Chapter 3

Description Logics

Franz Baader, Ian Horrocks and Ulrike Sattler

Abstract

In this chapter we will introduce description logics, a family of logic-based knowledge representation languages that can be used to represent the terminological knowledge of an application domain in a structured way. We will first review their provenance and history, and show how the field has developed. We will then introduce the basic description logic $\mathcal{ALC}$ in some detail, including definitions of syntax, semantics and basic reasoning services, and describe important extensions such as inverse roles, number restrictions, and concrete domains. Next, we will discuss the relationship between description logics and other formalisms, in particular first order and modal logics; the most commonly used reasoning techniques, in particular tableaux, resolution and automata based techniques; and the computational complexity of basic reasoning problems. After reviewing some of the most prominent applications of description logics, in particular ontology language applications, we will conclude with an overview of other aspects of description logic research, and with pointers to the relevant literature.

3.1 Introduction

Description logics (DLs) [14, 25, 50] are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way. The name *description logics* is motivated by the fact that, on the one hand, the important notions of the domain are described by concept *descriptions*, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the particular DL; on the other hand, DLs differ from their predecessors, such as semantic networks and frames, in that they are equipped with a formal, *logic*-based semantics.

We will first illustrate some typical constructors by an example; formal definitions will be given in Section 3.2. Assume that we want to define the concept of "A man that is married to a doctor, and all of whose children are either doctors or professors." This concept can be described with the following concept description:

$$\mathsf{Human} \sqcap \neg\mathsf{Female} \sqcap (\exists \mathsf{married}.\mathsf{Doctor}) \sqcap (\forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Professor})).$$

This description employs the Boolean constructors *conjunction* ($\sqcap$), which is interpreted as set intersection, *disjunction* ($\sqcup$), which is interpreted as set union, and *negation* ($\neg$), which is interpreted as set complement, as well as the *existential restriction* constructor ($\exists r.C$), and the *value restriction* constructor ($\forall r.C$). An individual, say Bob, belongs to $\exists$married.Doctor if there exists an individual that is married to Bob (i.e., is related to Bob via the married role) and is a doctor (i.e., belongs to the concept Doctor). Similarly, Bob belongs to $\forall$hasChild.(Doctor $\sqcup$ Professor) if all his children (i.e., all individuals related to Bob via the hasChild role) are either doctors or professors.

Concept descriptions can be used to build statements in a DL knowledge base, which typically comes in two parts: a terminological and an assertional one. In the *terminological* part, called the TBox, we can describe the relevant notions of an application domain by stating properties of concepts and roles, and relationships between them—it corresponds to the *schema* in a database setting. In its simplest form, a TBox statement can introduce a name (abbreviation) for a complex description. For example, we could introduce the name HappyMan as an abbreviation for the concept description from above:

$$\mathsf{HappyMan} \equiv \mathsf{Human} \sqcap \neg\mathsf{Female} \sqcap (\exists \mathsf{married.Doctor}) \sqcap (\forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Professor})).$$

More expressive TBoxes allow the statement of more general *axioms* such as

$$\exists \mathsf{hasChild.Human} \sqsubseteq \mathsf{Human},$$

which says that only humans can have human children. Note that, in contrast to the abbreviation statement from above, this statement does not define a concept. It just constrains the way in which concepts and roles (in this case, Human and hasChild) can be interpreted.

Obviously, all the knowledge we have described in our example could easily be represented by formulae of first-order predicate logic (see also Section 3.3). The variable-free syntax of description logics makes TBox statements easier to read than the corresponding first-order formulae. However, the main reason for using DLs rather than predicate logic is that DLs are carefully tailored such that they combine interesting means of expressiveness with decidability of the important reasoning problems (see below).

The *assertional* part of the knowledge base, called the ABox, is used to describe a concrete situation by stating properties of individuals—it corresponds to the *data* in a database setting. For example, the assertions

$$\mathsf{HappyMan}(\mathsf{BOB}), \quad \mathsf{hasChild}(\mathsf{BOB}, \mathsf{MARY}), \quad \neg\mathsf{Doctor}(\mathsf{MARY})$$

state that Bob belongs to the concept HappyMan, that Mary is one of his children, and that Mary is not a doctor. Modern DL systems all employ this kind of restricted ABox formalism, which basically can be used to state ground facts. This differs from the use of the ABox in the early DL system KRYPTON [38], where ABox statements could be arbitrary first-order formulae. The underlying idea was that the ABox could then be used to represent knowledge that was not expressible in the restricted TBox formalism of KRYPTON, but this came with a cost: reasoning about ABox knowledge required the use of a general theorem prover, which was quite inefficient and could lead to non-termination of the reasoning procedure.

Modern description logic systems provide their users with reasoning services that can automatically deduce implicit knowledge from the explicitly represented knowledge, and

always yield a correct answer in finite time. In contrast to the database setting, such inference capabilities take into consideration *both* the terminological statements (schema) *and* the assertional statements (data). The *subsumption* algorithm determines subconcept-superconcept relationships: C is subsumed by D if all instances of C are necessarily instances of D, i.e., the first description is always interpreted as a subset of the second description. For example, given the definition of HappyMan from above plus the axiom Doctor $\sqsubseteq$ Human, which says that all doctors are human, HappyMan is subsumed by $\exists$married.Human—since instances of HappyMan are married to some instance of Doctor, and all instances of Doctor are also instances of Human. The *instance* algorithm determines instance relationships: the individual i is an instance of the concept description C if i is always interpreted as an element of the interpretation of C. For example, given the assertions from above and the definition of HappyMan, MARY is an instance of Professor (because BOB is an instance of HappyMan, so all his children are either Doctors or Professors, MARY is a child of BOB, and MARY is not a Doctor). The *consistency* algorithm determines whether a knowledge base (consisting of a set of assertions and a set of terminological axioms) is non-contradictory. For example, if we add $\neg$Professor(MARY) to the three assertions from above, then the knowledge base containing these assertions together with the definition of HappyMan from above is inconsistent.

In a typical application, one would start building the TBox, making use of the reasoning services provided to ensure that all concepts in it are satisfiable, i.e., are not subsumed by the bottom concept, which is always interpreted as the empty set. Moreover, one would use the subsumption algorithm to compute the subsumption hierarchy, i.e., to check, for each pair of concept names, whether one is subsumed by the other. This hierarchy would then be inspected to make sure that it coincides with the intention of the modeller. Given, in addition, an ABox, one would first check for its consistency with the TBox and then, for example, compute the most specific concept(s) that each individual is an instance of (this is often called *realizing* the ABox). We could also use a concept description as a query, i.e., we could ask the DL system to identify all those individuals that are instances of the given, possibly complex, concept description.

In order to ensure a reasonable and predictable behavior of a DL system, these inference problems should at least be decidable for the DL employed by the system, and preferably of low complexity. Consequently, the expressive power of the DL in question must be restricted in an appropriate way. If the imposed restrictions are too severe, however, then the important notions of the application domain can no longer be expressed. Investigating this trade-off between the expressivity of DLs and the complexity of their inference problems has been one of the most important issues in DL research. This investigation has included both theoretical research, e.g., determining the worst case complexities for various DLs and reasoning problems, and practical research, e.g., developing systems and optimisation techniques, and empirically evaluating their behaviour when applied to benchmarks and used in various applications. The emphasis on decidable formalisms of restricted expressive power is also the reason why a great variety of extension of basic DLs have been considered. Some of these extensions leave the realm of classical first-order predicate logic, such as DLs with modal and temporal operators, fuzzy DLs, and probabilistic DLs (see [22] for details), but the goal of this research was still to design decidable extensions. If an application requires more expressive power than can be supplied by a decidable DL, then one usually embeds the DL into an application program or another KR formalism (see Section 3.8) rather than using an undecidable DL.

In the remainder of this section we will first give a brief overview of the history of DLs, and then describe the structure of this chapter. Research in Description Logics can be roughly classified into the following phases.

Phase 0 (1965–1980) is the pre-DL phase, in which *semantic networks* [139] and *frames* [122] were introduced as specialized approaches for representing knowledge in a structured way, and then criticized because of their lack of a formal semantics [164, 35, 84, 85]. An approach to overcome these problems were Brachman's *structured inheritance networks* [36], which were realized in the system KL-ONE, the first DL system.

Phase 1 (1980–1990) was mainly concerned with implementation of systems, such as KL-ONE, K-REP, KRYPTON, BACK, and LOOM [41, 119, 38, 138, 118]. These systems employed so-called *structural subsumption algorithms*, which first normalize the concept descriptions, and then recursively compare the syntactic structure of the normalized descriptions [126]. These algorithms are usually relatively efficient (polynomial), but they have the disadvantage that they are complete only for very inexpressive DLs, i.e., for more expressive DLs they cannot detect all subsumption/instance relationships. During this phase, the first logic-based accounts of the semantics of the underlying representation formalisms were given [38, 39], which made formal investigations into the complexity of reasoning in DLs possible. For example, in [39] it was shown that seemingly small additions to the expressive power of the representation formalism can cause intractability of the subsumption problem. In [149] it was shown that subsumption in the representation language underlying KL-ONE is even undecidable, and in [127] it was shown that the use of a TBox formalism that allows to introduce abbreviations for complex concept description makes subsumption intractable if the underlying DL has the constructors conjunction and value restriction (these constructors were supported by all the DL systems available at that time). As a reaction to these negative complexity results, the implementors of the CLASSIC system (the first industrial-strength DL system) carefully restricted the expressive power of their DL [136, 37].

Phase 2 (1990–1995) started with the introduction of a new algorithmic paradigm into DLs, so-called *tableaux based algorithms* [150, 63, 89]. They work on propositionally closed DLs (i.e., DLs with all Boolean operators), and are complete also for expressive DLs. To decide the consistency of a knowledge base, a tableaux based algorithm tries to construct a model of it by structurally decomposing the concepts in the knowledge base, thus inferring new constraints on the elements of this model. The algorithm either stops because all attempts to build a model failed with obvious contradictions, or it stops with a "canonical" model. Since, in propositionally closed DLs, the subsumption and the instance problem can be reduced to consistency, a consistency algorithm can solve all the inference problems mentioned above. The first systems employing such algorithms (KRIS and CRACK) demonstrated that optimized implementations of these algorithm led to an acceptable behavior of the system, even though the worst-case complexity of the corresponding reasoning problems is no longer in polynomial time [18, 44]. This phase also saw a thorough analysis of the complexity of reasoning in various DLs [63, 64, 62], and the important observation that DLs are very closely related to modal logics [145].

Phase 3 (1995–2000) is characterized by the development of inference procedures for very expressive DLs, either based on the tableau approach [100, 92], or on a translation into modal logics [57, 58, 56, 59]. Highly optimized systems (FaCT, RACE, and DLP

[95, 80, 133]) showed that tableau-based algorithms for expressive DLs led to a good practical behavior of the system even on (some) large knowledge bases. In this phase, the relationship to modal logics [57, 147] and to decidable fragments of first-order logic [33, 129, 79, 77, 78] was also studied in more detail, and applications in databases (like schema reasoning, query optimization, and integration of databases) were investigated [45, 47, 51].

We are now in *Phase 4*, where the results from the previous phases are used to develop industrial strength DL systems employing very expressive DLs, with applications like the Semantic Web or knowledge representation and integration in medical- and bio-informatics in mind. On the academic side, the interest in less expressive DLs has been revived, with the goal of developing tools that can deal with very large terminological and/or assertional knowledge bases [6, 23, 53, 1].

The structure of the remainder of the chapter is as follows. In Section 3.2 we introduce syntax and semantics of the prototypical DL $\mathcal{ALC}$, and some important extensions of $\mathcal{ALC}$. In Section 3.3 we discuss the relationship between DLs and other logical formalisms. In Section 3.4 we describe tableau-based reasoning techniques for $\mathcal{ALC}$, and in Section 3.5 we investigate the computation complexity of reasoning in $\mathcal{ALC}$. In Section 3.6 we introduce other reasoning techniques that can be used for DLs. In Section 3.7 we discuss the use of DLs in ontology language applications. Finally, in Section 3.8, we sketch important areas of DL research that have not been mentioned so far, and provide pointers to the literature.

Although we have endeavoured to cover the most important areas of DL research, we have decided to treat some areas in more detail rather than giving a comprehensive survey of the whole field. Readers seeking such a survey are directed to [14].

3.2 A Basic DL and its Extensions

In this section we will define the syntax and semantics of the basic DL $\mathcal{ALC}$, and the most widely used DL reasoning services. We will also introduce important extensions to $\mathcal{ALC}$, including inverse roles, number restrictions, and concrete domains. The name $\mathcal{ALC}$ stands for "*A*ttributive concept *L*anguage with *C*omplements." It was first introduced in [150], where also a first naming scheme for DLs was proposed: starting from a basic DL $\mathcal{AL}$, the addition of a constructors is indicated by appending a corresponding letter; e.g., $\mathcal{ALC}$ is obtained from $\mathcal{AL}$ by adding the complement operator ($\neg$) and $\mathcal{ALE}$ is obtained from $\mathcal{AL}$ by adding existential restrictions ($\exists r.C$) (for more details on such naming schemes for DLs, see [10]).

3.2.1 Syntax and Semantics of $\mathcal{ALC}$

In following, we give formal definitions of the syntax and semantics of the constructors that we have described informally in the introduction. The DL that includes just this set of constructors (i.e., conjunction, disjunction, negation, existential restriction and value restriction) is called $\mathcal{ALC}$.

Definition 1 ($\mathcal{ALC}$ syntax) *Let N_C be a set of* concept names *and N_R be a set of* role names*. The sets of $\mathcal{ALC}$-*concepts *is the smallest sets such that*

1. $\top$, $\bot$, *and every concept name* $A \in N_C$ *is an* $\mathcal{ALC}$*-concept,*

2. *if* C *and* D *are* $\mathcal{ALC}$*-concepts and* $r \in N_R$*, then* $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall r.C$, *and* $\exists r.C$ *are* $\mathcal{ALC}$*-concepts.*

The semantics of $\mathcal{ALC}$ (and of DLs in general) is given in terms of *interpretations*.

Definition 2 ($\mathcal{ALC}$ **semantics)** *An interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *consists of a non-empty set* $\Delta^{\mathcal{I}}$*, called the* domain *of* $\mathcal{I}$*, and a function* $\cdot^{\mathcal{I}}$ *that maps every* $\mathcal{ALC}$*-concept to a subset of* $\Delta^{\mathcal{I}}$*, and every role name to a subset of* $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ *such that, for all* $\mathcal{ALC}$*-concepts* C, D *and all role names* r*,*

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}}, \qquad \bot^{\mathcal{I}} = \emptyset \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \qquad \neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\
(\exists r.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \textit{There is some } y \in \Delta^{\mathcal{I}} \textit{ with } \langle x, y\rangle \in r^{\mathcal{I}} \textit{ and } y \in C^{\mathcal{I}}\}, \\
(\forall r.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \textit{For all } y \in \Delta^{\mathcal{I}}\textit{, if } \langle x, y\rangle \in r^{\mathcal{I}}\textit{, then } y \in C^{\mathcal{I}}\}.
\end{aligned}$$

We say that $C^{\mathcal{I}}$ *(*$r^{\mathcal{I}}$*) is the* extension *of the concept* C *(role name* r*) in the interpretation* $\mathcal{I}$*. If* $x \in C^{\mathcal{I}}$*, then we say that* x *is an* instance *of* C *in* $\mathcal{I}$*.*

As mentioned in the introduction, a DL knowledge base (KB) is made up of two parts, a terminological part (called the TBox) and an assertional part (called the ABox), each part consisting of a set of axioms. The most general form of TBox axioms are so-called general concept inclusions.

Definition 3 *A* general concept inclusion (GCI) *is of the form* $C \sqsubseteq D$*, where* C, D *are* $\mathcal{ALC}$*-concepts. A finite set of GCIs is called a* TBox*. An interpretation* $\mathcal{I}$ *is a* model *of a GCI* $C \sqsubseteq D$ *if* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$*;* $\mathcal{I}$ *is a model of a TBox* $\mathcal{T}$ *if it is a model of every GCI in* $\mathcal{T}$*.*

We use $C \equiv D$ as an abbreviation for the symmetrical pair of GCIs $C \sqsubseteq D$ and $D \sqsubseteq C$.

