Description Logics for Ontologies

Ulrike Sattler

TU Dresden

1 Introduction to Description Logics

Description logics (DLs) [6, 8, 21] are a family of logic-based knowledge representation formalisms designed to represent and reason about the knowledge of an application domain in a structured and well-understood way.

The basic notions in DLs are *concepts* (unary predicates) and *roles* (binary relations), and a specific DL is mainly characterised by the *constructors* it provides to form complex concepts and roles from atomic ones. Intuitively, the following concept describes "A cooler connected to a reactor which, in turn, has a part that is a stirrer and whose functionality is to stir or to cool (or both)":

$$\mathsf{Cooler} \sqcap \exists \mathsf{connectedTo}.(\mathsf{Reactor} \sqcap (\exists \mathsf{hasPart}.\mathsf{Stirrer} \sqcap \forall \mathsf{functionality}.(\mathsf{Cooling} \sqcup \mathsf{Stirring}))) \quad (1)$$

In addition to such a set of constructors, DLs are usually equipped with a terminological component, often called a *TBox*. In its simplest form, a TBox can be used to introduce names (abbreviations) for complex concepts. For example, we could introduce the abbreviation CooledStirringReactor for the concept in Concept 1 from above. More expressive TBox formalisms allow the statement of *general concepts inclusion axioms* (GCIs) such as

$$\exists \mathsf{hasPart}.\mathsf{Stirrer} \dot{\sqsubseteq} \mathsf{Reactor} \sqcap \exists \mathsf{functionality}.\mathsf{Stirring},$$

which says that only stirring reactors can have stirrers.

Description logic systems provide their users with various reasoning capabilities that deduce implicit knowledge from the one explicitly stated in the TBox. The *subsumption* algorithm determines subconcept-superconcept relationships: a concept C is subsumed by a concept D w.r.t. a TBox if, in each model of the TBox, each instance of C is also an instance of D. Such an algorithm can be used to compute the *taxonomy* of a TBox, i.e., the subsumption hierarchy of all those concepts introduced in the TBox. The *satisfiability* algorithm tests whether a given concept can ever be instantiated.

Unsurprisingly, the higher the expressive power of a DL is, the more complex are the subsumption and the satisfiability problem. To use a DL for a certain application, it has to provide enough expressive power to describe the relevant properties of the objects in this application. On the other hand, the system services should be "practical" in that they run in realistic time and space. Thus, we are confronted with the well-known trade-off between expressivity and complexity, as in many other areas of computer science.

In the last decade, a lot of work was devoted to investigate DLs w.r.t. their expressive power and computational complexity. It turned out that the first DL systems were based on undecidable logics [74, 61]. As a reaction, the expressive power was restricted severely, thus yielding a DL with polynomial reasoning problems. Then, in parallel with the discovery of the close relation between description and modal logics [73, 23], PSPACE-complete DLs were specified [75] and a tableau-based reasoning algorithm was implemented for such a DL [7]. After certain optimisation, it turned out that this implementation behaves much better than the high worst-case complexity of the underlying reasoning problem suggests. As a reaction, tableau-based reasoning algorithms for EXPTIME-complete DLs were implemented [41, 37]. Again, these implementations proved to be amenable to optimisation and behave surprisingly well in practice. This fostered the design and investigation of other EXPTIME-complete DLs together with tableau-based, "practicable" reasoning algorithms. In parallel, the investigation of the complexity of description logics continued successfully such that, today, we have a good understanding of the effects of the combination of concept and role constructors on the computational complexity and the expressive power of DLs; see, e.g., [26, 18, 24, 21, 20, 84, 54, 57, 56].

Today, industrial strength DL systems are being developed for very expressive DLs with system services being based on highly optimised tableau algorithms and with applications like the Semantic Web or knowledge representation and integration in bio-informatics in mind.

1.1 Preliminaries

In this section, we define the basic description logic $\mathcal{ALC}$, TBox formalisms, and reasoning problems.

Definition 1. *Let* $\mathbf{C}$ *and* $\mathbf{R}$ *be disjoint sets of* concept *and* role name*. The set of $\mathcal{ALC}$-concepts is the smallest set such that each concept name $A \in \mathbf{C}$ is an $\mathcal{ALC}$-concept and, if C and D are $\mathcal{ALC}$-concepts and r is a role name, then*

$$\neg C,\ C \sqcap D,\ C \sqcup D,\ \exists r.C,\ \text{and}\ \forall r.C \ \text{are also}\ \mathcal{ALC}\text{-concepts.}$$

A general concept inclusion axiom (GCI) *is of the form $C \dot{\sqsubseteq} D$ for C, D $\mathcal{ALC}$-concepts. A* TBox *is a finite set of GCIs.*

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, the interpretation domain*, and a mapping $\cdot^{\mathcal{I}}$ which associates, with each concept name A, a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and, with each role name r, a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation of complex concepts is defined as follows:*

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, \quad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \quad \neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},$$
$$(\exists r.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \textit{There exists an } e \in \Delta^{\mathcal{I}} \textit{ with } \langle d, e\rangle \in r^{\mathcal{I}} \textit{ and } e \in C^{\mathcal{I}}\},$$
$$(\forall r.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \textit{For all } e \in \Delta^{\mathcal{I}}\textit{, if } \langle d, e\rangle \in r^{\mathcal{I}}\textit{, then } e \in C^{\mathcal{I}}\}.$$

An interpretation $\mathcal{I}$ satisfies *a GCI $C \dot{\sqsubseteq} D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; $\mathcal{I}$ satisfies a TBox $\mathcal{T}$ if $\mathcal{I}$ satisfies all GCIs in $\mathcal{T}$—in this case, $\mathcal{I}$ is called* a model of $\mathcal{T}$*. An*

element $d \in C^{\mathcal{I}}$ *is called an* instance *of* C *and, if* $\langle d, e\rangle \in r^{\mathcal{I}}$*, then* e *is called an* r-successor *of* d*.*

A concept C *is* satisfiable *w.r.t. a TBox* $\mathcal{T}$ *if there is a model* $\mathcal{I}$ *of* $\mathcal{T}$ *with* $C^{\mathcal{I}} \neq \emptyset$*. A concept* C *is* subsumed *by a concept* D *w.r.t.* $\mathcal{T}$ *(written* $C \sqsubseteq_{\mathcal{T}} D$*) if, for each model* $\mathcal{I}$ *of* $\mathcal{T}$*,* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$*. Two concepts are equivalent if they mutually subsume each other.*

As usual, we use $\top$ as an abbreviation for $A \sqcup \neg A$, $\bot$ for $\neg\top$, $C \Rightarrow D$ for $\neg C \sqcup D$, and $C \Leftrightarrow D$ for $(C \Rightarrow D) \sqcap (D \Rightarrow C)$. Moreover, we use $C \doteq D$ as an abbreviation for $C \dot{\sqsubseteq} D$ and $D \dot{\sqsubseteq} C$.

Some remarks are in order here. Firstly, in $\mathcal{ALC}$, the two reasoning problems satisfiability and subsumption can be mutually reduced to each other: C is satisfiable w.r.t. $\mathcal{T}$ iff C is *not* subsumed by $\bot$ w.r.t. $\mathcal{T}$. And $C \sqsubseteq_{\mathcal{T}} D$ iff $C \sqcap \neg D$ is *not* satisfiable w.r.t. $\mathcal{T}$. Secondly, it can be shown that satisfiability (and thus subsumption) w.r.t. a general TBox is EXPTIME-complete [73], whereas these problems become PSPACE-complete when considered w.r.t. the empty TBox [75].

2 Description logics as ontology languages

A well-known attempt to define what constitutes an ontology is due to Gruber [35]: *an ontology is an explicit specification of a conceptualisation*, where "a conceptualisation" means an abstract model of some aspect of the world. This was later elaborated to "a formal specification of a shared conceptualisation" [16]. In this abstract model, relevant concepts of the aspect in question are defined, including a description of the interesting properties of their instances.

In the last decade, ontologies became rather popular through applications like the Semantic Web [15], enterprise knowledge management systems [85], and medical terminology systems [80, 66, 79] and through the growing amount of data available electronically.

