathcal{I})$;

(iv) $(R^{-})^{\mathcal{I}} = \{(b,a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}$. ∎

For example, the union of the roles hasSon and hasDaughter can be used to define the role hasChild, and the transitive closure of hasChild expresses the role hasOffspring. The inverse of hasChild yields the role hasParent.

The complexity of satisfiability and subsumption of concepts in the language $\mathcal{ALCN}^{\sqcap}$ (also called $\mathcal{ALCNR}$ in the literature), which extends $\mathcal{ALCN}$ by intersection of roles, has been investigated in [Donini *et al.*, 1997]. It is shown that these problems are still PSPACE-complete, provided that the numbers occurring in number restrictions are written in base 1 representation (where the size of the representation coincides with the number represented). Tobies [2001] shows that this result also hold for non-unary coding of numbers. Decidability of the extension of $\mathcal{ALCN}$ by the three Boolean operators and the inverse operator is an immediate consequence of the fact that concepts of the extended language can be expressed in $\mathcal{C}^2$, i.e., first-order predicate logic with two variables and counting quantifiers, which is known to be decidable in NEXPTIME [Grädel *et al.*, 1997; Pacholski *et al.*, 1997]. Lutz and Sattler [2000] show that $\mathcal{ALC}$ extended by role complement is EXPTIME-complete, whereas $\mathcal{ALC}$ extended by role intersection and (atomic) role complement is NEXPTIME-complete.

In [Baader, 1991], the DL $\mathcal{ALC}_{trans}$, which extends $\mathcal{ALC}$ by transitive-closure, composition, and union of roles, has been introduced, and subsumption and satisfiability of $\mathcal{ALC}_{trans}$-concepts has been shown to be decidable. Schild's observation [Schild, 1991] that $\mathcal{ALC}_{trans}$ is just a syntactic variant of propositional dynamic logic (PDL) [Fischer and Ladner, 1979] yields the exact complexity of subsumption and satisfiability in $\mathcal{ALC}_{trans}$: they are EXPTIME-complete [Fischer and Ladner, 1979; Pratt, 1979; 1980]. The extension of $\mathcal{ALC}_{trans}$ by the inverse constructor corresponds to converse PDL [Fischer and Ladner, 1979], which can also be shown to be decidable in deterministic exponential time [Vardi, 1985]. Whereas this extension of $\mathcal{ALC}_{trans}$ does not change the properties of the obtained DL in a significant way, things become more complex if both number restrictions and the inverse of roles are added to $\mathcal{ALC}_{trans}$. Whereas $\mathcal{ALC}_{trans}$ and $\mathcal{ALC}_{trans}$ with inverse still have the finite model property, $\mathcal{ALC}_{trans}$ extended by inverse and number restrictions does not. Indeed, it is easy to see that the concept

$$\neg A \sqcap \exists R^{-}.A \sqcap (\leqslant 1\, R) \sqcap \forall (R^{-})^{+}.(\exists R^{-}.A \sqcap (\leqslant 1\, R))$$

is satisfiable in an infinite interpretation, but not in a finite one. Nevertheless, this DL still has an EXPTIME-complete subsumption and satisfiability problem. In fact, in [De Giacomo, 1995], number restrictions, the inverse of roles, and Boolean operators on roles are added to $\mathcal{ALC}_{trans}$, and EXPTIME-decidability is shown by a rather ingenious reduction to the decision problem for $\mathcal{ALC}_{trans}$. It should be noted, however, that in this work only atomic roles and their inverse may occur in number restrictions, and that the complement of roles is built with respect to a fixed role any, which must contain all other roles, but need not be interpreted as the universal role (i.e., $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$). As we shall see below, allowing for more complex roles inside number restrictions may easily cause undecidability.

2.4.2 Expressive number restrictions

There are three different ways in which the expressive power of number restrictions can be enhanced.