An axiom of the form $A \equiv C$, where A is a concept name, is called a *definition*. A TBox $\mathcal{T}$ is called *definitorial* if it contains only definitions, with the additional restriction that (i) $\mathcal{T}$ contains at most one definition for any given concept name, and (ii) $\mathcal{T}$ is acyclic, i.e., the definitions of any concept A in $\mathcal{T}$ does not refer (directly or indirectly) to A itself. Definitorial TBoxes are also called *acyclic* TBoxes in the literature. Given a definitorial TBox $\mathcal{T}$, concept names occurring on the left-hand side of such a definition are called *defined* concepts, whereas the others are called *primitive* concepts. The name "definitorial" is motivated by the fact that, in such a TBox, the extensions of the defined concepts are uniquely determined by the extensions of the primitive concepts and the role names. From a computational point of view, definitorial TBox are interesting since they may allow for the use of simplified reasoning techniques (see Section 3.4), and reasoning w.r.t. such TBoxes is often of a lower complexity than reasoning w.r.t. a general TBox (see Section 3.5).

The ABox can contain two kinds of axiom, one for asserting that an individual is an instance of a given concept, and the other for asserting that a pair of individuals is an instance of a given role.

Definition 4 *An* assertional axiom *is of the form $x : C$ or $(x, y) : r$, where C is an $\mathcal{ALC}$-concept, r is an $\mathcal{ALC}$-role, and x and y are individual names. A finite set of assertional axioms is called an* ABox. *An interpretation $\mathcal{I}$ is a model of an assertional axiom $x : C$ if $x^{\mathcal{I}} \in C^{\mathcal{I}}$, and $\mathcal{I}$ is a model of an assertional axiom $(x, y) : r$ if $\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$; $\mathcal{I}$ is a model of an ABox $\mathcal{A}$ if it is a model of every axiom in $\mathcal{A}$.*

Several other notations for writing ABox axioms can be found in the literature, e.g., $C(x)$, $r(x, y)$ and $\langle x, y \rangle : r$.

Definition 5 *A* knowledge base *(KB) is a pair $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. An interpretation $\mathcal{I}$ is a model of a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ if $\mathcal{I}$ is a model of $\mathcal{T}$ and $\mathcal{I}$ is a model of $\mathcal{A}$.*

We will write $\mathcal{I} \models \mathcal{K}$ (resp. $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \mathcal{A}$, $\mathcal{I} \models a$) to denote that $\mathcal{I}$ is a model of a KB $\mathcal{K}$ (resp. TBox $\mathcal{T}$, ABox $\mathcal{A}$, axiom a).

3.2.2 Important Inference Problems

We define inference problems w.r.t. a KB consisting of a TBox and an ABox. Later on, we will also consider special cases where the TBox or/and ABox is empty, or where the TBox satisfies additional restrictions, such as being definitorial.

Definition 6 *Given a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox, $\mathcal{K}$ is called* consistent *if it has a model. A concept C is called* satisfiable with respect to $\mathcal{K}$ *if there is a model $\mathcal{I}$ of $\mathcal{K}$ with $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a* model of C *w.r.t. $\mathcal{K}$. The concept D* subsumes *the concept C w.r.t. $\mathcal{K}$ (written $\mathcal{K} \models C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{K}$. Two concepts C, D are* equivalent *w.r.t. $\mathcal{K}$ (written $\mathcal{K} \models C \equiv D$) if they subsume each other w.r.t. $\mathcal{K}$. An individual a is an* instance of a concept C with respect to $\mathcal{K}$ *(written $\mathcal{K} \models a : C$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{K}$. A pair of individuals (a, b) is an* instance of a role r with respect to $\mathcal{K}$ *(written $\mathcal{K} \models (a, b) : r$) if $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{K}$.*

For a DL providing all the Boolean operators, like $\mathcal{ALC}$, all of the above reasoning problems can be reduced to KB consistency. For example, $(\mathcal{T}, \mathcal{A}) \models a : C$ iff $(\mathcal{T}, \mathcal{A} \cup \{a : \neg C\})$ is inconsistent. We will talk about satisfiability (resp. subsumption and equivalence) with respect to a TBox $\mathcal{T}$, meaning satisfiability (resp. subsumption and equivalence) with respect to the KB $(\mathcal{T}, \emptyset)$. This is often referred to as *terminological* reasoning. In many cases (e.g. in the case of $\mathcal{ALC}$), the ABox has no influence on terminological reasoning, i.e., satisfiability (resp. subsumption and equivalence) with respect to $(\mathcal{T}, \mathcal{A})$ coincides with satisfiability (resp. subsumption and equivalence) with respect to $\mathcal{T}$, as long as the ABox $\mathcal{A}$ is consistent (i.e., has a model).

3.2.3 Important Extensions to $\mathcal{ALC}$

One prominent application of DLs is as the formal foundation for ontology languages. Examples of DL based ontology languages include OIL [69], DAML+OIL [97, 98], and

OWL [134], a recently emerged ontology language standard developed by the W3C Web-Ontology Working Group.[1]

High quality ontologies are crucial for many applications, and their construction, integration, and evolution greatly depends on the availability of a well-defined semantics and powerful reasoning tools. Since DLs provide for both, they should be ideal candidates for ontology languages. That much was already clear ten years ago, but at that time there was a fundamental mismatch between the expressive power and the efficiency of reasoning that DL systems provided, and the expressivity and the large knowledge bases that users needed [67]. Through basic research in DLs over the last 10–15 years, as summarized in the introduction, this gap between the needs of ontologist and the systems that DL researchers provide has finally become narrow enough to build stable bridges. In particular, $\mathcal{ALC}$ has been extended with several features that are important in an ontology language, including (qualified) number restrictions, inverse roles, transitive roles, subroles, concrete domains, and nominals.

With *number restrictions*, it is possible to describe the number of relationships of a particular type that individuals can participate in. For example, we may want to say that a person can be married to at most one other individual:

$$\mathsf{Person} \sqsubseteq \leqslant 1 \mathsf{married},$$

and we may want to extend our definition of HappyMan to include the fact that instances of HappyMan have between two and four children:

$$\begin{aligned} \mathsf{HappyMan} \quad \equiv \quad & \mathsf{Human} \sqcap \neg \mathsf{Female} \sqcap (\exists \mathsf{married}.\mathsf{Doctor}) \sqcap \\ & (\forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Professor})) \sqcap \\ & \geqslant 2 \mathsf{hasChild} \sqcap \leqslant 4 \mathsf{hasChild}. \end{aligned}$$

With *qualified number restrictions*, we can additionally describe the type of individuals that are counted by a given number restriction. For example, using qualified number restrictions, we could further extend our definition of HappyMan to include the fact that instances of HappyMan have at least two children who are doctors:

$$\begin{aligned} \mathsf{HappyMan} \quad \equiv \quad & \mathsf{Human} \sqcap \neg \mathsf{Female} \sqcap (\exists \mathsf{married}.\mathsf{Doctor}) \sqcap \\ & (\forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Professor})) \sqcap \\ & \geqslant 2 \mathsf{hasChild}.\mathsf{Doctor} \sqcap \leqslant 4 \mathsf{hasChild}. \end{aligned}$$

With *inverse roles*, *transitive roles*, and *subroles* we can, in addition to hasChild, also use its inverse hasParent, specify that hasAncestor is transitive, and specify that hasParent is a subrole of hasAncestor.

Concrete domains [16, 115] integrate DLs with concrete sets such as the real numbers, integers, or strings, as well as concrete predicates defined on these sets, such as numerical comparisons (e.g., $\leq$), string comparisons (e.g., isPrefixOf), or comparisons with constants (e.g., ≤ 17). This supports the modelling of concrete properties of abstract objects such as the age, the weight, or the name of a person, and the comparison of these concrete properties. Unfortunately, in their unrestricted form, concrete domains can have dramatic effects on the decidability and computational complexity of the underlying DL [17, 115].

[1] `http://www.w3.org/2001/sw/WebOnt/`

For this reason, a more restricted form of concrete domain, known as *datatypes* [101], is often used in practice.

The *nominal* constructor allows us to use individual names also within concept descriptions: if a is an individual name, then $\{a\}$ is a concept, called nominal, which is interpreted by a singleton set. Using the individual Turing, we can describe all those computer scientists that have met Turing by $\mathsf{CScientist} \sqcap \exists \mathsf{hasMet}.\{\mathsf{Turing}\}$. The so-called "one-of" constructor extends the nominal constructor to a finite set of individual. In the presence of disjunction, it can, however, be expressed using nominals: $\{a_1, \ldots, a_n\}$ is equivalent to $\{a_1\} \sqcup \ldots \sqcup \{a_n\}$. The presence of nominals can have dramatic effects on the complexity of reasoning [160].

An additional comment on the naming of DLs is in order. Recall that the name given to a particular DL usually reflects its expressive power, with letters expressing the constructors provided. For expressive DLs, starting with the basic DL $\mathcal{AL}$ would lead to quite long names. For this reason, the letter $\mathcal{S}$ is often used as an abbreviation for the "basic" DL consisting of $\mathcal{ALC}$ extended with transitive roles (which in the $\mathcal{AL}$ naming scheme would be called $\mathcal{ALC}_{R^+}$).[2] The letter $\mathcal{H}$ represents subroles (role $\mathcal{H}$ierarchies), $\mathcal{O}$ represents nominals (n$\mathcal{O}$minals), $\mathcal{I}$ represents inverse roles ($\mathcal{I}$nverse), $\mathcal{N}$ represent number restrictions ($\mathcal{N}$umber), and $\mathcal{Q}$ represent qualified number restrictions ($\mathcal{Q}$ualified). The integration of a concrete domain/datatype is indicated by appending its name in parenthesis, but sometimes a "generic" $\mathbf{D}$ is used to express that some concrete domain/datatype has been integrated. The DL corresponding to the OWL DL ontology language includes all of these constructors and is therefore called $\mathcal{SHOIN}(\mathbf{D})$.

3.3 Relationships with Other Formalisms

In this section, we discuss the relationships between DLs and predicate logic, and between DLs and Modal Logic. This is intended for readers who are familiar with these logics; those not familiar with these logics might want to skip the following subsection(s), since we do not introduce modal or predicate logic here—we simply use standard terminology. Here, we only describe the relationship of the basic DL $\mathcal{ALC}$ and some of its extensions to these other logics (for a more detailed analysis, see [33] and Chapter 4 of [14]).

3.3.1 DLs and Predicate Logic

Most DLs can be seen as fragments of first-order predicate logic, although some provide operators such as transitive closure of roles or fixpoints that require second-order logic [33]. The main reason for using Description Logics rather than general first-order predicate logic when representing knowledge is that most DLs are actually *decidable* fragments of first-order predicate logic, i.e., there are effecitive procedures for deciding the inference problems introduced above.

Viewing role names as binary relations and concept names as unary relations, we define two translation functions, π_x and π_y, that inductively map $\mathcal{ALC}$-concepts into first order

[2] The use of $\mathcal{S}$ is motivated by the close connection between this DL and the modal logic $\mathbf{S4}$.

formulae with one free variable, x or y:

$$\begin{array}{rclrcl}
\pi_x(A) & = & A(x), & \pi_y(A) & = & A(y), \\
\pi_x(C \sqcap D) & = & \pi_x(C) \wedge \pi_x(D), & \pi_y(C \sqcap D) & = & \pi_y(C) \wedge \pi_y(D), \\
\pi_x(C \sqcup D) & = & \pi_x(C) \vee \pi_x(D), & \pi_y(C \sqcup D) & = & \pi_y(C) \vee \pi_y(D), \\
\pi_x(\exists r.C) & = & \exists y.r(x,y) \wedge \pi_y(C), & \pi_y(\exists r.C) & = & \exists x.r(y,x) \wedge \pi_x(C), \\
\pi_x(\forall r.C) & = & \forall y.r(x,y) \Rightarrow \pi_y(C), & \pi_y(\forall r.C) & = & \forall x.r(y,x) \Rightarrow \pi_x(C).
\end{array}$$

Given this, we can translate a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ as follows, where $\psi[x/a]$ denotes the formula obtained from ψ by replacing all free occurrences of x with a:

$$\begin{array}{rcl}
\pi(\mathcal{T}) & = & \bigwedge_{C \sqsubseteq D \in \mathcal{T}} \forall x.(\pi_x(C) \Rightarrow \pi_x(D)) \\
\pi(\mathcal{A}) & = & \bigwedge_{a:C \in \mathcal{A}} \pi_x(C)[x/a] \wedge \bigwedge_{(a,b):r \in \mathcal{A}} r(a,b).
\end{array}$$

This translation preserves the semantics: we can obviuosly view DL interpretations as first-order interpretations and vice versa, and it is easy to show that the translation preserves models. As an easy consequence, we have that reasoning in DLs corresponds to first-order inference:

Theorem 1 *Let $(\mathcal{T}, \mathcal{A})$ be an $\mathcal{ALC}$-knowledge base, C, D possibly complex $\mathcal{ALC}$-concepts, and a an individual name. Then*

1. $(\mathcal{T}, \mathcal{A})$ *is consistent* *iff* $\pi(\mathcal{T}) \wedge \pi(\mathcal{A})$ *is consistent,*
2. $(\mathcal{T}, \mathcal{A}) \models C \sqsubseteq D$ *iff* $(\pi(\mathcal{T}) \wedge \pi(\mathcal{A})) \Rightarrow (\pi(\{C \sqsubseteq D\}))$ *is valid,*
3. $(\mathcal{T}, \mathcal{A}) \models a : C$ *iff* $(\pi(\mathcal{T}) \wedge \pi(\mathcal{A})) \Rightarrow (\pi(\{a : C\}))$ *is valid.*

This translation not only provides an alternative way of defining the semantics of $\mathcal{ALC}$, but also tells us that all the introduced reasoning problems for $\mathcal{ALC}$ knowledge bases are decidable. In fact, the translation of a knowledge base uses only variables x and y, and thus yields a formula in the *two variable fragment of first-order logic*, which is known to be decidable in non-deterministic exponential time [79]. Alternatively, we can use the fact that this translation uses quantification only in a restricted way, and therefore yields a formula in the *guarded fragment* [2], which is known to be decidable in deterministic exponential time [78]. Thus, the exploration of the relationship between DLs and first-order logics even gives us upper complexity bounds "for free". However, for $\mathcal{ALC}$ and also many other DLs, the upper bounds obtained this way are not necessarily optimal, which justifies the development of dedicated reasoning procedures for DLs.

The translation of more expressive DLs may be straightforward, or more difficult, depending on the additional constructs. Inverse roles can be captured easily in both the guarded and the two variable fragment by simply swapping the variable places; e.g., $\pi_x(\exists R^-.C) = \exists y.R(y,x) \wedge \pi_y(C)$. Number restrictions can be captured using (in)equality or so-called *counting quantifiers*. It is known that the two-variable fragment with counting quantifiers is still decidable in non-deterministic exponential time [130]. Transitive roles, however, cannot be expressed with two variables only, and the three variable fragment is known to be undecidable. The guarded fragment, when restricted carefully to the so-called *action guarded fragment* [75], can still capture a variety of features such as number restrictions, inverse roles, and fixpoints, while remaining decidable in deterministic exponential time.

3.3.2 DLs and Modal Logic

Description Logics are closely related to Modal Logics, yet they have been developed independently. This close relationship was discovered relatively late [145], but has since then been exploited quite successfully to transfer complexity and decidability results as well as reasoning techniques [146, 57, 90, 3]. It is not hard to see that $\mathcal{ALC}$-concepts can be viewed as syntactic variants of formulae of the (multi) modal logic **K**: Kripke structures can easily be viewed as DL interpretations and, vice versa, DL interpretations as Kripke structures; we can then view concept names as propositional variables, and role names as modal parameters, and realize this correspondence through the rewriting $\leftrightsquigarrow$, which allows to translate $\mathcal{ALC}$-concepts into modal formulae and vice versa modal formulae into $\mathcal{ALC}$-concepts, as follows:

$\mathcal{ALC}$-concepts		Modal **K** formulae
A	$\leftrightsquigarrow$	a, for concepts names A and propositional variables a
$C \sqcap D$	$\leftrightsquigarrow$	$C \wedge D$,
$C \sqcup D$	$\leftrightsquigarrow$	$C \vee D$,
$\neg C$	$\leftrightsquigarrow$	$\neg C$,
$\forall r.C$	$\leftrightsquigarrow$	$[r]C$,
$\exists r.C$	$\leftrightsquigarrow$	$\langle r \rangle C$.