An ontology is *built*—possibly by a group of domain experts—and *evolves* over time in any application that changes over time. Moreover, it is advisable to *integrate* existing ontologies if a larger aspect of the world is to be covered—instead of building a new one from scratch. Finally, if an ontology is *deployed*, knowledge is shared using the concepts defined in the ontology, e.g., concrete objects are described using the vocabulary defined in an ontology. Each of these tasks is rather complex: e.g. building and evolution involves a huge amount of creativity, integration requires knowledge in a large aspect, and all three tasks involve co-operation, thus risking misunderstanding, redundancy, etc.

The increasing importance of ontologies and their processing in computers has led to the development of ontology editors [64, 22, 12]. Due to the above mentioned complexity of ontology engineering tasks, it is highly desirable that these editors support the user in the design, evolution, integration, and deployment of ontologies through corresponding, intelligent system services. Moreover, an *unambiguous* language, e.g., a logic-based language, not only decreases the risk

of misunderstandings among the domain experts, but also enables the design of provably correct or optimal such services. Now *description logics* are such a class of logic-based knowledge representation languages that come with a knowledge base formalism which makes DLs good candidates for ontology languages: an ontology can be formalised in a TBox, which can be divided into the following two, disjoint parts.

Background Knowledge GCIs of the form $C \dot{\sqsubseteq} D$ for C and D complex concepts can be used to formalise background knowledge of the application domain and thus to constrain the set of models.
For example, we can express that two concepts A and B are disjoint by $A \dot{\sqsubseteq} \neg B$ and that each individual having an r-successor which is an instance of B is an instance of A by $\exists r.B \dot{\sqsubseteq} A$.

Definitiorial Part For each concept relevant in the application domain, we can introduce a concept name A and a *concept definition* $A \dot{\sqsubseteq} C$ or $A \doteq C$ for C a complex concept describing necessary or necessary and sufficient conditions for individuals to be an instance of A. We say that A is a *primitively defined* or a *defined* concept.
For example, we can primitively define connections as being devices having some input and some output, and then define a hose as a flexible connection:

$$\texttt{Connection} \dot{\sqsubseteq} \texttt{Device} \sqcap \exists \texttt{hasComp.Output} \sqcap \exists \texttt{hasComp.Input}$$
$$\texttt{Hose} \doteq \texttt{Connection} \sqcap \texttt{Flexible}$$

System services provided by DL-based knowledge representation systems include

- a satisfiability test for each concept defined in a TBox.
- the computation of the *taxonomy*: for each pair A_1, A_2 of concepts defined in the definitorial part of the TBox $\mathcal{T}$, we test whether $A_1 \dot{\sqsubseteq}_{\mathcal{T}} A_2$ and $A_2 \dot{\sqsubseteq}_{\mathcal{T}} A_1$. A taxonomy is the partial order of the defined concepts w.r.t. $\dot{\sqsubseteq}_{\mathcal{T}}$, and is often presented as the corresponding Hasse-diagram.

Clearly, unsatisfiable defined concepts and unintended or missing subsumption relationships are signs of modelling flaws, and thus these system services can be used to support the engineering of ontologies: in the design phase and when modifying or integrating an ontology, we can repeatedly use both system services to ensure that the TBox is consistent, that it reflects our intuition, and that it does not contain unintended redundancies, i.e., equivalent defined concepts. Unsurprisingly, it turned out that, in applications where the knowledge engineer is no description logic expert, ontology engineering requires more support [69, 58], e.g., the domain expert wants to see automatically generated suggestions for a new concept definition as a generalisation of a set of example instances. This observation lead to the investigation of *non-standard inferences* in description logics such as computing the least common subsumer of several concepts, matching a concept that contains concept variables to concept expressions, or computing the approximation of a concept expressed in a more expressive logic in a less expressive logic [52, 5, 9, 17].

State-of-the-art DL-based systems such as FaCT or Racer [41, 37] provide the above *standard* system services such as deciding the satisfiability and computing the taxonomy, and are based on the DL $\mathcal{SHIQ}$ [49] that is an extension of $\mathcal{ALC}$ with a variety of expressive means that turned out to be quite useful [70]; $\mathcal{SHIQ}$ is discussed in detail in Section 5. Despite these additional expressive means, $\mathcal{SHIQ}$ is of the the same worst-case complexity as $\mathcal{ALC}$, namely EXPTIME-complete [83]. This high complexity implies that, in the worst-case, the computation might take far too much time. However, the algorithms in these DL-based systems proved to be amenable to a wide range of optimisations, as a consequence of which these systems behave surprisingly well in many realistic applications [40, 41, 37, 48].

The suitability of DLs as ontology languages has been highlighted by their role as the foundation for several web ontology languages, including OIL [28], DAML+OIL [43], and OWL, a newly emerging ontology language standard being developed by the W3C Web-Ontology Working Group.[1] All of these languages have a syntax based on RDF Schema, but the basis for their design is a combination of the DLs $\mathcal{SHIQ}$ (mentioned above) and $\mathcal{SHOQ}(D)$ [46]. Both are DLs that were designed with the goal to find a good compromise between expressiveness and the complexity of reasoning.

3 Standard expressive means in DLs

To give the reader an impression of what DLs are, we present a variety of expressive means that are commonly used in DLs and discuss, if appropriate, their modal logic equivalent and their influence on the computational complexity. For a detailed description of the relationship between modal and description logics, see [73, 23]: $\mathcal{ALC}$ (without TBoxes) is a notational variant of the multi modal logic $\mathbf{K}_n$ [38]. To see the connection between $\mathbf{K}_n$ and $\mathcal{ALC}$, it suffices to view *elements* of a DL interpretation domain as *worlds* in a Kripke structure, roles as modal parameters, universal value restrictions as box formulae, and existential restrictions as diamond formulae. Then, for example, it can be easily seen that $A \sqcap \exists r.(C \sqcup \forall s.D)$ is equivalent to $A \wedge \langle r \rangle (C \vee [s]D)$.

TBoxes were introduced in Section 1.1, and it was mentioned in Section 2 that they are divided into a *background knowledge* part and a *definitorial* part. Some DLs only allow for the definitorial part and possibly require this part to be free of "definitorial cycles". Now reasoning w.r.t. acyclic concept definitions can be reduced to pure concept reasoning: one can either use a (sub-optimal) technique, called *unfolding*, which reduces reasoning w.r.t. acyclic concept definition to pure concept reasoning [59], or use more direct techniques [54]. As a result of the latter, it turned out that, for a variety of logics, reasoning w.r.t. acyclic concept definitions is as complex as pure concept reasoning [54].

In modal logics, the closest relative to a TBox is the *universal role*, a role that is interpreted as $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$; for more details about this relationship, see [55].

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

Number Restrictions are an expressive means rather popular in DLs: they are present in almost all implemented DL systems. They are concepts of the form $(\geqslant nr.C)$ (atleast restriction) or $(\leqslant nr.C)$ (atmost restriction), for n a non-negative integer, r a role, and C a (possibly complex) concept, and are interpreted as follows:

$$(\geqslant nr.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \#\{e \in C^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\} \geq n\},$$
$$(\leqslant nr.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \#\{e \in C^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\} \leq n\},$$

where $\#M$ denotes the cardinality of a set M. They can be used, e.g., to described pipes as those connections having exactly one input and one output (we use $(= nr.C)$ as an abbreviation for $(\geqslant nr.C) \sqcap (\leqslant nr.C)$):

$$\texttt{Connection} \sqcap (= 1\texttt{hasComp.Input}) \sqcap (= 1\texttt{hasComp.Output})$$

In their simpler form, number restrictions only allow for the concept $\top$ in the place of C above. A further restrictions only allows for 2 in atleast restrictions and 1 in atmost restrictions. Finally, *features* are role names that are to be interpreted as partial functions—they can be viewed as a "globalised" version of a simple form of number restrictions. Number restrictions rarely seem to have effects on the complexity of DLs: for a variety of logics, extending them with number restrictions does not change their complexity, even if such an extension yields the loss of the *finite model property* (see Section 5.4 for a more detailed discussion). For example, when extended with number restrictions, $\mathcal{ALC}$ remains in PSPACE [84] and $\mathcal{ALC}$ with TBoxes remains in EXPTIME, even if further extended with other expressive means such as inverse roles (see below) [83].

Number restrictions are known in modal logics as *graded modalities* [29], whereas features play an important role in dynamic logic: they are syntactic variants of *deterministic programs* [13].

