13

DESCRIPTION LOGIC

Franz Baader and Carsten Lutz

1 INTRODUCTION

Description logics (DLs) [12] are a family of knowledge representation languages which can be used to represent the terminological 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. For example, the concept of "a man that is married to a doctor, and has only happy children" can be expressed using the concept description

$$\mathsf{Man} \sqcap \exists \mathsf{married}.\mathsf{Doctor} \sqcap \forall \mathsf{child}.\mathsf{Happy}.$$

On the other hand, DLs differ from their predecessors in that they are equipped with a formal, *logic*-based semantics, which can, e.g., be given by a translation into first-order predicate logic. For example, the above concept description can be translated into the following first-order formula (with one free variable x):

$$\mathsf{Man}(x) \wedge \exists y(\mathsf{married}(x, y) \wedge \mathsf{Doctor}(y)) \wedge \forall y(\mathsf{child}(x, y) \rightarrow \mathsf{Happy}(y)).$$

The motivation for introducing the early predecessors of DLs, such as semantic networks and frames [133, 125], actually was to develop means of representation that are closer to the way humans represent knowledge than a representation in formal logics, like first-order predicate logic. Minsky [125] even combined his introduction of the frame idea with a general rejection of logic as an appropriate formalism for representing knowledge. However, once people tried to equip these "formalisms" with a formal semantics, it turned out that they can be seen as syntactic variants of (subclasses of) first-order predicate logic [83, 144].

The immediate precursors of DLs, Brachman's structured inheritance networks [42], were an attempt to define a formalism that allows for a structured representation of knowledge in the spirit of semantics networks and frames, but nevertheless is equipped with a formal semantics. The original description logics used in systems that implemented these ideas in the 1980ies [45, 132, 124, 123] turned out to correspond to rather inexpressive and somewhat unusual subclasses of first-order predicate logic. On the one hand, none of them was propositionally closed since they did not allow for disjunction or negation. On the other hand, they were equipped with certain other complex constructors (like number restrictions and role-value-maps), which, though expressible in first-order predicate logic, are not considered as atomic constructors there. For example, the number restriction $(\geqslant 5\,\mathsf{child})$ describes people having at least five children, and the role-value-map $\mathsf{child} \circ \mathsf{friend} \subseteq \mathsf{know}$ describes people that know all their children's friends.

The main inference problem to be solved in description logics is the subsumption problem, i.e., deciding whether one concept is a subconcept of another one. The early DL systems cited above employed so-called structural subsumption algorithms, which first normalise the concept descriptions, and then recursively compare the syntactic structure of the normalised descriptions. These algorithms are usually very efficient (polynomial), but they have the disadvantage that they are complete only for rather inexpressive DLs, i.e., for more expressive DLs they cannot detect all the existing subsumption relationships. To overcome this problem, Schmidt-Schauß and Smolka [143] made DLs into "real" logics by introducing negation. Their main motivation for this was that they wanted to reduce the subsumption problem to the satisfiability problem. They introduced a basic propositionally closed DL, which they called $\mathcal{ALC}$, developed a tableau-like algorithm for satisfiability in $\mathcal{ALC}$, and showed that the subsumption and satisfiability problem in $\mathcal{ALC}$ are PSPACE-complete.

A reader of the Handbook of Modal Logic who followed us so far may rightfully ask: *And what has all this to do with Modal Logic?* The answer was given by Schild, who noticed that $\mathcal{ALC}$ is just a syntactic variant of multi-modal K, i.e., the basic modal logic of Kripke frames with several accessibility relations (and thus several pairs of box- and diamond operators). In fact, the translations of $\mathcal{ALC}$ and of K into first-order predicate logic yield exactly the same class of first-order formulae. This connection between DLs and modal logic was used by Schild and others (see, e.g., [139, 140, 54, 55]) to transfer decidability and complexity results from modal logic to DLs, but also to extend these results to logics with other DL constructors. At the same time, tableau-based algorithms were developed for more and more expressive DLs (see [30] for an overview), and highly-optimized implementations of these algorithms [92] turned out to behave quite well on artificial benchmarks from modal logic [131] and also in practice [78].