Let us use $\dot{C}$ for the modal formula obtained by rewriting the $\mathcal{ALC}$-concept C. The translation of DL knowledge bases is slightly more tricky: a TBox $\mathcal{T}$ is satisfied only in those structures where, for each $C \sqsubseteq D$, $\neg\dot{C} \vee \dot{D}$ holds *globally*, i.e., in each world of our Kripke structure (or equivalently, in each element of our interpretation domain). We can express this using the universal modality, that is, a special modal parameter U that is interpreted as the total relation in all Kripke structures. Before we discuss ABoxes, let us first state the properties of our correspondence so far.

Theorem 2 *Let $\mathcal{T}$ be an $\mathcal{ALC}$-TBox and E, F possibly complex $\mathcal{ALC}$-concepts. Then*

1. *F is satisfiable w.r.t. $\mathcal{T}$* *iff* *$\dot{F} \wedge \bigwedge_{C \sqsubseteq D \in \mathcal{T}} [U]\neg\dot{C} \vee \dot{D}$ is satisfiable,*
2. *$\mathcal{T} \models E \sqsubseteq F$* *iff* *$(\bigwedge_{C \sqsubseteq D \in \mathcal{T}} [U](\neg\dot{C} \vee \dot{D})) \wedge \dot{E} \sqcap \neg\dot{F}$ is unsatisfiable.*

Like TBoxes, ABoxes do not have a direct correspondence in modal logic, but they can be seen as a special case of a modal logic constructor, namely *nominals*. These are special propositional variables that hold in exactly one world; they are the basic ingredient of *hybrid logics* [4], and usually come with a special modality, the @-operator, that allows one to refer to the (only) world in which the nominal a holds. For example, $@_a\psi$ holds if, in the world where a holds, ψ holds as well. Hence an ABox assertion of the form $a : C$ corresponds to the modal formula $@_a\dot{C}$, and an ABox assertion $(a, b) : r$ corresponds to $@_a\langle r \rangle b$. In this latter formula, we see that nominals can act both as a parameter to the @ operator, like a, and as a propositional variables, like b. Please note that the usage of individual names in ABoxes corresponds to formulae where nominals are used in a rather restricted form only—some DLs, such as $\mathcal{SHOIN}$ or $\mathcal{SHOIQ}$, allow for a more general use of nominals, which is normally indicated by the letter $\mathcal{O}$ in a DL's name.

As in the case of first-order logic, some DL constructors have close relatives in modal logics and some do not. Number restrictions correspond to so-called *graded modalities*

[70], which in modal logic have found only limited attention until the connection with DLs was found. In some variants of propositional dynamic logic [71], a modal logic for reasoning about programs, we find *deterministic programs*, which correspond to (unqualified) number restrictions of the form $\leqslant 1R.\top$ [29]. Similarly, we find there *converse programs*, which correspond to *inverse roles*, and *regular expressions of programs*, which correspond to roles built using transitive-reflexive closure, union, and composition.

3.4 Tableau Based Reasoning Techniques

A variety of reasoning techniques can be used to solve the reasoning problems introduced in Section 3.2. These include resolution based approaches [102, 104], automata based approaches [49, 162], and structural approaches (for sub-Boolean DLs) [6]. The most widely used technique, however, is the tableau based approach first introduced by Schmidt-Schauß and Smolka [150]. In this section, we described this technique for the case of our basic DL $\mathcal{ALC}$.

3.4.1 A Tableau Algorithm for $\mathcal{ALC}$

We will concentrate on knowledge base consistency because, as we have seen in Section 3.2, this is a very general problem to which many others can be reduced. For example, given a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a concept C is subsumed by a concept D with respect to $\mathcal{K}$ ($\mathcal{K} \models C \sqsubseteq D$) iff $(\mathcal{T}, \mathcal{A} \cup \{x : (C \sqcap \neg D)\})$ is not consistent, where x is a new individual name (i.e., one that does not occur in $\mathcal{K}$). For $\mathcal{ALC}$ with a general TBox, i.e., one where the TBox is not restricted to contain only definitorial axioms (see Section 3.2), this problem is known to be EXPTIME-complete [145].

The tableau based decision procedure for the consistency of general $\mathcal{ALC}$ knowledge bases sketched below (and described in more detail in [12, 14]), runs in worst-case *non-deterministic* double exponential time.[3] However, according to the current state of the art, procedures such as this work well in practice, and are the basis for highly optimised implementations of DL systems such as FaCT [95], FaCT++ [161], Racer [81] and Pellet [152].

Given a knowledge base $(\mathcal{T}, \mathcal{A})$, we can assume without loss of generality that all of the concepts occurring in $\mathcal{T}$ and $\mathcal{A}$ are in *negation normal form* (NNF), i.e., that negation is applied only to concept names. An arbitrary $\mathcal{ALC}$ concept can be transformed to an equivalent one in NNF by pushing negations inwards using a combination of de Morgan's laws and the duality between existential and universal restrictions ($\neg\exists r.C \equiv \forall r.\neg C$ and $\neg\forall r.C \equiv \exists r.\neg C$). For example, the concept $\neg(\exists r.A \sqcap \forall s.B)$, where A, B are concept names, can be transformed to the equivalent NNF concept $(\forall r.\neg A) \sqcup (\exists s.\neg B)$. For a concept C, we will use $\dot{\neg}C$ to denote the NNF of $\neg C$.

The idea behind the algorithm is that it tries to prove the consistency of a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ by constructing (a representation of) a model of $\mathcal{K}$. It does this by starting from the concrete situation described in $\mathcal{A}$, and explicating additional constraints on the model that are implied by the concepts in $\mathcal{A}$ and the axioms in $\mathcal{T}$. Since $\mathcal{ALC}$ has

[3] This is due to the algorithm searching a tree of worst-case exponential depth. By re-using previously computed search results, a similar algorithm can be made to run in exponential time [66], but this introduces a considerable overhead which turns out to be not always useful in practice.

a so-called forest model property, we can assume that this model has the form of a set of (potentially infinite) trees, the root nodes of which can be arbitrarily interconnected. If we want to obtain a decision procedure, we can only construct finite trees representing the (potentially) infinite ones (assuming that a model exists at all); this can be done such that the finite representation can be *unravelled* into an infinite forest model $\mathcal{I}$ of $(\mathcal{T}, \mathcal{A})$.

In order to construct such a finite representation, the algorithm works on a data structure called a *completion forest*. This consists of a labelled directed graph, each node of which is the root of a *completion tree*. Each node x in the completion forest (which is either a root node or a node in a completion tree) is labelled with a set of concepts $\mathcal{L}(x)$, and each edge $\langle x, y\rangle$ (which is either one between root nodes or one inside a completion tree) is labelled with a set of role names $\mathcal{L}(\langle x, y\rangle)$. If $\langle x, y\rangle$ is an edge in the completion forest, then we say that x is a predecessor of y (and that y is a successor of x); in case $\langle x, y\rangle$ is labelled with a set containing the role name r, then we say that x is an r-predecessor of y (and that y is an r-successor of x).

When started with a knowledge base $(\mathcal{T}, \mathcal{A})$, the completion forest $\mathcal{F}_\mathcal{A}$ is initialised such that it contains a root node x_a, with $\mathcal{L}(x_a) = \{C \mid a : C \in \mathcal{A}\}$, for each individual name a occurring in $\mathcal{A}$, and an edge $\langle x_a, x_b\rangle$, with $\mathcal{L}(\langle x_a, x_b\rangle) = \{r \mid (a, b) : r \in \mathcal{A}\}$, for each pair (a, b) of individual names for which the set $\{r \mid (a, b) : r \in \mathcal{A}\}$ is non-empty.

The algorithm then applies so-called *expansion rules*, which syntactically decompose the concepts in node labels, either inferring new constraints for a given node, or extending the tree according to these constraints (see Figure 3.1). For example, if $C_1 \sqcap C_2 \in \mathcal{L}(x)$, and either $C_1 \notin \mathcal{L}(x)$ or $C_2 \notin \mathcal{L}(x)$, then the $\sqcap$-rule adds both C_1 and C_2 to $\mathcal{L}(x)$; if $\exists r.C \in \mathcal{L}(x)$, and x does not yet have an r-successor with C in its label, then the $\exists$-rule generates a new r-successor node y of x with $\mathcal{L}(y) = \{C\}$. Note that the $\sqcup$-rule is different from the other rules in that it is *non-deterministic*: if $C_1 \sqcup C_2 \in \mathcal{L}(x)$ and neither $C_1 \in \mathcal{L}(x)$ nor $C_2 \in \mathcal{L}(x)$, then it adds *either* C_1 *or* C_2 to $\mathcal{L}(x)$. In practice this is the main source of complexity in tableau algorithms, because it may be necessary to explore all possible choices of rule applications.

The algorithm stops if it encounters a *clash*: a completion forest in which $\{A, \neg A\} \subseteq \mathcal{L}(x)$ for some node x and some concept name A. In this case, the completion forest contains an obvious inconsistency, and thus does not represent a model. If the algorithm stops without having encountered a clash, then the obtained completion forest yields a finite representation of a forest model, and the algorithm answers "$(\mathcal{T}, \mathcal{A})$ is consistent"; if none of the possible non-deterministic choices of the $\sqcup$-rule leads to such a representation of a forest model, i.e., all of them lead to a clash, then the algorithm answers "$(\mathcal{T}, \mathcal{A})$ is inconsistent".

Please note that we have two different kinds of non-determinism in this algorithm. The non-deterministic choice between the two disjuncts in the $\sqcup$-rule is "don't know" non-deterministic, i.e., if the first choice leads to a clash, then the second one must be explored. In contrast, the choice of which rule to apply next to a given completion forest is "don't care" non-deterministic, i.e., one can choose an arbitrary applicable rule without the need to backtrack and explore alternative choices.

It remains to explain the meaning of "blocked" in the formulation of the expansion rules. Without the $\sqsubseteq$-rule (i.e., in case the TBox is empty), the tableau algorithm for $\mathcal{ALC}$ would always terminate, even without blocking. In order to guarantee termination of the expansion process even in the presence of GCIs, the algorithm uses a technique

$\sqcap$-rule: if 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not blocked, and
2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$

$\sqcup$-rule: if 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, and
2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

$\exists$-rule: if 1. $\exists r.C \in \mathcal{L}(x)$, x is not blocked, and
2. x has no r-successor y with $C \in \mathcal{L}(y)$,
then create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{r\}$ and $\mathcal{L}(y) = \{C\}$

$\forall$-rule: if 1. $\forall r.C \in \mathcal{L}(x)$, x is not blocked, and
2. there is an r-successor y of x with $C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

$\sqsubseteq$-rule: if 1. $C_1 \sqsubseteq C_2 \in \mathcal{T}$, x is not blocked, and
2. $C_2 \sqcup \dot{\neg} C_1 \notin \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2 \sqcup \dot{\neg} C_1\}$

Figure 3.1: The tableau expansion rules for $\mathcal{ALC}$

called *blocking*.[4] Blocking prevents application of expansion rules when the construction becomes repetitive; i.e., when it is obvious that the sub-tree rooted in some node x will be "similar" to the sub-tree rooted in some predecessor y of x. To be more precise, we say that a node y is an *ancestor* of a node x if they both belong to the same completion tree and either y is a predecessor of x, or there exists a predecessor z of x such that y is an ancestor of z. A node x is *blocked* if there is an ancestor y of x such that $\mathcal{L}(x) \subseteq \mathcal{L}(y)$ (in this case we say that y blocks x), or if there is an ancestor z of x such that z is blocked; if a node x is blocked and none of its ancestors is blocked, then we say that x is *directly blocked*. When the algorithm stops with a clash free completion forest, a branch that contains a directly blocked node x represents an infinite branch in the corresponding model having a regular structure that corresponds to an infinite repetition (or "unravelling") of the section of the graph between x and the node that blocks it (see Section 3.6.1).

Theorem 3 *The above algorithm is a decision procedure for the consistency of $\mathcal{ALC}$ knowledge bases.*

A complete proof of this theorem is beyond the scope of this chapter, and we will only sketch the idea behind the proof: the interested reader can refer to [12, 14] for more details. Firstly, it is easy to see that the algorithm terminates: expansion rule applications always extend node labels or add new nodes, and we can fix an upper bound on the size of node labels (they can only contain concepts that are derivable from the syntactic decomposition of concepts occurring in the input KB), on the fan-out of trees in the completion forest (a node can have at most one successor for each existential restriction occurring in its label),

[4] In description logics, blocking was first employed in [8] in the context of an algorithm that can handle the transitive closure of roles, and was improved on in [13, 46, 12, 92].

and on the length of their branches (due to blocking). Secondly, soundness follows from the fact that we can transform a fully expanded and clash free completion forest into a model of the input KB by "throwing away" all blocked nodes and "bending" each edge from a non-blocked into a blocked node to the node it is blocked by.[5] Finally, completeness follows from the fact that, given a model of the input KB, we could use it to guide applications of the $\sqcup$-rule so as to produce a fully expanded and clash free completion forest.

The procedure described above can be simplified if the TBox is definitorial, i.e., if it contains only unique and acyclic definitions (see Section 3.2). In this case, reasoning with a knowledge base can be reduced to the problem of reasoning with an ABox only (equivalently, a knowledge base with an empty TBox) by *unfolding* the concepts used in ABox axioms [126]: given a KB $(\mathcal{T}, \mathcal{A})$, where the definition $A \equiv C$ occurs in $\mathcal{T}$, all occurrences of A in $\mathcal{A}$ can be replaced with C. Repeated application of this procedure can be used to eliminate from $\mathcal{A}$ all those concept names for which there is a definition in $\mathcal{T}$. As mentioned above, when the TBox is empty the $\sqsubseteq$-rule is no longer required and blocking can be dispensed with. This is because the other rules only introduce concepts that are smaller than the concept triggering the rule application; we will come back to this in Section 3.5.1.

It is easy to see that the above *static* unfolding procedure can lead to an exponential increase in the size of the ABox [126]. In general, this cannot be avoided since there are DLs where reasoning w.r.t. definitorial TBoxes is harder than without TBoxes [127, 114]. For $\mathcal{ALC}$, however, we can avoid an increase in the complexity of the algorithm by unfolding definitions not a priori, but only as required by the progress of the algorithm. This so called *lazy* unfolding [15, 95, 114] is achieved by substituting the $\sqsubseteq$-rule by the following two $\equiv_i$-rules:

$\equiv_1$-rule: if 1. $A \equiv C \in \mathcal{T}$, $A \in \mathcal{L}(x)$,	$\equiv_2$-rule: if 1. $A \equiv C \in \mathcal{T}$, $\neg A \in \mathcal{L}(x)$,
2. and $C \notin \mathcal{L}(x)$,	2. and $\doteq C \notin \mathcal{L}(x)$,
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$	then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{\doteq C\}$

As in the case of static unfolding, blocking is not required: the acyclicity condition on the TBox means that if a concept C is added to $\mathcal{L}(x)$ as a result of an application of one of the $\equiv_i$-rules to the concept A or $\neg A$ and axiom $A \equiv C$, then further unfolding of C cannot lead to the introduction of another occurrence of A in the sub-tree below x.

The tableau algorithm can also be extended to deal with a wide range of other DLs, including those supporting, e.g., (qualified) number restrictions, inverse roles, transitive roles, subroles, concrete domains and nominals. Extending the algorithm to deal with such features is mainly a matter of adding expansion rules to deal with the new constructors (e.g., number restrictions), adding new clash conditions (e.g., to deal with obviously unsatisfiable number restrictions), and using a more sophisticated blocking condition in order to guarantee both termination and soundness when using the extended rule set.