Nominals are, in their simplest form, special concept names that are to be interpreted as singleton sets. For example, the concept $\exists\texttt{partOf.BrentSpar}$ describes those objects that are part of the oil platform Brent Spar provided that `BrentSpar` is a nominal. In DLs, a weak form of nominals, ABoxes ("A" for assertional), are widely known and used: using individual names $a, b, \ldots$, we can assert that an an individual is an instance of a concept C by $a : C$ and that two individuals are related via a role r by $\langle a, b\rangle : r$. Interpretations associate, additionally, an element of the interpretation domain with each individual name. Please note that individual names are only to be used in assertions, in contrast to nominals that can be used in the place of concepts in concepts. Whereas ABox consistency is often as complex as satisfiability of concepts [72], extending a description logic with nominals often increases its complexity. For example, $\mathcal{ALC}$ with inverse roles (see below) is PSPACE-complete, but becomes EXPTIME-complete when extended with inverse roles [1]. If, additionally, number restrictions are present, the complexity leaps from EXPTIME-completeness to NEXPTIME-completeness [83]. A reason for this increase in complexity might be that nominals destroy the *tree model property* [87]: a logic enjoys the tree model property if every satisfiable concept/formula has a model whose rela-

tional structure forms a tree. For example, for nominals N_1 and N_2, the concept $N_1 \sqcap \exists r.(N_2 \sqcap \exists r.N_1)$ only has models with a cycle of length two.

Nominals originate in *hybrid* logic [63, 1, 2], and are known in DLs as an elegant and powerful generalisation of ABoxes.

Inverse Roles In various applications, one wants to use both "directions" of a role, e.g., one wants to use both `hasPart` and `isPartOf`. To model these roles adequately, i.e., to ensure that $\langle x, y\rangle \in \texttt{hasPart}^{\mathcal{I}}$ iff $\langle y, x\rangle \in \texttt{isPartof}^{\mathcal{I}}$, some description logics provide *inverse roles*: for r a role name, r^- is an inverse role, which is interpreted as $(r^-)^{\mathcal{I}} = \{\langle y, x\rangle \mid \langle x, y\rangle \in r^-\}$.

A variety of DLs can be extended with inverse roles without affecting their computational complexity: examples are $\mathcal{ALC}$ with or without TBoxes and possibly with number restrictions [21, 83]. However, there are counter-examples such as $\mathcal{ALC}$ with concrete domains, which becomes NEXPTIME-complete when extended with inverse roles [56] or $\mathcal{ALC}$ with nominals and without TBoxes, which becomes EXPTIME-complete when extended with inverse roles [1].

Inverse roles are closely related to the tense logic "past" modality [67, 78, 88] and are syntactic variants of converse programs in dynamic logics [81, 86].

Transitive Roles are special role names $r \in \mathbf{R}_+ \subseteq \mathbf{R}$ that are to be interpreted as *transitive relations* [68]. Transitive roles can be used to model transitive relations such as `isAncestorOf` or `isPartOf`. Another way to extend DLs with transitivity is to allow for the *transitive closure operator* on roles, i.e., to allow for roles r^* in the place of roles [3, 24], where $(r^*)^{\mathcal{I}}$ is to be interpreted as the transitive closure of $r^{\mathcal{I}}$. We will discuss the expressiveness of transitive roles in more detail in Section 5.1.

Adding transitive roles to $\mathcal{ALC}$ without TBoxes yields a DL whose reasoning problems are still PSPACE-complete [68], whereas adding the transitive closure operator on roles yields an EXPTIME-complete logics [30]. Transitive roles are notational variants of transitive accessibility relations in modal logics [38], whereas a transitive closure operator is also present in the dynamic logic PDL [30], which is a notational variant of $\mathcal{ALC}$ with regular role expressions [73].

Boolean Operator on Roles So far, we considered DLs with full Boolean operators on concepts, but no Boolean operators on roles. In DLs, Boolean operators on roles are mostly restricted to intersection [26], or to union and difference [24, 19]. They are interpreted in the obvious way, i.e., $(r \sqcap s)^{\mathcal{I}} = r^{\mathcal{I}} \cap s^{\mathcal{I}}$, etc., and are an interesting expressive means. For example, role negation allows to express the so-called *window* operator from modal logic [32]. The window operator can be viewed as the dual of universal value restrictions: an instance of $\forall$`connectedTo.Pipe` is connected *only* to pipes, whereas an instance of the concept $\forall$`Pipe.connectedTo` using the window operator is connected to *all* pipes. It can be easily seen that the concept $\forall\neg$`connectedTo.`$\neg$`Pipe` using role negation is equivalent to $\forall$`Pipe.connectedTo`. For a complete description of the (mostly dramatic) effects of adding Boolean operators on roles to the computational complexity of $\mathcal{ALC}$, see [57].

In dynamic logic, union of programs is present in all logics allowing for regular programs [30], and Boolean operators on modilities are discussed, e.g., in [32].

Role Inclusion Axioms Another expressive means on roles are role inclusion axioms, which are of the form $r \dot{\sqsubseteq} s$ for r, s roles, and force interpretations to map r to a sub-relation of s. Such axioms can be used, for example, to introduce a sub-role `hasComponent` of `hasPart`. In the presence of inverse roles, role hierarchies can be used to enforce symmetric roles using $r^- \dot{\sqsubseteq} r$ and $r \dot{\sqsubseteq} r^-$.

It should be noted that role hierarchies are a weak form of role intersection: replacing each role expression $r_1 \sqcap r_2$ with a new role name $r_{1,2}$ and adding $r_{1,2} \dot{\sqsubseteq} r_1$ and $r_{1,2} \dot{\sqsubseteq} r_2$ to the role hierarchy yields a "weakened" form of intersection since $r_{1,2}^{\mathcal{I}} \subseteq r_1^{\mathcal{I}} \cap r_2^{\mathcal{I}}$. Moreover, for s a transitive role, the role inclusion axioms $r \dot{\sqsubseteq} s$ yields a weakened form of the transitive closure: s is interpreted as *some* transitive role containing r, whereas r^* is interpreted as *the smallest* transitive role containing r. The latter observation implies that pure concept satisfiability of $\mathcal{ALC}$, when extended with both transitive roles and role inclusion axioms, becomes EXPTIME-hard [68].

General Role Inclusion Axioms (g-RIAs) are a generalisation of the above role inclusion axioms to the form $r_1 \dots r_m \dot{\sqsubseteq} s_1 \dots s_n$ for r_i, s_j role names [47]. A model of such an axiom satisfies $r_1^{\mathcal{I}} \circ \dots \circ r_m^{\mathcal{I}} \subseteq s_1^{\mathcal{I}} \circ \dots \circ s_n^{\mathcal{I}}$, where $\circ$ denotes the composition of binary relations. *Role value maps*, i.e., concepts of the form $r_1 \dots r_m \Rightarrow s_1 \dots s_n$ with the semantics

$$(r_1 \dots r_m \Rightarrow s_1 \dots s_n)^{\mathcal{I}} = \{x \mid \forall y. \langle x, y\rangle \in r_1^{\mathcal{I}} \circ \dots \circ r_m^{\mathcal{I}} \Rightarrow \langle x, y\rangle \in s_1^{\mathcal{I}} \circ \dots \circ s_n^{\mathcal{I}}\},$$

can be viewed as a "local" form of g-RIAs. Both constructors have dramatic effects on the decidability of a description logic: it was shown in [74] that subsumption of a very weak DL becomes undecidable when extended with role value maps. DLs with g-RIAs are closely related to *grammar logics* [25, 10, 11], i.e., the multi modal logic $\mathbf{K}_n$ with accessibility relations being constrained by a grammar: a production rule of the form $s_1 \dots s_n \rightarrow r_1 \dots r_m$ can be viewed as a notational variant of the g-RIA $r_1 \dots r_m \dot{\sqsubseteq} s_1 \dots s_n$ enforcing models to interpret $r_1 \dots r_m$ as s sub-relation of $s_1 \dots s_n$. Since each context-free grammar can be transformed into an equivalent one in Chomsky normal form, and $\mathcal{ALC}$ becomes undecidable with context-free grammars [10, 11], the satisfiability of $\mathcal{ALC}$-concepts w.r.t. g-RIAs of the form $r_1 r_2 \dot{\sqsubseteq} s$ is undecidable.

