Description Logics

Franz Baader[1], Ian Horrocks[2], and Ulrike Sattler[3]

[1] Institut für Theoretische Informatik, TU Dresden, Germany,
baader@tcs.inf.tu-dresden.de
[2] Computing Laboratory, Oxford University, Oxford, UK,
ian.horrocks@comlab.ox.ac.uk
[3] Department of Computer Science, University of Manchester, Manchester, UK,
sattler@cs.man.ac.uk

Summary. In this chapter, we explain what description logics are and why they make good ontology languages. In particular, we introduce the description logic $\mathcal{SHIQ}$, which has formed the basis of several well-known ontology languages, including OWL. We argue that, without the last decade of basic research in description logics, this family of knowledge representation languages could not have played such an important rôle in this context.

Description logic reasoning can be used both during the design phase, in order to improve the quality of ontologies, and in the deployment phase, in order to exploit the rich structure of ontologies and ontology based information. We discuss the extensions to $\mathcal{SHIQ}$ that are required for languages such as OWL and, finally, we sketch how novel reasoning services can support building ontologies.

1 Introduction

The aim of this section is to give a brief introduction to description logics, and to argue why they are well-suited as ontology languages. In the remainder of the chapter we will put some flesh on this skeleton by providing more technical details with respect to the theory of description logics, and their relationship to state of the art ontology languages. More detail on these and other matters related to description logics can be found in [6].

1.1 Ontologies

There have been many attempts to define what constitutes an ontology, perhaps the best known (at least amongst computer scientists) being due to Gruber: "an ontology is an explicit specification of a conceptualisation" [49].[1] In this context, a conceptualisation means an abstract model of some aspect

[1] This was later elaborated to "a formal specification of a shared conceptualisation" [21].

S. Staab and R. Studer (eds.), *Handbook on Ontologies,* International Handbooks on Information Systems, DOI 10.1007/978-3-540-92673-3,

of the world, taking the form of a definition of the properties of important concepts and relationships. An explicit specification means that the model should be specified in some unambiguous language, making it amenable to processing by machines as well as by humans.

Ontologies are becoming increasingly important in fields such as knowledge management, information integration, cooperative information systems, information retrieval and electronic commerce. One application area which has recently seen an explosion of interest is the so called *Semantic Web* [18], where ontologies are set to play a key rôle in establishing a common terminology between agents, thus ensuring that different agents have a shared understanding of terms used in semantic markup.

The effective use of ontologies requires not only a well-designed and well-defined ontology language, but also support from reasoning tools. Reasoning is important both to ensure the quality of an ontology, and in order to exploit the rich structure of ontologies and ontology based information. It can be employed in different phases of the ontology life cycle. During ontology design, it can be used to test whether concepts are non-contradictory and to derive implied relations. In particular, one usually wants to compute the concept hierarchy, i.e. the partial ordering of named concepts based on the subsumption relationship. Information on which concept is a specialization of another, and which concepts are synonyms, can be used in the design phase to test whether the concept definitions in the ontology have the intended consequences or not. This information is also very useful when the ontology is deployed.

Since it is not reasonable to assume that all applications will use the same ontology, interoperability and integration of different ontologies is also an important issue. Integration can, for example, be supported as follows: after the knowledge engineer has asserted some inter-ontology relationships, the integrated concept hierarchy is computed and the concepts are checked for consistency. Inconsistent concepts as well as unintended or missing subsumption relationships are thus signs of incorrect or incomplete inter-ontology assertions, which can then be corrected or completed by the knowledge engineer.

Finally, reasoning may also be used when the ontology is deployed. As well as using the pre-computed concept hierarchy, one could, for example, use the ontology to determine the consistency of facts stated in annotations, or infer relationships between annotation instances and ontology classes. More precisely, when searching web pages annotated with terms from the ontology, it may be useful to consider not only exact matches, but also matches with respect to more general or more specific terms – where the latter choice depends on the context. However, in the deployment phase, the requirements on the efficiency of reasoning are much more stringent than in the design and integration phases.