3.4.2 Implementation and Optimisation Techniques

Although reasoning in $\mathcal{ALC}$ (w.r.t. an arbitrary KB) is of a relatively high complexity (EXPTIME-complete), the pathological cases that lead to such high *worst case* complex-

[5] For $\mathcal{ALC}$, we can always construct a finite cyclical model in this way; for more expressive DLs, we may need different blocking conditions, and we may need to unravel such cycles in order to construct an infinite model.

ity are rather artificial, and rarely occur in practice [127, 86, 155, 95]. Even in realistic applications, however, problems can occur that are much too hard to be solved by naive implementations of theoretical algorithms such as the one sketched in Section 3.4.1. Modern DL systems, therefore, include a wide range of optimisation techniques, the use of which has been shown to improve *typical case* performance by several orders of magnitude [96]. These systems exhibit good typical case performance, and work well in realistic applications [15, 44, 95, 81, 133].

A detailed description of optimisation techniques is beyond the scope of this chapter, and the interested reader is referred to Chapter 8 of [14] for further information. It will, however, be interesting to sketch a couple of the key techniques: absorption and dependency directed backtracking.

Absorption

Whereas definitorial TBoxes can be dealt with efficiently by using lazy unfolding (see Section 3.4.1 above), more general axioms are not amenable to this optimization technique. In particular, GCIs $C \sqsubseteq D$, where C is non-atomic, must be dealt with by explicitly making every individual in the model an instance of $D \sqcup \doteq C$ (see Figure 3.1). Large numbers of such GCIs result in a very high degree of non-determinism due to the introduction of these disjunctions, and thus to catastrophic performance degradation [95].

Absorption is a rewriting technique that tries to reduce the number of GCIs in the TBox by absorbing them into axioms of the form $A \sqsubseteq C$, where A is a concept name. The basic idea is that an axiom of the form $A \sqcap D \sqsubseteq D'$ can be rewritten as $A \sqsubseteq D' \sqcup \neg D$ and absorbed into an existing $A \sqsubseteq C$ axiom to give $A \sqsubseteq C \sqcap (D' \sqcup \neg D)$ [93]. Although the disjunction is still present, lazy unfolding applied to this axiom (where only the $\equiv_1$ rule needs to be applied) ensures that the disjunction is only introduced for individuals that are already known to be instances of A.

Dependency Directed Backtracking

Inherent unsatisfiability concealed in sub-descriptions can lead to large amounts of unproductive backtracking search known as thrashing. For example, expanding the description $(C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \sqcap \exists R.(A \sqcap B) \sqcap \forall R.\neg A$ could lead to the fruitless exploration of 2^n possible expansions of $(C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n)$ before the inherent unsatisfiability of $\exists R.(A \sqcap B) \sqcap \forall R.\neg A$ is discovered. This problem is addressed by adapting a form of dependency directed backtracking called *backjumping*, which has been used in solving constraint satisfiability problems [27].

Backjumping works by labelling concepts with a dependency set indicating the non-deterministic expansion choices on which they depend. When a clash is discovered, the dependency sets of the clashing concepts can be used to identify the most recent non-deterministic expansion where an alternative choice might alleviate the cause of the clash. The algorithm can then jump back over intervening non-deterministic expansions *without* exploring any alternative choices. Similar techniques have been used in first-order theorem provers, e.g., the "proof condensation" technique employed in the HARP theorem prover [128].

3.5 Complexity

In this section, we discuss the computational complexity of some of the reasoning problems we have specified. Since introducing complexity classes and other notions of computational complexity would go beyond the scope of this chapter, we expect the reader to be familiar with the complexity classes PSpace and ExpTime, the notions of membership in and hardness for such a class, and what it means for a problem to be undecidable. Those readers who want to learn more about computational complexity are referred to [131], or any other textbook covering computational complexity.

3.5.1 $\mathcal{ALC}$ ABox Consistency is PSpace-complete

In Section 3.4.1, we have seen a tableau based algorithm that decides the consistency of $\mathcal{ALC}$ ABoxes w.r.t. TBoxes. Here, we will first consider ABoxes only and explain how this algorithm can be implemented to use polynomial space only; that is, we will show that consistency of $\mathcal{ALC}$ ABoxes is *in* PSpace. Then we will show that we cannot do better; that is, that consistency of $\mathcal{ALC}$ ABoxes is PSpace-*hard*.

For these considerations, we need to agree how to measure the size of the input. For $\mathcal{A}$ an ABox $\mathcal{A}$, intuitively its size $|\mathcal{A}|$ is the length required to write $\mathcal{A}$ down, where we assume that the length required to write concept and role names is 1. Formally, we define the size of ABoxes as follows:

$$\begin{aligned}
|\mathcal{A}| &= \sum_{a:C\in\mathcal{A}} (|C|+1) + \sum_{(a,b):r\in\mathcal{A}} 3 \\
|A| &= 1 \text{ for a concept name } A \text{ (including } \top, \bot) \\
|\neg D| &= |D| + 1 \\
|D_1 \sqcap D_2| &= |D_1 \sqcup D_2| = |D_1| + |D_2| + 1 \\
|\exists R.D| &= |\forall R.D| = |D| + 2
\end{aligned}$$

Next, let us look again at the tableau algorithm. First, note that, in the absence of a TBox, neither the $\sqsubseteq$-rule not the $\equiv_i$-rules is applicable. Second, observe that the tableau algorithm builds a completion forest in a monotonic way; that is, all expansion rules either add concepts to node labels or new nodes to the forest, but never remove anything. The forest it constructs consists of two parts: for each individual name in $\mathcal{A}$, the forest contains a root node, which we will call an *old node* in the following. The edges between old nodes all stem from role assertions in $\mathcal{A}$, and thus may occur without restrictions. Other nodes (i.e., the notes in the completion tress that are not root nodes) are generated by the $\exists$-rule, and we call them *new nodes*; we call the other rules *augmenting* rules, because they only augment the labels of existing nodes. In contrast to edges between old nodes, edges between new nodes are of a particular shape: each new node is found in a completion tree with an old node at its root.

Let us consider the node labels. Initially, for an old node x_a, $\mathcal{L}(x_a)$ contains the concepts C from the assertions $a : C \in \mathcal{A}$. Other concepts are added by the expansion rules, and we observe that these expansion rules only add subconcepts of the concepts occurring in $\mathcal{A}$. Since there are at most $|\mathcal{A}|$ such subconcepts, each node label can be stored in space polynomial in $|\mathcal{A}|$. Moreover, for each concept D in the label of a new node x, the (unique) predecessor of x contains a larger concept. Hence the maximum size of concepts in node

labels strictly decreases along a path of new nodes, and thus the depth of each completion tree in our completion graph is bounded by $\max\{|C| \mid a : C \in \mathcal{A}\}$.

Finally, we note that the expansion rules can be applied in an arbitrary order: the correctness proof for the algorithm does not rely on a specific application order. Hence we can use the following order: first, all augmenting rules are exhaustively applied to old nodes. Next, we treat each old node in turn, and build the tree rooted at it in a depth first manner. That is, for an old node x_a, we deal in turn with each existential restriction $\exists r.C \in \mathcal{L}(x_a)$: we apply the $\exists$-rule in order to generate an r-successor x_0 with $\mathcal{L}(x_0) = \{C\}$, apply the $\forall$-rule exhaustively to this r-successor of x_a (which may add further concepts to $\mathcal{L}(x_0)$), and recursively apply the same procedure to x_0, i.e., exhaustively apply the augmenting rules, and then deal with the existential restrictions one at a time. As usual, the algorithm stops if a clash occurs; otherwise, when all of a new node's existential restrictions have been dealt with, we can delete it, including its label, and re-use the space. Using this technique, we can investigate the whole tree rooted at our old node x_a while only keeping a single branch in memory at any time. This branch is of length linear in $|\mathcal{A}|$, and can thus be stored with all its labels in size polynomial in $|\mathcal{A}|$. Continuing the investigation of all trees in the same manner, our algorithm only requires space polynomial in $|\mathcal{A}|$. This technique is called the *trace* technique since it only "traces" the tree-shaped part of a completion tree [150].

To show that we cannot do better, we will prove that consistency of $\mathcal{ALC}$ ABoxes is PSpace-hard, even for ABoxes that consist of a single assertion $\{a : C\}$. This proof is by a reduction of the validity problem for *quantified Boolean formulae*, which is known to be PSpace-hard [156]. A *quantified Boolean formula* (QBF for short) Φ is of the form

$$Q_1 p_1 . Q_2 p_2 . \ldots Q_n p_n . \varphi$$

for $Q_i \in \{\forall, \exists\}$ and φ a Boolean formula over $p_1, \ldots, p_n$. The validity of QBFs is defined inductively:

$\exists p.\Phi$	is valid if	$\Phi[p/t]$ or $\Phi[p/f]$ is valid
$\forall p.\Phi$	is valid if	$\Phi[p/t]$ and $\Phi[p/f]$ are valid

For example, $\forall p.\exists q.(p \vee q)$ is valid, whereas $\forall p.\forall q.\exists r.((p \vee r) \Rightarrow q)$ is not valid.

Since validity of QBFs is PSpace-hard, it remains to show that, for a given QBF Φ, we can construct in polynomial time an $\mathcal{ALC}$-concept C_Φ such that Φ is valid iff $\{a : C_\Phi\}$ is consistent. As an immediate consequence, consistency of $\mathcal{ALC}$ ABoxes and satisfiability of $\mathcal{ALC}$ concepts are PSpace-hard.

The idea underlying our reduction is to build, for a QBF as above, a concept C_Φ such that each instance x_0 of C_Φ is the root of a tree of depth n such that, for each $1 \leq i \leq n$, we have the following:

1. if $Q_i = \exists$, each $\underbrace{r \ldots r}_{i-1 \text{ times}}$-successor of x_0 has one r-successor, which can be in p_i or in $\neg p_i$, and

2. if $Q_i = \forall$, each $\underbrace{r \ldots r}_{i-1 \text{ times}}$-successor of x_0 has two r-successors one in p_i, one in $\neg p_i$.

To this end, for a QBF $\Phi = Q_1 p_1 . Q_2 p_2 . \ldots Q_n p_n . \varphi$, we define C_Φ as follows, where $\hat{\varphi}$ is the DL counterpart of φ obtained by replacing $\wedge$ with $\sqcap$ and $\vee$ with $\sqcup$ in φ:

$$
\begin{aligned}
C_\Phi &:= L_1 \sqcap \forall r.(L_2 \sqcap \forall r.(L_3 \sqcap \ldots \forall r.(L_n \sqcap \hat{\varphi}))\ldots)), \text{ where} \\
L_i &:= D_i \sqcap \begin{cases} \exists r.\top & \text{if } Q_i = \exists \\ \exists r.p_i \sqcap \exists r.\neg p_i & \text{if } Q_i = \forall \end{cases} \\
D_i &:= \bigsqcap_{j<i} (p_j \Rightarrow \forall r.p_j) \sqcap (\neg p_j \Rightarrow \forall r.\neg p_j)
\end{aligned}
$$

Through this definition we ensure that, if $x_0 \in C_\Phi^\mathcal{I}$ and there is a path $(x_0, x_1) \in r^\mathcal{I}, \ldots, (x_{i-1}, x_i) \in r^\mathcal{I}$, then $x_i \in L_i^\mathcal{I}$, and L_i is responsible for the branching pattern described above. The concepts D_i ensure that, if some x_j is (is not) an instance of p_j for $j < i$, then so is (neither is) x_{j+1}. These observations, together with the fact that x_n must be an instance of $\hat{\varphi}$, ensure that Φ is valid iff $\{a : C_\Phi\}$ is consistent.

Theorem 4 *Satisfiability and subsumption of $\mathcal{ALC}$ concepts and consistency of $\mathcal{ALC}$ ABoxes are PSpace-complete problems.*

3.5.2 Adding General TBoxes Results in ExpTime-hardness

We will see in Section 3.6.1 that satisfiability of $\mathcal{ALC}$ concepts w.r.t. (general) TBoxes can be decided in exponential time, i.e., that this problem is in ExpTime. Again, one can show that we cannot do better, i.e., that this problem is also ExpTime-hard. Unfortunately, this proof goes beyond the scope of this chapter since, to the best of our knowledge, all proofs require the introduction of some "complexity theory machinery": one possible proof is by adapting the proof of ExpTime-hardness of propositional dynamic logic (PDL) in [71]. This proof uses a polynomial reduction of the word problem for *polynomially space-bounded, alternating Turing machines* to the satisfiability of PDL formulae. When translated into its DL counterpart, the reduction formula of this proof is of the form $C \sqcap \forall r^*.D$, where C and D are $\mathcal{ALC}$ concepts and r^* is the transitive-reflexive closure of r, i.e., this concept involves a constructor not available in $\mathcal{ALC}$. It is not hard to see, however, that $C \sqcap \forall r^*.D$ is satisfiable iff C is satisfiable w.r.t. the TBox $\{\top \sqsubseteq D\}$. This is the case since r is the only role name occurring in C and D. For more information on the relationship between TBoxes and PDL see, e.g., [145, 57] or Chapter 4 of [14].

It is worth noting that, for *definitorial* TBoxes and $\mathcal{ALC}$, this blow-up of the complexity from PSpace to ExpTime does not occur [114]. Yet, we will see in Section 3.6.2 that there are DLs where even the presence of definitorial TBoxes can lead to an increased complexity.

3.5.3 The Effect of Other Constructors

In Section 3.2.3 we have seen various extensions of $\mathcal{ALC}$, and we will now briefly describe the influence they have on the computational complexity.

In general, number restrictions are "harmless": with only one exception, which we will come back to later, even qualified number restrictions can be added to a DL without

increasing its complexity. For example, concept satisfiability in $\mathcal{ALC}$Q is still in PSpace [160], and consistency of of general $\mathcal{ALC}$Q knowledge bases is still in ExpTime [56, 160].

Transitive roles are mostly harmless: all DLs between $\mathcal{ALC}$ and $\mathcal{ALCQIO}$ can be extended with transitive roles without increasing their computational complexity [143, 160]. One "dangerous" interaction we are aware of is with role hierarchies: concept satisfiability of $\mathcal{ALC}$ with transitive roles and role hierarchies is ExpTime-hard, whereas concept satisfiability in $\mathcal{ALC}$ with either transitive roles or role hierarchies is in PSpace [143]. The increase in complexity is due to the fact that transitive roles and role hierarchies can be used to internalise TBoxes [145]: given a TBox $\mathcal{T}$ and an $\mathcal{ALC}$ concept E that use role names $r_1, \ldots, r_n$, we have that E is satisfiable w.r.t. $\mathcal{T}$ if and only if the concept

$$\exists r.E \sqcap \forall r. \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} (\neg C \sqcup D)$$

is satisfiable w.r.t. $\{r_1 \sqsubseteq r, \ldots, r_n \sqsubseteq r\}$, where r is a new, transitive role. The first conjunct ensures that the extention of E is indeed non-empty, and the second conjunct ensures that every element in a (connected) model satisfies each GCI in $\mathcal{T}$. Thus, in $\mathcal{ALC}$ with transitive roles and role hierarchies, we can polynomially reduce reasoning w.r.t. a TBox to pure concept reasoning, and hence pure concept reasoning is already ExpTime-hard. In the additional presence of number restrictions, we need to take special care not to use super-roles of transitive roles inside number restrictions since this leads to undecidability [92]. As a consequence, expressive DLs such as $\mathcal{SHIQ}$ allow only so-called *simple* roles to be used in number restrictions.

Nominals and inverse roles are also mostly harmless: concept satisfiability in $\mathcal{ALCQO}$ and $\mathcal{ALCI}$ with transitive roles is still in PSpace [92, 7], but concept satisfiability of $\mathcal{ALCIO}$ is ExpTime-hard [4]. This increase in complexity is again due to the fact that, with inverse roles and nominals, we can internalise TBoxes. Intuitively, we use a nominal as a "spy point", i.e., an individual that has all other elements of a (connected) model as t-successors, and we use inverse roles to ensure this spy-point behaviour. More precisely, a concept E is satisfiable w.r.t. a TBox $\mathcal{T}$ if and only if the following concept is satisfiable, where o is a nominal, R is the set of roles r occurring in $\mathcal{T}$ or E and their inverses r^-, and t is a role that is not in R:

$$o \sqcap (\exists t.E) \sqcap (\forall t. \bigsqcap_{r \in R} \forall r.\exists t^-.o) \sqcap \forall t. \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} (\neg C \sqcup D).$$

The third conjunct ensures that o indeed "sees" all elements in a connected model, i.e., if x_o is an instance of the above concept in a connected model $\mathcal{I}$ and there is an element $y \in \Delta^{\mathcal{I}}$, then $(x_o, y) \in t^{\mathcal{I}}$

Finally, we would like to point out that nominals, inverse roles, and number restrictions together have a dramatic influence on complexity: satisfiability of $\mathcal{ALCQIO}$ concepts is NExpTime-hard [160], even though satisfiability of $\mathcal{ALCQI}$, $\mathcal{ALCIO}$, and $\mathcal{ALCOQ}$ concepts w.r.t. TBoxes is in ExpTime [56, 144, 7].

