Description Logics

Franz Baader

Theoretical Computer Science, TU Dresden, Germany
baader@inf.tu-dresden.de

Abstract. Description Logics (DLs) are a well-investigated family of logic-based knowledge representation formalisms, which can be used to represent the conceptual knowledge of an application domain in a structured and formally well-understood way. They are employed in various application domains, such as natural language processing, configuration, and databases, but their most notable success so far is the adoption of the DL-based language OWL as standard ontology language for the semantic web.
This article concentrates on the problem of designing reasoning procedures for DLs. After a short introduction and a brief overview of the research in this area of the last 20 years, it will on the one hand present approaches for reasoning in expressive DLs, which are the foundation for reasoning in the Web ontology language OWL DL. On the other hand, it will consider tractable reasoning in the more light-weight DL $\mathcal{EL}$, which is employed in bio-medical ontologies, and which is the foundation for the OWL 2 profile OWL 2 EL.

1 Introduction

In their introduction to The Description Logic Handbook [11], Brachman and Nardi point out that the general goal of knowledge representation (KR) is to "develop formalisms for providing high-level descriptions of the world that can be effectively used to build intelligent applications" [32]. This sentence states in a compact way some of the key requirements that a KR formalism needs to satisfy. In order to be accepted as a *formalism* in this sense, a knowledge representation language needs to be equipped with a well-defined syntax and a formal, unambiguous semantics, which was not always true for early KR approaches such as semantic networks [101] and frames [90]. A *high-level* description concentrates on the representation of those aspects relevant for the application at hand while ignoring irrelevant details. In particular, this facilitates the use of relatively inexpressive languages even though they may not be able to faithfully represent the whole application domain. *Intelligent* applications should be able to reason about the knowledge and infer implicit knowledge from the explicitly represented knowledge, and thus the *effective* use of the knowledge depends on the availability of practical reasoning tools.

Description logics (DLs) [11] are a family of logic-based knowledge representation formalisms that are tailored towards representing the terminological knowledge of an application domain in a structured and formally well-understood way. They allow their users to define the important notions (classes, relations, objects) of the domain using concepts, roles, and individuals; to state constraints on the way these notions can be interpreted; and to deduce consequences such as subclass and instance relationships from the definitions and constraints. The name *description logics* is motivated by the fact that, on the one hand, classes 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. For example, in a conference domain, we may have classes (concepts) like Person, Speaker, Author, Talk, Participant, PhD_student, Workshop, Tutorial; relations (roles) like gives, attends, attended_by, likes; and objects (individuals) like Richard, Frank, Paper_176. A speaker can be defined as a person that gives a talk:

$$\mathsf{Speaker} \equiv \mathsf{Person} \sqcap \exists \mathsf{gives}.\mathsf{Talk},$$

we can say that Frank is a speaker and attends the DL tutorial using the assertions:

$$\mathsf{Speaker}(\mathsf{FRANK}), \quad \mathsf{attends}(\mathsf{FRANK}, \mathsf{DL\_TUTORIAL}), \quad \mathsf{Tutorial}(\mathsf{DL\_TUTORIAL}),$$

and state the constraints that tutorials are only attended by PhD students:

$$\mathsf{Tutorial} \sqsubseteq \forall \mathsf{attended\_by}.\mathsf{PhD\_student},$$

and that the relation attended_by is the inverse of the relation attends:

$$\mathsf{attended\_by} \equiv \mathsf{attends}^{-1}.$$

DLs have been employed in various application domains, such as natural language processing, configuration, databases, and biomedical ontologies, but their most notable success so far is probably the adoption of the DL-based language OWL[1] as standard ontology language for the semantic web [69, 15]. The three main reasons for the adoption of DLs as ontology languages are

- the availability of a formal, unambiguous semantics, which is based on the Tarski-style semantics of first-order predicate logic, and is thus fairly easy to describe and comprehend;
- the fact that DLs provide their users with various carefully chosen means of expressiveness for constructing concepts and roles, for further constraining their interpretations, and for instantiating concepts and roles with individuals;
- the fact that DL systems provide their users with highly-optimized inference procedures that allow them to deduce implicit knowledge from the explicitly represented knowledge.

[1] http://www.w3.org/TR/owl-features/

The formal semantics of DLs and typical concept and role constructors as well as the formalism for expressing constraints will be introduced in the next section. In the remainder of this section, we concentrate on the inference capabilities of DL systems. The *subsumption* algorithm determines subconcept-superconcept relationships: C is subsumed by D iff all instances of C are necessarily instances of D, i.e., the first concept is always interpreted as a subset of the second concept. For example, given the definition of $\mathsf{Speaker}$ from above, $\mathsf{Speaker}$ is obviously subsumed by Person. In general, however, induced subsumption relationships may be much harder to detect. The *instance* algorithm determines induced instance relationships: the individual i is an instance of the concept description C iff i is always interpreted as an element of C. For example, given the assertions for Frank and the DL tutorial from above, the constraint for tutorials, and the constraint expressing that attends is the inverse of attended by, we can deduce that FRANK is an instance of $\mathsf{Phd\_student}$. The *consistency* algorithm determines whether a knowledge base (consisting of a set of assertions and a set of terminological axioms, i.e., concept definitions and constraints) is non-contradictory. For example, if we added a disjointness constraint

$$\mathsf{Speaker} \sqcap \mathsf{PhD\_student} \sqsubseteq \bot$$

for speakers and PhD students to the conference knowledge base introduced so far, then this knowledge base would become inconsistent since it follows from the knowledge base that Frank is both a speaker and a PhD students, contradicting the stated disjointness of these two concepts.

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. The research related to this issue can be classified into the following five phases.[2]

Phase 1 (1980–1990) was mainly concerned with implementation of systems, such as Klone, K-Rep, Back, and Loom [33, 88, 100, 87]. 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 [93]. These algorithms are usually quite 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 the existing subsumption/instance relationships. At the end of this phase, early formal investigations into the complexity of reasoning in DLs showed that most DLs do not have polynomial-time inference problems [30, 94]. As a reaction, the

[2] Note, however, that the assigned temporal intervals are only rough estimates, and thus should not be taken too seriously.

implementors of the Classic system (the first industrial-strength DL system) carefully restricted the expressive power of their DL [99, 29].

Phase 2 (1990–1995) started with the introduction of a new algorithmic paradigm into DLs, so-called *tableau-based algorithms* [108, 50, 66]. They work on propositionally closed DLs (i.e., DLs with full Boolean operators) and are complete also for expressive DLs. To decide the consistency of a knowledge base, a tableau-based algorithm tries to construct a model of it by breaking down 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, subsumption and satisfiability can be reduced to consistency, a consistency algorithm can solve all inference problems mentioned above. The first systems employing such algorithms (Kris and Crack) demonstrated that optimized implementations of these algorithm lead to an acceptable behavior of the system, even though the worst-case complexity of the corresponding reasoning problems is no longer in polynomial time [14, 35]. This phase also saw a thorough analysis of the complexity of reasoning in various DLs [50, 51, 49, 47]. Another important observation was that DLs are very closely related to modal logics [103].

Phase 3 (1995–2000) is characterized by the development of inference procedures for very expressive DLs, either based on the tableau-approach [70, 71] or on a translation into modal logics [44, 45, 43, 46]. Highly optimized systems (FaCT, Race, and Dlp [67, 61, 98]) showed that tableau-based algorithms for expressive DLs lead to a good practical behavior of the system even on (some) large knowledge bases. In this phase, the relationship to modal logics [44, 104] and to decidable fragments of first-order logic was also studied in more detail [28, 96, 59, 57, 58, 75], and applications in databases (like schema reasoning, query optimization, and integration of databases) were investigated [36, 40, 42].

During *Phase 4* (2000–2005), industrial strength DL systems employing very expressive DLs and tableau-based algorithms were developed [115, 62, 109], with applications like the Semantic Web or knowledge representation and integration in bio-informatics in mind. In this phase, the Web Ontology Language OWL, whose sublanguages OWL DL and OWL Lite are based on expressive DLs, became an official W3C recommendation,[3] thus boosting the use of DLs for the definition of ontologies. On the more foundational side, this phase saw the development of alternative approaches for reasoning in expressive DLs, such as resolution-based approaches [73, 74, 2, 72, 78], which use an optimized translation of DLs into first-order predicate logic and then apply appropriate first-order resolution provers, and automata-based approaches [41, 86, 84, 114, 25, 13], which are often more convenient for showing ExpTime complexity upper-bounds than tableau-based approaches.

We are now in *Phase 5*, where on the one hand even more expressive DLs with highly-optimized tableau-based algorithms [68] are proposed as basis for the

[3] http://www.w3.org/TR/owl-features/

new Web Ontology Language OWL 2.[4] On the other hand, more light-weight DLs are investigated and proposed as profiles of OWL 2,[5] such as members of the $\mathcal{EL}$ family [7, 8], for which the subsumption and the instance problem are polynomial, and of the DL Lite family [38, 39], for which the instance problem and query answering are polynomial w.r.t. data complexity. Another important development in this phase is that inference problems other than the classical ones (subsumption, instance, consistency) are gaining importance, such as query answering (i.e., answering conjunctive queries w.r.t. DL knowledge bases) [1, 55, 85, 95], pinpointing (i.e., exhibiting the axioms responsible for a given consequence) [105, 97, 89, 22, 24], and modularization (i.e., extracting a part of a knowledge base that has the same consequence as the full knowledge base, for consequences formulated using a certain restricted vocabulary) [60, 79, 110].

2 Basic definitions

As mentioned above, a key component of a DL is the *description language*, which allows its users to build complex concepts (and roles) out of atomic ones. These descriptions can then be uses in the *terminological part* of the knowledge base (TBox) to introduce the terminology of an application domain, by defining concepts and imposing additional (non definitional) constraints on their interpretation. In the *assertional part* of the knowledge base (ABox), facts about a specific application situation can be stated, by introducing named individuals and relating them to concepts and roles. *Reasoning* then allows us to derive implicit knowledge from the explicitly represented one. In the following, we introduce these four components of a DL more formally.

2.1 The basic description language $\mathcal{ALC}$ and some extensions

Starting with a set of concept names (atomic concepts) and role names (atomic roles), concept descriptions are built using concept constructors. The semantics of concept descriptions is defined using the notion of an interpretation, which assigns sets to concepts and binary relations to roles. First, we introduce the constructors available in the basic description language $\mathcal{ALC}$,[6] together with their semantics.