First, one can consider so-called *qualified number restrictions*, where the number restrictions are concerned with role-fillers belonging to a certain concept. For example, given the role $\mathsf{hasChild}$, the simple number restrictions introduced above can only state that the number of all children is within certain limits, such as in the concept $\geqslant 2\,\mathsf{hasChild} \sqcap \leqslant 5\,\mathsf{hasChild}$. Qualified number restrictions can also express that there are at least 2 sons and at most 5 daughters:

$$\geqslant 2\,\mathsf{hasChild.Male} \sqcap \leqslant 5\,\mathsf{hasChild.Female}.$$

Adding qualified number restrictions to $\mathcal{ALC}$ leaves the important inference problems (like subsumption and satisfiability of concepts, and consistency of ABoxes) decidable: the worst-case complexity is still PSPACE-complete. Membership in PSPACE was first shown for the case where numbers occurring in number restrictions are written in base 1 representation [Hollunder and Baader, 1991; Hollunder, 1996]. More recently, this has been proved even for the case of binary (or, equivalently, decimal) representation of numbers [Tobies, 1999; 2001]. The language stays decidable if general sets of inclusion axioms are allowed [Buchheit *et al.*, 1993].

Second, one can allow for *complex role expressions inside number restrictions.* As already mentioned above, allowing for the three Boolean operators and the inverse operator in number restrictions of $\mathcal{ALCN}$ leaves us within $\mathcal{C}^2$, which is known to be decidable. In [Baader and Sattler, 1996b; 1999], languages that allow for composition of roles in number restrictions have been considered.[1] The extension of $\mathcal{ALC}$ by number restrictions involving composition has a decidable satisfiability and subsumption problem. On the other hand, if either number restrictions involving composition, union and inverse, or number restrictions involving composition and intersection are added, then satisfiability and subsumption become undecidable [Baader and Sattler, 1996b; 1999]. For $\mathcal{ALC}_{trans}$, the extension by number restrictions involving composition is already undecidable [Baader and Sattler, 1999].

Third, one can replace the explicit numbers n in number restrictions by variables α

[1] Note that composition cannot be expressed within $\mathcal{C}^2$.

that stand for arbitrary nonnegative integers [Baader and Sattler, 1996a; 1999]. This allows one, for example, to define the concept of all persons having at least as many daughters as sons, without explicitly saying how many sons and daughters the person has:

$$\mathsf{Person} \sqcap \geqslant \alpha\, \mathsf{hasDaughter} \sqcap \leqslant \alpha\, \mathsf{hasSon}.$$

The expressive power of this language can further be increased by introducing explicit quantification of the numeric variables. For example, it is important to know whether the numeric variables are introduced before or after a value restriction. This is illustrated by the following concept

$$\mathsf{Person} \sqcap \downarrow\alpha.(\forall \mathsf{hasChild}.(\geqslant \alpha\, \mathsf{hasChild} \sqcap \leqslant \alpha\, \mathsf{hasChild})),$$

in which introducing the numerical variable before the universal value restriction makes sure that all the children of the person have the same number of children. Here, $\downarrow\alpha$ stands for an existential quantification of α. Universal quantification of numerical variables comes in via negation. In [Baader and Sattler, 1996a; 1999] it is shown that $\mathcal{ALCN}$ extended by such *symbolic number restrictions* with universal and existential quantification of numerical variables has an undecidable satisfiability and subsumption problem. If one restricts this language to existential quantification of numerical variables and negation on atomic concepts, then satisfiability becomes decidable, but subsumption remains undecidable.

2.4.3 Role-value-maps

Role-value-maps are a family of very expressive concept constructors, which were, however, available in the original KL-ONE-system. They allow one to relate the sets of role fillers of role chains.

Definition 2.29 (Role-value-maps) A role chain is a composition $R_1 \circ \cdots \circ R_n$ of role names. If R, S are role chains, then $R \subseteq S$ and $R = S$ are concepts (role-value-maps). The former is called a *containment* role-value-map, while the latter is called an *equality* role-value-map.