Though there is a very close connection between DLs and modal logics (MLs), the underlying intuition as well as the intended applications differ significantly. As a consequence, the focus of research in DL and in modal logic also differs. While mentioning the similarities, this chapter will focus on topics that are specific for DLs.

Section 2 formally introduces syntax and semantics of the basic DL $\mathcal{ALC}$, and shows its relationship to multi-modal K. It then introduces additional DL constructors, and describes their ML counterparts. In addition to these constructors, DLs provide their users with a terminological formalism, which (in its simplest form) allows to introduce names for complex concepts, and an assertional formalism, which allows to state facts about specific individuals/objects. Though these components are usually not available in ML, there are some connections to things known in ML (such as nominals, the universal modality, fixpoint operators, etc.).

In *Section 3*, we introduce the standard inference problems in description logics, show how they can be reduced to each other, and how they relate to inference problems in ML. The standard way of solving these problems in propositionally closed DLs is using tableau-based algorithms. Since these algorithms are treated in other chapters, we only give some references to the relevant chapters.

Section 4 considers DLs that are not propositionally closed, and where consequently subsumption cannot be reduced to satisfiability. We review the known complexity results for such DLs, and then describe (complete) structural subsumption algorithms for some of them. In addition, we mention bi-simulation characterizations of the corresponding ML fragments.

Section 5 is concerned with so-called non-standard inferences in DLs, like computing the least common subsumer and the most specific concept, and rewriting, unification, and matching of concepts. These inferences have been introduced with the goal of supporting the user when building and maintaining large DL knowledge bases. With the exception of unification, none of them have been investigated in ML.

Finally, *Section 6* introduces means of expressiveness that do not have immediate ML counterparts.

2 BASIC DEFINITIONS AND CONNECTION TO MODAL LOGIC

In this section, we introduce the basic components of description logics: concept languages, terminological formalisms, and assertional formalisms.

2.1 Concept Languages

We first define the basic propositionally closed concept language $\mathcal{ALC}$ introduced by Schmidt-Schauß and Smolka [143], and then describe a number of natural extensions that are important for many applications and offered by modern DL reasoners. Assume that a countably infinite supply of *concept names*, usually denoted A and B, and of *role names*, usually denoted r and s, are available. *Concept descriptions* in $\mathcal{ALC}$ are formed

Name	Syntax	Semantics
top concept	$\top$	$\Delta^{\mathcal{I}}$
bottom concept	$\bot$	$\emptyset$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
value restriction	$\forall r.C$	$\{d \in \Delta^{\mathcal{I}} \mid \forall e.(d,e) \in r^{\mathcal{I}} \rightarrow e \in C^{\mathcal{I}}\}$
existential restriction	$\exists r.C$	$\{d \in \Delta^{\mathcal{I}} \mid \exists e.(d,e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}$
transitive role	r	$r^{\mathcal{I}}$ transitive
inverse role	r^-	$\{(d,e) \mid (e,d) \in r^{\mathcal{I}}\}$
nominal	I	$I^{\mathcal{I}}$ singleton
qualifying number restrictions	$\leqslant n\ r\ C$ $\geqslant n\ r\ C$	$\{d \in \Delta^{\mathcal{I}} \mid \#\{(d,e) \in r^{\mathcal{I}} \mid d \in C^{\mathcal{I}}\} \leq n\}$ $\{d \in \Delta^{\mathcal{I}} \mid \#\{(d,e) \in r^{\mathcal{I}} \mid e \in C^{\mathcal{I}}\} \geq n\}$
number restrictions	$\leqslant n\ r$ $\geqslant n\ r$	$\{d \in \Delta^{\mathcal{I}} \mid \#\{(d,e) \in r^{\mathcal{I}}\} \leq n\}$ $\{d \in \Delta^{\mathcal{I}} \mid \#\{(d,e) \in r^{\mathcal{I}}\} \geq n\}$