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

– *all concept names are $\mathcal{ALC}$ concept descriptions;*

[4] http://www.w3.org/TR/2009/WD-owl2-overview-20090327/

[5] http://www.w3.org/TR/owl2-profiles/

[6] Following the usage in the literature, we will sometimes call description languages like $\mathcal{ALC}$ “Description Logics,” thereby ignoring the additional ingredients of a DL, such as the terminological formalism.

– *if C and D are $\mathcal{ALC}$ concept descriptions, then so are $\neg C$, $C \sqcup D$, and $C \sqcap D$;*
– *if C is an $\mathcal{ALC}$ concept description and $r \in N_R$, then $\exists r.C$ and $\forall r.C$ are $\mathcal{ALC}$ concept descriptions.*

An interpretation *is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where the* domain $\Delta^{\mathcal{I}}$ *is a non-empty set and $\cdot^{\mathcal{I}}$ is a function that assigns to every concept name A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every role name r a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. This function is extended to $\mathcal{ALC}$ concept descriptions as follows:*

– $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$;
– $(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid$ *there is a* $y \in \Delta^{\mathcal{I}}$ *with* $(x, y) \in r^{\mathcal{I}}$ *and* $y \in C^{\mathcal{I}}\}$;
– $(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid$ *for all* $y \in \Delta^{\mathcal{I}}$, $(x, y) \in r^{\mathcal{I}}$ *implies* $y \in C^{\mathcal{I}}\}$.

As usual, the Boolean constructors $\sqcap, \sqcup, \neg$ are respectively called *conjunction*, *disjunction*, and *negation*. We call a concept description of the form $\exists r.C$ an *existential restriction*, and a concept description of the form $\forall r.C$ a *value restriction*. In the following, we will us $\top$ as an abbreviation for $A \sqcup \neg A$, where A is an arbitrary concept name (*top concept*, which is always interpreted as the whole domain), $\bot$ as an abbreviation for $\neg\top$ (*bottom concept*, which is always interpreted as the empty set), and $C \Rightarrow D$ as an abbreviation for $\neg C \sqcup D$ (implication).

The following are examples of $\mathcal{ALC}$ concept descriptions that may be of interest in the conference domain. Assume that Participant, Talk, Boring, DL are concept names, and attends, gives, topic are role names. The description

$$\mathsf{Participant} \sqcap \exists \mathsf{attends}.\mathsf{Talk}$$

describes conference participants that attend at least on talk,

$$\mathsf{Participant} \sqcap \forall \mathsf{attends}.(\mathsf{Talk} \sqcap \neg \mathsf{Boring})$$

describes conference participants that attend only non-boring talks, and

$$\mathsf{Speaker} \sqcap \exists \mathsf{gives}.(\mathsf{Talk} \sqcap (\mathsf{Boring} \sqcup \forall \mathsf{topic}.\mathsf{DL}))$$

describes speakers giving a talk that is boring or has as its only topic DL.

Relationship with first-order logic. Given the semantics of $\mathcal{ALC}$ concept descriptions, it is easy to see that $\mathcal{ALC}$ can be viewed as a fragment of first-order predicate logic.[7] Indeed, concept names (which are interpreted as sets) are simply unary predicates, and role names (which are interpreted as binary relations) are simply binary predicates. For a given first-order variable x, an $\mathcal{ALC}$ concept description C is translated into a formula $\tau_x(C)$ with free variable x:

– $\tau_x(A) := A(x)$ for concept names A;

[7] More information about the connection between DLs and first-order predicate logic can be found in [28].

- $\tau_x(C \sqcap D) := \tau_x(C) \wedge \tau_x(D)$;
- $\tau_x(C \sqcup D) := \tau_x(C) \vee \tau_x(D)$;
- $\tau_x(\neg C) := \neg\tau_x(C)$;
- $\tau_x(\forall r.C) := \forall y.(r(x,y) \rightarrow \tau_y(C))$ where y is a variable different from x;
- $\tau_x(\exists r.C) := \exists y.(r(x,y) \wedge \tau_y(C))$ where y is a variable different from x.

Regarding the semantics, any first-order interpretation $\mathcal{I}$ (over the signature consisting of the concept names in N_C as unary predicates and the role names in N_R as binary predicates) can be viewed as an $\mathcal{ALC}$ interpretation and vice versa. Intuitively, the first-order formula $\tau_x(C)$ describes all domain elements $d \in \Delta^{\mathcal{I}}$ that make $\tau_x(C)$ true if x is replaced by them. It is easy to see that this set coincides with the interpretation of the concept description C, i.e.,

$$C^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \mathcal{I} \models \tau_x(C)[x \leftarrow d]\}.$$

The resolution-based approaches for reasoning in DLs are based on such a translation to first-order predicate logic. It should be noted, however, that the translation sketched above does not yield arbitrary first-order formulae. Instead, we obtain formulae belonging to known decidable fragments of first-order predicate logic: the guarded fragment [58] and the two-variable fragment [91, 59]. Intuitively, the formulae of the form $\tau_x(C)$ belong to the guarded fragment since every quantified variable y is guarded by a role $r(x,y)$. Regarding membership in the two-variable fragment, it is easy to see that it is enough to use just two first-order variables x, y in the translation: in τ_x one uses y as the variable different from x, and in τ_y one uses x for this purpose.

Relationship with modal logics. There is also a close connection between DLs and modal logics. In particular, $\mathcal{ALC}$ is just a syntactic variant of the basic multimodal logic K [103], where "*multi*modal" means that one has several pairs of box and diamond operators, which are indexed with the name of the corresponding transition relation. In the following, we assume that the reader is familiar with the basic notions of modal logics (see, e.g., [27] for more details). Intuitively, concept names A correspond to propositional variables a and role names r to names for transition relations r. An $\mathcal{ALC}$ concept description C is translated into a modal formula $\theta(C)$ as follows:

- $\theta(A) := a$ for concept names A;
- $\theta(C \sqcap D) := \theta(C) \wedge \theta(D)$;
- $\theta(C \sqcup D) := \theta(C) \vee \theta(D)$;
- $\theta(\neg C) := \neg\theta(C)$;
- $\theta(\forall r.C) := \Box_r\theta(C))$;
- $\theta(\exists r.C) := \Diamond_r\theta(C))$.

Regarding the semantics, any $\mathcal{ALC}$ interpretation $\mathcal{I}$ can be viewed as a Kripke structure $K_{\mathcal{I}}$ (and vice versa): every element w of $\Delta^{\mathcal{I}}$ is a possible world of $K_{\mathcal{I}}$, the world w makes the propositional variable a true iff $w \in A^{\mathcal{I}}$ for the concept name A corresponding to a, and there is a transition from world w to world

w' with the transition relation r iff $(w, w') \in r^{\mathcal{I}}$. The translation function θ preserves the semantics in the following sense: $C^{\mathcal{I}}$ is the set of worlds that make $\theta(C)$ true in $K_{\mathcal{I}}$.

The translation-based approaches that reduce reasoning in DLs to reasoning in appropriate modal logics are based on (extensions of) this translation.

Additional constructors. $\mathcal{ALC}$ is only one example of a description language. DL researchers have introduced many additional constructors and investigated various description languages obtained by combining such constructors. Here, we only introduce qualified number restrictions as examples for additional concept constructors, and inverse roles as example for a role constructor (see [5] for an extensive list of additional concept and role constructors).

Qualified number restrictions are of the form $(\geq n\, r.C)$ (at-least restriction) and $(\leq n\, r.C)$ (at-most restriction), where $n \geq 0$ is a non-negative integer, $r \in N_R$ is a role name, and C is a concept description. The semantics of these additional constructors is defined as follows:

$$(\geq n\, r.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \mathit{card}(\{e \mid (d, e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}) \geq n\},$$
$$(\leq n\, r.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \mathit{card}(\{e \mid (d, e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}) \leq n\},$$

where $\mathit{card}(X)$ yields the cardinality of the set X. Using qualified number restrictions, we can define the concept of all persons that attend at most 20 talks, of which at least 3 have the topic DL:

$$\mathsf{Person} \sqcap (\leq 20\, \mathsf{attends.Talk}) \sqcap (\geq 3\, \mathsf{attends.}(\mathsf{Talk} \sqcap \exists \mathsf{topic.DL})).$$

The *inverse role* constructor applies to a role name r and yields its inverse r^{-1}, where the semantics is the obvious one, i.e.,

$$(r^{-1})^{\mathcal{I}} := \{(e, d) \mid (d, e) \in r^{\mathcal{I}}\}.$$

Inverse roles can be used like role names within concept descriptions. Using the inverse of the role attends, we can define the concept of a speaker giving a boring talk as

$$\mathsf{Speaker} \sqcap \exists \mathsf{gives.}(\mathsf{Talk} \sqcap \forall \mathsf{attends}^{-1}.(\mathsf{Bored} \sqcup \mathsf{Sleeping})).$$

In the following, we will use the notion "concept description" to refer to a description built using the (concept and role) constructors of some description language. Indeed, the definitions of the other three components of a DL (terminological formalism, assertional formalism, reasoning) is independent of the description language. Accordingly, we will also use the notion "role description" to refer to a role name or a role description (such as r^{-1}) built using the constructors of some description language.