Before arguing why description logics are good candidates for such an ontology language, we provide a brief introduction to and history of description logics.

1.2 Description Logics

Description logics (DLs) [6, 16, 30] are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way. The name *description logics* is motivated by the fact that, on the one hand, the important notions of the domain are described by concept *descriptions*, i.e. expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the particular DL. On the other hand, DLs differ from their predecessors, such as semantic networks and frames, in that they are equipped with a formal, *logic*-based semantics.

In this introduction, we only illustrate some typical constructors by an example. Formal definitions are given in Sect. 2. Assume that we want to define the concept of "A man that is married to a doctor and has at least five children, all of whom are professors." This concept can be described with the following concept description:

$$\mathsf{Human} \sqcap \neg\mathsf{Female} \sqcap \exists\mathsf{married}.\mathsf{Doctor} \sqcap (\geq 5\,\mathsf{hasChild}) \sqcap \forall\mathsf{hasChild}.\mathsf{Professor}$$

This description employs the Boolean constructors *conjunction* ($\sqcap$), which is interpreted as set intersection, and *negation* ($\neg$), which is interpreted as set complement, as well as the *existential restriction* constructor ($\exists R.C$), the *value restriction* constructor ($\forall R.C$), and the *number restriction* constructor ($\geq n\,R$). An individual, say Bob, belongs to $\exists\mathsf{married}.\mathsf{Doctor}$ if there exists an individual that is married to Bob (i.e. is related to Bob via the $\mathsf{married}$ role) and is a doctor (i.e. belongs to the concept Doctor). Similarly, Bob belongs to $(\geq 5\,\mathsf{hasChild})$ iff he has at least five children, and he belongs to $\forall\mathsf{hasChild}.\mathsf{Professor}$ iff all his children (i.e. all individuals related to Bob via the $\mathsf{hasChild}$ role) are professors.

In addition to this description formalism, DLs are usually equipped with a terminological and an assertional formalism. In its simplest form, *terminological axioms* can be used to introduce names (abbreviations) for complex descriptions. For example, we could introduce the abbreviation $\mathsf{HappyMan}$ for the concept description from above. More expressive terminological formalisms allow the statement of constraints such as

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

which says that only humans can have human children. A set of terminological axioms is called a TBox. The *assertional formalism* can be used to state properties of individuals. For example, the assertions

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

state that Bob belongs to the concept $\mathsf{HappyMan}$ and that Mary is one of his children. A set of such assertions is called an ABox, and the named individuals that occur in ABox assertions are called ABox individuals.

Description logic systems provide their users with various inference capabilities that deduce implicit knowledge from the explicitly represented knowledge. 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 description is always interpreted as a subset of the second description. For example, given the definition of HappyMan from above, HappyMan is subsumed by ∃hasChild.Professor – since instances of HappyMan have at least five children, all of whom are professors, they also have a child that is a professor. The *instance* algorithm determines 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 from above and the definition of HappyMan, MARY is an instance of Professor. The *consistency* algorithm determines whether a knowledge base (consisting of a set of assertions and a set of terminological axioms) is non-contradictory. For example, if we add ¬Professor(MARY) to the two assertions from above, then the knowledge base containing these assertions together with the definition of HappyMan from above is inconsistent.

In order to ensure 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. Roughly, the research related to this issue can be classified into the following four phases.

Phase 1 (1980–1990) was mainly concerned with implementation of systems, such as KLONE, K-REP, BACK, and LOOM [24, 70, 71, 80]. 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 [73]. These algorithms are usually relatively efficient (polynomial), but they have the disadvantage that they are complete only for very inexpressive DLs, i.e. for more expressive DLs they cannot detect all 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 [23, 74]. As a reaction, the implementors of the CLASSIC system (the first industrial-strength DL system) carefully restricted the expressive power of their DL [22, 79].