Fixpoint Operators are the first expressive means mentioned here that are not first order definable, and they are known in DLs in at least three forms: a restricted form includes the transitive closure operator on roles [3, 24] (see above) and an operator that allows to enforce that a role is interpreted as a well-founded relation [19]. Secondly, general least and greatest fixpoints operators in DLs [20] are notational variants of the fixpoint operators in the μ-calculus [51]. Thirdly, *cyclic* concept definitions such as

$$\begin{aligned} \texttt{Device} &\doteq \texttt{TechThing} \sqcap \neg\texttt{Connection} \sqcap \forall\texttt{connectedTo.Connection} \\ \texttt{Connection} &\doteq \texttt{TechThing} \sqcap \neg\texttt{Device} \sqcap \forall\texttt{connectedTo.Device} \end{aligned}$$

can be read with least or greatest *fixpoint semantics* [59, 4]: in contrast to the *descriptive* semantics, which takes into account *all* fixpoints of such GCIs, one might chose to take into account only the least or the greatest fixpoints.

A variety of EXPTIME-complete description and modal logics exist that have some form of fixpoints, e.g. the dynamic logic PDL [30, 62] or $\mathcal{DLR}_\mu$, a generalisation of the μ-calculus with n-ary relations [20].

4 Introduction to tableau algorithms for DLs

For several expressive DLs, there exist efficient tableau-based implementations that decide satisfiability of concepts w.r.t. a TBox [41, 37]. In the following, we will give an intuitive description of description logic tableau algorithms; for an extensive survey of tableau algorithms for description logics, see, e.g., [8]. In general, they work on trees whose nodes stand for individuals of an interpretation. Nodes are labelled with sets of concepts, namely those they are assumed to be an instance of. Edges between nodes are labelled with role names or sets of role names, namely those that hold between the corresponding individuals.

Intuitively, to decide the satisfiability of a concept C, a tableau algorithm starts with an instance x_0 of C, i.e., a tree consisting of a root node x_0 with C as its node label (written $\mathcal{L}(x_0) = \{C\}$). Then the algorithm breaks down concepts in node labels syntactically, thus inferring new constraints on the model of C to be built, and possibly generating new individuals, i.e., new nodes. For example, if $(D \sqcap E) \in \mathcal{L}(y)$ has already been inferred, it adds D and E to $\mathcal{L}(y)$. For $\exists r.F \in \mathcal{L}(y)$, it generates a new r-successor node of y, say z, and sets $\mathcal{L}(z) = \{F\}$. If a node y has some r-successor z and it finds $\forall r.G \in \mathcal{L}(y)$, then G is added to $\mathcal{L}(z)$. Finally, in the presence of a TBox $\mathcal{T}$, it adds, for each GCI $C_i \dot{\sqsubseteq} D_i \in \mathcal{T}$, and for each node y, the concept $(\neg C_i \sqcup D_i)$ to $\mathcal{L}(y)$. Now, for logics with disjunctions, various tableau algorithms non-deterministically choose whether to add D or E to $\mathcal{L}(y)$ for $(D \sqcup E) \in \mathcal{L}(y)$. The answer behaviour is as follows: if this "completion" can be carried out exhaustively without encountering a node with both a concept and its negation in its label—a so-called *clash*,[2] then the algorithm answers that the input concept was satisfiable, and unsatisfiable, otherwise.

Thus, disjunctions are often treated non-deterministically. Avoiding this non-determinism in a way that is more efficient than naive back-tracking proves to be a complicated task for many logics—to the best of our knowledge, the algorithm in [27] is the only known worst-case optimal tableau algorithm for an EXPTIME-complete description logic. In contrast, the tableau algorithm implemented in state-of-the-art DL systems such as FaCT and Racer is 2NEXPTIME even though the underlying logic is EXPTIME-complete [49, 83]. Despite this sub-optimality, these tableau algorithms allow for a set of well-known efficient optimisations, so that they perform much better in practice than their worst-case complexity suggests; see [41, 49, 36, 37, 48] for descriptions of these optimisations. An interesting open question is whether an implementation of a worst-case optimal algorithm would behave better in practice—so far, only implementations of worst-case sub-optimal algorithms exist.

[2] We assume that the description logic in question is propositionally closed, and we can thus work on concepts in negation normal form.

Since we are talking about decision procedures, *termination* is an important issue. Even though tableau algorithms for some inexpressive DLs terminate "automatically", this is not the case for more expressive ones. For example, consider the algorithm sketched above on the input concept A and TBox $\{A \dot{\sqsubseteq} \exists r.A\}$: it would create an infinite r-chain of nodes with labels $\{A, \exists r.A\}$. To guarantee termination, the tableau algorithm needs to be stopped explicitly. Intuitively, the processing of an element z is stopped if all "relevant" concepts in the label of z are also present in the label of an "older" element z'. In this case, z' is said to *block* z. The definition of "relevant" has to be chosen carefully since it is crucial for the correctness of the algorithm [3, 49] and for the efficiency of the implementation [48, 39].

Correctness of DL tableau algorithms are often proved as follows: first, termination is proved by, roughly spoken, showing that the algorithm builds a (tree) structure of bounded size in a monotonic manner. Soundness is proved by constructing a model (or an abstraction of a model) of the input concept (and TBox) in case that the algorithm stops without having generated a clash. Completeness can be proved by using a model of the input concept (and TBox) to steer the application of the non-deterministic rules and proving that no clash occurs using this control.

Summing up, tableau algorithms are successfully used in state-of-the-art implementations, and many well-understood optimisations are available. However, they involve special techniques to ensure termination and avoid non-determinism, and are thus rarely optimal for logics complete for deterministic complexity classes.

4.1 Other reasoning techniques

For several expressive description and modal logics, there exist optimal automata-based algorithms that decide satisfiability (and thus subsumption) of concepts w.r.t. a TBox [89, 82, 88, 20, 57, 71]: for a concept C and a TBox $\mathcal{T}$, we define an automaton $\mathcal{A}_{C,\mathcal{T}}$ which accepts exactly the (abstractions of) models of C and $\mathcal{T}$. Thus, the satisfiability problem is reduced to the emptiness problem of automata. In summary, automata-based approaches often allow for a very elegant and natural translation of a logic and provide ExpTime upper complexity bounds and are thus optimal for ExpTime-hard logics. Equally important, they handle infinite structures and non-determinism implicitly.

For certain DLs that are *not* propositionally closed such as the one used in the system Classic [60], one can use a reasoning technique called *structural subsumption*: roughly speaking, to decide the subsumption between two concepts C and D, both concepts are transformed into a certain normal form C' and D', and then subsumption can be decided by a syntactic comparison of C' and D', see Section 2.3.1 of [6]. This technique yields a polynomial time decision procedure for a sub-Boolean fragment of $\mathcal{ALC}$ with number restrictions, but seems to be applicable only to DLs without disjunction and existential restrictions.

Finally, the successful resolution-based theorem prover SPASS was modified into a decision procedure for expressive modal and description logics, then called

MSPASS [50]. Interestingly, it is well-suited for DLs extending $\mathcal{ALC}$ with Boolean operators on roles and can be extended to n-ary description logics [33].

5 DLs with expressive operators on roles

In various ontology applications such as engineering or medicine, *aggregated objects* play a central role, that is, objects that are composed of various parts, which again can be composite, etc. It is natural to describe an aggregated object by means of its parts and, vice versa, to describe parts by means of the aggregate they belong to. For example, the following statements describe a control rod and a reactor core by means of their parts and wholes:

$$\begin{aligned} \texttt{ControlRod} &\dot{\sqsubseteq} \texttt{Device} \sqcap \exists \texttt{partOf}.\texttt{ReactorCore} \\ \texttt{ReactorCore} &\dot{\sqsubseteq} \texttt{Device} \sqcap \exists \texttt{hasPart}.\texttt{ControlRod} \sqcap \exists \texttt{partOf}.\texttt{NuclReactor} \end{aligned}$$

In contrast to, for example, the relation `likes`, the part-whole relation has a variety of properties; for a complete collection of these properties, we refer to [77]. Most importantly, the general part-whole relation is a strict partial order, i.e., it is *transitive* and *asymmetric* (and hence irreflexive). Moreover, an aggregated object has at least two parts where none is a part of the other. Next, we might consider to assume that two objects consisting of the same parts are identical. As a last example, we might assume the existence of atoms, i.e., indivisible objects of which all other objects are composed. This is equivalent to assuming that `hasPart` is well-founded.