3.6 Other Reasoning Techniques

Although the tableau based approach is currently the most widely used technique for reasoning in DLs, other approaches have been developed as well. In general, a reasoning algorithm can be used in an implementation, or to prove a decidability or computational

complexity result. Certain approaches may (for a given logic) be better suited for the former task, whereas others may be better suited for the latter—and it is sometimes hard to find one that is well-suited for both. Examples of other approaches are the automata based approach, the structural subsumption approach, the resolution based approach, and translation based approaches. For certain logics and tasks, other approaches turn out to be superior to the tableau based approach. For example, it is not clear how the polynomiality result for subsumption in $\mathcal{EL}$ with GCIs [42, 6], which uses a structural subsumption algorithm, could be obtained with the help of a tableau based algorithm. Similarly, the automata based approach can be used to show that satisfiability and subsumption of $\mathcal{ALC}$ concepts w.r.t. TBoxes can be decided within exponential time [49, 117, 116, 160],[6] whereas this is very hard to prove using a tableau based approach [66]. Resolution based approaches [103, 5, 104, 107], which use the translation of DLs into first-order predicate logic, may have the advantage that they simultaneously yield a decision procedure for a certain decidable DL, and a semidecision procedure for a more expressive logic (such as OWL Full or first-order predicate logic). Moreover, some of them are worst-case optimal [104], and others can be implemented through appropriate parameterization of existing first-order theorem provers [103]. Finally, the translation of very expressive DLs into propositional dynamic logic or the propositional mu-calculus [57, 58, 56, 59] allows one to transfer known decidability and complexity results for these logics to very expressive DLs. It is not clear how these results could be obtained with the help of the tableau based approach.

In this section, we restrict our attention to the automata based approach for $\mathcal{ALC}$ with GCIs, and to structural subsumption algorithms for the sub-Boolean DLs[7] $\mathcal{EL}$ and $\mathcal{FL}_0$.

3.6.1 The Automata Based Approach

In this subsection, we restrict our attention to concept satisfiability, possibly w.r.t. (general) TBoxes. This is not a severe restriction since most of the other interesting inference problem can be reduced to satisfiability.[8] There are various instances of the automata based approach, which differ not only w.r.t. the DL under consideration, but also w.r.t. the employed automaton model. However, in principle all these instances have the following general ideas in common:

- First, one shows that the DL in question has the *tree model property*.
- Second, one devises a translation from pairs $C, \mathcal{T}$, where C is a concept and $\mathcal{T}$ is a TBox, into an appropriate *tree automata* $\mathcal{A}_{C,\mathcal{T}}$ such that $\mathcal{A}_{C,\mathcal{T}}$ accepts exactly the tree models of C w.r.t. $\mathcal{T}$.
- Third, one applies the *emptiness test* for the employed automaton model to $\mathcal{A}_{C,\mathcal{T}}$ to test whether C has a (tree) model w.r.t. $\mathcal{T}$.

The complexity of the satisfiability algorithm obtained this way depends on the complexity of the translation and the complexity of the emptiness tests. The latter complexity in turn depends on which automaton model is employed.

[6] The cited papers actually use automata based approaches to show EXPTIME results for *extensions* of $\mathcal{ALC}$.

[7] Sub-Boolean DLs are DLs that are not equipped with all Boolean operators.

[8] Using the so-called pre-completion technique [88], this is also the case for inference problems involving ABoxes.

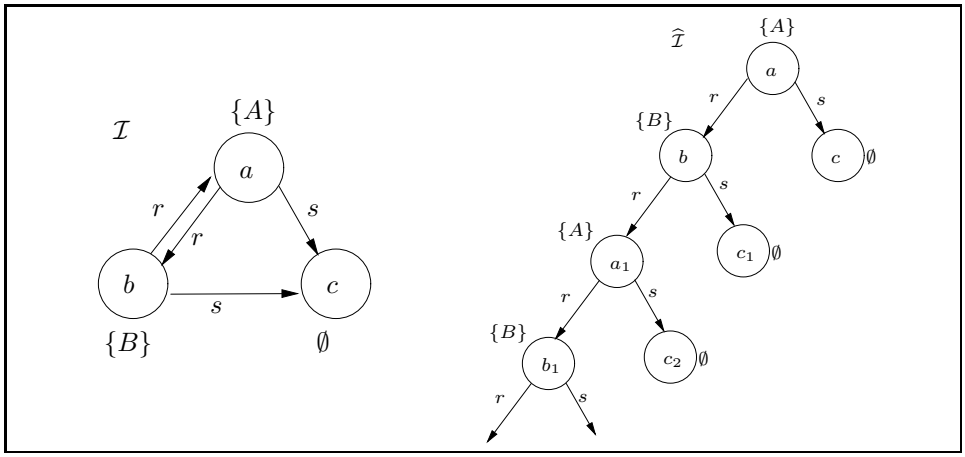

Figure 3.2: Unraveling of a model into a tree-shaped model.

Below, we will use a simple form of non-deterministic automata working on infinite trees of fixed arity, so-called *looping automata* [163]. In this case, the translation is exponential, but the emptiness test is polynomial (in the size of the already exponentially large automaton obtained through the translation). Thus, the whole algorithm runs in deterministic exponential time. Alternatively, one could use alternating tree automata [125], where a polynomial translation is possible, but the emptiness test is exponential.

Instead of considering automata working on trees of fixed arity, one could also consider so-called amorphous tree automata [31, 105], which can deal with arbitrary branching. This simplifies defining the translation, but uses a slightly more complicated automaton model. For some very expressive description logics (e.g., ones that allow for transitive closure of roles [8]), the simple looping automata introduced below are not sufficient since one needs additional acceptance conditions such as the Büchi condition [159] (which requires the occurrence of infinitely many final states in every path).

The Tree Model Property

The first step towards showing that satisfiability in $\mathcal{ALC}$ w.r.t. general TBoxes can be decided with the automata based approach is to establish the tree model property, i.e., to show that any $\mathcal{ALC}$-concept C satisfiable w.r.t. an $\mathcal{ALC}$-TBox $\mathcal{T}$ has a tree-shaped model. Note that this model may, in general, be infinite. One way of seeing this is to consider the tableau algorithm introduced in Section 3.4.1, applied to the knowledge base $(\mathcal{T}, \{a : C\})$, and just dispose of blocking. Possibly infinite runs of the algorithm then generate tree-shaped models. However, one can also show the tree model property of $\mathcal{ALC}$ by using the well-known unraveling technique [32], in which an arbitrary model of C w.r.t. $\mathcal{T}$ is unraveled into a bisimilar tree-shaped interpretation. Invariance of $\mathcal{ALC}$ under bisimulation [110] (which it inherits from its syntactic variant multi modal $\mathbf{K}_{(\mathbf{m})}$) then implies that the tree shaped interpretation obtained by unraveling is also a model of C w.r.t. $\mathcal{T}$.

Instead of defining unraveling in detail, we just give an example in Fig. 3.2, and refer the reader to [32] for formal definitions and proofs. The graph on the left-hand side of

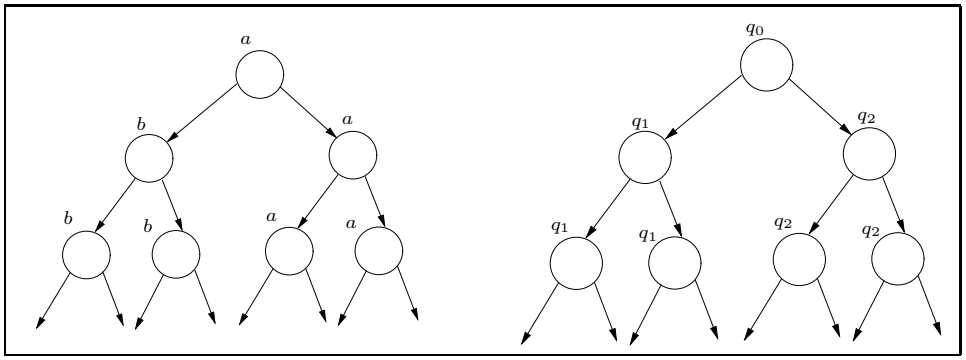

Figure 3.3: A tree and a run on it.

Fig. 3.2 describes an interpretation $\mathcal{I}$: the nodes of the graph are the elements of $\Delta^{\mathcal{I}}$, the node labels express to which concept names the corresponding element belongs, and the labelled edges of the graph express the role relationships. For example, $a \in \Delta^{\mathcal{I}}$ belongs to $A^{\mathcal{I}}$, but not to $B^{\mathcal{I}}$, and it has r-successor b and s-successor c. It is easy to check that $\mathcal{I}$ is a model of the concept A w.r.t. the TBox

$$\mathcal{T} := \{A \sqsubseteq \exists r.B,\ \ B \sqsubseteq \exists r.A,\ \ A \sqcup B \sqsubseteq \exists s.\top\}.$$

The graph on the right-hand side of Fig. 3.2 describes (a finite part of) the corresponding unraveled model, where a was used as the start node for the unraveling. Basically, one considers all paths starting with a in the original model, but whenever one would re-enter a node one makes a copy of it. Like $\mathcal{I}$, the corresponding unraveled interpretation $\widehat{\mathcal{I}}$ is a model of $\mathcal{T}$ and it satisfies $a \in A^{\widehat{\mathcal{I}}}$.

Looping Tree Automata

As mentioned above, we consider automata working on *infinite trees* of some fixed arity k. To be more precise, the nodes of the trees are labelled by elements from some finite alphabet Σ, whereas the edges are unlabelled, but ordered, i.e., there is a first, second, to kth successor for each node. Such trees, which we call k-ary Σ-trees, can formally be represented as mappings $T : \{0, \ldots, k-1\}^* \to \Sigma$. Thus, nodes are represented as words over $\{0, \ldots, k-1\}$, the root is the word ε, and a node u has exactly k successor nodes $u0, \ldots, u(k-1)$, and its label is $T(u)$. For example, the binary tree that has root label a, whose left subtree contains only nodes labelled by b, and whose right subtree has only nodes labelled by a (see the left-hand side of Fig. 3.3) is formally represented as the mapping

$$T : \{0,1\}^* \to \{a, b\} \text{ with } T(u) = \begin{cases} b & \text{if } u \text{ starts with } 0 \\ a & \text{otherwise} \end{cases}$$

A *looping automaton* working on k-ary Σ-trees is of the form $\mathcal{A} = (Q, \Sigma, I, \Delta)$, where

- Q is a finite set of states and $I \subseteq Q$ is the set of initial states;
- Σ is a finite alphabet;

- $\Delta \subseteq Q \times \Sigma \times Q^k$ is the transition relation.

We will usually write tuples $(q, a, q_1, \ldots, q_k) \in \Delta$ in the form $(q, a) \to (q_1, \ldots, q_k)$.

A *run* of $\mathcal{A} = (Q, \Sigma, I, \Delta)$ on the tree $T : \{0, \ldots, k-1\}^* \to \Sigma$ is a k-ary Q-tree $R : \{0, \ldots, k-1\}^* \to Q$ such that $(R(u), T(u)) \to (R(u0), \ldots, R(u(k-1))) \in \Delta$ for all $u \in \{0, \ldots, k-1\}^*$. This run is called *accepting* if $T(\varepsilon) \in I$.

For example, consider the automaton $\mathcal{A} = (Q, \Sigma, I, \Delta)$, where

- $Q = \{q_0, q_1, q_2\}$ and $I = \{q_0\}$;
- $\Sigma = \{a, b\}$;
- Δ consists of the transitions
  $(q_0, a) \to (q_1, q_2)$, $(q_0, a) \to (q_2, q_1)$, $(q_1, b) \to (q_1, q_1)$, $(q_2, a) \to (q_2, q_2)$.

The k-ary Q-tree R from the right-hand side of Fig. 3.3 maps ε to q_0, nodes starting with 0 to q_1, and nodes starting with 1 to q_2. This tree R is an accepting run of $\mathcal{A}$ on the tree T on the left hand side of Figure 3.3.

The *tree language accepted* by a given looping automaton $\mathcal{A} = (Q, \Sigma, I, \Delta)$ is

$$L(\mathcal{A}) := \{T : \{0, \ldots, k-1\}^* \to \Sigma \mid \text{there is an accepting run of } \mathcal{A} \text{ on } T\}.$$

In our example, the language accepted by the automaton consists of two trees, the tree T defined above and the symmetric tree where the left subtree contains only nodes labelled with a and the right subtree contains only nodes labelled with b.

The Emptiness Test

Given a looping tree automaton $\mathcal{A}$, the emptiness test decides whether $L(\mathcal{A}) = \emptyset$ or not. Based on the definition of the accepted language, one might be tempted to try to solve the problem in a *top-down* manner, by first choosing an initial state to label the root, then choose a transition starting with this state to label its successors, etc. However, the algorithm obtained this way is non-deterministic since one may have several initial states, and also several possible transitions for each state.

To obtain a *deterministic polynomial time emptiness test*, it helps to work *bottom-up*. The main idea is that one wants to compute the set of *bad states*, i.e., states that do not occur in any run of the automaton. Obviously, any state q that does not occur on the left-hand side of a transition $(q, \cdot) \to (\cdots)$ is bad. Starting with this set, one can then extend the set of states known to be bad using the fact that a state q is bad if all transitions $(q, \cdot) \to (q_1, \ldots, q_k)$ starting with q contain a bad state q_j in their right-hand side. Obviously, this process of extending the set of known bad states terminates after a linear number of additions of states to the set of known bad states, and it is easy to show that the final set obtained this way is indeed the set of all bad states. The accepted language is then empty iff all initial states are bad. By using appropriate data structures, one can ensure that the overall complexity of the algorithm is linear in the size of the automaton. A more detailed description of this emptiness test for looping tree automata can be found in [26].

The Reduction

Recall that we want to reduce the satisfiability problem for $\mathcal{ALC}$-concepts w.r.t. general TBoxes to the emptiness problem for looping tree automata by constructing, for a given input $C, \mathcal{T}$, an automaton $\mathcal{A}_{C,\mathcal{T}}$ that accepts exactly the tree-shaped models of C w.r.t. $\mathcal{T}$.

Before this is possible, however, we need to overcome the *mismatch between the underlying kinds of trees*. The tree-shaped models of C w.r.t. $\mathcal{T}$ are trees with labelled edges, but without a fixed arity. In order to express such trees as k-ary Σ-trees for an appropriate k, where Σ consists of all sets of concept names, we consider all the existential restriction occurring in C and $\mathcal{T}$. The number of these restrictions determines k. Using the bisimulation invariance of $\mathcal{ALC}$ [110], it is easy to show that the existence of a tree-shaped model of C w.r.t. $\mathcal{T}$ also implies the existence of a tree-shaped model where every node has at most k successor nodes. To get exactly k successors, we can do some padding with dummy nodes if needed. The edge label is simply pushed into the label of the successor node, i.e., each node label contains, in addition to concept names, exactly one role name, which expresses with which role the node is reached from its unique predecessor. For the root, an arbitrary role can be chosen.