Figure 1.1. Semantics of concept and role constructors

according to the following syntax rule:

$$C, D \longrightarrow A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall r.C \mid \exists r.C$$

where A ranges over concept names and r ranges over role names. In examples, we will usually use uppercase names for concept names and lowercase names for role names, thus obtaining $\mathcal{ALC}$ concept descriptions such as the one given in the introduction:

$$\mathsf{Man} \sqcap \exists \mathsf{married}.\mathsf{Doctor} \sqcap \forall \mathsf{child}.\mathsf{Happy}.$$

The semantics of $\mathcal{ALC}$ is based on *interpretations*, i.e., pairs $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set (the *domain* of $\mathcal{I}$), and $\cdot^{\mathcal{I}}$ is the *interpretation function*, assigning to each concept name A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to each role name r a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is inductively extended to concept descriptions as shown in (the upper part of) Figure 1.1, which also lists the names that we use for $\mathcal{ALC}$ constructors. An interpretation $\mathcal{I}$ is a *model* of a concept description C if $C^{\mathcal{I}} \neq \emptyset$. In the following, we will sometimes call concept languages "description logics", ignoring further ingredients to DLs such as the terminological formalism.

As first observed by Schild [138], $\mathcal{ALC}$ is a notational variant of the multi-modal logic K. Syntactically, concept names can simply be viewed as propositional variables and role names can be viewed as names for accessibility relations. Then, interpretations of $\mathcal{ALC}$ are obviously just Kripke structures with $\Delta^{\mathcal{I}}$ the set of worlds and $\cdot^{\mathcal{I}}$ providing both the accessibility relations and the valuation of the propositional variables. With this reading, the value restriction $\forall r.C$ becomes a box operator $\Box_r C$ referring to the accessibility relation denoted by r, and $\exists r.C$ becomes a diamond operator $\Diamond_r C$. This connection is also witnessed by the usual translation of $\mathcal{ALC}$ to first-order predicate logic [37, 38], which is identical to the standard translation for modal logic presented in Chapter 1.

The concept language $\mathcal{ALC}$ is only one member of a large family of concept languages. These languages can be obtained from $\mathcal{ALC}$ by disallowing certain constructors (thus obtaining the *sub-Boolean* description logics discussed in Section 4) and/or adding various combinations of additional constructors. Such additional constructors can be concept constructors, or they can be role constructors allowing to construct compound *role descriptions* to be used in place of role names. We now discuss several additional constructors that are related to expressive means common in modal logic. Constructors that have been considered in the context of description logics, but lack a modal counterpart will be discussed in more depth in Section 6.

When the connection between DLs and modal logic was discovered, one of it's first uses was to transfer results from propositional dynamic logic (PDL) to description logics [138, 139, 54]. The description logic counterpart of PDL is called $\mathcal{ALC}_{reg}$, which stands for "$\mathcal{ALC}$ with *reg*ular expressions on roles" [3, 138]. $\mathcal{ALC}_{reg}$ extends $\mathcal{ALC}$ by allowing compound role descriptions inside value restrictions and existential restrictions. Such role descriptions are built using the binary constructors for union ("$\sqcup$") and composition (";") and a unary constructor for reflexive-transitive closure ("$\cdot^*$"). The semantics is given in the straightforward way by interpreting the constructors using the corresponding relational operations. For example, the additional constructors could be used in the concept description

$$\mathsf{Man} \sqcap \exists \mathsf{child}.\mathsf{Human} \sqcap \forall(\mathsf{child};\mathsf{child}^*).\mathsf{Happy}$$

where $(\mathsf{child};\mathsf{child}^*)$ describes the transitive closure of the role child, i.e., the $\mathsf{descendant}$ relation. The work on $\mathcal{ALC}_{reg}$ has led to several variants and extensions whose expressive power goes beyond that of PDL [54, 56, 57]. However, many of today's most used concept languages do not include the role constructors of $\mathcal{ALC}_{reg}$. The main reason is that applications demand an implementation of description logic reasoning, and the presence of the reflexive-transitive closure constructor makes obtaining efficient implementations much harder.