Phase 2 (1990–1995) started with the introduction of a new algorithmic paradigm into DLs, so-called *tableau-based algorithms* [40, 54, 88]. 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 the 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 [9, 27]. This phase also saw a thorough analysis of the complexity of reasoning in various DLs [39–41]. Another important observation was that DLs are very closely related to modal logics [85].

Phase 3 (1995–2000) is characterized by the development of inference procedures for very expressive DLs, either based on the tableau-approach [58, 60] or on a translation into modal logics [35–38]. Highly optimized systems (FaCT, RACE, and DLP [50, 55, 78]) 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 [36, 86] and to decidable fragments of first-order logic was also studied in more detail [19, 45–47, 76], and applications in databases (like schema reasoning, query optimization, and integration of databases) were investigated [28, 29, 31].

We are now at the beginning of *Phase 4*, where industrial strength DL systems employing very expressive DLs and tableau-based algorithms are being developed, with applications like the Semantic Web or knowledge representation and integration in bio-informatics in mind.

1.3 Description Logics as Ontology Languages

As already mentioned above, high quality ontologies are crucial for many applications, and their construction, integration, and evolution greatly depends on the availability of a well-defined semantics and powerful reasoning tools. Since DLs provide for both, they should be ideal candidates for ontology languages. That much was already clear ten years ago, but at that time there was a fundamental mismatch between the expressive power and the efficiency of reasoning that DL systems provided, and the expressivity and the large knowledge bases that ontologists needed [42]. Through the basic research in DLs of the last 10–15 years that we have summarized above, this gap between the needs of ontologist and the systems that DL researchers provide has finally become narrow enough to build stable bridges.

The suitability of DLs as ontology languages has been highlighted by their role as the foundation for several web ontology languages, including OWL, an ontology language standard developed by the W3C Web-Ontology Working Group[2] (see chapter "Web Ontology Language: OWL"). OWL has a syntax

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

based on RDF Schema, but the basis for its design is the expressive DL $\mathcal{SHIQ}$ [60],[3] and the developers have tried to find a good compromise between expressiveness and the complexity of reasoning. Although reasoning in $\mathcal{SHIQ}$ is decidable, it has a rather high worst-case complexity (ExpTime). Nevertheless, highly optimized $\mathcal{SHIQ}$ reasoners such as FaCT++ [95], Racer [52] and Pellet [91] behave quite well in practice.

Let us point out some of the features of $\mathcal{SHIQ}$ that make this DL expressive enough to be used as an ontology language. Firstly, $\mathcal{SHIQ}$ provides number restrictions that are more expressive than the ones introduced above (and employed by earlier DL systems). With the *qualified number restrictions* available in $\mathcal{SHIQ}$, as well as being able to say that a person has at most two children (without mentioning the properties of these children):

$$(\leq 2\,\mathsf{hasChild}),$$

one can also specify that there is at most one son and at most one daughter:

$$(\leq 1\,\mathsf{hasChild}.\neg\mathsf{Female}) \sqcap (\leq 1\,\mathsf{hasChild}.\mathsf{Female}).$$

Secondly, $\mathcal{SHIQ}$ allows the formulation of complex terminological axioms like "humans have human parents":

$$\mathsf{Human} \sqsubseteq \exists \mathsf{hasParent}.\mathsf{Human}.$$

Thirdly, $\mathcal{SHIQ}$ also allows for *inverse roles*, *transitive roles*, and *subroles*. For example, in addition to hasChild one can also use its inverse hasParent, one can specify that hasAncestor is transitive, and that hasParent is a subrole of hasAncestor.

It has been argued in the DL and the ontology community that these features play a central role when describing properties of aggregated objects and when building ontologies [43, 83, 93]. The actual use of a DL providing these features as the underlying logical formalism of the web ontology language OWL [57] substantiates this claim [93].[4]

Finally, we would like to briefly mention three extensions to $\mathcal{SHIQ}$ that are often used in ontology languages (we will discuss them in more detail in Sect. 4).