A given interpretation $\mathcal{I}$ is extended to role-value-maps as follows:

(i) $(R \subseteq S)^\mathcal{I} = \{a \in \Delta^\mathcal{I} \mid \forall b.\ (a, b) \in R^\mathcal{I} \rightarrow (a, b) \in S^\mathcal{I}\}$,
(ii) $(R = S)^\mathcal{I} = \{a \in \Delta^\mathcal{I} \mid \forall b.\ (a, b) \in R^\mathcal{I} \leftrightarrow (a, b) \in S^\mathcal{I}\}$. ∎

For example, the concept

$$\mathsf{Person} \sqcap (\mathsf{hasChild} \circ \mathsf{hasFriend} \subseteq \mathsf{knows})$$

describes the persons knowing all the friends of their children, and

$$\mathsf{Person} \sqcap (\mathsf{marriedTo} \circ \mathsf{likesToEat} = \mathsf{likesToEat})$$

describes persons having the same favorite foods as their spouse.

Unfortunately, in the presence of role-value-maps, the subsumption problem is undecidable, even if the language allows only for conjunction and value restriction as additional constructors [Schmidt-Schauß, 1989] (see also Chapter 3).

To avoid this problem, one may restrict the attention to role chains of functional roles, also called *attributes* or *features* in the literature. An interpretation $\mathcal{I}$ interprets the role R as a *functional role* iff $\{(a,b),(a,c)\} \subseteq R^{\mathcal{I}}$ implies $b = c$. In the following, we assume that the set of role names is partitioned into the set of functional roles and the set of ordinary roles. Any interpretation must interpret the functional roles as such. Usually, we write functional roles with small letters f, g, possibly with index.

Definition 2.30 (Agreements) If f, g are role chains of functional roles, then $f \doteq g$ and $f \not\doteq g$ are concepts (agreement and disagreement).

A given interpretation $\mathcal{I}$ is extended to agreements and disagreements as follows:

(i) $(f \doteq g)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b.\ (a,b) \in f^{\mathcal{I}} \wedge (a,b) \in g^{\mathcal{I}}\}$,
(ii) $(f \not\doteq g)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b_1, b_2.\ b_1 \neq b_2 \wedge (a,b_1) \in f^{\mathcal{I}} \wedge (a,b_2) \in g^{\mathcal{I}}\}$. ■

In the literature, the agreement constructor is sometimes also called the *same-as* constructor. Note that, since f, g are role chains between functional roles, there can be at most one role filler for a w.r.t. the respective role chain. Also note that the semantics of agreements and disagreements requires these role fillers to exist (and be equal or distinct) for a to belong to the concept.

For example, hasMother, hasFather, and hasLastName with their usual interpretation are functional roles, whereas hasParent and hasChild are not. The concept

$$\begin{aligned}\mathsf{Person} \sqcap{} & (\mathsf{hasLastName} \doteq \mathsf{hasMother} \circ \mathsf{hasLastName}) \\ \sqcap{} & (\mathsf{hasLastName} \not\doteq \mathsf{hasFather} \circ \mathsf{hasLastName})\end{aligned}$$

describes persons whose last name coincides with the last name of their mother, but not with the last name of their father.

The restriction to functional roles makes reasoning in $\mathcal{ALC}$ extended by agreements and disagreements decidable [Hollunder and Nutt, 1990]. A structural subsumption algorithm for the language provided by the CLASSIC-system, which includes the same-as constructor, can be found in [Borgida and Patel-Schneider, 1994]. However, if general inclusion axioms (or transitive closure of functional roles or cyclic definitions) are allowed, then agreements and disagreements between chains of functional roles again cause subsumption to become undecidable [Nebel, 1991;

Baader *et al.*, 1993]. Additional types of role interaction constructors similar to agreements and role-value-maps are investigated in [Hanschke, 1992].

Acknowledgement

We would like to thank Maarten de Rijke for his pointers to the literature on Beth definability in modal logics.