Besides the properties mentioned above, it might be useful to distinguish various sub-relations of the part-whole relation such as, for example, the relation between a *component* and its *composite* (e.g. between a motor and the car the motor is in), the relation between *stuff* and an *object* containing this stuff (e.g. between metal and a car), or the relation between a *member* and a *collection* it belongs to (e.g. between a tree and the forest this tree belongs to) [91, 34].

In this section, we describe expressive means relevant for the representation of aggregated objects and the development of the DLs $\mathcal{SHIQ}$ and $\mathcal{RIQ}$.

5.1 Adding transitivity

Coming back to representing aggregated objects in ontologies using DLs, we observe that the DL $\mathcal{ALC}$ provides no means to express that a relation is *transitive*. For example, in $\mathcal{ALC}$, the concept

$$\texttt{Device} \sqcap \exists \texttt{hasPart}.(\texttt{ReactorCore} \sqcap \exists \texttt{hasPart}.\texttt{ControlRod})$$

is *not* subsumed by $\texttt{Device} \sqcap \exists \texttt{hasPart}.\texttt{ControlRod}$, although the first concept is a specialisation of the second one under the assumption that `hasPart` is interpreted as a transitive relation.

Thus the adequate modelling of aggregated objects asks for the extension of $\mathcal{ALC}$ with some form of transitivity. As mentioned in Section 3, there are

at least two such possible extensions. After investigating their expressive power and complexity, we have chosen the "cheaper" possibility: by $\mathcal{S}$, we refer to the description logic $\mathcal{ALC}$ extended with transitive roles.[3] Obviously, $\mathcal{S}$ provides the means to represent the general part-whole relation as a transitive relation by asserting that `partOf` is a transitive role. Additionally, since $\mathcal{S}$ has a tree model property, all satisfiable concepts and TBoxes have a model in which `partOf` is interpreted as a strict partial order.

Tableau algorithm for $\mathcal{S}$ A naive extension of the tableau algorithm for $\mathcal{ALC}$ sketched in Section 4 to transitive roles does not necessarily terminate: assume the algorithm is started with the concept $C_0 := C \sqcap \exists r.C \sqcap \forall r.(\exists r.C)$ for r a transitive role. After some rule applications, the algorithm has generated three nodes, x, y, and z where y is an r-successor of x, z is an r-successor of y, and $C_0, \forall r.(\exists r.C) \in \mathcal{L}(x)$, $\exists r.C \in \mathcal{L}(y)$, and $C \in \mathcal{L}(z)$. Since r is a transitive role, we could make z an r-successor of x, but this would destroy the tree structure that turned out to be quite useful. Instead, we do something which has the same effect: we add $\forall r.(\exists r.C)$ to $\mathcal{L}(y)$. More precisely, if $\forall r.C \in \mathcal{L}(x)$ and x has an r-successor y, we add both C and $\forall r.C$ to y's label. In this case, this yields $\exists r.C \in \mathcal{L}(z)$. It can easily be seen that the repeated application of this modification builds an infinite r-chain, and thus leads to non-termination. To regain termination without corrupting soundness or completeness of the algorithm, we use the blocking technique mentioned in Section 4: we stop generating new successors of a node z in case there is another node z' with $\mathcal{L}(z) \subseteq \mathcal{L}(z')$. In this case, we say that z' *blocks* z, and we can build a model by "merging" z and z' (and all other nodes which z' blocks), thus building a finite, possibly cyclic model.

5.2 Further adding inverse roles

When modelling aggregated objects using $\mathcal{S}$ and using both `partOf`and `hasPart`, we might end up with an inadequate representation in the following sense: for example, extending the TBox in the beginning of Section 5 with

$$\texttt{NuclReactor} \sqcap \exists \texttt{hasPart.Faulty} \dot{\sqsubseteq} \texttt{Dangerous},$$

we would assume that `ControlRod`$\sqcap$`Faulty` is subsumed by $\exists$`partOf.Dangerous` w.r.t. to this TBox—which is only the case if `partOf` were *the inverse* of `hasPart`, i.e., if $\langle x, y\rangle \in \texttt{hasPart}^{\mathcal{I}}$ iff $\langle y, x\rangle \in \texttt{partOf}^{\mathcal{I}}$. Thus we extend $\mathcal{S}$ with inverse roles, which yields the DL called $\mathcal{SI}$ and allows to describe both objects by means of the wholes they belong to and by means of the parts they have. Substituting `hasPart` by $\texttt{partOf}^{-}$ in the last example yields a TBox with respect to which `ControlRod` $\sqcap$ `Faulty` is indeed subsumed by $\exists$`partOf.Dangerous`.

Tableau algorithm for $\mathcal{SI}$ Intuitively, we can extend the tableau algorithm for $\mathcal{S}$ as follows to yield a decision procedure for satisfiability of $\mathcal{SI}$-concepts: if

[3] The logic $\mathcal{S}$ has previously been called $\mathcal{ALC}_{R^+}$, but this becomes too cumbersome when adding letters to represent additional features.

$\forall r.C \in \mathcal{L}(w)$, instead of adding C only to r-successors, we also add C to $\mathsf{Inv}(r)$-*predecessors*.[4] For example, for the concept $\exists r^-.(C \sqcap \forall r.B) \in \mathcal{L}(x)$, we would first create an r^--successor y of x with $C \sqcap \forall r.B \in \mathcal{L}(y)$. For $\forall r.B \in \mathcal{L}(y)$, we would then add B to $\mathcal{L}(x)$ since x is an r-predecessor of y. Moreover, the blocking condition has to be more strict: for z' to block z, they must have identical labels, i.e., $\mathcal{L}(z) = \mathcal{L}(z')$. Finally, blocking becomes necessarily "dynamic": in the presence of inverse roles, node labels influence each other up and down the completion tree. Thus the label of a node x blocking some node y further down the tree can change due to some of its other predecessors, the node labels of x and y become different, and we must "unblock" them.

This tableau algorithm decides satisfiability (and thus subsumption) of $\mathcal{SI}$-concepts w.r.t. TBoxes. Moreover, we were able to prove that, in the absence of a TBox and employing a certain strategy and a more intricate blocking condition, it uses polynomial space only. This is one example for the fact that the definition of the blocking condition is not only crucial for the correctness of the algorithm, but also for its complexity. As a consequence, $\mathcal{ALC}$ without TBoxes and with transitive and inverse roles is of the same complexity as pure $\mathcal{ALC}$, namely PSPACE-complete [49].

5.3 Further adding role inclusion axioms

To represent, beside the general part-whole relation, certain sub-part-whole relations such as "is a component of" or "is an ingredient of", we can use role inclusion axioms [42].

A *role inclusion axiom* is an expression of the form $r \dot{\sqsubseteq} s$, where r and s are (possibly inverse) roles. A *role hierarchy* is a finite set of role inclusion axioms. An interpretation $\mathcal{I}$ *satisfies* a role hierarchy $\mathcal{R}$ iff $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ for each $r \dot{\sqsubseteq} s$ in $\mathcal{R}$. Such an interpretation is called a *model* of $\mathcal{R}$. Satisfiability and subsumption w.r.t. role hierarchies are defined in the obvious way. $\mathcal{SHI}$ is the extension of $\mathcal{SI}$ with role hierarchies.

Adding role hierarchies to $\mathcal{SI}$ has mainly two consequences: firstly, we can introduce (possibly transitive—depending on the additional relation) role names such as `hasComp` or `hasIngredient` and add role inclusion axioms `hasComp` $\dot{\sqsubseteq}$ `hasPart` and `hasIngredient` $\dot{\sqsubseteq}$ `hasPart`. This turns out to be quite useful in various applications since it allows for a concise and natural description not only of aggregated objects.