2.2 Terminological knowledge

In its simplest form, a TBox introduces names (abbreviations) for complex descriptions.

$$\begin{aligned}
\mathsf{Woman} &\equiv \mathsf{Person} \sqcap \mathsf{Female}\\
\mathsf{Man} &\equiv \mathsf{Person} \sqcap \neg\mathsf{Female}\\
\mathsf{Talk} &\equiv \exists\mathsf{topic}.\top\\
\mathsf{Speaker} &\equiv \mathsf{Person} \sqcap \exists\mathsf{gives}.\mathsf{Talk}\\
\mathsf{Participant} &\equiv \mathsf{Person} \sqcap \exists\mathsf{attends}.\mathsf{Talk}\\
\mathsf{BusySpeaker} &\equiv \mathsf{Speaker} \sqcap (\geq 3\,\mathsf{gives}.\mathsf{Talk})\\
\mathsf{BadSpeaker} &\equiv \mathsf{Speaker} \sqcap \forall\mathsf{gives}.(\forall\mathsf{attends}^{-1}.(\mathsf{Bored} \sqcup \mathsf{Sleeping}))
\end{aligned}$$

Fig. 1. A TBox for the conference domain

Definition 2. *A* concept definition *is of the form $A \equiv C$ where A is a concept name and C is a concept description. Given a set $\mathcal{T}$ of concept definitions, we say that the concept name A* directly uses *the concept name B if $\mathcal{T}$ contains a concept definition $A \equiv C$ such that B occurs in C. Let* uses *be the transitive closure of the relation "directly uses." We say that $\mathcal{T}$ is* cyclic *if there is a concept name A that uses itself, and* acyclic *otherwise.*

A TBox *is a finite set $\mathcal{T}$ of concept definitions that is acyclic and such that every concept name occurs at most once on the left-hand side of a concept definition in $\mathcal{T}$. Given a TBox $\mathcal{T}$, we call the concept name A a* defined concept *if A occurs on the left-hand side of a definition in $\mathcal{T}$. All other concept names are called* primitive concepts. *An interpretation $\mathcal{I}$ is a* model *of the TBox $\mathcal{T}$ if it satisfies all its concept definitions, i.e., $A^{\mathcal{I}} = C^{\mathcal{I}}$ holds for all $A \equiv C$ in $\mathcal{T}$.*

Fig. 1 shows a small TBox with concept definitions relevant in our example domain. Modern DL systems allow their users to state more general constraints for the interpretation of concepts and roles.

Definition 3. *A* general concept inclusion *axiom (GCI) is of the form $C \sqsubseteq D$ where C, D are concept descriptions. A finite set of GCIs is called a* general TBox.

An interpretation $\mathcal{I}$ is a model *of the general TBox $\mathcal{T}$ if it satisfies all its GCIs, i.e., $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all GCIs $C \sqsubseteq D$ in $\mathcal{T}$.*

Obviously, the concept definition $A \equiv C$ is equivalent (in the sense that is has the same models) to the pair of GCIs $A \sqsubseteq C, C \sqsubseteq A$, which shows that TBoxes can be expressed using general TBoxes. Thus, we assume in the following that the notion of a general TBox subsumes the notion of a TBox. In general, GCIs with complex concept descriptions on their left-hand side cannot be expressed with the help of TBoxes. Using GCIs we can, e.g., say that talks in which all

$$\begin{array}{lll} \mathsf{Lecturer(FRANZ)}, & \mathsf{teaches(FRANZ, Tut03)}, & \\ \mathsf{Tutorial(Tut03)}, & \mathsf{topic(Tut03, ReasoningInDL)}, & \mathsf{DL(ReasoningInDL)} \end{array}$$

Fig. 2. An ABox for the conference domain

attendants are sleeping are boring

$$\mathsf{Talk} \sqcap \forall \mathsf{attends}^{-1}.\mathsf{Sleeping} \sqsubseteq \mathsf{Boring},$$

and that PC chairs cannot as well be authors

$$\mathsf{Author} \sqcap \mathsf{PCchair} \sqsubseteq \bot.$$

In some applications, it also makes sense to consider a generalization of TBoxes where one only allows the use of unambiguous definitions, but dispenses with the acyclicity requirement. Such *cyclic TBoxes* $\mathcal{T}$ are thus finite sets of concept definitions such that every concept name occurs at most once on the left-hand side of a concept definition in $\mathcal{T}$ (see, e.g., [4, 21] for details).

2.3 Assertional knowledge

Assertions can be used to state facts about named individuals. Thus, we assume that there is a third set N_I of names, called *individual names*, which is disjoint with the sets of concept and role names. An interpretation additional assigns an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to every individual name $a \in N_I$.

Definition 4. *Let C be a concept description, r be a role description, and $a, b \in N_I$. An* assertion *is of the form $C(a)$ (concept assertion) or $r(a, b)$ (role assertion). An* ABox *is a finite set of assertions.*

An interpretation $\mathcal{I}$ is a model *of the ABox $\mathcal{A}$ if it satisfies all its assertions, i.e., $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all concept assertions $C(a) \in \mathcal{A}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ holds for all role assertions $r(a, b) \in \mathcal{A}$.*

Fig. 2 shows a small ABox with assertions describing a specific DL tutorial.

2.4 Inference problems

DL systems provide their users with inference capabilities that allow them to derive implicit knowledge from the one explicitly represented. The following are the most important "classical" inference problems supported by DL systems.

Definition 5. *Let $\mathcal{T}$ be a generalized TBox, $\mathcal{A}$ an ABox, C, D concept descriptions, and a an individual name.*

- *C is* subsumed by D w.r.t. $\mathcal{T}$ *($C \sqsubseteq_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$.*
- *C is* equivalent to D w.r.t. $\mathcal{T}$ *($C \equiv_{\mathcal{T}} D$) iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$.*

- C *is* satisfiable w.r.t. $\mathcal{T}$ *iff* $C^{\mathcal{I}} \neq \emptyset$ *for some model* $\mathcal{I}$ *of* $\mathcal{T}$*.*
- $\mathcal{A}$ *is* consistent w.r.t. $\mathcal{T}$ *iff it has a model that is also a model of* $\mathcal{T}$*.*
- a *is an* instance of C w.r.t. $\mathcal{A}$ and $\mathcal{T}$ *(*$\mathcal{A} \models_{\mathcal{T}} C(a)$*) iff* $a^{\mathcal{I}} \in C^{\mathcal{I}}$ *for all models* $\mathcal{I}$ *of* $\mathcal{T}$ *and* $\mathcal{A}$*.*

One might think that, in order to realize the inference component of a DL system, on needs to design and implement five algorithms, each solving one of the above inference problems. Fortunately, this is not the case since there exist the following polynomial time reductions, which only require the availability of the concept constructors conjunction and negation in the description language:

- Subsumption can be be reduced in polynomial time to equivalence:
$$C \sqsubseteq_{\mathcal{T}} D \text{ iff } C \sqcap D \equiv_{\mathcal{T}} C$$
- Equivalence can be be reduced in polynomial time to subsumption:
$$C \equiv_{\mathcal{T}} D \text{ iff } C \sqsubseteq_{\mathcal{T}} D \text{ and } D \sqsubseteq_{\mathcal{T}} C$$
- Subsumption can be be reduced in polynomial time to (un)satisfiability:
$$C \sqsubseteq_{\mathcal{T}} D \text{ iff } C \sqcap \neg D \text{ is unsatisfiable w.r.t. } \mathcal{T}$$
- Satisfiability can be be reduced in polynomial time to (non-)subsumption:
$$C \text{ is satisfiable w.r.t. } \mathcal{T} \text{ iff not } C \sqsubseteq_{\mathcal{T}} \bot$$
- Satisfiability can be be reduced in polynomial time to consistency:
$$C \text{ is satisfiable w.r.t. } \mathcal{T} \text{ iff } \{C(a)\} \text{ is consistent w.r.t. } \mathcal{T}$$
- The instance problem can be reduced in polynomial time to (in)consistency:
$$\mathcal{A} \models_{\mathcal{T}} C(a) \text{ iff } \mathcal{A} \cup \{\neg C(a)\} \text{ is inconsistent w.r.t. } \mathcal{T}$$
- Consistency can be reduced in polynomial time to the (non-)instance problem:
$$\mathcal{A} \text{ is consistent w.r.t. } \mathcal{T} \text{ iff } \mathcal{A} \not\models_{\mathcal{T}} \bot(a)$$

Thus, if one is only interested in terminological reasoning (i.e., satisfiability, equivalence, and subsumption), it is enough to have a satisfiability algorithm. If one is additionally interested in assertional reasoning (i.e., consistency and instance), then it is enough to have a consistency algorithm.

Another important observation is that reasoning w.r.t. a normal (i.e., not general) TBox can be reduced to reasoning w.r.t. the empty TBox.[8] Intuitively,

[8] Instead of saying "w.r.t. $\emptyset$" one usually says "without a TBox," and omits the index $\mathcal{T}$ for subsumption, equivalence, and instance, i.e., writes $\equiv$, $\sqsubseteq$, $\models$ instead of $\equiv_{\mathcal{T}}$, $\sqsubseteq_{\mathcal{T}}$, and $\models_{\mathcal{T}}$.

$$\begin{aligned} A_0 &\equiv \forall r.A_1 \sqcap \forall s.A_1 \\ A_1 &\equiv \forall r.A_2 \sqcap \forall s.A_2 \\ &\vdots \\ A_{n-1} &\equiv \forall r.A_n \sqcap \forall s.A_n \end{aligned}$$

Fig. 3. A TBox $\mathcal{T}$ that causes exponential blow-up during expansion

TBoxes merely state that defined concepts are abbreviations for certain complex concept descriptions. These complex descriptions can be made explicit by *expanding* the definitions from $\mathcal{T}$: given a concept description C, its expansion $exp(C, \mathcal{T})$ w.r.t. $\mathcal{T}$ is obtained by exhaustively replacing all defined concept names A occurring on the left-hand side of concept definitions $A \equiv C$ by their defining concept descriptions C. For example, w.r.t. the TBox of Fig. 1, the concept description $\mathsf{Woman} \sqcap \mathsf{BusySpeaker}$ is expanded to

$$\mathsf{Person} \sqcap \mathsf{Female} \sqcap \mathsf{Person} \sqcap \exists \mathsf{gives.Talk} \sqcap (\geq 3\,\mathsf{gives.Talk}),$$

which is equivalent to $\mathsf{Person} \sqcap \mathsf{Female} \sqcap (\geq 3\,\mathsf{gives.Talk})$.

It is easy to show that $C \sqsubseteq_{\mathcal{T}} D$ iff $exp(C, \mathcal{T}) \sqsubseteq exp(D, \mathcal{T})$. Similar reduction are possible for the other inference problems. It should be noted, however, that these reductions are in general exponential. For example, expanding the concept description A_0 w.r.t. the TBox of Fig. 3 yields an expanded description $exp(A_0, \mathcal{T})$ that contains the concept name A_n 2^n times. This exponential blow-up can sometimes be avoided by devising satisfiability algorithms that explicitly take acyclic TBoxes into account. For example, satisfiability of $\mathcal{ALC}$ concept descriptions w.r.t. TBoxes is PSPACE-complete, and without TBoxes this problem is of exactly the same complexity [107, 83]. However this is not always the case: in Section 4, we will introduce the DL $\mathcal{FL}_0$, for which reasoning w.r.t. TBoxes is considerably more difficult than reasoning without them [94].