The states of $\mathcal{A}_{C,\mathcal{T}}$ are sets of subexpressions of the concepts occurring in C and $\mathcal{T}$. Intuitively, a run of the automaton on a tree-shaped model of C w.r.t. $\mathcal{T}$ labels a node not only with the concept names to which this element of the model belongs, but also with all the subexpressions to which it belongs. For technical reasons, we need to normalize the input concept and TBox before we build these subexpressions. First, we ensure that all GCIs in $\mathcal{T}$ are of the form $\top \sqsubseteq D$ by using the fact that the GCIs $C_1 \sqsubseteq C_2$ and $\top \sqsubseteq \neg C_1 \sqcup C_2$ are equivalent. Second, we transform the input concept C and every concept D in a GCI $\top \sqsubseteq D$ into negation normal form as described in Section 3.4.1. In our example, the normalized TBox consists of the GCIs

$$\top \sqsubseteq \neg A \sqcup \exists r.B, \quad \top \sqsubseteq \neg B \sqcup \exists r.A, \quad \top \sqsubseteq (\neg A \sqcap \neg B) \sqcup \exists s.\top,$$

whose subexpressions are $\top, \neg A \sqcup \exists r.B, \neg A, A, \exists r.B, B, \neg B \sqcup \exists r.A, \neg B, \exists r.A, (\neg A \sqcap \neg B) \sqcup \exists s.\top, \neg A \sqcap \neg B, \exists s.\top$. Of these, the node a in the tree-shaped model depicted on the right-hand side of Fig. 3.2 belongs to $\top, \neg A \sqcup \exists r.B, A, \exists r.B, \neg B \sqcup \exists r.A, \neg B, (\neg A \sqcap \neg B) \sqcup \exists s.\top, \exists s.\top$.

We are now ready to give a *formal definition of the automaton* $\mathcal{A}_{C,\mathcal{T}} = (Q, \Sigma, I, \Delta)$. Let $S_{C,\mathcal{T}}$ denote the set of all subexpressions of C and $\mathcal{T}$, $R_{C,\mathcal{T}}$ denote the set of all role names occurring in C and $\mathcal{T}$, and k the number of existential restrictions contained in $S_{C,\mathcal{T}}$. The *alphabet* Σ basically consists of all subsets of the set of concept names occurring in C and $\mathcal{T}$. As mentioned above, in order to encode the edge labels (i.e., express for which role r the node is a successor node), each "letter" contains, additionally, exactly one role name. Finally, the alphabet contains the empty set (not even containing a role name), which is used to label nodes that are introduced for padding purposes.

The set of *states* Q of $\mathcal{A}_{C,\mathcal{T}}$ consists of the *Hintikka sets* for $C, \mathcal{T}$, i.e., subsets q of $S_{C,\mathcal{T}} \cup R_{C,\mathcal{T}}$ such that $q = \emptyset$ or

- q contains exactly one role name;
- if $\top \sqsubseteq D \in \mathcal{T}$ then $D \in q$;
- if $C_1 \sqcap C_2 \in q$ then $\{C_1, C_2\} \subseteq q$;

- if $C_1 \sqcup C_2 \in q$ then $\{C_1, C_2\} \cap q \neq \emptyset$; and
- $\{A, \neg A\} \not\subseteq q$ for all concept names A.

The set of *initial states* I consists of those states containing C.

Finally, the *transition relation* Δ consists of those transitions $(q, \sigma) \rightarrow (q_1, \ldots, q_k)$ satisfying the following properties:

- q and σ coincide w.r.t. the concept and role names contained in them;
- if $q = \emptyset$, then $q_1 = \ldots = q_k = \emptyset$;
- if $\exists r.D \in q$, then there is an i such that $\{D, r\} \subseteq q_i$; and
- if $\forall r.D \in q$ and $r \in q_i$, then $D \in q_i$.

It is not hard to show that the construction of $\mathcal{A}_{C,\mathcal{T}}$ indeed yields a reduction of satisfiability w.r.t. general TBoxes in $\mathcal{ALC}$ to the emptiness problem for looping tree automata.

Proposition 5 *An $\mathcal{ALC}$-concept C is satisfiable w.r.t. a general $\mathcal{ALC}$-TBox $\mathcal{T}$ iff $L(\mathcal{A}_{C,\mathcal{T}}) \neq \emptyset$.*

Obviously, the number of states of $\mathcal{A}_{C,\mathcal{T}}$ is exponential in the size of C and $\mathcal{T}$. Since the emptiness problem for looping tree automata can be decided in polynomial time, we obtain an deterministic exponential upper-bound for the time complexity of the satisfiability problem. Together with the EXPTIME-hardness result sketched in Section 3.5 we thus know the exact worst-case complexity of the problem.

Theorem 6 *Satisfiability in $\mathcal{ALC}$ w.r.t. general TBoxes is* EXPTIME*-complete.*

3.6.2 Structural Approaches

As mentioned in the introduction, early DL systems were based on so-called structural subsumption algorithms, which first normalize the concepts to be tested for subsumption, and then compare the syntactic structure of the normalized concepts. The claim was that these algorithms can decide subsumption in polynomial time. However, the first complexity results for DLs, also mentioned in the introduction, showed that these algorithms were neither polynomial nor decision procedures for subsumption. For example, all early systems used unfolding of concept definitions, which can cause an exponential blow-up of the size of concepts. Nebel's coNP-hardness result [127] for subsumption w.r.t. definitorial TBoxes showed that this blow-up cannot be avoided whenever the constructors conjunction and value restriction are available. In addition, the early structural subsumption algorithms were not complete, i.e., they were not able to detect all valid subsumption relationships. These negative results for structural subsumption algorithms together with the advent of tableau based algorithms for expressive DLs, which behaved well in practice, was probably the main reason why structural approaches—and with them the quest for DLs with a polynomial subsumption problem—were largely abandoned during the 1990s. More recent results [11, 42, 6] on the complexity of reasoning in DLs with existential restrictions, rather than value restrictions, have led to a partial rehabilitation of structural approaches.

When trying to find a DL with a polynomial subsumption problem, it is clear that one cannot allow for all Boolean operations, since then one would inherit NP-hardness from propositional logic. It should also be clear that conjunction cannot be dispensed with since one must be able to state that more than one property should hold when defining a concept. Finally, if one wants to call the logic a DL, one needs a constructor using roles. This leads to the following two minimal candidate DLs:

- the DL $\mathcal{FL}_0$, which offers the concept constructors conjunction, value restriction ($\forall r.C$), and the top concept;
- the DL $\mathcal{EL}$, which offers the concept constructors conjunction, existential restriction ($\exists r.C$), and the top concept.

In the following, we will look at the subsumption problem[9] in these two DLs in some detail. Whereas subsumption without a TBox turns out to be polynomial in both cases, we will also see that $\mathcal{EL}$ exhibits a more robust behaviour w.r.t. the complexity of the subsumption problem in the presence of TBoxes.

Subsumption in $\mathcal{FL}_0$

First, we consider the case of subsumption of $\mathcal{FL}_0$-concepts without a TBox. There are basically two approaches for obtaining a structural subsumption algorithm in this case, which are based on two different normal forms. One can either use the equivalence $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as a rewrite rule from left-to-right or from right-to-left. Here we will consider the approach based on the left-to-right direction, whereas all of the early structural subsumption algorithms were based on a normal form obtained by rewriting in the other direction.[10]

By using the rewrite rule $\forall r.(C \sqcap D) \rightarrow \forall r.C \sqcap \forall r.D$ together with associativity, commutativity and idempotence[11] of $\sqcap$, any $\mathcal{FL}_0$-concept can be transformed into an equivalent one that is a conjunction of concepts 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 $\forall r_1 \ldots r_m.A$, where $r_1 \ldots r_m$ is viewed as a word over the alphabet of all role names. In addition, instead of $\forall w_1.A \sqcap \ldots \sqcap \forall w_\ell.A$ 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 $\mathcal{FL}_0$-concepts C, D containing the concept names $A_1, \ldots, A_k$ can be rewritten as

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

where U_i, V_i are finite sets of words over the alphabet of all role names. This normal form provides us with the following *characterization of subsumption* of $\mathcal{FL}_0$-concepts [24]:

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

[9] Note that the satisfiability problem is trivial in $\mathcal{FL}_0$ and $\mathcal{EL}$, since any concept expressed in these languages is satisfiable. The reduction of subsumption to satisfiability is not possible due to the absence of negation.

[10] A comparison between the two approaches can be found in [21].

[11] I.e., $(A \sqcap B) \sqcap C \equiv A \sqcap (B \sqcap C)$, $A \sqcap B \equiv B \sqcap A$, and $A \sqcap A \equiv A$.

$$
\begin{array}{lcl}
A & \equiv & C \sqcap \forall r.B \sqcap \forall s.\forall r.P \\
B & \equiv & \forall s.C \\
C & \equiv & \forall r.P
\end{array}
$$

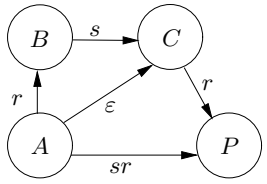

Figure 3.4: A definitorial $\mathcal{FL}_0$-TBox and the corresponding acyclic automaton.

Since the size of the normal forms is polynomial in the size of the original concepts, 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 $\mathcal{FL}_0$.

This characterization of subsumption via inclusion of finite sets of words can be extended to definitorial TBoxes as follows. A given TBox $\mathcal{T}$ can be translated into a finite (word) automaton[12] $\mathcal{A}_\mathcal{T}$, whose states are the concept names occurring in $\mathcal{T}$, and whose transitions are induced by the value restrictions occurring in $\mathcal{T}$ (see Fig. 3.4 for an example). A formal definition of this translation can be found in [9], where the more general case of cyclic TBoxes is treated. In the case of definitorial TBoxes, which are by definition acyclic, the resulting automata are also acyclic.

Let us call a concept name a *defined concept* in a definitorial TBox if it occurs on the left-hand side of a concept definition, and a *primitive concept* otherwise. For a defined concept A and a primitive concept 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, it should not be surprising that subsumption w.r.t. definitorial $\mathcal{FL}_0$-TBoxes can be characterized in terms of inclusion of languages accepted by acyclic automata. Indeed, the following is a *characterization of subsumption* in $\mathcal{FL}_0$ w.r.t. definitorial TBoxes:

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

In the example of Fig. 3.4, we have $L_{\mathcal{A}_\mathcal{T}}(A, P) = \{r, sr, rsr\} \supset \{sr\} = L_{\mathcal{A}_\mathcal{T}}(B, P)$, and thus $A \sqsubseteq_\mathcal{T} B$, but $B \not\sqsubseteq_\mathcal{T} A$.

Since the inclusion problem for languages accepted by acyclic finite automata is coNP-complete [73], this reduction shows that the subsumption problem in $\mathcal{FL}_0$ w.r.t. definitorial TBoxes is in coNP. As shown by Nebel [127], the reduction also works in the opposite direction, which yields the matching lower bound. In the presence of general TBoxes, the subsumption problem in $\mathcal{FL}_0$ actually becomes as hard as for $\mathcal{ALC}$, namely ExpTime-hard [6, 87].

Theorem 7 *Subsumption in $\mathcal{FL}_0$ is polynomial without TBox, coNP-complete w.r.t. definitorial TBoxes, and* EXPTIME*-complete w.r.t. general TBoxes.*

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

Subsumption in $\mathcal{EL}$

In contrast to the negative complexity results for subsumption w.r.t. TBoxes in $\mathcal{FL}_0$, subsumption in $\mathcal{EL}$ remains polynomial even in the presence of general TBoxes [42]. The polynomial-time subsumption algorithm for $\mathcal{EL}$ that will be sketched below actually classifies a given TBox $\mathcal{T}$, i.e., it simultaneously computes all subsumption relationships between the concept names occurring in $\mathcal{T}$. This algorithm proceeds in four steps:

1. Normalize the TBox.
2. Translate the normalized TBox into a graph.
3. Complete the graph using completion rules.
4. Read off the subsumption relationships from the normalized graph.

A general $\mathcal{EL}$-TBox is *normalized* if it only contains GCIs of the following form:

$$A_1 \sqcap A_2 \sqsubseteq B, \quad A \sqsubseteq \exists r.B, \text{ or } \exists r.A \sqsubseteq B,$$

where A, A_1, A_2, B are concept names or the top-concept $\top$. One can transform a given TBox into a normalized one by applying normalization rules. Instead of describing these rules in the general case, we just illustrate them by an example, where we underline GCIs that need further rewriting:

$$\begin{aligned}
\exists r.A \sqcap \exists r.\exists s.A \sqsubseteq A \sqcap B &\rightsquigarrow \underline{\exists r.A \sqsubseteq B_1},\ \underline{B_1 \sqcap \exists r.\exists s.A \sqsubseteq A \sqcap B} \\
B_1 \sqcap \exists r.\exists s.A \sqsubseteq A \sqcap B &\rightsquigarrow \underline{\exists r.\exists s.A \sqsubseteq B_2},\ \underline{B_1 \sqcap B_2 \sqsubseteq A \sqcap B} \\
\exists r.\exists s.A \sqsubseteq B_2 &\rightsquigarrow \exists s.A \sqsubseteq B_3,\ \exists r.B_3 \sqsubseteq B_2, \\
B_1 \sqcap B_2 \sqsubseteq A \sqcap B &\rightsquigarrow B_1 \sqcap B_2 \sqsubseteq A,\ B_1 \sqcap B_2 \sqsubseteq B
\end{aligned}$$

For example, in the first normalization step we introduce the abbreviation B_1 for the description $\exists r.A$. One might think that one must make B_1 equivalent to $\exists r.A$, i.e., also add the GCI $B_1 \sqsubseteq \exists r.A$. However, it can be shown that adding just $\exists r.A \sqsubseteq B_1$ is sufficient to obtain a *subsumption-equivalent* TBox, i.e., a TBox that induces the same subsumption relationships between the concept names occurring in the original TBox. All normalization rules preserve equivalence in this sense, and if one uses an appropriate strategy (which basically defers the applications of the rule applied in the last step of our example to the end), then the normal form can be computed in linear time.

In the next step, we build the *classification graph* $G_{\mathcal{T}} = (V, V \times V, S, R)$ where

- V is the set of concept names (including $\top$) occurring in the normalized TBox $\mathcal{T}$;
- S labels nodes with sets of concept names (again including $\top$);
- R labels edges with sets of role names.

It can be shown that the label sets satisfy the following *invariants*:

- $B \in S(A)$ implies $A \sqsubseteq_{\mathcal{T}} B$, i.e., $S(A)$ contains only subsumers of A w.r.t. $\mathcal{T}$.
- $r \in R(A, B)$ implies $A \sqsubseteq_{\mathcal{T}} \exists r.B$, i.e., $R(A, B)$ contains only roles r such that $\exists r.B$ subsumes A w.r.t. $\mathcal{T}$.

(R1)	$A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{T}$	and $A_1, A_2 \in S(A)$	then add B to $S(A)$
(R2)	$A_1 \sqsubseteq \exists r.B \in \mathcal{T}$	and $A_1 \in S(A)$	then add r to $R(A, B)$
(R3)	$\exists r.B_1 \sqsubseteq A_1 \in \mathcal{T}$	and $B_1 \in S(B), r \in R(A, B)$	then add A_1 to $S(A)$

Figure 3.5: The completion rules for subsumption in $\mathcal{EL}$ w.r.t. general TBoxes.

Initially, we set $S(A) := \{A, \top\}$ for all nodes $A \in V$, and $R(A, B) := \emptyset$ for all edges $(A, B) \in V \times V$. Obviously, the above invariants are satisfied by these initial label sets.

The labels of nodes and edges are then extended by applying the rules of Fig. 3.5, where we assume that a rule is only applied if it really extends a label set. It is easy to see that these rules preserve the above invariants. For example, consider the (most complicated) rule (R3). Obviously, $\exists r.B_1 \sqsubseteq A_1 \in \mathcal{T}$ implies $\exists r.B_1 \sqsubseteq_{\mathcal{T}} A_1$, and the assumption that the invariants are satisfied before applying the rule yields $B \sqsubseteq_{\mathcal{T}} B_1$ and $A \sqsubseteq_{\mathcal{T}} \exists r.B$. The subsumption relationship $B \sqsubseteq_{\mathcal{T}} B_1$ obviously implies $\exists r.B \sqsubseteq_{\mathcal{T}} \exists r.B_1$. By applying transitivity of the subsumption relation $\sqsubseteq_{\mathcal{T}}$, we thus obtain $A \sqsubseteq_{\mathcal{T}} A_1$.

The fact that subsumption in $\mathcal{EL}$ w.r.t. general TBoxes can be decided in polynomial time is an immediate consequence of the following statements:

1. Rule application terminates after a polynomial number of steps.
2. If no more rules are applicable, then $A \sqsubseteq_{\mathcal{T}} B$ iff $B \in S(A)$.