Secondly, $\mathcal{SHI}$ (as well as $\mathcal{SH}$ and $\mathcal{SHIQ}$) has the expressive power for the *internalisation* of TBoxes [3, 45]. This technique polynomially reduces reasoning w.r.t. a general TBox to pure concept reasoning as follows. We introduce a new transitive role name $u \in \mathbf{R}_+$ and specify that u is a super-role of all roles and their respective inverses. This implies that, in connected models, u behaves like a universal role, i.e., u relates all elements of the interpretation domain; cf. Section 3. Since each satisfiable $\mathcal{SHI}$ concept is satisfiable in a connected model,

[4] To avoid considering roles such as r^{--}, we use $\mathsf{Inv}(r)$ to denote r^- if r is a role name and s if $r = s^-$ for a role name s.

it can be shown that a concept C is satisfiable w.r.t. $\{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\}$ iff $\exists u.C \sqcap \forall u. \bigsqcap_{1 \leq i \leq n} (C_i \Rightarrow D_i)$ is satisfiable.

Tableau algorithm for $\mathcal{SHI}$ Basically, the extension of the $\mathcal{SI}$ tableau algorithm to $\mathcal{SHI}$ involves an adaption of the notion of an "r-successor" to take into account role hierarchies [45]: if y is an r-successor of x and $r \dot{\sqsubseteq} s$ is in the role hierarchy, then y is also an s-successor of x. An analogous adaption for predecessors is also required in the presence of inverse roles, and transitive roles require a further, rather complex adaption of the propagation of universal value restrictions $\forall r.C$. Moreover, the correctness proof of the tableau algorithm becomes more complex since the tree structure the algorithm works on does no longer correspond to the relational structure that is to be built in case that the algorithm answers "satisfiable": this is already the case in the presence of transitive roles, but becomes more notable if, additionally, role hierarchies are taken into account.

5.4 Further adding number restrictions

In general, when describing the relevant concepts of an application domain, it seems to be natural to describe an object by restricting the number of objects it is related to via a certain relation. For example, the following are concept definitions for pipes and forks:

$$\texttt{Pipe} \doteq \texttt{Connection} \sqcap (= 1\ \texttt{partOf}^-\ \texttt{Input}) \sqcap (= 1\ \texttt{partOf}^-\ \texttt{Output})$$
$$\texttt{Fork} \doteq \texttt{Connection} \sqcap (= 1\ \texttt{partOf}^-\ \texttt{Input}) \sqcap (\geq 2\ \texttt{partOf}^-\ \texttt{Output})$$

Before adding number restriction to $\mathcal{SHI}$, we have to define *simple* roles since only simple roles are allowed in number restrictions—without that restriction, satisfiability of $\mathcal{SHI}$ extended with number restrictions is undecidable [49].

A (possibly inverse) role is called *simple* if it is neither transitive nor has a transitive sub-role. $\mathcal{SHIQ}$ is obtained from $\mathcal{SHI}$ by allowing, additionally, for concepts of the form $(\geqslant ns.C)$ and $(\leqslant ns.C)$ for n a non-negative integer, s a simple role, and C a $\mathcal{SHIQ}$-concept. The semantics of number restrictions is given in Section 3.

In contrast to $\mathcal{SHI}$, $\mathcal{SHIQ}$ lacks the finite model property. That is, there are concepts that are satisfiable, but only in *infinite* models. For example, for r a transitive role and $s \dot{\sqsubseteq} r$, each model of the following concept contains an infinite, acyclic s-chain: $\neg A \sqcap \exists s.A \sqcap \forall r.((\exists s.A) \sqcap (\geqslant 1 s^-.\top))$.

As mentioned in Section 2, state-of-the-art DL reasoners such as FaCT and Racer implement tableau algorithms for $\mathcal{SHIQ}$ [41, 37]. Thus, $\mathcal{SHIQ}$ forms the logical basis of ontology editors Rice and Oiled [22, 12], and of the intelligent conceptual modelling tool Icom [31].

Tableau algorithm for $\mathcal{SHIQ}$ It is not difficult to see that, in the presence of number restrictions, we have to add two new rules to our tableau algorithm:

1. if $(\geqslant nr.C) \in \mathcal{L}(x)$ and x has less than n r-neighbours with C in their label, then generate these missing r-neighbours and set their labels to $\{C\}$.

2. if $(\leqslant nr.C) \in \mathcal{L}(x)$ and x has more than n r-neighbours with C in their label, then merge some of them, so that only n remain.

However, this is not sufficient. Firstly, such a naive extension might easily yield a "yo-yo" effect: for example, if applied to a node x with $(\geqslant 3r.C \sqcap D), (\leqslant 2r.C) \in \mathcal{L}(x)$, the above tableau algorithm would generate three r-successors y_i with $C \sqcap D \in \mathcal{L}(y_i)$, break down the conjunctions in $C \sqcap D \in \mathcal{L}(y_i)$, and then notice that there are too many r-successors y_i of x with $C \in \mathcal{L}(y_i)$ for $(\leqslant 2r.C) \in \mathcal{L}(x)$. Thus two of them would be merged into a single one. Now there are not enough r-successors for $(\geqslant 3r.C \sqcap D)$, so one would be generated, and so on, thus leading to non-termination. To re-gain termination, we can use, for example, an explicit inequality relation $\neq$ that prevents nodes that were introduced for one $(\geqslant nr.C)$ from being merged again later. Moreover, we extend the notion of a "clash" to cases where $(\leqslant nr.C) \in \mathcal{L}(x)$ and x has more than n $\neq$-distinct r-successors with C in their label.

Secondly, consider the concept $C := (\geqslant 3r.B) \sqcap (\leqslant 1r.A) \sqcap (\leqslant 1r.\neg A)$. So far, for $C \in \mathcal{L}(x)$, the tableau algorithm would generate three r-successors y_i of x with $\{B\} = \mathcal{L}(y_i)$, and stop with the answer "C is satisfiable". However, the concept C is obviously unsatisfiable: the algorithm's unsoundness is due to its ignorance of which of the y_i are instances of A and which are instances of $\neg A$. To overcome this problem, we add a third rule

3. if $(\leqslant nr.C) \in \mathcal{L}(x)$ and y is an r-neighbour of x, then non-deterministically add C or $\neg C$ to $\mathcal{L}(y)$.

Thirdly, one also needs to modify the blocking condition—otherwise, the algorithm would still be unsound. Roughly spoken, the $\mathcal{SHIQ}$ blocking condition involves two pairs of subsequent nodes whose labels must coincide pairwise. Together, these three modifications indeed yield a decision procedure for the satisfiability of $\mathcal{SHIQ}$ [49].

Interestingly, the first proposal of the $\mathcal{SHIQ}$ blocking condition was so strict that it delayed blocking severely, thus enlarging the search space for a model dramatically and degrading the performance of FaCT. Investigating the soundness and completeness proof of the $\mathcal{SHIQ}$ tableau algorithm more closely, we were able to devise an new blocking condition which still ensures soundness, completeness, and termination, but was less strict [48]. Intuitively, node labels have only to be equal for "relevant concepts" in the respective nodes, a fact that made the formulation of the new blocking condition rather intricate. However, an empirical evaluation of the new tableau algorithm in FaCT showed that this more intricate but less strict blocking condition payes off: it improves performance up to two orders of magnitude.

Concerning worst-case complexity, both the original and the optimised $\mathcal{SHIQ}$ tableau algorithm are far from being optimal: in the worst case, they run in 2NExptime, whereas satisfiability of $\mathcal{SHIQ}$-concepts is known to be in ExpTime, even with numbers in number restrictions coded in binary [83]. Despite this worst-case sub-optimality, its implementation in the FaCT and Racer systems behave surprisingly well in practice [48, 37]. However, the worst-case com-

plexity implies that there exist rather small example inputs for which these systems need so much time that they are practically not terminating [14].

5.5 Further adding more expressive role inclusion axioms

Although $\mathcal{SHIQ}$ is rather expressive, there is a common phenomenon that $\mathcal{SHIQ}$ is not able to express, and that would be useful in many applications, especially for those involving aggregated objects. This phenomenon is sometimes coined *propagation of properties*: for example, one wants to express that a fracture located in the shaft of the femur (which is a division of the femur) is a fracture located in the femur. Or one might want to express that the owner of a thing also owns the parts of this thing. The importance of this expressive means is illustrated by the fact that the Grail DL [44, 66], which was designed for medical terminologies, is able to express these propagations (although it is quite weak in other respects). In two other medical terminology applications, rather complex work arounds to represent propagations can be found: SEP-triplets[5] in [76] and right-identities in [79]. Finally, the CycL language provides the `transfersThro` statement for similar propagations [53]. So far and to the best of our knowledge, none of these systems were proven to handle these propagations in a sound and complete way.