For some expressive DLs it is possible to reduce reasoning w.r.t. a general TBox to reasoning without a TBox [10, 70], but for $\mathcal{ALC}$ this is not possible, i.e., one really needs to design algorithms that take GCIs into account.

Compound inferences. Some of the most important inference problems in DLs are of a compound nature in the sense that, in principle, they can be reduced to multiple invocations of the more basic inference problems mentioned above. However, when the goal is to achieve an efficient implementation, it is vital to consider compound inferences as first-class citizens since a naïve reduction to the basic inference problems may be too inefficient [12]. Here, we define two of these compound inference problems, but do not deal with the efficiency issue.

Classification. Given a (general) TBox $\mathcal{T}$, compute the restriction of the subsumption relation "$\sqsubseteq_{\mathcal{T}}$" to the set of concept names used in $\mathcal{T}$.

Realization. Given an ABox $\mathcal{A}$, a (general) TBox $\mathcal{T}$, and an individual name a, compute the set $R_{\mathcal{A},\mathcal{T}}(a)$ of those concept names A that are used in $\mathcal{T}$, satisfy

$\mathcal{A} \models_\mathcal{T} A(a)$, and are minimal with this property w.r.t. the subsumption relation "$\sqsubseteq_\mathcal{T}$".

Complexity of reasoning. In the 1980ies, it was a commonly held belief that reasoning in knowledge representation systems should be tractable, i.e., of polynomial time complexity. The precursor of all DL systems, KLONE [33], as well as its early successor systems, like K-REP [88], BACK [100], and LOOM [87], indeed employed polynomial-time subsumption algorithms. Later on, however, it turned out that subsumption in rather inexpressive DLs may be intractable [82], that subsumption in KLONE is even undecidable [106], and that subsumption w.r.t. a TBox in a description language with conjunction ($\sqcap$) and value restriction ($\forall r.C$)[9] is intractable [94]. The reason for the discrepancy between the complexity of the subsumption algorithms employed in the above mention early DL systems and the worst-case complexity of the subsumption problems these algorithms were supposed to solve was, as mentioned in the introduction, due to the fact that these systems employed sound, but incomplete subsumption algorithms, i.e., algorithms whose positive answers to subsumption queries are correct, but whose negative answers may be incorrect.

The use of incomplete algorithms has since then largely been abandoned in the DL community, mainly because of the problem that the behavior of the systems is no longer determined by the semantics of the description language: an incomplete algorithm may claim that a subsumption relationship does not hold, although it should hold according to the semantics. This left the DL community with two ways out of the complexity dilemma:

– Employ expressive DLs with sound and complete, but intractable inference procedures.
– Employ inexpressive DLs that allow the use of sound, complete, and tractable inference procedures.

In the next two sections, we treat these two approaches in more detail.

It should be noted that here we have barely scratched the surface of the research on the complexity of reasoning in DLs. Indeed, DL researchers have investigated the complexity of reasoning in a great variety of DLs in detail. Giving an overview of the results obtained in this direction in the last 20 years is beyond the scope of this article. We refer the reader to overview articles such as [47, 18] and the Description Logic Complexity Navigator[10] for more details.

3 Reasoning in expressive DLs

As mentioned in the introduction, a variety of of reasoning techniques have been introduced for expressive DLs. Here, we describe tableau-based and automata-

[9] All the systems mentioned above supported these two concept constructors, which were at that time viewed as being indispensable for a DL.

[10] http://www.cs.man.ac.uk/∼ezolin/dl/

based approaches in some detail, but do not treat approaches based on translations to first-order or modal logic.

Before looking at specific inference procedures in detail, let us first state some general requirements on the behavior of such procedures:

- The procedure should be a *decision procedure* for the problem, which means that it should be:
  - *sound*, i.e., the positive answers should be correct;
  - *complete*, i.e., the negative answers should be correct;
  - *terminating*, i.e., it should always give an answer in finite time
- The procedure should be as *efficient* as possible. Preferably, it should be optimal w.r.t. the (worst-case) complexity of the problem.
- The procedure should be *practical*, i.e., easy to implement and optimize, and behave well in applications.

Both tableau-based and automata-based approaches to reasoning in DLs yield decision procedures. Tableau-based approaches often yield practical procedures: optimized implementations of such procedures have turned out to behave quite well in applications even for expressive DLs with a high worst-case complexity. However, these practical procedures are often not optimal w.r.t. the worst-case complexity of the problem: in particular, satisfiability in $\mathcal{ALC}$ w.r.t. general TBoxes is ExpTime-complete, but it is very hard to design a tableau-based procedure for it that runs in deterministic exponential time. In contrast, it is quite easy to design an ExpTime automata-based procedure for this problem, but there are no practical implementations for this procedure.

3.1 Tableau-based approaches

The most widely used reasoning technique for DLs is the tableau-based approach, which was first introduced in the context of DLs by Schmidt-Schauß and Smolka [108], though it had already been used for modal logics long before that [53]. In this section, we first describe this technique for the case of consistency of an ABox (without a TBox[11]) in our basic DL $\mathcal{ALC}$. Then we show how the approach can be extended to deal with qualified number restrictions and with general TBoxes.

Given an $\mathcal{ALC}$ ABox $\mathcal{A}_0$, the tableau algorithm for consistency tries to construct a finite interpretation $\mathcal{I}$ that is a model of $\mathcal{A}_0$. Before we can describe the algorithm more formally, we need to introduce an appropriate data structure in which to represent the (partial descriptions of) finite interpretations that are generated during the run of the algorithm. The original paper by Schmidt-Schauß and Smolka [108], and also many other papers on tableau algorithms for DLs, introduce the new notion of a constraint system for this purpose. However, if we look at the information that must be expressed (namely, the elements

[11] As mentioned above, inference problems w.r.t. a TBox can be reduced to the corresponding ones without TBoxes by expanding the concept definitions from $\mathcal{T}$.

of the interpretation, the concept descriptions they belong to, and their role relationships), we see that ABox assertions are sufficient for this purpose.

It will be convenient to assume that all concept descriptions are in *negation normal form* (NNF), i.e., that negation occurs only directly in front of concept names. Using de Morgan's rules and the usual rules for quantifiers, any $\mathcal{ALC}$ concept description can be transformed (in linear time) into an equivalent description in NNF. An ABox is in NNF if all the concept descriptions occurring in it are in NNF.

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

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

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

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

Fig. 4. Tableau rules of the consistency algorithm for $\mathcal{ALC}$.

Let $\mathcal{A}_0$ be an $\mathcal{ALC}$ ABox in NNF. In order to test consistency of $\mathcal{A}_0$, the algorithm starts with $\mathcal{A}_0$, and applies consistency preserving transformation rules (see Fig. 4) to this ABox. The transformation rule that handles disjunction is *nondeterministic* in the sense that a given ABox is transformed into two new ABoxes such that the original ABox is consistent iff *one of* the new ABoxes is so. For this reason we will consider finite sets of ABoxes $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$ instead of single ABoxes. Such a set is *consistent* iff there is some i, $1 \leq i \leq k$, such that $\mathcal{A}_i$ is consistent. A rule of Fig. 4 is applied to a given finite set of ABoxes $\mathcal{S}$ as follows: it takes an element $\mathcal{A}$ of $\mathcal{S}$, and replaces it by one ABox $\mathcal{A}'$ or by two ABoxes $\mathcal{A}'$ and $\mathcal{A}''$.

Definition 6. *An ABox $\mathcal{A}$ is called* complete *iff none of the transformation rules of Fig. 4 applies to it. The ABox $\mathcal{A}$ contains a* clash *iff $\{A(x), \neg A(x)\} \subseteq \mathcal{A}$ for some individual name x and some concept name A. An ABox is called* closed *if it contains a clash, and* open *otherwise.*

The *consistency algorithm for $\mathcal{ALC}$* works as follows. It starts with the singleton set of ABoxes $\{\mathcal{A}_0\}$, and applies the rules of Fig. 4 (in arbitrary order) until no more rules apply. It answers "consistent" if the set $\widehat{\mathcal{S}}$ of ABoxes obtained this way contains an open ABox, and "inconsistent" otherwise. The fact that this algorithm is a decision procedure for consistency of $\mathcal{ALC}$ ABoxes is an easy consequence of the following lemma.

Lemma 1. *Let $\mathcal{A}_0$ be an $\mathcal{ALC}$ ABox in negation normal form.*

1. Local correctness: *the rules preserve consistency, i.e., if $\mathcal{S}'$ is obtained from the finite set of ABoxes $\mathcal{S}$ by application of a transformation rule, then $\mathcal{S}$ is consistent iff $\mathcal{S}'$ is consistent.*
2. Termination: *there cannot be an infinite sequence of rule applications*

$$\{\mathcal{A}_0\} \rightarrow \mathcal{S}_1 \rightarrow \mathcal{S}_2 \rightarrow \cdots .$$

3. Soundness:[12] *any complete and open ABox $\mathcal{A}$ is consistent.*
4. Completeness:[13] *any closed ABox $\mathcal{A}$ is inconsistent.*

Proof. 1. Local correctness: We treat the $\rightarrow_\exists$-rule and the $\rightarrow_\sqcup$-rule in detail. The other rules can be handled similarly.

First, assume that $\mathcal{S}'$ is obtained from $\mathcal{S}$ by an application of the $\rightarrow_\exists$-rule. Then there is an ABox $\mathcal{A} \in \mathcal{S}$ containing an assertion of the form $(\exists r.C)(x)$, and $\mathcal{S}'$ is obtained from $\mathcal{S}$ by replacing $\mathcal{A}$ by $\mathcal{A}' := \mathcal{A} \cup \{C(y), r(x,y)\}$ where y is an individual name not occurring in $\mathcal{A}$.