Complex roles are often required in ontologies. For example, when describing complex physically composed structures it may be desirable to express the fact that damage to a part of the structure implies damage to the structure as a whole. This feature is particularly important in medical ontologies: it is supported in the Grail DL [81], which was specifically designed for use with medical terminology, and in another medical terminology application using

[3] To be exact, it is based on $\mathcal{SHOIN}$.

[4] The more expressive qualified number restrictions are not supported by OWL, but are featured in the proposed OWL 2 extension (see Sect. 4).

the comparatively inexpressive DL $\mathcal{ALC}$, a rather complex "work around" is performed in order to capture this kind of information [89].[5]

It is quite straightforward to extend $\mathcal{SHIQ}$ so that this kind of propagation can be expressed: simply allow for the use of complex roles in role inclusion axioms. E.g. $\mathsf{hasLocation} \circ \mathsf{partOf} \sqsubseteq \mathsf{hasLocation}$ expresses the fact that things located in part of something are also located in the thing as a whole. Although this leads to undecidability in general, syntactic restrictions can be devised that lead to a decidable logic [59].

Concrete domains [7,69] integrate DLs with concrete sets such as the real numbers, integers, or strings, and built-in predicates such as comparisons $\leq$, comparisons with constants ≤ 17, or $\mathsf{isPrefixOf}$. This supports the modelling of concrete properties of abstract objects such as the age, the weight, or the name of a person, and the comparison of these concrete properties. Unfortunately, in their unrestricted form, concrete domains can have dramatic effects on the decidability and computational complexity of the underlying DL [69].

Nominals are special concept names that are to be interpreted as singleton sets. Using a nominal Turing, we can describe all those computer scientists that have met Turing by $\mathsf{CSientist} \sqcap \exists \mathsf{hasMet}.\mathsf{Turing}$. Again, nominals can have dramatic effects on the complexity of a logic [94]. The extension of $\mathcal{SHIQ}$ with nominals is usually called $\mathcal{SHOIQ}$.

2 The Expressive Description Logic $\mathcal{SHIQ}$

In this section, we present syntax and semantics of the expressive DL $\mathcal{SHIQ}$ [60] (although, as can be seen in chapter "Web Ontology Language: OWL", the DL underlying OWL is, in some respects, slightly more expressive).

In contrast to most of the DLs considered in the literature, which concentrate on constructors for defining concepts, the DL $\mathcal{SHIQ}$ also allows for rather expressive roles. Of course, these roles can then be used in the definition of concepts.

Definition 1 (Syntax and semantics of $\mathcal{SHIQ}$-roles and concepts). *Let* $\mathbf{R}$ *be a set of* role names*, which is partitioned into a set* $\mathbf{R}_+$ *of transitive roles and a set* $\mathbf{R}_\mathsf{P}$ *of normal roles. The set of all* $\mathcal{SHIQ}$-roles *is* $\mathbf{R} \cup \{r^- \mid r \in \mathbf{R}\}$*, where* r^- *is called the* inverse *of the role* r*.*

Let $\mathbf{C}$ *be a set of* concept names*. The set of* $\mathcal{SHIQ}$-concepts *is the smallest set such that*

1. *every concept name* $A \in \mathbf{C}$ *is a* $\mathcal{SHIQ}$*-concept,*
2. *if* C *and* D *are* $\mathcal{SHIQ}$*-concepts and* r *is a* $\mathcal{SHIQ}$*-role, then* $C \sqcap D$*,* $C \sqcup D$*,* $\neg C$*,* $\forall r.C$*, and* $\exists r.C$ *are* $\mathcal{SHIQ}$*-concepts,*

[5] In this approach, so-called *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.