It is rather straightforward to extend $\mathcal{SHIQ}$ to allow for the propagation of properties: obviously, it suffices to extend role hierarchies to the general role inclusion axioms, see Section 3. For the first example, one would introduce an axiom $\texttt{hasLocation} \circ \texttt{divisionOf} \dot{\sqsubseteq} \texttt{hasLocation}$ and, indeed, w.r.t. this axiom,

$$\texttt{Fracture} \sqcap \exists \texttt{hasLocation}.(\texttt{Shaft} \sqcap \exists \texttt{divisionOf}.\texttt{Femur})$$

is subsumed by $\texttt{Fracture} \sqcap \exists \texttt{hasLocation}.\texttt{Femur}$. For the second example, one would introduce an axiom $\texttt{owns} \circ \texttt{hasPart} \sqsubseteq \texttt{owns}$ and, w.r.t. this axiom,

$$\exists \texttt{owns}.(\texttt{Bicycle} \sqcap \exists \texttt{hasPart}.\texttt{SuspensionFork})$$

is subsumed by $\exists \texttt{owns}.\texttt{SuspensionFork}$.

As mentioned in Section 3, results in grammar and description logics imply that extending $\mathcal{ALC}$ with role inclusion axioms of the form $r \circ s \dot{\sqsubseteq} t$ yields a logic for which satisfiability and subsumption are undecidable [10, 11, 90]. However, for expressing propagation of properties, we only need axioms of the form $r \circ s \dot{\sqsubseteq} s$ or $s \circ r \dot{\sqsubseteq} s$ [44, 65]. Unfortunately, extending $\mathcal{SHIQ}$ with this restricted form of axioms still yields an undecidable logic [47].

One way to re-gain decidability would be to restrict the underlying logic $\mathcal{SHIQ}$. Since we have argued that, especially for the representation of aggregated objects, the concept- and role-forming operators of $\mathcal{SHIQ}$ are crucial, we have chosen a different approach, namely to further restrict the role inclusion axioms: further restricting role hierarchies to not contain "affecting cycles" of length

[5] *SEP-triplets* are used both to compensate for the absence of transitive roles in $\mathcal{ALC}$, and to express the propagation of properties across a distinguished "part-of" role.

greater than one finally yields a decidable logic. Roughly speaking, "affecting" is the transitive closure of the relation "directly affecting", and r directly affects s if $r \circ s \dot{\sqsubseteq} s$, $s \circ r \dot{\sqsubseteq} s$, or $r \dot{\sqsubseteq} s$ is contained in the role hierarchy. A role hierarchy containing no "affecting" cycles of length greater than one is called *acyclic*, and the extension of $\mathcal{SHIQ}$ with acyclic role hierarchies is called $\mathcal{RIQ}$.

In $\mathcal{RIQ}$, we can model the propagation of properties as mentioned above, and the restriction to acyclicity does not seem to be too severe since non-trivial cycles seem to indicate modelling flaws [65].

Tableau algorithm for $\mathcal{RIQ}$ The tableau algorithm for $\mathcal{RIQ}$ [47] involves two pre-processing steps that transform the role hierarchy into a more explicit and manageable structure. Firstly, acyclic role hierarchies are unfolded in a similar way as acyclic TBoxes can be unfolded [59], thus making all implicit implications explicit. As a result of this unfolding, we obtain, for each role name r, a regular expressions τ_r on role names. Secondly, we construct, for each τ_r, a non-deterministic finite automata $\mathcal{A}_r$ which accepts $L(\tau_r)$.

Then, in the tableau rules, we add three rules

1. if $\forall r.C \in \mathcal{L}(x)$, then we add $\forall \mathcal{A}_r.C$.
2. if $\forall \mathcal{A}.C \in \mathcal{L}(x)$ and x has an s-successor y, then we add $\forall \mathcal{A}'.C$ to $\mathcal{L}(y)$ for each automaton $\mathcal{A}'$ that is the result of $\mathcal{A}$ reading s, i.e., $\mathcal{A}'$ is obtained from $\mathcal{A}$ by simply changing the initial state to a state that is reachable from $\mathcal{A}$'s initial state by an s transition.
3. if $\forall \mathcal{A}.C \in \mathcal{L}(x)$ and $\varepsilon \in L(\mathcal{A})$, then add C to $\mathcal{L}(x)$.

The pre-processing together with these three rules can be shown to yield a decision procedure for $\mathcal{RIQ}$.

This tableau algorithm for $\mathcal{RIQ}$ is implemented successfully in FaCT. The additional overhead introduced by using automata in tableau rules seems to pay off since it does not degrade the performance of FaCT, yields a more readable algorithm, and can draw additional inferences like the medical one above [47].

References

1. C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In *Proc. of CSL'99*, vol. 1683 of *LNCS*. Springer-Verlag, 1999.
2. C. Areces, P. Blackburn, and M. Marx. The computational complexity of hybrid temporal logics. *Logic Journal of the IGPL*, 8(5), 2000.
3. F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of IJCAI-91*, Sydney, 1991.
4. F. Baader. Using automata theory for characterizing the semantics of terminological cycles. *Annals of Mathematics and Artificial Intelligence*, 18(2–4):175–219, 1996.
5. F. Baader, S. Brandt, and R. Küsters. Matching under side conditions in description logics. In B. Nebel, ed., *Proc. of IJCAI-01*. Morgan Kaufmann, 2001.
6. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

7. F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems, or: Making KRIS get a move on. *Applied Artificial Intelligence*, 4:109–132, 1994.
8. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001. An abridged version appeared in *Tableaux 2000*, vol. 1847 of LNAI, 2000. Springer-Verlag.
9. F. Baader and A.-Y. Turhan. On the problem of computing small representations of least common subsumers. In *Proc. of KI 2002*, vol. 2479 of *LNAI*. Springer-Verlag, 2002.
10. M. Baldoni. *Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension*. PhD thesis, Dipartimento di Informatica, Università degli Studi di Torino, Italy, 1998.
11. M. Baldoni, L. Giordano, and A. Martelli. A tableau calculus for multimodal logics and some (un)decidability results. In *Proc. of TABLEAUX-98*, vol. 1397 of *LNAI*. Springer-Verlag, 1998.
12. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a reason-able ontology editor for the semantic web. In *Proc. of DL 2001*. CEUR (http://ceur-ws.org/), 2001.
13. M. Ben-Ari, J.Y. Halpern, and A. Pnueli. Deterministic propositional dynamic logic: finite models, complexity and completeness. *J. of Computer and System Science*, 25:402–417, 1982.
14. D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML Class Diagrams using Description Logic Based Systems. In *Proc. of the KI'2001 Workshop on Applications of Description Logics*. CEUR (http://ceur-ws.org/), 2001.
15. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, 284(5):34–43, 2001.
16. P. Borst, H. Akkermans, and J. Top. Engineering ontologies. *International Journal of Human-Computer Studies*, 46:365–406, 1997.
17. S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In *Proc. of KR-02*. Morgan Kaufmann, 2002.
18. D. Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms*. PhD thesis, Dip. di Inf. e Sist., Univ. di Roma "La Sapienza", 1996.
19. D. Calvanese, G. De Giacomo, and M. Lenzerini. Structured objects: Modeling and reasoning. In *Proc. of DOOD-95*, vol. 1013 of *LNCS*, pages 229–246, 1995.
20. D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *Proc. of IJCAI-99*. Morgan Kaufmann, 1999.
21. D. Calvanese, G. De Giacomo, M. Lenzerini, and D. Nardi. Reasoning in expressive description logics. In A. Robinson and A. Voronkov, eds., *Handbook of Automated Reasoning*. Elsevier Science Publishers, 1999.
22. R. Cornet. Rice ontology editor. Homepage at `http://www.b1g-systems.com/ronald/rice/`.
23. G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics (extended abstract). In *Proc. of AAAI-94*. AAAI Press, 1994.
24. G. De Giacomo and M. Lenzerini. Tbox and Abox reasoning in expressive description logics. In *Proc. of KR-96*. Morgan Kaufmann, 1996.
25. S. Demri. The complexity of regularity in grammar logics and related modal logics. *J. of Logic and Computation*, 11(6), 2001.