Obviously, it is enough to show that $\mathcal{A}$ has a model iff $\mathcal{A}'$ has a model. The if-direction is trivial since $\mathcal{A} \subseteq \mathcal{A}'$. To show the only-if direction, assume that $\mathcal{I}$ is a model of $\mathcal{A}$. Since $(\exists r.C)(x) \in \mathcal{A}$, there is a $d \in \Delta^{\mathcal{I}}$ such that

$$(x^{\mathcal{I}}, d) \in r^{\mathcal{I}} \quad \text{and} \quad d \in C^{\mathcal{I}}.$$

Let $\mathcal{I}'$ be the interpretation that coincides with $\mathcal{I}$, with the exception that $y^{\mathcal{I}'} = d$. Since y does not occur in $\mathcal{A}$, $\mathcal{I}'$ is a model of $\mathcal{A}$. By the definition of $y^{\mathcal{I}'}$, it is also a model of $\{r(x,y), C(y)\}$, and thus of $\mathcal{A}'$.

Second, assume that $\mathcal{S}'$ is obtained from $\mathcal{S}$ by an application of the $\rightarrow_\sqcup$-rule. Then there is an ABox $\mathcal{A} \in \mathcal{S}$ containing an assertion of the form $(C_1 \sqcup C_2)(x)$, and $\mathcal{S}'$ is obtained from $\mathcal{S}$ by replacing $\mathcal{A}$ by $\mathcal{A}' := \mathcal{A} \cup \{C_1(x)\}$ and $\mathcal{A}'' := \mathcal{A} \cup \{C_2(x)\}$.

It is enough to show that $\mathcal{A}$ has a model iff $\mathcal{A}'$ has a model or $\mathcal{A}''$ has a model. The if-direction is again trivial since $\mathcal{A} \subseteq \mathcal{A}'$ and $\mathcal{A} \subseteq \mathcal{A}''$. To show the

[12] Recall that *soundness* means that the positive answers of the algorithm are correct, i.e., if the algorithm says "consistent," then the input ABox $\mathcal{A}_0$ is indeed consistent. This follows from the part 3. of the lemma together with part 1. (local correctness).

[13] Recall that *completeness* means that the negative answers of the algorithm are correct, i.e., if the algorithm says "inconsistent," then the input ABox $\mathcal{A}_0$ is indeed inconsistent. This follows from the part 4. of the lemma together with part 1. (local correctness).

only-if direction, assume that $\mathcal{I}$ is a model of $\mathcal{A}$. Since $(C_1 \sqcup C_2)(x) \in \mathcal{A}$, we have

$$x^{\mathcal{I}} \in (C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}.$$

If $x^{\mathcal{I}} \in C_1^{\mathcal{I}}$, then $\mathcal{I}$ is a model of $\mathcal{A}'$. If $x^{\mathcal{I}} \in C_2^{\mathcal{I}}$, then $\mathcal{I}$ is a model of $\mathcal{A}''$.

2. Termination: Define the label $\mathcal{L}_{\mathcal{A}}(x)$ of an individual name x in an ABox $\mathcal{A}$ to consist of the concept descriptions in concept assertions for x, i.e.,

$$\mathcal{L}_{\mathcal{A}}(x) := \{C \mid C(x) \in \mathcal{A}\}.$$

Let $\mathcal{S}$ be a set of ABoxes reached by a finite number of rule applications, starting with $\{\mathcal{A}_0\}$, and let $\mathcal{A} \in \mathcal{S}$. The following are easy consequences of the definition of the tableau rules.

1. rule application is monotonic, i.e., every application of a rule to $\mathcal{A}$ extends the label of an individual, by adding a new concept assertion, and does not remove any element from a label;
2. concept descriptions occurring in labels in $\mathcal{A}$ are subdescriptions of concept descriptions occurring in the initial ABox $\mathcal{A}_0$.

Clearly, these two facts imply that there can only be a finite number of rule applications per individual. Thus, it remains to show that the number of newly introduced individuals in a chain of rule applications is bounded as well. Let us call an individual name occurring in $\mathcal{A}$ a *new* individual if it is not one of the individuals already present in $\mathcal{A}_0$. We say that y is an r-successor of x if $r(x, y) \in \mathcal{A}$.

3. for a given individual x, an existential restriction in the label of x can trigger at most one introduction of a new individual, and thus the number of new individuals that are r-successors of an individual in $\mathcal{A}$ is bounded by the number of existential restrictions in $\mathcal{A}_0$;
4. the length of successor chains of new individuals in $\mathcal{A}$ is bounded by the maximal size of the concept descriptions occurring in $\mathcal{A}_0$. This is an immediate consequence of the following two facts:
   - if x is a new individual in $\mathcal{A}$, then it has a unique predecessor y
   - the maximal size of concept descriptions in $\mathcal{L}_{\mathcal{A}}(x)$ is strictly smaller than the maximal size of concept descriptions in $\mathcal{L}_{\mathcal{A}}(y)$

Facts 3. and 4. yield an overall bound on the number of new individuals in $\mathcal{A}$. Since only a finite number of individuals can be introduced during rule application, and only finitely many rules can be applied to a fixed individual, this shows that overall we can have only a finite number of rule applications, which completes the proof of termination.

3. Soundness: Let $\mathcal{A}$ be a complete and open ABox. To prove that $\mathcal{A}$ is consistent, we define the *canonical interpretation* $\mathcal{I}_{\mathcal{A}}$, and show that it is a model of $\mathcal{A}$:

1. The domain $\Delta^{\mathcal{I}_{\mathcal{A}}}$ of $\mathcal{I}_{\mathcal{A}}$ consists of the individual names occurring in $\mathcal{A}$.
2. For all individual names x we define $x^{\mathcal{I}_{\mathcal{A}}} := x$.

3. For all concept names A we define $A^{\mathcal{I}_\mathcal{A}} := \{x \mid A(x) \in \mathcal{A}\}$.
4. For all role names r we define $r^{\mathcal{I}_\mathcal{A}} := \{(x,y) \mid r(x,y) \in \mathcal{A}\}$.

By definition, $\mathcal{I}_\mathcal{A}$ satisfies all the role assertions in $\mathcal{A}$. To prove that $\mathcal{I}_\mathcal{A}$ satisfies the concept assertions as well, we consider $C(x) \in \mathcal{A}$ and show $x^{\mathcal{I}_\mathcal{A}} = x \in C^{\mathcal{I}_\mathcal{A}}$ by *induction* on the size of C:

– $C = A$ for $A \in N_C$: $x \in A^{\mathcal{I}_\mathcal{A}}$ is an immediate consequence of the definition of $A^{\mathcal{I}_\mathcal{A}}$.
– $C = \neg A$ for $A \in N_C$: since $\mathcal{A}$ is *open*, $A(x) \notin \mathcal{A}$, and thus $x \notin A^{\mathcal{I}_\mathcal{A}}$ by the definition of $A^{\mathcal{I}_\mathcal{A}}$.
– $C = C_1 \sqcap C_2$: since $\mathcal{A}$ is *complete*, $(C_1 \sqcap C_2)(x) \in \mathcal{A}$ implies that $C_1(x) \in \mathcal{A}$ and $C_2(x) \in \mathcal{A}$; by *induction*, this yields $x \in C_1^{\mathcal{I}_\mathcal{A}}$ and $x \in C_2^{\mathcal{I}_\mathcal{A}}$, and thus $x \in (C_1 \sqcap C_2)^{\mathcal{I}_\mathcal{A}}$.
– the other constructors can be treated similarly.

4. Completeness: the fact that a closed ABox cannot have a model is an immediate consequence of the definition of a clash. □

Theorem 1. *The tableau algorithm introduced above is a decision procedure for consistency of $\mathcal{ALC}$ ABoxes.*

Proof. Started with a finite $\mathcal{ALC}$ ABox $\mathcal{A}_0$ in NNF, the algorithm always terminates with a finite set of complete ABoxes $\mathcal{A}_1, \dots, \mathcal{A}_n$. Local correctness implies that $\mathcal{A}_0$ is consistent iff one of $\mathcal{A}_1, \dots, \mathcal{A}_n$ is consistent.

If the algorithm answers "inconsistent," then all the ABoxes $\mathcal{A}_1, \dots, \mathcal{A}_n$ are closed. Completeness then yields that all the ABoxes $\mathcal{A}_1, \dots, \mathcal{A}_n$ are inconsistent, and thus $\mathcal{A}_0$ is inconsistent, by local correctness.

If the algorithm answers "consistent," then one of the complete ABoxes $\mathcal{A}_1, \dots, \mathcal{A}_n$, say $\mathcal{A}_i$, is open. Soundness then yields that $\mathcal{A}_i$ is consistent, and thus $\mathcal{A}_0$ is consistent, by local correctness.

To sum up, we have shown that the algorithm always terminates, and that both the positive answers ("consistent") and the negative answers ("inconsistent") are correct. □

Adding qualified number restrictions. The description language obtained from $\mathcal{ALC}$ by adding qualified number restrictions is called $\mathcal{ALCQ}$. In order to transform also $\mathcal{ALCQ}$ ABoxes into negation normal form, we additionally use the following equivalence preserving rules:

$$\begin{aligned}
\neg(\geq n+1\, r.C) &\rightsquigarrow (\leq n\, r.C) \\
\neg(\geq 0\, r.C) &\rightsquigarrow \bot \\
\neg(\leq n\, r.C) &\rightsquigarrow (\geq n+1\, r.C)
\end{aligned}$$

In the following, we assume that all $\mathcal{ALCQ}$ ABoxes are in NNF.

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

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

Fig. 5. Tableau rules for qualified number restrictions.

The main idea underlying the extension of the tableau algorithm for $\mathcal{ALC}$ to $\mathcal{ALCQ}$ is quite simple. At-least restrictions are treated by generating the required role successors as new individuals. At-most restrictions that are currently violated are treated by (non-deterministically) identifying some of the role successors. To avoid running into a generate-identify cycle, we introduce explicit inequality assertions that prohibit the identification of individuals that were introduced to satisfy the same at-least restriction. This use of inequality assertions also creates new types of clashes, which occur when an at-most restriction requires some identification, but all identifications are prohibited by inequality assertions.