3. if C is a $\mathcal{SHIQ}$-concept, r is a simple[6] *$\mathcal{SHIQ}$-role, and $n \in \mathbb{N}$, then $(\leqslant n\, r.C)$ and $(\geqslant n\, r.C)$ are $\mathcal{SHIQ}$-concepts.*

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *consists of a set $\Delta^{\mathcal{I}}$, called the* domain *of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ that maps every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for all $p \in \mathbf{R}$ and $r \in \mathbf{R}_+$,*

$$\langle x, y\rangle \in p^{\mathcal{I}} \quad \text{iff} \quad \langle y, x\rangle \in (p^-)^{\mathcal{I}},$$
$$\text{if } \langle x, y\rangle \in r^{\mathcal{I}} \text{ and } \langle y, z\rangle \in r^{\mathcal{I}} \text{ then } \langle x, z\rangle \in r^{\mathcal{I}}.$$

The interpretation function $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ maps, additionally, every concept to a subset of $\Delta^{\mathcal{I}}$ such that

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \qquad \neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},$$
$$(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{There is some } y \in \Delta^{\mathcal{I}} \text{ with } \langle x, y\rangle \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\},$$
$$(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{For all } y \in \Delta^{\mathcal{I}}, \text{ if } \langle x, y\rangle \in r^{\mathcal{I}}, \text{then } y \in C^{\mathcal{I}}\},$$
$$(\leqslant n\, r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp r^{\mathcal{I}}(x, C) \leqslant n\},$$
$$(\geqslant n\, r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp r^{\mathcal{I}}(x, C) \geqslant n\},$$

where $\sharp M$ denotes the cardinality of the set M, and $r^{\mathcal{I}}(x, C) := \{y \mid \langle x, y\rangle \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$. If $x \in C^{\mathcal{I}}$, then we say that x is an instance *of C in $\mathcal{I}$, and if $\langle x, y\rangle \in r^{\mathcal{I}}$, then y is called an* r-successor *of x in $\mathcal{I}$.*

So far, we have fixed the syntax and semantics of concepts and roles. Next, we define how they can be used in a $\mathcal{SHIQ}$ TBox. Please note that authors sometimes distinguish between a role hierarchy or RBox and a TBox – we do not make this distinction here.

Definition 2 (TBox). *A* role inclusion axiom *is of the form $r \sqsubseteq s$, where r, s are $\mathcal{SHIQ}$-roles. A* general concept inclusion *(GCI) is of the form $C \sqsubseteq D$, where C, D are $\mathcal{SHIQ}$-concepts.*

A finite set of role inclusion axioms and GCIs is called a TBox.

An interpretation $\mathcal{I}$ is a model *of a TBox $\mathcal{T}$ if it satisfies all axioms in $\mathcal{T}$, i.e. $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each $C \sqsubseteq D \in \mathcal{T}$ and $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ holds for each $r \sqsubseteq s \in \mathcal{T}$.*

A concept definition is of the form $A \equiv C$, where A is a concept name; it can be seen as an abbreviation for the two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$.

In addition to describing the relevant notions of an application domain, a DL knowledge base may also contain knowledge about the properties of specific individuals (or objects) existing in this domain. This done in the assertional part of the knowledge base (ABox).

[6] We refer the interested reader to [60] for a definition of simple roles: roughly speaking, a role is simple if it is neither transitive nor has a transitive sub-role. Only simple roles are allowed in number restrictions to ensure decidability [60].