Regarding the first statement, note that the number of nodes is linear and the number of edges is quadratic in the size of $\mathcal{T}$. In addition, the size of the label sets is bounded by the number of concept names and role names, and each rule application extends at least one label. Regarding the equivalence in the second statement, the "if" direction follows from the fact that the above invariants are preserved under rule application. To show the "only-if" direction, assume that $B \notin S(A)$. Then the following interpretation $\mathcal{I}$ is a model of $\mathcal{T}$ in which $A \in A^{\mathcal{I}}$, but $A \notin B^{\mathcal{I}}$:

- $\Delta^{\mathcal{I}} := V$;
- $r^{\mathcal{I}} := \{(A', B') \mid r \in R(A', B')\}$ for all role names r;
- $B'^{\mathcal{I}} := \{A' \mid B' \in S(A')\}$ for all concept names A'.

More details can be found in [42, 6].

Theorem 8 *Subsumption in $\mathcal{EL}$ is polynomial w.r.t. general TBoxes.*

In [6] this result is extended to the DL $\mathcal{EL}^{++}$, which extends $\mathcal{EL}$ with the bottom concept, nominals, a restricted form of concrete domains, and a restricted form of so-called role-value maps. In addition, it is shown in [6] that basically all other additions of typical DL constructors to $\mathcal{EL}$ make subsumption w.r.t. general TBoxes EXPTIME-complete.

It should be noted that these results are not only of theoretical interest. In fact, large biomedical ontologies such as the Gene Ontology [55] and SNOMED [154] can be expressed in $\mathcal{EL}$, and a first implementation of the subsumption algorithm for $\mathcal{EL}$ sketched above behaves very well on these very large knowledge bases [23].

3.7 DLs in Ontology Language Applications

Description Logics are (or have been) used in a wide range of applications, including configuration (e.g., of telecommunications equipment) [120], and software information and documentation systems [61]. DLs have also been extensively applied in the area of databases [34], where they have been used to support schema design [54, 30], schema and data integration [51, 52], and query answering [47, 48, 91].

Perhaps the most prominent application of DLs is, however, as the basis for ontology languages such as OIL, DAML+OIL and OWL [99]. In the following section we will briefly examine the motivation for and realisation of a DL based ontology language, with particular reference to OWL; in Section 3.7.2 we will mention some ontology tools and applications that exploit DL reasoning.

3.7.1 The OWL Ontology Language

OWL is a semantic web ontology language, developed by the W3C Web-Ontology working group, whose semantics can be defined via a translation into an expressive DL.[13] This is not a coincidence—it was a design goal. The mapping allows OWL to exploit results from DL research (e.g., regarding the decidability and complexity of key inference problems), and to use implemented DL reasoners (e.g., FaCT [94] and RACER [81]) in order to provide reasoning services for OWL applications.

An OWL (Lite or DL) ontology can be seen to correspond to a DL TBox together with a role hierarchy, describing the domain in terms of *classes* (corresponding to concepts) and *properties* (corresponding to roles). An ontology consists of a set of *axioms* that assert, e.g., subsumption relationships between classes or properties.

As in a standard DL, OWL classes may be names or expressions built up from simpler classes and properties using a variety of constructors. The set of constructors supported by OWL, along with the equivalent DL syntax, is summarised in Figure 3.6.[14] The full XML serialisation of the RDF syntax is not shown as it is rather verbose, e.g., $\mathsf{Human} \sqcap \mathsf{Male}$ would be written as

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Human"/>
    <owl:Class rdf:about="#Male"/>
  </owl:intersectionOf>
</owl:Class>
```

while $(\geqslant 2\ \mathsf{hasChild}.\mathsf{Thing})$ would be written as

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasChild"/>
  <owl:minCardinality
    rdf:datatype="&xsd;NonNegativeInteger">2
```

[13] In fact there are 3 "species" of OWL: OWL Lite, OWL DL and OWL full, only the first two of which have DL based semantics. The semantics of OWL full is given by an extension of the RDF model theory [83].

[14] In fact, there are a few additional constructors provided as "syntactic sugar", but all are trivially reducible to the ones described in Figure 3.6.

Constructor	DL Syntax	Example
`intersectionOf`	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male
`unionOf`	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer
`complementOf`	$\neg C$	$\neg$Male
`oneOf`	$\{x_1 \ldots x_n\}$	{john, mary}
`allValuesFrom`	$\forall P.C$	$\forall$hasChild.Doctor
`someValuesFrom`	$\exists r.C$	$\exists$hasChild.Lawyer
`hasValue`	$\exists r.\{x\}$	$\exists$citizenOf.{USA}
`minCardinality`	$(\geqslant n\ r)$	$(\geqslant 2$ hasChild$)$
`maxCardinality`	$(\leqslant n\ r)$	$(\leqslant 1$ hasChild$)$
`inverseOf`	r^-	hasChild$^-$

Figure 3.6: OWL constructors

```
    </owl:minCardinality>
  </owl:Restriction>
```

Prefixes such as `owl:` and `&xsd;` specify XML namespaces for resources, while `rdf:parseType="Collection"` is an extension to RDF that provides a "shorthand" notation for lisp style lists defined using triples with the properties first and rest (it can be eliminated, but with a consequent increase in verbosity). E.g., the first example above consists of the triples $\langle \mathsf{r}_1, \mathsf{owl:intersectionOf}, \mathsf{r}_2 \rangle$, $\langle \mathsf{r}_2, \mathsf{owl:first}, \mathsf{Human} \rangle$, $\langle \mathsf{r}_2, \mathsf{rdfs:type}, \mathsf{Class} \rangle$, $\langle \mathsf{r}_2, \mathsf{owl:rest}, \mathsf{r}_3 \rangle$, etc., where r_i is an anonymous resource, Human stands for a URI naming the resource "Human", and owl : intersectionOf, owl : first, owl : rest and rdfs : type stand for URIs naming the properties in question.

An important feature of OWL is that, besides "abstract" classes defined by the ontology, one can also use XML Schema *datatypes* (e.g., string, decimal and float) in `someValuesFrom`, `allValuesFrom`, and `hasValue` restrictions. This can be seen as a restricted version of *concrete domains* as mentioned in Section 3.2.3. The kinds of datatype that can be used in OWL are, however, very limited (see [135]), essentially being limited to built-in XML datatypes, and so only allowing for concepts such as $\exists$age.xsd : nonNegativeInteger; this could, e.g., be used in an axiom Person $\sqcap$ $\exists$age.xsd : nonNegativeInteger to assert that all persons have an age that is a non-negative integer.

As already mentioned, an OWL ontology consists of a set of axioms. Figure 3.7 summarises the axioms supported by OWL. These axioms make it possible to assert subsumption or equivalence with respect to classes or properties, the disjointness of classes, and the equivalence or non-equivalence of individuals (resources). Moreover, OWL also allows properties of properties (i.e., DL roles) to be asserted. In particular, it is possible to assert that a property is transitive, functional, inverse functional or symmetric.

It is easy to see that, except for individuals and datatypes, the constructors and axioms of OWL can be translated into $\mathcal{SHIQ}$; in fact, OWL Lite is equivalent to $\mathcal{SHIN}(\mathbf{D})$ and OWL DL is equivalent to $\mathcal{SHOIN}(\mathbf{D})$ (see Section 3.2.3).

Axiom	DL Syntax	Example
`subClassOf`	$C_1 \sqsubseteq C_2$	$\mathsf{Human} \sqsubseteq \mathsf{Animal} \sqcap \mathsf{Biped}$
`equivalentClass`	$C_1 \equiv C_2$	$\mathsf{Man} \equiv \mathsf{Human} \sqcap \mathsf{Male}$
`subPropertyOf`	$P_1 \sqsubseteq P_2$	$\mathsf{hasDaughter} \sqsubseteq \mathsf{hasChild}$
`equivalentProperty`	$P_1 \equiv P_2$	$\mathsf{cost} \equiv \mathsf{price}$
`disjointWith`	$C_1 \sqsubseteq \neg C_2$	$\mathsf{Male} \sqsubseteq \neg\mathsf{Female}$
`sameAs`	$\{x_1\} \equiv \{x_2\}$	$\{\mathsf{Pres\_Bush}\} \equiv \{\mathsf{G\_W\_Bush}\}$
`differentFrom`	$\{x_1\} \sqsubseteq \neg\{x_2\}$	$\{\mathsf{john}\} \sqsubseteq \neg\{\mathsf{peter}\}$
`TransitiveProperty`	P transitive role	hasAncestor is a transitive role
`FunctionalProperty`	$\top \sqsubseteq (\leqslant 1\ P)$	$\top \sqsubseteq (\leqslant 1\ \mathsf{hasMother})$
`InverseFunctionalProperty`	$\top \sqsubseteq (\leqslant 1\ P^-)$	$\top \sqsubseteq (\leqslant 1\ \mathsf{isMotherOf}^-)$
`SymmetricProperty`	$P \equiv P^-$	$\mathsf{isSiblingOf} \equiv \mathsf{isSiblingOf}^-$

Figure 3.7: OWL axioms

3.7.2 OWL Tools and Applications

As mentioned in Section 3.7.1, the ability to use DL reasoners to provide reasoning services for OWL applications was one of the motivations for basing the design of OWL on a DL. Several ontology design tools, both "academic" and commercial, now exploit the correspondence between OWL and $\mathcal{SHOIN}(\mathbf{D})$ in order to support ontology design and maintenance by, for example, highlighting inconsistent classes and implicit subsumption relationships. Examples of such tools include Protégé [109], Swoop [106], OilEd [28] and TopBraid Composer.[15] Reasoning support for such tools is typically provided by a DL reasoner such as FaCT++ [161], Racer [81] or Pellet [152].

The availability of such tools has contributed to the increasingly widespread use of OWL, not only in the Semantic Web per se, but as a popular language for ontology development in fields as diverse as biology [151], medicine [74], geography [76], geology [157], astronomy [60], agriculture [153] and defence [112]. Applications of OWL are particularly prevalent in the life sciences where it has been used by the developers of several large biomedical ontologies, including the Biological Pathways Exchange (BioPAX) ontology [142], the GALEN ontology [140], the Foundational Model of Anatomy (FMA) [74], and the National Cancer Institute thesaurus [82].

The importance of reasoning support in such applications was highlighted in [108], which describes a project in which the Medical Entities Dictionary (MED), a large ontology (100,210 classes and 261 properties) that is used at the Columbia Presbyterian Medical Center, was converted into OWL, and checked using an OWL reasoner. This check revealed "systematic modelling errors", and a significant number of missed subClass relationships which, if not corrected, "could have cost the hospital many missing results in various decision support and infection control systems that routinely use MED to screen patients".

[15] `http://www.topbraidcomposer.com/`

3.8 Further Reading

As mentioned in Section 3.1, we have endeavoured to cover the most important areas of DL research, but the subject is a very large one, and many interesting topics have not been covered. We will briefly list a few of these here, and provide pointers into the literature.

Besides the ones discussed in Section 3.2, a great many other operators and extensions have been investigated. These include feature chain agreements, role value maps, fixpoint constructors, n-ary predicates, various role constructors (including intersection, union, complement, composition, (reflexive-) transitive closure and identity), probabilistic extensions and defaults. Details of all of these (and more) can be found in [14].

There is currently a great deal of interest in the idea of combining DLs with other KR formalisms such as rules and Answer Set Programming (ASP). With an appropriate integration, the advantages of the different paradigms might be combined, e.g., by extending the powerful schema language provided by DLs with the ability to describe more complex relationships between named individuals, or by adding support for non-monotonic features such as negation as failure. Important contributions in this area include work on rule support in the Classic system [40], the integration of Datalog with DLs in $\mathcal{AL}$-log and CARIN [65, 113], the integration of answer set programming with DLs [68], and the extension of DLs with so-called DL-safe rules [124, 141].

As well as studying the formal properties of DLs, considerable energy has been devoted to investigating the implementation and optimisation of DL systems. Modern systems include CEL [6], FaCT++ [161], KAON2 [123], Pellet [137] and Racer [81]; for information on older systems, and on optimisation techniques, the reader is referred to [14]. A number of tools are now available that use the above mentioned reasoners to support, e.g., ontology design or schema integration. These include Swoop [106], Protégé [109], OilEd [28], and ICom [72].

Finally, in Section 3.2 we focused on standard reasoning problems such as satisfiability and subsumption testing. These are not, however, the only reasoning problems that might be of interest in applications, and several other "non-standard" inference problems have also been investigated. These include matching [20, 19], least common subsumer (lcs) [111], approximation and difference [43], axiom pinpointing [148, 132, 121], and conjunctive query answering [47, 158].

Bibliography

[1] Andrea Acciarri, Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati. Quonto: Querying ontologies. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI-05)*, pages 1670–1671. AAAI Press/The MIT Press, 2005.

[2] Hajnal Andréka, Johan van Benthem, and Istvaán Németi. Modal languages and bounded fragments of predicate logic. Technical Report ML-96-03, ILLC, University of Amsterdam, 1996.

[3] Carlos Areces. *Logic Engineering. The Case of Description and Hybrid Logics*. PhD thesis, ILLC, University of Amsterdam, 2000. ILLC Dissertation Series 2000–5.

[4] Carlos Areces, Patrick Blackburn, and Maarten Marx. A road-map on complexity for hybrid logics. In *Proc. of the Annual Conf. of the Eur. Assoc. for Computer Science Logic (CSL'99)*, volume 1683 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 1999.

[5] Carlos Areces, Maarten de Rijke, and Hans de Nivelle. Resolution in modal, description and hybrid logic. *J. of Logic and Computation*, 11(5):717–736, 2001.

[6] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, 2005.

[7] F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. Integrating description logics and action formalisms: First results. In AAAI Press/The MIT Press, editor, *Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05)*, 2005.

[8] 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)*, pages 446–451, 1991.

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

[10] Franz Baader. Description logic terminology. In *[14]*, pages 485–495. 2003.

[11] Franz Baader. Terminological cycles in a description logic with existential restrictions. In Georg Gottlob and Toby Walsh, editors, *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 325–330, Acapulco, Mexico, 2003. Morgan Kaufmann, Los Altos.

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

[13] Franz Baader, Hans-Jürgen Bürckert, Bernhard Hollunder, Werner Nutt, and Jörg H. Siekmann. Concept logics. In John W. Lloyd, editor, *Computational Logics, Symposium Proceedings*, pages 177–201. Springer, 1990.

[14] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[15] 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.

[16] 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.

[17] Franz Baader and Philipp Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of the 16th German Workshop on Artificial Intelligence (GWAI'92)*, volume 671 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 1992.

[18] 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.

[19] Franz Baader and Ralf Küsters. Matching in description logics with existential restrictions. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 261–272, 2000.

[20] Franz Baader, Ralf Küsters, Alex Borgida, and Deborah L. McGuinness. Matching in description logics. *J. of Logic and Computation*, 9(3):411–447, 1999.

[21] 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)*, volume 11 of *CEUR (*`http://ceur-ws.org/`*)*, 1998.

[22] Franz Baader, Ralf Küsters, and Frank Wolter. Extensions to description logics. In *[14]*, pages 219–261. 2003.

[23] Franz Baader, Carsten Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. SV, 2006.

[24] Franz Baader and Paliath Narendran. Unification of concepts terms in description logics. *J. of Symbolic Computation*, 31(3):277–305, 2001.

[25] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1):5–40, October 2001.

[26] Franz Baader and Stephan Tobies. The inverse method implements the automata approach for modal satisfiability. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 92–106. Springer, 2001.

[27] A. B. Baker. *Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results*. PhD thesis, University of Oregon, 1995.

[28] Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. OilEd: A Reason-able ontology editor for the semantic web. In *Proc. of the Joint German/Austrian Conf. on Artificial Intelligence (KI 2001)*, number 2174 in Lecture Notes in Artificial Intelligence, pages 396–408. Springer, 2001.

[29] Mordechai Ben-Ari, Joseph Y. Halpern, and Amir Pnueli. Deterministic propositional dynamic logic: Finite models, complexity, and completeness. *J. of Computer and System Sciences*, 25:402–417, 1982.

[30] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams using description logic based systems. In *Proc. of the KI'2001 Workshop on Applications of Description Logics*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-44/`, 2001.

[31] Orna Bernholtz and Orna Grumberg. Branching time temporal logic and amorphous tree automata. In Eike Best, editor, *Proc. of the Int. Conf. on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 262–277, 1993.

[32] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.