To be more precise, the tableau algorithm for consistency of $\mathcal{ALC}$ ABoxes is extended to $\mathcal{ALCQ}$ as follows:

– For each of the new concept constructors, we add a *new tableau rule:* the $\rightarrow_{\geq}$-rule and the $\rightarrow_{\leq}$-rule are shown in Fig. 5.
– In the formulation of these rules, we have used *inequality assertions*, which are of the form $x \not\doteq y$ for individual names x, y, and have the obvious semantics that an interpretation $\mathcal{I}$ satisfies such an assertion iff $x^{\mathcal{I}} \neq y^{\mathcal{I}}$.
– Finally, there are *new types of clashes*:
  - $x \not\doteq x \in \mathcal{A}$ for an individual name x.
  - $\{(\leq n\, r.C)(x)\} \cup \{r(x, y_i), C(y_i) \mid 1 \leq i \leq n+1\} \cup \{y_i \not\doteq y_j \mid 1 \leq i < j \leq n+1\} \subseteq \mathcal{A}$ for individual names $x, y_1, \ldots, y_{n+1}$, an $\mathcal{ALCQ}$ concept description C, a role name r, and a non-negative integer n.

The main question is then, of course, whether this extended algorithm really yields a decision procedure for consistency of $\mathcal{ALCQ}$ ABoxes. To prove this, it would be enough to show that the four properties stated in Lemma 1 also hold for the extended algorithm. Local correctness and completeness are easy to show. Unfortunately, neither soundness nor termination hold.

To see that the algorithm is not sound, consider the ABox

$$\mathcal{A}_0 := \{(\geq 3\,\mathsf{child}.\top)(x),\ (\leq 1\,\mathsf{child}.\mathsf{Female})(x),\ (\leq 1\,\mathsf{child}.\neg\mathsf{Female})(x)\}.$$

Obviously, this ABox is inconsistent, but the algorithm does not find this out. In fact, it would introduce three new individuals y_1, y_2, y_3 as r-successors of x, each belonging to $\top$. In an interpretation, the element $y_i^{\mathcal{I}}$ $(i = 1, 2, 3)$ belongs to either $\mathsf{Female}^{\mathcal{I}}$ or to $(\neg\mathsf{Female})^{\mathcal{I}}$, but in the ABox $\mathcal{A}_1$ obtained by applying the $\rightarrow_{\geq}$-rule to $\mathcal{A}_0$, the only concept assertion for y_i $(i = 1, 2, 3)$ is $\top(y_i)$. Thus, the $\rightarrow_{\leq}$-rule does not apply to $\mathcal{A}_1$, and $\mathcal{A}_1$ also does not contain a clash. The ABox $\mathcal{A}_1$ is thus complete and open, but it is not consistent.

The soundness problem illustrated by this example can be avoided by adding as a third rule the $\rightarrow_{\text{choose}}$-rule shown in Fig. 6, where $\sim C$ denotes the negation normal form of C. It is easy to show that this rule preserves local correctness, and that its addition allows us to regain soundness.

The $\rightarrow_{\text{choose}}$-rule
Condition: $\mathcal{A}$ contains $(\leq n\,r.C)(x)$ and $r(x, y)$, but neither $C(y)$ nor $\neg C(y)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C(y)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{\sim C(y)\}$.

Fig. 6. The $\rightarrow_{\text{choose}}$-rule for qualified number restrictions.

However, we still need to deal with the termination problem. This problem is illustrated in the following example. Consider the ABox

$$\mathcal{A}_0 := \{A(a),\ r(a, a),\ (\exists r.A)(a),\ (\leq 1\,r.\top)(a),\ (\forall r.\exists r.A)(a), r(a, x),\ A(x)\}.$$

The $\rightarrow_{\forall}$-rule can be used to add the assertion $(\exists r.A)(x)$, which yields the new ABox

$$\mathcal{A}_1 := \mathcal{A}_0 \cup \{(\exists r.A)(x)\}.$$

This triggers an application of the $\rightarrow_{\exists}$-rule to x. Thus, we obtain the new ABox

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

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

$$\mathcal{A}_3 := \{A(a),\ r(a, a),\ (\exists r.A)(a),\ (\leq 1\,r.\top)(a),\ (\forall r.\exists r.A)(a), r(a, y),\ A(y)\},$$

Except for the individual names (i.e., y instead of x), $\mathcal{A}_3$ is identical to $\mathcal{A}_1$. For this reason, we can continue as above to obtain an infinite chain of rule applications.

We can easily regain termination by requiring that *generating rules* (i.e., the rules $\rightarrow_{\exists}$ and $\rightarrow_{\geq}$, which generate new individuals) may only be applied if none of the other rules is applicable. In the above example, this strategy would prevent the application of the $\rightarrow_{\exists}$-rule to x in the ABox $\mathcal{A}_1$ since the $\rightarrow_{\leq}$-rule is also applicable. After applying the $\rightarrow_{\leq}$-rule (which replaces x by a), the $\rightarrow_{\exists}$-rule is no longer applicable since a already has an r-successor that belongs to A.

Consistency w.r.t. general TBoxes. Let $\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$ be a general TBox. It is easy to see that the general TBox consisting of the single GCI

$$\top \sqsubseteq (\neg C_1 \sqcup D_1) \sqcap \ldots \sqcap (\neg C_n \sqcup D_n)$$

is equivalent to $\mathcal{T}$ in the sense that it has the same models. Thus, it is sufficient to deal with the case where the general TBox consists of a single GCI of the form $\top \sqsubseteq C$ for a concept description C. Obviously, this GCI says that every element of the model belongs to C. Thus, to reason w.r.t. a general TBox consisting of this GCI, it makes sense to add a new rule, the $\rightarrow_{\top\sqsubseteq C}$-rule, which adds the concept assertion $C(x)$ in case the individual name x occurs in the ABox, and this assertion is not yet present.

Does the addition of the $\rightarrow_{\top\sqsubseteq C}$-rule yield a decision procedure for ABox consistency w.r.t. the general TBox $\{\top \sqsubseteq C\}$? Local correctness, soundness, and completeness can indeed easily be shown, but the procedure does not terminate, as illustrated by the following example. Consider the ABox $\mathcal{A}_0 := \{(\exists r.A)(x_0)\}$, and assume that we want to test its consistency w.r.t. the general TBox $\{\top \sqsubseteq \exists r.A\}$. The procedure generates an infinite sequence of ABoxes $\mathcal{A}_1, \mathcal{A}_2, \ldots$ and individuals $x_1, x_2, \ldots$ such that $\mathcal{A}_{i+1} := \mathcal{A}_i \cup \{r(x_i, x_{i+1}),\ A(x_{i+1}),\ (\exists r.A)(x_{i+1})\}$. Since all individuals x_i ($i \geq 1$) receive the same concept assertions as x_1, we may say that the procedure has run into a cycle.

Termination can be regained by using a mechanism that detects cyclic computations, and then blocking the application of generating rules: the application of the $\rightarrow_{\exists}$- and the $\rightarrow_{\geq}$-rule to an individual x is *blocked* by an individual y in an ABox $\mathcal{A}$ iff $\mathcal{L}_{\mathcal{A}}(x) \subseteq \mathcal{L}_{\mathcal{A}}(y)$.[14] The main idea underlying blocking is that the blocked individual x can use the role successors of y instead of generating new ones. For example, instead of generating a new r-successor for x_2 in the above example, one can simply use the r-successor of x_1. This yields an interpretation $\mathcal{I}$ with $\Delta^{\mathcal{I}} := \{x_0, x_1, x_2\}$, $A^{\mathcal{I}} := \{x_1, x_2\}$, and $r^{\mathcal{I}} := \{(x_0, x_1), (x_1, x_2), (x_2, x_2)\}$. Obviously, $\mathcal{I}$ is a model of both $\mathcal{A}_0$ and the general TBox $\{\top \sqsubseteq \exists r.A\}$.

To avoid cyclic blocking (of x by y and vice versa), we consider an enumeration of all individual names, and require that an individual x may only be blocked by individuals y that occur before x in this enumeration. This, together with some other technical assumptions, makes sure that a tableau algorithm using this notion of blocking is sound and complete as well as terminating both for $\mathcal{ALC}$ and $\mathcal{ALCQ}$ (see, e.g., [37, 9] for details).

Complexity of reasoning. For $\mathcal{ALC}$, the satisfiability and the consistency problem (without TBox) are PSpace-complete [107, 64]. The tableau algorithm

[14] Recall that $\mathcal{L}_{\mathcal{A}}(z) = \{C \mid C(z) \in \mathcal{A}\}$ for any individual z occurring in $\mathcal{A}$.

as described above needs exponential space, but it can be modified such that it needs only polynomial space [107]. Both TBoxes [83] and qualified number restrictions [65, 113] can be added without increasing the complexity. W.r.t. general TBoxes, the satisfiability and the consistency problem are ExpTime-complete [103]. However, it is not easy to show the ExpTime-upper bound using tableau algorithms, though it is in principle possible [48, 56]. As we will see in the next section, automata-based algorithms are well-suited to show such ExpTime-upper bounds. The tableau algorithms implemented in systems like FaCT, Racer, and Pellet are not worst-case optimal, but they are nevertheless highly optimized and behave quite well on large knowledge bases from applications.

3.2 Automata-based approaches

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 may be developed with different intentions in mind, such as using it for an optimized implementation or using it 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. As mentioned above, the tableau-based approach often yields practical algorithms, whereas it is not well-suited for proving ExpTime-upper bounds. In contrast, such upper bounds can often be shown in a very elegant way using automata-based approach [41, 86, 84, 114].[15]

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 problems can be reduced to satisfiability.[16] 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 description 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.

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

[16] Using the so-called pre-completion technique [64], this is also the case for inference problems involving ABoxes.

Below, we will use a simple form of non-deterministic automata working on infinite trees of fixed arity, so-called *looping automata* [116]. 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 [92], 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 [26, 76], 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 [3]), the simple looping automata introduced below are not sufficient since one needs additional acceptance conditions such as the Büchi condition [112] (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 description C satisfiable w.r.t. a general $\mathcal{ALC}$ 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 above, applied to the ABox $\{C(x)\}$ w.r.t. the representation of the general TBox $\mathcal{T}$ as a single GCI, 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 of modal logic [27], 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 [80] (which it inherits from its syntactic variant multimodal K) 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. 7, and refer the reader to [27] for formal definitions and proofs. The graph on the left-hand side of Fig. 7 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

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

The graph on the right-hand side of Fig. 7 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 $a \in A^{\widehat{\mathcal{I}}}$.