Definition 3. *Let* $\mathbf{I}$ *be a set of* individual names *disjoint from* $\mathbf{R}$ *and* $\mathbf{C}$*. For* $a, b \in \mathbf{I}$ *individual names,* C *a possibly complex* $\mathcal{SHIQ}$ *concept, and* r *a* $\mathcal{SHIQ}$ *role, an expression of the form*

- $C(a)$ *is called a* concept assertion*, and*
- $r(a, b)$ *is called a* role assertion*.*

A finite set of concept and role assertions is called an ABox*.*

An interpretation function $\cdot^{\mathcal{I}}$*, additionally, is required to map every individual name* $a \in \mathbf{I}$ *to an element* $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$*. An interpretation* $\mathcal{I}$ satisfies

- *a concept assertion* $C(a)$ *if* $a^{\mathcal{I}} \in C^{\mathcal{I}}$*, and*
- *a role assertion* $r(a, b)$ *if* $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$*.*

An interpretation that satisfies each concept assertion and each role assertion in an ABox $\mathcal{A}$ *is called a* model *of* $\mathcal{A}$*.*

Inference problems for concepts are defined w.r.t. a TBox. Inference problems for individuals additionally involve an ABox.

Definition 4. *The concept* C *is called* satisfiable with respect to the TBox $\mathcal{T}$ *iff there is a model* $\mathcal{I}$ *of* $\mathcal{T}$ *with* $C^{\mathcal{I}} \neq \emptyset$*. Such an interpretation is called a* model of C *w.r.t.* $\mathcal{T}$*. The concept* D subsumes *the concept* C *w.r.t.* $\mathcal{T}$ *(written* $C \sqsubseteq_{\mathcal{T}} D$*) if* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ *holds for all models* $\mathcal{I}$ *of* $\mathcal{T}$*. Two concepts* C, D *are* equivalent *w.r.t.* $\mathcal{T}$ *(written* $C \equiv_{\mathcal{T}} D$*) if they subsume each other.*

The ABox $\mathcal{A}$ *is called* consistent with respect to the TBox $\mathcal{T}$ *iff there exists a model of* $\mathcal{T}$ *and* $\mathcal{A}$*. The individual* a *is called an* instance of the concept C with respect to the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ *iff* $a^{\mathcal{I}} \in C^{\mathcal{I}}$ *holds for all models of* $\mathcal{I}$ *of* $\mathcal{T}$ *and* $\mathcal{A}$*.*

By definition, equivalence can be reduced to subsumption. In addition, subsumption can be reduced to satisfiability since $C \sqsubseteq_{\mathcal{T}} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{T}$. Satisfiability and the instance problem can be reduced to the consistency problem since C is satisfiable w.r.t. $\mathcal{T}$ if the ABox $\{C(a)\}$ is consistent w.r.t. $\mathcal{T}$, and a is an instance of C w.r.t. $\mathcal{T}$ and $\mathcal{A}$ if the ABox $\mathcal{A} \cup \{\neg C(a)\}$ is inconsistent w.r.t. $\mathcal{T}$.

As mentioned above, most DLs are (decidable) fragments of (first-order) predicate logic [5, 19]. Viewing role names as binary relations, concept names as unary relations, and individual names as constants, for example, the role inclusion axiom $r \sqsubseteq s^-$ translates into $\forall x \forall y.r(x, y) \Rightarrow s(y, x)$, the concept assertion $(A \sqcap B)(a)$ translates into $A(a) \wedge B(a)$, and the GCI $A \sqcap \exists r.C \sqsubseteq D \sqcup \forall s^-.E$ translates into

$$\forall x.(A(x) \wedge \exists y.r(x, y) \wedge C(y)) \Rightarrow (D(x) \vee \forall y.s(y, x) \Rightarrow E(y)).$$

This translation preserves the semantics: we can easily view DL interpretations as predicate logic interpretations, and then prove, e.g. that each model of a concept C w.r.t. a TBox $\mathcal{T}$ is a model of the translation of C conjoined with the (universally quantified) translations of $\mathcal{T}$.

The reasoning services that can decide the inference problems introduced above can be implemented using various algorithmic techniques, including tableaux-based techniques (see chapter "Tableau-Based Reasoning") and resolution-based techniques (see "Resolution-Based Reasoning for Ontologies").