Another important family of description logics is obtained by considering fragments of the concept language $\mathcal{SHOIQ}$ [94, 98, 95], which extends $\mathcal{ALC}$ with several expressive means that are discussed in detail below.[1] The importance of $\mathcal{SHOIQ}$ stems from the fact that it and its fragments are used in two of the most influential application areas of description logics: reasoning about conceptual database models [52] and reasoning in the semantic web [19]. In the latter application, the fragment $\mathcal{SHOIN}$ roughly corresponds to the ontology language OWL-DL [93], which was recommended by the W3C as the standard web ontology language. The fragment $\mathcal{SHIQ}$ is the concept language supported by modern description logic systems such as FaCT and RACER [91, 79]. A tableau algorithm for full $\mathcal{SHOIQ}$ was introduced in [97], and optimized implementations of this algorithm are under development.

Compared to $\mathcal{ALC}$, the additional expressive means provided by $\mathcal{SHOIQ}$ are transitive roles, role hierarchies, inverse roles, qualifying number restrictions, and nominals. With the exception of role hierarchies, the formal semantics of these extensions can be found in the lower part of Figure 1.1. Below, we discuss each means of expressiveness in more detail. Before that, a remark on the naming scheme used to describe fragments of $\mathcal{SHOIQ}$ is in order. To avoid long sequences of letters, the abbreviation $\mathcal{S}$ was introduced for $\mathcal{ALC}$ with transitive roles. The additional presence of role hierarchies is indicated by

[1] The naming of description logics is historically grown, and there are several naming schema in use; see the Appendix of [12].

Symbol	Syntax	$\mathcal{SHIQ}$	$\mathcal{SHOIQ}$	$\mathcal{SHIN}$	$\mathcal{SHOIN}$
$\mathcal{H}$	$r \sqsubseteq s$	x	x	x	x
$\mathcal{I}$	r^-	x	x	x	x
$\mathcal{N}$	$(\leqslant n\, r), (\geqslant n\, r)$	x	x	x	x
$\mathcal{Q}$	$(\leqslant n\ r\ C), (\geqslant n\ r\ C)$	x	x		
$\mathcal{O}$	I		x		x

Figure 1.2. Some members of the $\mathcal{S}$ family of DLs.

the letter $\mathcal{H}$, of inverse roles by $\mathcal{I}$, of (qualifying) number restrictions by $\mathcal{N}$ ($\mathcal{Q}$), and of nominals by $\mathcal{O}$ (see Figure 1.2).

Transitive roles. $\mathcal{ALC}$ can be extended with transitive roles by adding a new sort of role names whose interpretation is required to be transitive [135]. The resulting description logic is a notational variant of the fusion of multi-modal K and multi-modal $\mathsf{K4}$. One of the most important uses of transitive roles is for the representation of knowledge about parts and wholes by means of a transitive role $\mathsf{part\text{-}of}$ [136].

Inverse roles extend $\mathcal{ALC}$ with a unary role constructor $\cdot^-$. Roles of the form r^- correspond to "backwards modalities" as known from temporal logic and converse PDL [154]. They allow, for example, to define the converse parent of the relation child, and the converse $\mathsf{has\text{-}part}$ of $\mathsf{part\text{-}of}$.

Role hierarchies are not a part of the concept language, but rather "external" to it [91]. Formally, a role hierarchy is a finite set of inclusion statement $r \sqsubseteq s$ with r and s role descriptions. Intuitively, the presence of a role hierarchy puts constraints on the class of accepted interpretations: if $r \sqsubseteq s$ is in the hierarchy, then we only accept interpretations in which $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$. Thus, role hierarchies are much closer in spirit to TBoxes (see below) than to concept or role constructors. The connection between role hierarchies and modal logics will be discussed in a more general context in Section 6.2.

Nominals are an additional sort of concept names that are required to be interpreted as singleton sets. The name has been adopted from modal logics, where nominals appear e.g. in the context of hybrid logic, c.f. Chapter 14 and [1, 74]. There are several natural concepts, such as Pope, that require nominals for an adequate modelling. In description logics, nominals sometimes occur in the form of two concept constructors called "one of" and "fills", see [44] for more details.