Bibliography

[Baader and Hanschke, 1991] Franz Baader and Philipp Hanschke. A schema for integrating concrete domains into concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.

[Baader and Hollunder, 1991] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge (PDK'91)*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 67–86. Springer, 1991.

[Baader and Narendran, 1998] Franz Baader and Paliath Narendran. Unification of concept terms in description logics. In H. Prade, editor, *Proc. of the 13th Eur. Conf. on Artificial Intelligence (ECAI'98)*, pages 331–335. John Wiley & Sons, 1998.

[Baader and Sattler, 1996a] Franz Baader and Ulrike Sattler. Description logics with symbolic number restrictions. In *Proc. of the 12th Eur. Conf. on Artificial Intelligence (ECAI'96)*, pages 283–287. John Wiley & Sons, 1996.

[Baader and Sattler, 1996b] Franz Baader and Ulrike Sattler. Number restrictions on complex roles in description logics: A preliminary report. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 328–338, 1996.

[Baader and Sattler, 1999] Franz Baader and Ulrike Sattler. Expressive number restrictions in description logics. *J. of Logic and Computation*, 9(3):319–350, 1999.

[Baader *et al.*, 1993] Franz Baader, Hans-Jürgen Bürckert, Bernhard Nebel, Werner Nutt, and Gert Smolka. On the expressivity of feature logics with negation, functional uncertainty, and sort equations. *J. of Logic, Language and Information*, 2:1–18, 1993.

[Baader *et al.*, 1994] Franz Baader, Enrico Franconi, Bernhard Hollunder, Bernhard Nebel, and Hans-Jürgen Profitlich. 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, 1994.

[Baader *et al.*, 1996] Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.

[Baader *et al.*, 1998] Franz Baader, Ralf Küsters, and Ralf Molitor. Structural subsumption considered from an automata theoretic point of view. In *Proc. of the 1998 Description Logic Workshop (DL'98)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-11/`, 1998.

[Baader, 1991] Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, 1991.

[Baader, 1996] Franz Baader. Using automata theory for characterizing the semantics of terminological cycles. *Ann. of Mathematics and Artificial Intelligence*, 18:175–219, 1996.

[Borgida and Patel-Schneider, 1994] Alexander Borgida and Peter F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *J. of Artificial Intelligence Research*, 1:277–308, 1994.

[Borgida, 1996] Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[Brachman and Schmolze, 1985] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Brachman *et al.*, 1991] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alexander Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 401–456. Morgan Kaufmann, Los Altos, 1991.

[Brachman, 1977] Ronald J. Brachman. What's in a concept: Structural foundations for semantic networks. *Int. Journal of Man-Machine Studies*, 9(2):127–152, 1977.

[Brachman, 1978] Ronald J. Brachman. Structured inheritance networks. In W. A. Woods and R. J. Brachman, editors, *Research in Natural Language Understanding*, Quarterly Progress Report No. 1, BBN Report No. 3742, pages 36–78. Bolt, Beranek and Newman Inc., Cambridge, Mass., 1978.

[Bresciani *et al.*, 1995] P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive description logics: Preliminary report. In *Proc. of the 1995 Description Logic Workshop (DL'95)*, pages 131–139, 1995.

[Buchheit *et al.*, 1993] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *J. of Artificial Intelligence Research*, 1:109–138, 1993.

[De Giacomo and Lenzerini, 1994a] Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI'94)*, pages 205–212. AAAI Press/The MIT Press, 1994.

[De Giacomo and Lenzerini, 1994b] Giuseppe De Giacomo and Maurizio Lenzerini. Concept language with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. of the 11th Eur. Conf. on Artificial Intelligence (ECAI'94)*, pages 411–415, 1994.

[De Giacomo and Lenzerini, 1997] Giuseppe De Giacomo and Maurizio Lenzerini. A uniform framework for concept definitions in description logics. *J. of Artificial Intelligence Research*, 6:87–110, 1997.

[De Giacomo, 1995] Giuseppe De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.