3 Describing Ontologies in $\mathcal{SHIQ}$

In general, an ontology can be formalised in a DL knowledge base as follows. Firstly, we restrict the possible worlds by introducing restrictions on the allowed interpretations. For example, to express that, in our world, we want to consider humans, which are either muggles or sorcerers, we can use the GCIs

$$\mathsf{Human} \sqsubseteq \mathsf{Muggle} \sqcup \mathsf{Sorcerer} \quad \text{and} \quad \mathsf{Muggle} \sqsubseteq \neg\mathsf{Sorcerer}.$$

Next, to express that humans have exactly two parents and that all parents and children of humans are human, we can use the following GCI:

$$\mathsf{Human} \sqsubseteq \forall \mathsf{hasParent}.\mathsf{Human} \sqcap (\leqslant 2\ \mathsf{hasParent}.\top) \sqcap (\geqslant 2\ \mathsf{hasParent}.\top) \sqcap \forall \mathsf{hasParent}^{-}.\mathsf{Human},$$

where $\top$ is an abbreviation for the top concept $A \sqcup \neg A$.[7]

In addition, we consider the *transitive* role hasAncestor, and the role inclusion

$$\mathsf{hasParent} \sqsubseteq \mathsf{hasAncestor}.$$

The next GCI expresses that humans having an ancestor that is a sorcerer are themselves sorcerers:

$$\mathsf{Human} \sqcap \exists \mathsf{hasAncestor}.\mathsf{Sorcerer} \sqsubseteq \mathsf{Sorcerer}.$$

Secondly, we can define the relevant notions of our application domain using concept definitions. Recall that the concept definition $A \equiv C$ stands for the two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$. A concept name is called *defined* if it occurs on the left-hand side of a definition, and *primitive* otherwise.

We want our concept definitions to have definitional impact, i.e. the interpretation of the primitive concept and role names should uniquely determine the interpretation of the defined concept names. For this, the set of concept definitions together with the additional GCIs must satisfy three conditions:

1. There are no multiple definitions, i.e. each defined concept name must occur at most once as the left-hand side of a concept definition.

[7] When the qualifying concept is $\top$, this is equivalent to an unqualified restriction, and it will often be written as such, e.g. $(\leqslant 2\ \mathsf{hasParent})$.

2. There are no cyclic definitions, i.e. no cyclic dependencies between the defined names in the set of concept definitions.[8]
3. The defined names do not occur in any of the additional GCIs.

In contrast to concept definitions, the GCIs in $\mathcal{SHIQ}$ may well have cyclic dependencies between concept names. An example are the above GCIs describing humans.

As a simple example of a set of concept definitions satisfying the restrictions from above, we define the concepts grandparent and parent[9]:

$$\mathsf{Parent} \equiv \mathsf{Human} \sqcap \exists \mathsf{hasParent}^{-}.\top,$$
$$\mathsf{Grandparent} \equiv \exists \mathsf{hasParent}^{-}.\mathsf{Parent}.$$

The TBox consisting of the above concept definitions and GCIs, together with the fact that hasAncestor is a transitive superrole of hasParent, implies the following subsumption relationship:

$$\mathsf{Grandparent} \sqcap \mathsf{Sorcerer} \sqsubseteq \exists \mathsf{hasParent}^{-}.\exists \mathsf{hasParent}^{-}.\mathsf{Sorcerer},$$

i.e. grandparents who are sorcerers have a grandchild who is a sorcerer. Though this conclusion may sound reasonable given the assumptions, it requires quite some reasoning to obtain it. In particular, one must use the fact that hasAncestor (and thus also $\mathsf{hasAncestor}^{-}$) is transitive, that $\mathsf{hasParent}^{-}$ is the inverse of hasParent, and that we have a GCI that says that children of humans are again humans.

To sum up, a $\mathcal{SHIQ}$-TBox can, on the one hand, axiomatize the basic notions of an application domain (the primitive concepts) by GCIs, transitivity statements, and role inclusions, in the sense that these statements restrict the possible interpretations of the basic notions. On the other hand, more complex notions (the defined concepts) can be introduced by concept definitions. Given an interpretation of the basic notions, the concept definitions uniquely determine the interpretation of the defined notions.