Qualifying number restrictions. Corresponding to graded modalities in modal logic [70, 71, 153], qualifying number restrictions allow to put *counting* constraints on the number of domain elements that are related via a certain role and belong to a certain concept [88]. This constructor allows, e.g., the formulation of concepts such as $\mathsf{Father} \sqcap (\leqslant 1\ \mathsf{child}\ \mathsf{Female})$ describing fathers that have at most one daughter (but arbitrarily many sons). Number restrictions also appear in a non-qualifying variant $\leqslant n\ r$ and $\geqslant n\ r$, in which the third argument implicitly is the top concept. They are a very important means of expressivity that appeared already in early description logic systems such as KL-ONE [45]. In the case of $\mathcal{SHOIQ}$ and its fragments, number restrictions are usually restricted to *simple* roles, i.e. roles having no transitive subroles according to the role hierarchy. If this syntactic restriction is not adopted, reasoning in $\mathcal{SHOIQ}$ is undecidable [98].

For the sake of brevity, the list of concept and role constructors given above is not exhaustive. For example, $\mathcal{ALC}$ has also been extended with Boolean role constructors,

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{Woman} & \equiv & \mathsf{Person} \sqcap \mathsf{Female} & \mathsf{Man} & \equiv & \mathsf{Person} \sqcap \neg\mathsf{Woman} \\
\mathsf{Mother} & \equiv & \mathsf{Woman} \sqcap \exists\mathsf{child}.\mathsf{Person} & \mathsf{Father} & \equiv & \mathsf{Man} \sqcap \exists\mathsf{child}.\mathsf{Person} \\
\mathsf{Parent} & \equiv & \mathsf{Mother} \sqcup \mathsf{Father} & & &
\end{array}$$

Figure 1.3. An example TBox formulated in $\mathcal{ALC}$.

which corresponds to going from multi-modal K to Boolean modal logic [101, 122, 121].

2.2 Terminological Formalisms

The concept language is only one part of description logics. To capture the terminological knowledge of application domains in a structured way, it is not sufficient to formulate single concept descriptions. Additionally, we must be able to organize and interrelate multiple concept descriptions in a suitable way. This is achieved through the terminological formalism. Just like concept languages, terminological formalisms come in several flavors. One of the most fundamental variants is the following: a *TBox (terminological box)* $\mathcal{T}$ is a finite set of *concept definitions*

$$A \equiv C$$

with A a concept name and C a concept description, such that no concept name appears on the left-hand side of two different concept definitions in $\mathcal{T}$. An example of a TBox formulated in $\mathcal{ALC}$ is displayed in Figure 1.3. A concept name is called a *defined concept* if it appears on the left-hand side of a concept definition, and a *primitive concept* otherwise.

When defining the semantics, we face the difficulty of treating terminological cycles which may occur in the kind of TBoxes considered here. We say that a concept name A *directly uses* a concept name B w.r.t. a TBox $\mathcal{T}$ if there is a concept definition $A \equiv C \in \mathcal{T}$ with B occurring C. Let *uses* be the transitive closure of directly uses. Then a TBox $\mathcal{T}$ contains a *terminological cycle* if there is a concept name that uses itself w.r.t. $\mathcal{T}$; otherwise $\mathcal{T}$ is called *acyclic*. For example, the TBox displayed in Figure 1.3 is acyclic, whereas the following concept definition induces a terminological cycle (Adam and Eve are nominals):

$$\mathsf{Human} \equiv \mathsf{Adam} \sqcup \mathsf{Eve} \sqcup \exists\mathsf{parent}.\mathsf{Human}.$$

In the following, we sometimes call the general form of TBoxes introduced above *cyclic TBoxes* to distinguish them from acyclic ones. However, this does not imply that the TBoxes in question *necessarily* contains a terminological cycle.