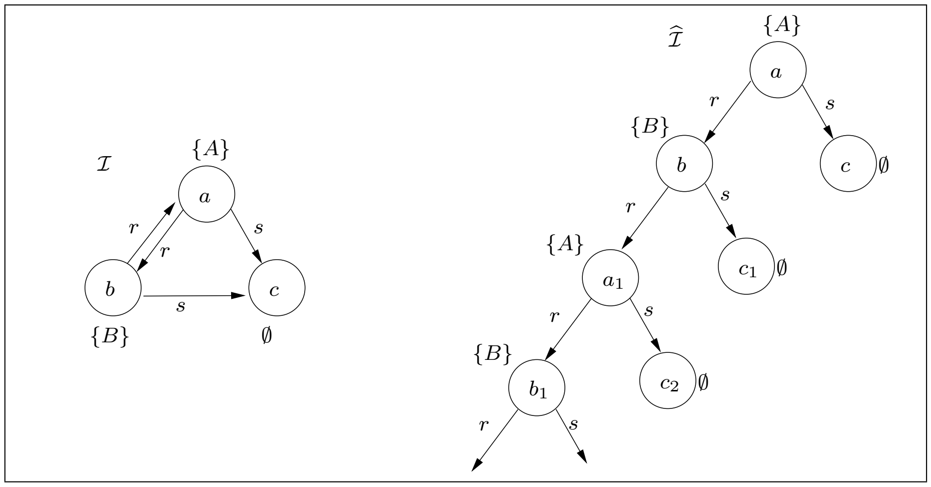

Fig. 7. Unraveling of a model into a tree-shaped model.

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 unlabeled, 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 $T : \{0, \dots, k-1\}^* \to \Sigma$. Thus, nodes are represented as words over $\{0, \dots, k-1\}$, the root is the word ε, and a node u has exactly k successor nodes $u0, \dots, u(k-1)$, 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. 8) is formally represented as the mapping

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

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 $(q, a, q_1, \dots, q_k) \in \Delta$ in the form $(q, a) \to (q_1, \dots, q_k)$.

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

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

- $Q = \{q_0, q_1, q_2\}$ and $I = \{q_0\}$;

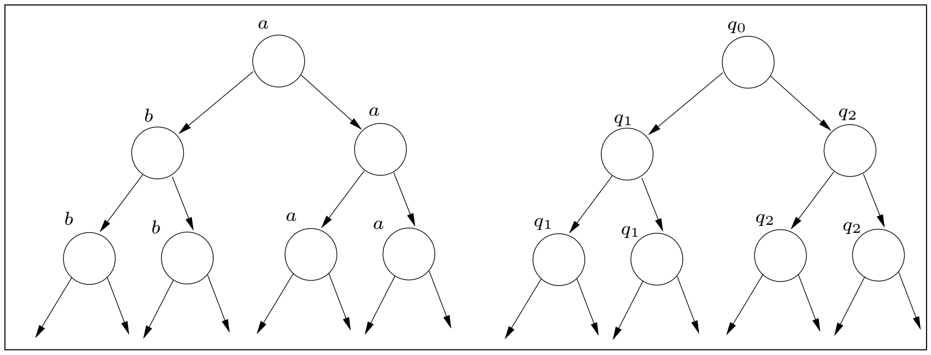

Fig. 8. A tree and a run on it.

- $\Sigma = \{a, b\}$;
- 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. 8 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 8.

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 choosing 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 [25].

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 restrictions occurring in C and $\mathcal{T}$. The number of these restrictions determines k. Using the bisimulation invariance of $\mathcal{ALC}$ [80], 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 description 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 description C and every concept D in a GCI $\top \sqsubseteq D$ into negation normal form as described in Section 3.1. 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. 7 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, and the *transition relation* Δ consists of those transitions $(q, \sigma) \to (q_1, \ldots, q_k)$ 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 1. *The $\mathcal{ALC}$ concept description C is satisfiable w.r.t. the 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. ExpTime-hardness of this problem can be shown by adapting the proof of ExpTime-hardness of satisfiability in propositional dynamic logic (PDL) in [52].

Theorem 2. *Satisfiability in $\mathcal{ALC}$ w.r.t. general TBoxes is* ExpTime*-complete.*

4 Reasoning in the light-weight DLs $\mathcal{EL}$ and $\mathcal{FL}_0$

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 expansion of concept definitions, which can cause an exponential blow-up of the size of concepts. Nebel's coNP-hardness result [94] for subsumption w.r.t. 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 [6, 34, 7, 8] on the complexity of reasoning in DLs with existential restrictions, rather than value restrictions, have led to a partial rehabilitation of structural approaches and light-weight DLs with polynomial reasoning problems (see the description of *Phase 5* in the introduction).

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 $\mathcal{FL}_0$ [4], which offers the concept constructors conjunction, value restriction ($\forall r.C$), and the top concept;
- the DL $\mathcal{EL}$ [7], 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[17] 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 behavior 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 $\mathcal{FL}_0$-concept descriptions 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.[18]

By using the rewrite rule $\forall r.(C \sqcap D) \rightarrow \forall r.C \sqcap \forall r.D$ together with associativity, commutativity and idempotence[19] of $\sqcap$, any $\mathcal{FL}_0$-concept can be transformed

[17] 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.

[18] A comparison between the two approaches can be found in [17].

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

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 $\mathcal{FL}_0$-concept descriptions C, D containing the concept names $A_1, \ldots, A_k$ can be rewritten as

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

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 $\mathcal{FL}_0$-concept descriptions [20]:

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

Since the size of the normal forms is polynomial in the size of the original concept descriptions, and since the inclusion tests $U_i \supseteq V_i$ can also be realized in polynomial time, this yields a polynomial-time decision procedure for subsumption in $\mathcal{FL}_0$.

This characterization of subsumption via inclusion of finite sets of words can be extended to TBoxes as follows. A given TBox $\mathcal{T}$ can be translated into a finite (word) automaton[20] $\mathcal{A}_\mathcal{T}$, 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. 9 for an example). A formal definition of this translation can be found in [4], where the more general case of cyclic TBoxes is treated. In the case of TBoxes, which are by definition acyclic, the resulting automata are also acyclic.

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. $\mathcal{FL}_0$ 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. 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. 9, 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 [54], this reduction shows that the subsumption problem in $\mathcal{FL}_0$ w.r.t. TBoxes is in coNP. As shown by Nebel [94], the reduction also works in the opposite direction, which yields the matching lower bound. For cyclic

[20] 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.

$$A \equiv C \sqcap \forall r.B \sqcap \forall s.\forall r.P$$
$$B \equiv \forall s.C$$
$$C \equiv \forall r.P$$

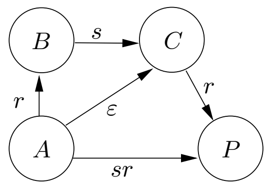

Fig. 9. An $\mathcal{FL}_0$ TBox and the corresponding acyclic automaton.

TBoxes, the subsumption problem corresponds to the inclusion problem for languages accepted by arbitrary finite automata, which is PSpace-complete, and thus the subsumption problem is also PSpace-complete [4, 77]. In the presence of general TBoxes, the subsumption problem in $\mathcal{FL}_0$ actually becomes as hard as for $\mathcal{ALC}$, namely ExpTime-hard [7, 63].

Theorem 3. *Subsumption in $\mathcal{FL}_0$ is polynomial without TBox, coNP-complete w.r.t. TBoxes, PSpace-complete w.r.t. cyclic TBoxes, and* ExpTime-*complete w.r.t. general TBoxes.*

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 [34].[21] 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:

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

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 on the right-hand side that need further rewriting:

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

[21] The special case of cyclic TBoxes was already treated in [6].

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 by a linear number of rule applications.

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. $\mathcal{T}$.

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. 10, 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,

(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 $S(A)$

Fig. 10. The completion rules for subsumption in $\mathcal{EL}$ w.r.t. general TBoxes.

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 [34, 7].

Theorem 4. *Subsumption in $\mathcal{EL}$ is polynomial w.r.t. general TBoxes.*

In [7] 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 [7] that almost all additions of other 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, both the large medical ontology Snomed ct[22] and the Gene Ontology[23] can be expressed in $\mathcal{EL}$, and the same is true for large parts of the medical ontology Galen [102]. First implementations of the subsumption algorithm for $\mathcal{EL}$ sketched above behave well on these very large knowledge bases [19, 81, 111].

In [8], the DL $\mathcal{EL}^{++}$ is extended with reflexive roles and range restrictions since these means of expressivity have turned out to be important in medical ontologies. It is shown that subsumption remains tractable if a certain syntactic restriction is adopted. The DL obtained this way corresponds closely to the OWL 2 profile OWL 2 EL.[24]

Acknowledgement

This article is, on the one hand, based on the Description Logic tutorial by the author, which he first taught at the 2005 Logic Summer School organized by the Research School of Information Sciences and Engineering, Australian National University, Canberra, Australia. This tutorial in turn took some inspirations from the Description Logic tutorial taught by Carsten Lutz and Ulrike Sattler at the 2005 ICCL Summer School on Logic-Based Knowledge Representation

[22] http://www.ihtsdo.org/snomed-ct/
[23] http://www.geneontology.org/
[24] http://www.w3.org/TR/owl2-profiles/

organized by the International Center for Computational Logic, TU Dresden, Germany. On the other hand, this article reuses some of the material from the overview articles [23, 18, 16], written by the author in collaboration with Ian Horrocks, Carsten Lutz, and Ulrike Sattler.

References

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 2005)*, pages 1670–1671. AAAI Press/The MIT Press, 2005.
2. 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.
3. Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, 1991.
4. Franz Baader. Using automata theory for characterizing the semantics of terminological cycles. *Ann. of Mathematics and Artificial Intelligence*, 18:175–219, 1996.
5. Franz Baader. Description logic terminology. In *[11]*, pages 485–495. 2003.
6. 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.
7. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 364–369, Edinburgh (UK), 2005. Morgan Kaufmann, Los Altos.
8. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope further. In Kendall Clark and Peter F. Patel-Schneider, editors, *In Proceedings of the Fifth International Workshop on OWL: Experiences and Directions (OWLED'08)*, Karlsruhe, Germany, 2008.
9. Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.
10. Franz Baader, Hans-Jürgen Bürckert, Bernhard Nebel, Werner Nutt, and Gert Smolka. On the expressivity of feature logics with negation, functional uncertainty, and sort equations. *J. of Logic, Language and Information*, 2:1–18, 1993.
11. 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.
12. 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.
13. Franz Baader, Jan Hladik, Carsten Lutz, and Frank Wolter. From tableaux to automata for description logics. *Fundamenta Informaticae*, 57(2–4):247–279, 2003.
14. 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-Verlag, 1991.

15. Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks in Information Systems, pages 3–28. Springer–Verlag, Berlin, Germany, 2003.
16. Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, pages 135–179. Elsevier, 2007.
17. Franz Baader, Ralf Küsters, and Ralf Molitor. Structural subsumption considered from an automata theoretic point of view. In *Proc. of the 1998 Description Logic Workshop (DL'98)*. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-11/, 1998.
18. Franz Baader and Carsten Lutz. Description logic. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *The Handbook of Modal Logic*, pages 757–820. Elsevier, 2006.
19. Franz Baader, Carsten Lutz, and Bontawee Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In Ulrich Furbach and Natarajan Shankar, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.
20. Franz Baader and Paliath Narendran. Unification of concepts terms in description logics. *J. of Symbolic Computation*, 31(3):277–305, 2001.
21. Franz Baader and Werner Nutt. Basic description logics. In *[11]*, pages 43–95. 2003.
22. Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}^+$. In *Proc. of the 30th German Annual Conf. on Artificial Intelligence (KI'07)*, volume 4667 of *Lecture Notes in Artificial Intelligence*, pages 52–67, Osnabrück, Germany, 2007. Springer-Verlag.
23. Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
24. Franz Baader and Boontawee Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proceedings of the International Conference on Representing and Sharing Knowledge Using SNOMED (KR-MED'08)*, Phoenix, Arizona, 2008.
25. 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-Verlag, 2001.
26. Orna Bernholtz and Orna Grumberg. Branching time temporal logic and amorphous tree automata. In Eike Best, editor, *Proc. of the 4th Int. Conf. on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 262–277, 1993.
27. Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.
28. Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.
29. 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.

30. Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI'84)*, pages 34–37, 1984.
31. Ronald J. Brachman and Hector J. Levesque, editors. *Readings in Knowledge Representation*. Morgan Kaufmann, Los Altos, 1985.
32. Ronald J. Brachman and Daniele Nardi. An introduction to description logics. In *[11]*, pages 1–40. 2003.
33. Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
34. 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.
35. 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.
36. 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.
37. Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *J. of Artificial Intelligence Research*, 1:109–138, 1993.
38. 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 2005)*, pages 602–607. AAAI Press/The MIT Press, 2005.
39. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Automated Reasoning*, 39(3):385–429, 2007.
40. 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.
41. 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.
42. 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.
43. Giuseppe De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.
44. Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th*

Nat. Conf. on Artificial Intelligence (AAAI'94), pages 205–212. AAAI Press/The MIT Press, 1994.

45. 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.
46. Giuseppe De Giacomo and Maurizio Lenzerini. TBox and ABox reasoning in expressive description logics. In Luigia C. Aiello, John Doyle, and Stuart C. Shapiro, editors, *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 316–327. Morgan Kaufmann, Los Altos, 1996.
47. Francesco Donini. Complexity of reasoning. In *[11]*, pages 96–136. 2003.
48. Francesco Donini and Fabio Massacci. EXPTIME tableaux for $\mathcal{ALC}$. *Acta Informatica*, 124(1):87–138, 2000.
49. 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.
50. Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 151–162. Morgan Kaufmann, Los Altos, 1991.
51. Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. Tractable concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 458–463, Sydney (Australia), 1991.
52. Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.
53. Melvin Fitting. Tableau methods of proof for modal logics. *Notre Dame J. of Formal Logic*, 13(2):237–247, 1972.
54. Michael R. Garey and David S. Johnson. *Computers and Intractability — A guide to NP-completeness.* W. H. Freeman and Company, San Francisco (CA, USA), 1979.
55. Birte Glimm, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In Manuela M. Veloso, editor, *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 399–404, Hyderabad, India, 2007.
56. Rajeev Goré and Linh Anh Nguyen. Exptime tableaux for $\mathcal{ALC}$ using sound global caching. In *Proc. of the 2007 Description Logic Workshop (DL 2007)*, Brixen-Bressanone, Italy, 2007.
57. Erich Grädel. Guarded fragments of first-order logic: A perspective for new description logics? In *Proc. of the 1998 Description Logic Workshop (DL'98)*. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-11/, 1998.
58. Erich Grädel. On the restraining power of guards. *J. of Symbolic Logic*, 64:1719–1742, 1999.
59. 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.
60. Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. A logical framework for modularity of ontologies. In Manuela M. Veloso, editor, *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 298–303, Hyderabad, India, 2007.
61. 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.

62. 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 701–706. Springer-Verlag, 2001.
63. Martin Hofmann. Proof-theoretic approach to description-logic. In Prakash Panangaden, editor, *Proc. of the 20th IEEE Symp. on Logic in Computer Science (LICS 2005)*, pages 229–237. IEEE Computer Society Press, 2005.
64. Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Ann. of Mathematics and Artificial Intelligence*, 18(2–4):133–157, 1996.
65. Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 335–346, 1991.
66. 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.
67. 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.
68. Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67, Lake District, UK, 2006. AAAI Press/The MIT Press.
69. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
70. 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.
71. Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei 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-Verlag, 1999.
72. Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing SHIQ-description logic to disjunctive datalog programs. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 152–162. Morgan Kaufmann, Los Altos, 2004.
73. Ullrich Hustadt and Renate A. Schmidt. On the relation of resolution and tableaux proof systems for description logics. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 110–117, 1999.
74. Ullrich Hustadt and Renate A. Schmidt. Issues of decidability for description logics in the framework of resolution. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *Lecture Notes in Artificial Intelligence*, pages 191–205. Springer-Verlag, 2000.
75. Ullrich Hustadt, Renate A. Schmidt, and Lilia Georgieva. A survey of decidable first-order fragments and description logics. *Journal of Relational Methods in Computer Science*, 1:251–276, 2004.
76. David Janin and Igor Walukiewicz. Automata for the modal mu-calculus and related results. In Jirí Wiedermann and Petr Hájek, editors, *Proc. of the 20th Int.*

Symp. on Foundations of Computer Science (MFCS'95), volume 969 of *Lecture Notes in Computer Science*, pages 552–562. Springer-Verlag, 1995.

77. Yevgeny Kazakov and Hans de Nivelle. Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*. CEUR Electronic Workshop Proceedings, http://CEUR-WS.org/Vol-81/, 2003.
78. 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-Verlag, 2006.
79. Boris Konev, Carsten Lutz, Dirk Walther, and Frank Wolter. Semantic modularity and module extraction in description logics. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikos Avouris, editors, *Proc. of the 18th Eur. Conf. on Artificial Intelligence (ECAI 2008)*, pages 55–59. IOS Press, 2008.
80. Natasha Kurtonina and Maarten de Rijke. Expressiveness of concept expressions in first-order description logics. *Artificial Intelligence*, 107(2):303–333, 1999.
81. Michael Lawley. Exploiting fast classification of SNOMED CT for query and integration of health data. In Ronald Cornet and Kent Spackman, editors, *Proc. of the 3rd Int. Conf. on Knowledge Representation in Medicine (KR-MED 2008)*, Phoenix (Arizona), USA, 2008.
82. Hector J. Levesque and Ron J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.
83. 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-Verlag, 1999.
84. 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.
85. Carsten Lutz. The complexity of conjunctive query answering in expressive description logics. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2008)*, Lecture Notes in Artificial Intelligence, pages 179–193. Springer-Verlag, 2008.
86. Carsten Lutz and Ulrike Sattler. Mary likes all cats. In *Proc. of the 2000 Description Logic Workshop (DL 2000)*, pages 213–226. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-33/, 2000.
87. 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.
88. E. Mays, R. Dionne, and R. Weida. K-REP system overview. *SIGART Bull.*, 2(3), 1991.
89. 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.
90. Marvin Minsky. A framework for representing knowledge. In John Haugeland, editor, *Mind Design*. The MIT Press, 1981. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [31].
91. Michael Mortimer. On languages with two variables. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:135–140, 1975.

92. D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.
93. Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1990.
94. Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.
95. Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data complexity of query answering in expressive description logics via tableaux. *J. of Automated Reasoning*, 41(1):61–98, 2008.
96. 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.
97. 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.
98. Peter F. Patel-Schneider. DLP. In *Proc. of the 1999 Description Logic Workshop (DL'99)*, pages 9–13. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-22/, 1999.
99. 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.
100. Christof Peltason. The BACK system — an overview. *SIGART Bull.*, 2(3):114–119, 1991.
101. M. Ross Quillian. Semantic memory. In M. Minsky, editor, *Semantic Information Processing*, pages 216–270. The MIT Press, 1968.
102. Alan Rector and Ian Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*, Stanford, CA, 1997. AAAI Press.
103. 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.
104. Klaus Schild. *Querying Knowledge and Data Bases by a Universal Description Logic with Recursion.* PhD thesis, Universität des Saarlandes, Germany, 1995.
105. 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.
106. 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.
107. Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with unions and complements. Technical Report SR-88-21, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern (Germany), 1988.
108. Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
109. Evren Sirin and Bijan Parsia. Pellet: An OWL DL reasoner. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 212–213, 2004.

110. Boontawee Suntisrivaraporn. Module extraction and incremental classification: A pragmatic approach for $\mathcal{EL}^+$ ontologies. In Sean Bechhofer, Manfred Hauswirth, Joerg Hoffmann, and Manolis Koubarakis, editors, *Proceedings of the 5th European Semantic Web Conference (ESWC'08)*, volume 5021 of *Lecture Notes in Computer Science*, pages 230–244. Springer-Verlag, 2008.
111. Boontawee Suntisrivaraporn. *Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies.* PhD thesis, Fakultät Informatik, TU Dresden, 2009. http://lat.inf.tu-dresden.de/research/phd/#Sun-PhD-2008.
112. Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 134–189. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.
113. Stephan Tobies. A PSPACE algorithm for graded modal logic. In Harald Ganzinger, editor, *Proc. of the 16th Int. Conf. on Automated Deduction (CADE'99)*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 52–66. Springer-Verlag, 1999.
114. Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation.* PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.
115. Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer-Verlag, 2006.
116. Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.