The *taxonomy* of such a TBox is then given by the subsumption hierarchy of the defined concepts. It can be computed using a subsumption algorithm for $\mathcal{SHIQ}$ (see chapters "Tableau-Based Reasoning" and "Resolution-Based Reasoning for Ontologies"). The knowledge engineer can test whether the TBox captures her intuition by checking the satisfiability of the defined concepts (since it does not make sense to give a complex definition for the empty concept), and by checking whether their place in the taxonomy corresponds to their intuitive place. The taxonomy of our example TBox would contain, for example, the fact that Grandparent is subsumed by Parent which is, in turn, subsumed by Human – if this is not intended, then the knowledge engineer

[8] In order to give cyclic definitions definitional impact, one would need to use fixpoint semantics for them [1, 75].

[9] In addition to the role hasParent, which relates children to their parents, we use the concept Parent, which describes all humans having children.

would need to go back and modify the TBox. The expressive power of $\mathcal{SHIQ}$ together with the fact that one can "verify" the TBox in the sense mentioned above is the main reason for $\mathcal{SHIQ}$ being well-suited as an ontology language [43, 83, 93].

In case we have, in addition to our TBox $\mathcal{T}$, also an ABox $\mathcal{A}$, we can first ask a DL reasoner to check whether $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ to make sure that our assertions in $\mathcal{A}$ conform with the axioms expressed in $\mathcal{T}$. Consider the following ABox:

$$\mathcal{A} = \{\, \mathsf{Human}(\mathsf{Harry}),\ \mathsf{Sorcerer}(\mathsf{Bob}) \\ \mathsf{hasParent}(\mathsf{Harry}, \mathsf{Bob})\},$$

and let $\mathcal{T}$ consist of all axioms in this section. We can first use a DL reasoner to prove that $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$. Next, we can *query* $\mathcal{A}$ through $\mathcal{T}$. For example, we can ask a DL reasoner to retrieve all instances of Human w.r.t. $\mathcal{T}$ and $\mathcal{A}$. This would result in Harry and Bob being returned: for the former, this information is explicit in $\mathcal{A}$, for the latter, this is implied by the GCI which states that parents of humans are humans. Similarly, both Harry and Bob are instances of Sorcerer w.r.t. $\mathcal{T}$ and $\mathcal{A}$: for Harry, this is a consequence of the GCI which states that offsprings of sorcerers are sorcerers. As a final example, let us point out that our ABox contains no instance of $\forall$hasParent.Sorcerer: even though all *explicitly known* parents of Harry are sorcerers, Harry could have other parents (and indeed must have another parent) who may or may not be a sorcerer – this feature of DL semantics is known as the "open world assumption" [5].

4 Extensions and Variants of $\mathcal{SHIQ}$

The ontology language OWL extends $\mathcal{SHIQ}$ with nominals and concrete datatypes; see chapter "Web Ontology Language: OWL." In this section, we discuss the consequences of these extensions on the reasoning problems in $\mathcal{SHIQ}$.

Concrete datatypes, as available in OWL, are a very restricted form of concrete domains [7]. For example, using the concrete domain of all nonnegative integers equipped with the $<$ predicate, a (functional) role age relating (abstract) individuals to their (concrete) age, and a (functional) subrole father of hasParent, the following axiom states that children are younger than their fathers:

$$\mathsf{Animal} \sqsubseteq (\mathsf{age} < (\mathsf{father} \circ \mathsf{age})).$$

Extending expressive DLs with concrete domains may easily lead to undecidability [8, 68]. In OWL, however, no datatype predicates are supported – only XML schema datatypes (such as integer and string) and enumerations (such as $\{1, 2, 5, 7\}$) can be used in descriptions. These restrictions are enough to