For acyclic TBoxes, the natural semantics is *descriptive semantics*: an interpretation $\mathcal{I}$ *satisfies* a concept definition $A \equiv C$ if $A^{\mathcal{I}} = C^{\mathcal{I}}$, and $\mathcal{I}$ is a *model* of the TBox $\mathcal{T}$ if it satisfies all concept definitions in $\mathcal{T}$. Intuitively, acyclic TBoxes merely state that defined concepts are abbreviations for certain compound concept descriptions. These compound concepts can be made explicit by *expanding* the acyclic TBox $\mathcal{T}$: exhaustively replace all concept names A on the left-hand side of concept definitions $A \equiv C$ by their defining concept descriptions C. After this expansion, the compound concept abbreviated by a defined concept can simply be read off from the corresponding concept definition. For example, the defined concept Father in Figure 1.3 abbreviates the compound concept

$$\mathsf{Person} \sqcap \neg(\mathsf{Person} \sqcap \mathsf{Female}) \sqcap \exists \mathsf{child}.\mathsf{Person}.$$

A *primitive interpretation* for a TBox $\mathcal{T}$ is an interpretation that interprets only the primitive concept names and role names, but not the defined concepts. A (full) interpretation $\mathcal{I}$ is called an *extension* of a primitive interpretation $\mathcal{J}$ if it agrees with $\mathcal{J}$ on the domain and the interpretation of the primitive concepts and role names. We say that $\mathcal{T}$ is *definitorial* if every primitive interpretation has exactly one extension that is a model of $\mathcal{T}$. Since we can expand them, acyclic TBoxes are clearly definitorial: if $\mathcal{T}$ is an acyclic TBox and $\mathcal{T}' = \{A_1 \equiv C_1, \ldots, A_k \equiv C_k\}$ has been obtained from $\mathcal{T}$ by expansion, then the unique extension of a primitive interpretation $\mathcal{J}$ that is a model of $\mathcal{T}$ is obtained by setting $A_i^{\mathcal{I}} := C_i^{\mathcal{J}}$ for $1 \leq i \leq k$.

If we do not require TBoxes to be acyclic, then TBoxes are no longer definitorial under descriptive semantics. For example, the TBox

$$\mathcal{T} \equiv \{\mathsf{Human} \equiv \forall \mathsf{parent}.\mathsf{Human}\}$$

has no primitive concept, and the primitive interpretation $\mathcal{J}$ with $\Delta^{\mathcal{J}} = \{d\}$ and $\mathsf{parent}^{\mathcal{J}} = \{(d,d)\}$ can be extended to two different models of $\mathcal{T}$. Thus, the above TBox does not provide an unequivocal definition of Human. To obtain definitorial TBoxes in the presence of terminological cycles, two steps are necessary [129]: first, descriptive semantics is changed to a (least/greatest) fixpoint semantics; and second, the syntax of TBoxes is restricted to ensure that least and greatest fixpoints indeed exist. To illustrate why fixpoints are a natural choice for defining TBox semantics, we note that they can be used to characterize models of a TBox in a straightforward way. Let $\mathcal{T}$ be a TBox and $\mathcal{J}$ a primitive interpretation for $\mathcal{T}$. We write $\mathcal{T}(A)$ to denote the concept description C if $A \equiv C \in \mathcal{T}$. With $\mathit{Ext}_{\mathcal{J}}$, we denote the set of all extensions of $\mathcal{J}$. Let $\mathcal{T}_{\mathcal{J}} : \mathit{Ext}_{\mathcal{J}} \to \mathit{Ext}_{\mathcal{J}}$ be the mapping that maps the extension $\mathcal{I}$ of $\mathcal{J}$ to the extension $\mathcal{T}_{\mathcal{J}}(\mathcal{I})$ of $\mathcal{J}$ defined by setting $A^{\mathcal{T}_{\mathcal{J}}(\mathcal{I})} := (\mathcal{T}(A))^{\mathcal{I}}$ for each defined concept A. It is trivial to verify that an interpretation $\mathcal{I}$ is a model of $\mathcal{T}$ if and only if $\mathcal{I}$ is a fixpoint of $\mathcal{T}_{\mathcal{J}}$ with $\mathcal{J}$ the restriction of $\mathcal{I}$ to a primitive interpretation.