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

[34] Alexander Borgida, Maurizio Lenzerini, and Riccardo Rosati. Description logics for data bases. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 16. Cambridge University Press, 2003.

[35] Ronald J. Brachman. What's in a concept: Structural foundations for semantic

networks. *Int. Journal of Man-Machine Studies*, 9(2):127–152, 1977.

[36] 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.

[37] Ronald J. Brachman. "Reducing" CLASSIC to practice: Knowledge representation meets reality. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 247–258. Morgan Kaufmann, Los Altos, 1992.

[38] Ronald J. Brachman, Richard E. Fikes, and Hector J. Levesque. KRYPTON: A functional approach to knowledge representation. *IEEE Computer*, October:67–73, 1983.

[39] Ronald J. Brachman and Hector J. Levesque, editors. *Readings in Knowledge Representation*. Morgan Kaufmann, Los Altos, 1985.

[40] 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.

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

[42] Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In Ramon López de Mántaras and Lorenza Saitta, editors, *Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI 2004)*, pages 298–302, 2004.

[43] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin Turhan. Approximation and difference in description logics. In *Proc. of the 8th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2002)*, pages 203–214, 2002.

[44] 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.

[45] Martin Buchheit, Francesco M. Donini, Werner Nutt, and Andrea Schaerf. A refined architecture for terminological systems: Terminology = schema + views. *Artificial Intelligence*, 99(2):209–260, 1998.

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

[47] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[48] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Modeling and querying semi-structured data. *Network and Information Systems*, 2(2), 1999.

[49] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 84–89, 1999.

[50] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi.

Reasoning in expressive description logics. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 23, pages 1581–1634. Elsevier Science Publishers (North-Holland), Amsterdam, 2001.

[51] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.

[52] Diego Calvanese, Giuseppe De Giacomo, and Riccardo Rosati. Data integration and reconciliation in data warehousing: Conceptual modeling and reasoning support. *Network and Information Systems*, 2(4), 1999.

[53] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI-05)*, pages 602–607. AAAI Press/The MIT Press, 2005.

[54] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publisher, 1998.

[55] The Gene Ontology Consortium. Gene Ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

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

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

[58] 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.

[59] Giuseppe De Giacomo and Maurizio Lenzerini. TBox and ABox reasoning in expressive description logics. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 316–327, 1996.

[60] S. Derriere, A. Richard, and A. Preite-Martinez. An ontology of astronomical object types for the virtual observatory. *Proc. of Special Session 3 of the 26th meeting of the IAU: Virtual Observatory in Action: New Science, New Technology, and Next Generation Facilities*, 2006.

[61] Premkumar Devambu, Ronald J. Brachman, Peter J. Selfridge, and Bruce W. Ballard. LASSIE: A knowledge-based software information system. *Communications of the ACM*, 34(5):36–49, 1991.

[62] Francesco M. Donini, Bernhard Hollunder, Maurizio Lenzerini, Alberto Marchetti Spaccamela, Daniele Nardi, and Werner Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 2–3:309–327, 1992.

[63] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 151–162, 1991.

[64] 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, 1991.

[65] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. $\mathcal{AL}$-log: Integrating Datalog and description logics. *J. of Intelligent Information Systems*, 10(3):227–252, 1998.

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

[67] Jon Doyle and Ramesh S. Patil. Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48:261–297, 1991.

[68] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. In *Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 141–151. Morgan Kaufmann, Los Altos, 2004.

[69] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[70] K. Fine. In so many possible worlds. *Notre Dame J. of Formal Logic*, 13(4):516–520, 1972.

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

[72] Enrico Franconi and Gary Ng. The i.com tool for intelligent conceptual modeling. In *Proc. of the 7th Int. Workshop on Knowledge Representation meets Databases (KRDB 2000)*, volume 29 of *CEUR (`http://ceur-ws.org/`)*, pages 45–53, 2000.

[73] 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.

[74] Christine Golbreich, Songmao Zhang, and Olivier Bodenreider. The foundational model of anatomy in OWL: Experience and perspectives. *J. of Web Semantics*, 4(3), 2006.

[75] E. Gonçalvès and E. Grädel. Decidability issues for action guarded logics. In *Proc. of the 2000 Description Logic Workshop (DL 2000)*, volume 33 of *CEUR (`http://ceur-ws.org/`)*, pages 123–132, 2000.

[76] John Goodwin. Experiences of using OWL at the ordnance survey. In *Proc. of the First OWL Experiences and Directions Workshop*, volume 188 of *CEUR Workshop Proceedings*. CEUR (`http://ceur-ws.org/`), 2005.

[77] Erich Grädel. Guarded fragments of first-order logic: A perspective for new description logics? In *Proc. of the 1998 Description Logic Workshop (DL'98)*, volume 11 of *CEUR (`http://ceur-ws.org/`)*, 1998.

[78] Erich Grädel. On the restraining power of guards. *J. of Symbolic Logic*, 64:1719–1742, 1999.

[79] Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

[80] Volker Haarslev and Ralf Möller. RACE system description. In *Proc. of the 1999 Description Logic Workshop (DL'99)*, pages 130–132. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-22/`, 1999.

[81] 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.

[82] Frank W. Hartel, Sherri de Coronado, Robert Dionne, Gilberto Fragoso, and Jennifer Golbeck. Modeling a description logic vocabulary for cancer research. *Journal of Biomedical Informatics*, 38(2):114–129, 2005.

[83] Patrick Hayes. RDF model theory. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-mt/`.

[84] Patrick J. Hayes. In defense of logic. In *Proc. of the 5th Int. Joint Conf. on Artificial Intelligence (IJCAI'77)*, pages 559–565, 1977. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [39].

[85] Patrick J. Hayes. The logic of frames. In D. Metzing, editor, *Frame Conceptions and Text Understanding*, pages 46–61. Walter de Gruyter and Co., 1979. Republished in [39].

[86] Jochen Heinsohn, Daniel Kudenko, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of terminological representation systems. *Artificial Intelligence*, 68:367–397, 1994.

[87] Martin Hofmann. Proof-theoretic approach to description-logic. In Prakash Panangaden, editor, *Proc. of the 20th Annual IEEE Symp. on Logic in Computer Science, LICS 2005*, pages 229–237. IEEE Computer Society Press, June 2005.

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

[89] 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.

[90] I. Horrocks and P. F. Patel-Schneider. Optimising propositional modal satisfiability for description logic subsumption. In *Proc. of the 4th Int. Conf. on Artificial Intelligence and Symbolic Computation (AISC'98)*, volume 1476 of *LNAI*, pages 234–246. SV, 1998.

[91] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to decide query containment under constraints using a description logic. In *Proc. of the 7th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR 2000)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2000.

[92] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.

[93] I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 285–296, 2000.

[94] Ian Horrocks. The FaCT system. In Harrie de Swart, editor, *Proc. of the 2nd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'98)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 307–312. Springer, 1998.

[95] 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.

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

[97] Ian Horrocks and Peter F. Patel-Schneider. The generation of DAML+OIL. In *Proc. of the 2001 Description Logic Workshop (DL 2001)*, volume 49 of *CEUR (http://ceur-ws.org/)*, pages 30–35, 2001.

[98] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI 2002)*, pages 792–797. AAAI Press, 2002.

[99] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[100] 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.

[101] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, Los Altos, 2001.

[102] U. Hustadt and R. A. Schmidt. Using resolution for testing modal satisfiability and building models. In I. P. Gent, H. van Maaren, and T. Walsh, editors, *SAT 2000: Highlights of Satisfiability Research in the Year 2000*, volume 63 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, 2000. Also to appear in a special issue of *Journal of Automated Reasoning*.

[103] U. Hustadt, R. A. Schmidt, and C. Weidenbach. MSPASS: Subsumption testing with SPASS. In *Proc. of the 1999 Description Logic Workshop (DL'99)*, pages 136–137. Linköping University, 1999.

[104] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing SHIQ-description logic to disjunctive datalog programs. In *Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 152–162, 2004.

[105] David Janin and Igor Walukiewicz. Automata for the modal mu-calculus and related results. In Jirí Wiedermann and Petr Hájek, editors, *Int. Symp. on the Mathematical Foundation of Computer Science*, volume 969 of *Lecture Notes in Computer Science*, pages 552–562. Springer, 1995.

[106] Aditya Kalyanpur, Bijan Parsia, and James Hendler. A tool for working with web ontologies. *International Journal on Semantic Web and Information Systems*, 1(1):36–49, 2005.

[107] Yevgeny Kazakov and Boris Motik. A resolution-based decision procedure for $\mathcal{SHOIQ}$. In Ulrich Furbach and Natarajan Shankar, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Computer Science*, pages 662–677. Springer, 2006.

[108] Aaron Kershenbaum, Achille Fokoue, Chintan Patel, Christopher Welty, Edith Schonberg, James Cimino, Li Ma, Kavitha Srinivas, Robert Schloss, and J William Murdock. A view of OWL from the field: Use cases and experiences. In *Proc. of the Second OWL Experiences and Directions Workshop*, volume 216 of *CEUR (http://ceur-ws.org/)*, 2006.

[109] Holger Knublauch, Ray Fergerson, Natalya Noy, and Mark Musen. The Protégé OWL Plugin: An open development environment for semantic web applications. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proc.*

of the 2004 International Semantic Web Conference (ISWC 2004), number 3298 in Lecture Notes in Computer Science, pages 229–243. Springer, 2004.

[110] Natasha Kurtonina and Maarten de Rijke. Classifying description logics. In *Proc. of the 1997 Description Logic Workshop (DL'97)*, pages 49–53, 1997.

[111] Ralf Küsters and Ralf Molitor. Computing least common subsumers in $\mathcal{ALEN}$. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 219–224, 2001.

[112] Lee Lacy, Gabriel Aviles, Karen Fraser, William Gerber, Alice Mulvehill, and Robert Gaskill. Experiences using OWL in military applications. In *Proc. of the First OWL Experiences and Directions Workshop*, volume 188 of *CEUR Workshop Proceedings*. CEUR (`http://ceur-ws.org/`), 2005.

[113] Alon Y. Levy and Marie-Christine Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[114] 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.

[115] Carsten Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2001.

[116] Carsten Lutz. Interval-based temporal reasoning with general TBoxes. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 89–94, 2001.

[117] Carsten Lutz and Ulrike Sattler. Mary likes all cats. In *Proc. of the 2000 Description Logic Workshop (DL 2000)*, volume 33 of *CEUR (`http://ceur-ws.org/`)*, pages 213–226, 2000.

[118] 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.

[119] Eric Mays, Robert Dionne, and Robert Weida. K-Rep system overview. *SIGART Bull.*, 2(3):93–97, 1991.

[120] Deborah L. McGuinness and Jon Wright. Conceptual modeling for configuration: A description logic-based approach. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 12(4):333–344, 1998. Special issue on Configuration.

[121] Thomas Meyer, Kevin Lee, Richard Booth, and Jeff Z. Pan. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006)*. AAAI Press/The MIT Press, 2006.

[122] Marvin Minsky. A framework for representing knowledge. In J. Haugeland, editor, *Mind Design*. The MIT Press, 1981. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [39].

[123] Boris Motik and Ulrike Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In *Proc. of the 13th International Conference on Logic for Programming, Artificial Intelligence (LPAR06)*, LNCS. Springer Verlag, 2006.

[124] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for owl-dl with rules. *J. of Web Semantics*, 3(1):41–60, 2005.

[125] D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[126] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume

422 of *Lecture Notes in Artificial Intelligence*. Springer, 1990.

[127] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43(2):235–249, 1990.

[128] F. Oppacher and E. Suen. HARP: A tableau-based theorem prover. *J. of Automated Reasoning*, 4:69–100, 1988.

[129] 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.

[130] Leszek Pacholski, Wiesław Szwast, and Lidia Tendera. Complexity results for first-order two-variable logic with counting. *SIAM J. on Computing*, 29(4):1083–1117, 2000.

[131] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley Publ. Co., Reading, Massachussetts, 1994.

[132] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In Allan Ellis and Tatsuya Hagino, editors, *Proc. of the 14th International Conference on World Wide Web (WWW'05)*, pages 633–640. ACM, 2005.

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

[134] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language semantics and abstract syntax. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-semantics/`.

[135] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language semantics and abstract syntax. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-semantics/`.

[136] Peter F. Patel-Schneider, Deborah L. McGuiness, Ronald J. Brachman, Lori Alperin Resnick, and Alexander Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bull.*, 2(3):108–113, 1991.

[137] Pellet OWL reasoner. Maryland Information and Network Dynamics Lab, 2003. `http://www.mindswap.org/2003/pellet/index.shtml`.

[138] Christof Peltason. The BACK system — an overview. *SIGART Bull.*, 2(3):114–119, 1991.

[139] M. Ross Quillian. Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science*, 12:410–430, 1967. Republished in [39].

[140] Alan Rector and Jeremy Rogers. Ontological and practical issues in using a description logic to represent medical concept systems: Experience from GALEN. In *Reasoning Web, Second International Summer School, Tutorial Lectures*, volume 4126 of *LNCS*, pages 197–231. SV, 2006.

[141] Riccardo Rosati. On the decidability and complexity of integrating ontologies and rules. *J. of Web Semantics*, 3(1):61–73, 2005.

[142] Alan Ruttenberg, Jonathan Rees, and Joanne Luciano. Experience using OWL DL for the exchange of biological pathway information. In *Proc. of the First OWL Experiences and Directions Workshop*, volume 188 of *CEUR Workshop Proceedings*. CEUR (`http://ceur-ws.org/`), 2005.

[143] U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, volume 1137 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1996.

[144] U. Sattler and M. Y. Vardi. The hybrid mu-calculus. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proc. of the International Joint Conference on Automated Reasoning (IJCAR-01)*, volume 2083 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2001.

[145] 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.

[146] Klaus Schild. Terminological cycles and the propositional μ-calculus. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 509–520, 1994.

[147] Klaus Schild. *Querying Knowledge and Data Bases by a Universal Description Logic with Recursion*. PhD thesis, Universität des Saarlandes, Germany, 1995.

[148] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In Georg Gottlob and Toby Walsh, editors, *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 355–362, Acapulco, Mexico, 2003. Morgan Kaufmann, Los Altos.

[149] 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.

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

[151] Amandeep Sidhu, Tharam Dillon, Elisabeth Chang, and Baldev Singh Sidhu. Protein ontology development using OWL. In *Proc. of the First OWL Experiences and Directions Workshop*, volume 188 of *CEUR Workshop Proceedings*. CEUR (`http://ceur-ws.org/`), 2005.

[152] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. To appear, 2005.

[153] Dagobert Soergel, Boris Lauser, Anita Liang, Frehiwot Fisseha, Johannes Keizer, and Stephen Katz. Reengineering thesauri for new applications: The AGROVOC example. *J. of Digital Information*, 4(4), 2004.

[154] K.A. Spackman, K.E. Campbell, and R.A. Cote. SNOMED RT: A reference terminology for health care. *J. of the American Medical Informatics Association*, pages 640–644, 1997. Fall Symposium Supplement.

[155] P.-H. Speel, F. van Raalte, P. E. van der Vet, and N. J. I. Mars. Runtime and memory usage performance of description logics. In G. Ellis, R. A. Levinson, A. Fall, and V. Dahl, editors, *Knowledge Retrieval, Use and Storage for Efficiency: Proc. of the 1st Int. KRUSE Symposium*, pages 13–27, 1995.

[156] Larry Stockmeyer and Albert Meyer. Word problems requiring exponential time(preliminary report). In *Proc. of the 5th Annual ACM symposium on Theory of computing (STOC'73)*, pages 1–9. ACM Press, 1973.

[157] Semantic web for earth and environmental terminology (SWEET). Jet Propulsion Laboratory, California Institute of Technology, 2006. `http://sweet.jpl.nasa.gov/`.

[158] Sergio Tessaris. *Questions and Answers: Reasoning and Querying in Description Logic*. PhD thesis, University of Manchester, Department of Computer Science, April 2001.

[159] Wolfgang Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 133–192. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

[160] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.

[161] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[162] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32:183–221, 1986.

[163] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[164] William A. Woods. What's in a link: Foundations for semantic networks. In D. G. Bobrow and A. M. Collins, editors, *Representation and Understanding: Studies in Cognitive Science*, pages 35–82. Academic Press, 1975. Republished in [39].