[Donini and Massacci, 2000] Francesco M. Donini and Fabio Massacci. EXPTIME tableaux for $\mathcal{ALC}$. *Artificial Intelligence*, 124(1):87–138, 2000.

[Donini *et al.*, 1991] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. Tractable concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 458–463, Sydney (Australia), 1991.

[Donini *et al.*, 1992] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. Adding epistemic operators to concept languages. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 342–353. Morgan Kaufmann, Los Altos, 1992.

[Donini *et al.*, 1994] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Deduction in concept languages: From subsumption to instance checking. *J. of Logic and Computation*, 4(4):423–452, 1994.

[Donini *et al.*, 1997] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. *Information and Computation*, 134:1–58, 1997.

[Donini *et al.*, 1998] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. An epistemic operator for description logics. *Artificial Intelligence*, 100(1–2):225–274, 1998.

[Fischer and Ladner, 1979] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.

[Fitting, 1993] Melvin Fitting. Basic modal logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1, pages 365–448. Oxford Science Publications, 1993.

[Gabbay, 1972] Dov M. Gabbay. Craig's interpolation theorem for modal logics. In *Proceedings of the Logic Conference*, volume 255 of *Lecture Notes in Mathematics*, pages 111–127. Springer, 1972.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability — A guide to NP-completeness.* W. H. Freeman and Company, San Francisco (CA, USA), 1979.

[Grädel *et al.*, 1997] Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *Proc. of the 12th IEEE Symp. on Logic in Computer Science (LICS'97)*, pages 306–317. IEEE Computer Society Press, 1997.

[Haarslev and Möller, 2000] Volker Haarslev and Ralf Möller. Expressive ABox reasoning with number restrictions, role hierarchies, and transitively closed roles. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 273–284, 2000.

[Haarslev and Möller, 2001a] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 161–168, 2001.

[Haarslev and Möller, 2001b] Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.

[Haarslev *et al.*, 1999] Volker Haarslev, Carsten Lutz, and Ralf Möller. A description logic with concrete domains and role-forming predicates. *J. of Logic and Computation*, 9(3):351–384, 1999.

[Hanschke, 1992] Philipp Hanschke. Specifying role interaction in concept languages. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 318–329. Morgan Kaufmann, Los Altos, 1992.

[Hollunder and Baader, 1991] Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. Technical Report RR-91-03, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern (Germany), 1991. An abridged version appeared in *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*.

[Hollunder and Nutt, 1990] Bernhard Hollunder and Werner Nutt. Subsumption algorithms for concept languages. Technical Report RR-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern (Germany), 1990.

[Hollunder *et al.*, 1990] Bernhard Hollunder, Werner Nutt, and Manfred Schmidt-Schauß. Subsumption algorithms for concept description languages. In *Proc. of the 9th Eur. Conf. on Artificial Intelligence (ECAI'90)*, pages 348–353, London (United Kingdom), 1990. Pitman.

[Hollunder, 1990] Bernhard Hollunder. Hybrid inferences in KL-ONE-based knowledge representation systems. In *Proc. of the German Workshop on Artificial Intelligence*, pages 38–47. Springer, 1990.

[Hollunder, 1996] Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Ann. of Mathematics and Artificial Intelligence*, 18(2–4):133–157, 1996.

[Horrocks and Patel-Schneider, 1999] Ian Horrocks and Peter F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999.

[Horrocks and Sattler, 1999] Ian Horrocks and Ulrike Sattler. A description logic with transitive and inverse roles and role hierarchies. *J. of Logic and Computation*, 9(3):385–410, 1999.

[Horrocks, 1998] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[Küsters, 1998] Ralf Küsters. Characterizing the semantics of terminological cycles in $\mathcal{ALN}$ using finite automata. In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 499–510, 1998.

[Levesque and Brachman, 1987] Hector J. Levesque and Ron J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[Lutz and Sattler, 2000] Carsten Lutz and Ulrike Sattler. The complexity of reasoning with boolean modal logic. In *Proc. of Advances in Modal Logic 2000 (AiML 2000)*, 2000.