26. F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Proc. of KR-91*, Boston, MA, USA, 1991.
27. F. M. Donini and F. Massacci. Exptime tableaux for $\mathcal{ALC}$. *Artificial Intelligence*, 124(1):87–138, 2000.
28. D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.
29. K. Fine. In so many possible worlds. *Notre Dame Journal of Formal Logics*, 13:516–520, 1972.
30. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Science*, 18:194–211, 1979.
31. E. Franconi and G. Ng. The i.com tool for intelligent conceptual modelling. In *Working Notes of the ECAI2000 Workshop KRDB2000*. CEUR (http://ceur-ws.org/), 2000.
32. G. Gargov, S. Passy, and T. Tinchev. Modal environment for Boolean speculations. In D. Skordev, ed., *Mathematical Logic and Applications*, pages 253–263. Plenum Publ. Co., New York, 1987.
33. L. Georgieva, U. Hustadt, and R. A. Schmidt. Hyperresolution for guarded formulae. *J. of Symbolic Logic*, 2003. To appear.
34. P. Gerstl and S. Pribbenow. Midwinters, end games and bodyparts. *International Journal of Human-Computer Studies*, 43:847–864, 1995.
35. T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, eds., *Formal Ontology in Conceptual Analysis and Knowledge Representation*, 1993. Kluwer Academic Publishers.
36. V. Haarslev and R. Möller. Consistency testing: The RACE experience. In *Proc. TABLEAUX 2000*, vol. 1847 of *LNAI*. Springer-Verlag, 2000.
37. V. Haarslev and R. Möller. RACER system description. In *IJCAR-01*, vol. 2083 of *LNAI*. Springer-Verlag, 2001.
38. J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logic of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.
39. J. Hladik. Implementation and optimisation of a tableau algorithm for the guarded frament. In *Proc. TABLEAUX 2002*, vol. 2381 of *LNAI*. Springer-Verlag, 2002.
40. I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.
41. I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of KR-98*. Morgan Kaufmann, 1998.
42. I. Horrocks and G. Gough. Description logics with transitive roles. In M.-C. Rousset, R. Brachmann, F. Donini, E. Franconi, I. Horrocks, and A. Levy, eds., *Proc. of DL'97*, pages 25–28, 1997.
43. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proc. of AAAI-02*, 2002.
44. I. Horrocks, A. Rector, and C. Goble. A description logic based schema for the classification of medical data. In *Working Notes of the ECAI-96 Workshop KRDB-96*. CEUR (http://ceur-ws.org/), 1996.
45. I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *J. of Logic and Computation*, 1999.
46. I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}(D)$ description logic. In B. Nebel, ed., *Proc. of IJCAI-01*. Morgan Kaufmann, 2001.
47. I. Horrocks and U. Sattler. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. In *Proc. of CADE-19*. LNAI, Springer-Verlag, 2003 (to appear).

48. I. Horrocks and U. Sattler. Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. of ECAI 2002*, 2002.
49. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, vol. 1705 of *LNAI*. Springer-Verlag, 1999.
50. U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In *Proc. TABLEAUX 2000*, vol. 1847 of *LNAI*, pages 67–71. Springer-Verlag, 2000.
51. D. Kozen. Results on the propositional μ-calculus. In *Automata, Languages and Programming, 9th Colloquium*, vol. 140 of *LNCS*. Springer-Verlag, 1982.
52. R. Küsters. *Non-Standard Inferences in Description Logics*, vol. 2100 of *LNAI*. Springer-Verlag, 2001.
53. D. B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems*. Addison Wesley Publ. Co., Reading, Massachussetts, 1989.
54. C. Lutz. Complexity of terminological reasoning revisited. In *Proc. of LPAR'99*, LNAI, pages 181–200. Springer-Verlag, 1999.
55. C. Lutz. *The Complexity of Description Logics with Concrete Domains*. PhD thesis, RWTH Aachen, 2002.
56. C. Lutz. Description logics with concrete domains—a survey. In *Advances in Modal Logics Vol. 4*. World Scientific Publishing Co. Pte. Ltd., 2003.
57. C. Lutz and U. Sattler. The complexity of reasoning with boolean modal logics. In F. Wolter, H. Wansing, M. de Rijke, and M. Zakharyaschev, eds., *Advances in Modal Logics 3*. CSLI Publications, Stanford, 2001.
58. R. Molitor. *Unterstützung der Modellierung verfahrenstechnischer Prozesse durch Nicht-Standardinferenzen in Beschreibungslogiken*. PhD thesis, RWTH Aachen, 2000.
59. B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, vol. 422 of *LNAI*. Springer-Verlag, 1990.
60. P. Patel-Schneider, D. McGuinness, R. Brachman, L. Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rationale. *SIGART Bulletin*, 2(3):108–113, 1991.
61. P. F. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39:263–272, 1989.
62. V. R. Pratt. Models of program logics. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, 1979.
63. A. Prior. *Past, Present and Future*. Oxford University Press, 1967.
64. Protégé. Homepage at `http://protege.stanford.edu/`, 2003.
65. A. Rector. Analysis of propagation along transitive roles: Formalisation of the galen experience with medical ontologies. In *Proc. of DL 2002*. CEUR (http://ceur-ws.org/), 2002.
66. A. Rector, S. Bechhofer, C. A. Goble, I. Horrocks, W. A. Nowlan, and W. D. Solomon. The GRAIL concept modelling language for medical terminology. *AI in Medicine*, 9:139–171, 1997.
67. N. Rescher and A. Urquhart. *Temporal Logic*. Springer-Verlag, 1971.
68. U. Sattler. A concept language extended with different kinds of transitive roles. In *Proc. of KI'96*, vol. 1137 of *LNAI*. Springer-Verlag, 1996.
69. U. Sattler. *Terminological knowledge representation systems in a process engineering application*. PhD thesis, RWTH Aachen, 1998.
70. U. Sattler. Description logics for the representation of aggregated objects. In W. Horn, ed., *Proc. of ECAI-2000*. IOS Press, Amsterdam, 2000.
71. U. Sattler and M. Y. Vardi. The hybrid μ-calculus. In *IJCAR-01*, vol. 2083 of *LNAI*. Springer-Verlag, 2001.

72. A. Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.
73. K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI-91*. Morgan Kaufmann, 1991.
74. M. Schmidt-Schauss. Subsumption in KL-ONE is undecidable. In *Proc. of KR-89*, Boston (USA), 1989.
75. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
76. S. Schulz and U. Hahn. Parts, locations, and holes - formal reasoning about anatomical structures. In *Proc. of AIME 2001*, vol. 2101 of *LNAI*. Springer-Verlag, 2001.
77. P. M. Simons. *Parts. A study in Ontology*. Oxford: Clarendon, 1987.
78. E. Spaan. The complexity of propositional tense logics. In M. de Rijke, ed., *Diamonds and Defaults*. Kluwer Academic Publishers, 1993.
79. K.A. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. *J. of the Amer. Med. Informatics Ass.*, 2000. Fall Symposium Special Issue.
80. R. Stevens, I. Horrocks, C. Goble, and S. Bechhofer. Building a bioinformatics ontology using OIL. *IEEE Inf. Techn. in Biomedicine.*, 6(2):135–141, 2002.
81. R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Computation*, 54:121–141, 1982.
82. Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional μ-calculus. *Information and Computation*, 81:249–264, 1989.
83. S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.
84. S. Tobies. PSPACE reasoning for graded modal logics. *J. of Logic and Computation*, 11(1):85–106, 2001.
85. M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13, 1998.
86. M. Y. Vardi. The taming of converse: Reasoning about two-way computations. In *Proc. of the 4th Workshop on Logics of Programs*, vol. 193 of *LNCS*. Springer-Verlag, 1985.
87. M. Y. Vardi. Why is modal logic so robustly decidable? In N. Immerman and P. G. Kolaitis, eds., *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS*. American Mathematical Society, 1997.
88. M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of ICALP'98*, vol. 1443 of *LNCS*. Springer-Verlag, 1998.
89. M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Science*, 32:183–221, 1986.
90. M. Wessel. Obstacles on the way to qualitative spatial reasoning with description logics: Some undecidability results. In *Proc. of DL 2001*. CEUR (http://ceur-ws.org/), 2001.
91. M.E. Winston, R. Chaffin, and D. Herrmann. A taxonomy of part whole relations. *Cognitive Science*, 11:417–444, 1987.