[Lutz, 1999] Carsten Lutz. Complexity of terminological reasoning revisited. In *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 181–200. Springer, 1999.

[MacGregor, 1991] Robert MacGregor. The evolving technology of classification-based knowledge representation systems. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 385–400. Morgan Kaufmann, Los Altos, 1991.

[Mortimer, 1975] Michael Mortimer. On languages with two variables. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:135–140, 1975.

[Nebel, 1988] Bernhard Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

[Nebel, 1990] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[Nebel, 1991] Bernhard Nebel. Terminological cycles: Semantics and computational properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–361. Morgan Kaufmann, Los Altos, 1991.

[Pacholski *et al.*, 1997] Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity of two-variable logic with counting. In *Proc. of the 12th IEEE Symp. on Logic in Computer Science (LICS'97)*, pages 318–327. IEEE Computer Society Press, 1997.

[Patel-Schneider, 1989] Peter F. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39:263–272, 1989.

[Patel-Schneider, 1999] Peter F. Patel-Schneider. DLP. In *Proc. of the 1999 Description Logic Workshop (DL'99)*, pages 9–13. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-22/`, 1999.

[Pratt, 1979] Vaugham R. Pratt. Models of program logic. In *Proc. of the 20th Annual Symp. on the Foundations of Computer Science (FOCS'79)*, pages 115–122, 1979.

[Pratt, 1980] Vaugham R. Pratt. A near-optimal method for reasoning about action. *J. of*

Computer and System Sciences, 20:231–255, 1980.

[Sattler, 1996] Ulrike Sattler. A concept language extended with different kinds of transitive roles. In Günter Görz and Steffen Hölldobler, editors, *Proc. of the 20th German Annual Conf. on Artificial Intelligence (KI'96)*, number 1137 in Lecture Notes in Artificial Intelligence, pages 333–345. Springer, 1996.

[Savitch, 1970] W. J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. of Computer and System Sciences*, 4:177–192, 1970.

[Schaerf, 1994] Andrea Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.

[Schild, 1991] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991.

[Schild, 1993] Klaus Schild. Combining terminological logics with tense logic. In *Proc. of the 6th Portuguese Conf. on Artificial Intelligence (EPIA'93)*, volume 727 of *Lecture Notes in Computer Science*, pages 105–120. Springer, 1993.

[Schild, 1994] Klaus Schild. Terminological cycles and the propositional μ-calculus. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 509–520, Bonn (Germany), 1994. Morgan Kaufmann, Los Altos.

[Schmidt-Schauß and Smolka, 1991] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[Schmidt-Schauß, 1989] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In Ron J. Brachman, Hector J. Levesque, and Ray Reiter, editors, *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'89)*, pages 421–431. Morgan Kaufmann, Los Altos, 1989.

[Smolka, 1988] Gert Smolka. A feature logic with subsorts. Technical Report 33, IWBS, IBM Deutschland, P.O. Box 80 08 80 D-7000 Stuttgart 80, Germany, 1988.

[Tarski, 1955] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Tobies, 1999] Stephan Tobies. A PSPACE algorithm for graded modal logic. In H. Ganzinger, editor, *Proc. of the 16th Int. Conf. on Automated Deduction (CADE'99)*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 52–66. Springer, 1999.

[Tobies, 2001] Stephan Tobies. PSPACE reasoning for graded modal logics. *J. of Logic and Computation*, 11(1):85–106, 2001.

[Vardi and Wolper, 1986] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32:183–221, 1986. A preliminary version appeared in *Proc. of the 16th ACM SIGACT Symp. on Theory of Computing (STOC'84)*.

[Vardi, 1985] Moshe Y. Vardi. The taming of converse: Reasoning about two-way computations. In R. Parikh, editor, *Proc. of the 4th Workshop on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 413–424. Springer, 1985.