To make the TBox formalism definitorial in the presence of terminological cycles, we restrict the set of fixpoints of $\mathcal{T}_{\mathcal{J}}$ that are intended as models. Let $\mathcal{I}$ be a model of $\mathcal{T}$ and $\mathcal{J}$ the restriction of $\mathcal{I}$ to a primitive interpretation. Then $\mathcal{I}$ is a *least fixpoint model* (*greatest fixpoint model*) of $\mathcal{T}$ if $A^{\mathcal{I}} \subseteq A^{\mathcal{I}'}$ ($A^{\mathcal{I}} \supseteq A^{\mathcal{I}'}$) for every defined concept A and every fixpoint $\mathcal{I}'$ of $\mathcal{T}_{\mathcal{J}}$. We obtain the *least fixpoint semantics* (*greatest fixpoint semantics*) by admitting only the least fixpoint models (greatest fixpoint models) of $\mathcal{T}$ as intended models. However, the obtained semantics is still not definitorial, at least not for all TBoxes: let $\mathcal{T} = \{A \equiv \forall r.\neg A\}$ and $\mathcal{J}$ the primitive interpretation with $\Delta^{\mathcal{J}} = \{d\}$ and $r^{\mathcal{J}} = \{(d,d)\}$. Then $\mathcal{J}$ has no extension that is a model of $\mathcal{T}$. The usual way to get around such a problem, as e.g. used in the modal μ-calculus [104], is to adopt a syntactic monotonicity restriction. In the setting of cyclic TBoxes, this restriction can be formulated as follows: a TBox $\mathcal{T}$ is called *monotone* if, on the right-hand side of concept definitions in $\mathcal{T}$, defined concepts appear only under an even number of negations. It is easy to show that, according to least or greatest fixpoint semantics every monotone TBox is definitorial: every primitive interpretation can be *uniquely* extended to a least or greatest fixpoint model of the TBox.

Whether least fixpoint semantics or greatest fixpoint semantics is preferable depends on the concept definition at hand: for the concept definition $\mathsf{Human} \equiv \mathsf{Adam} \sqcup \mathsf{Eve} \sqcup \exists \mathsf{parent}.\mathsf{Human}$ from above, we should use the least fixpoint semantics to avoid that individuals on cyclic or infinite parent-paths have to be Humans. In other cases, greatest fixpoint semantics can be more appropriate: say we want to define top researchers (in a somewhat incestuous way) as researchers who are renowned and collaborate only with top researchers:

$$\mathsf{TopResearcher} \equiv \mathsf{Researcher} \sqcap \mathsf{Renowned} \sqcap \forall \mathsf{collaborates\text{-}with}.\mathsf{TopResearcher}.$$

Then least fixpoint semantics is not convincing since two renowned researchers who collaborate mutually (but not with anybody else) will not be classified as top researchers. In contrast, greatest fixpoint semantics yields the intended models. These two examples illustrate that the most flexible solution is to use a mixed semantics: least fixpoints for some defined concepts, and greatest fixpoints for others [140]. Note that, in the case of an acyclic TBox, least fixpoint semantics, greatest fixpoint semantics, and descriptive semantics coincide in the sense that they admit exactly the same models.

It is also possible to use descriptive semantics for cyclic TBoxes. As discussed above, this implies that TBoxes will no longer be definitorial. While this is inappropriate if the goal is to *define* concepts, it poses no problem if we view TBoxes simply as formulating *constraints* on the intended models. This view of TBoxes, which is rather natural in a number of applications, leads to the idea of *general concept inclusion axioms (GCIs).* A GCI is an expression of the form

$$C \sqsubseteq D,$$

where both C and D are (possibly compound) concept descriptions. An interpretation $\mathcal{I}$ *satisfies* the GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. When working with GCIs as constraints on models, no syntactic restrictions such as unique left-hand sides, acyclicity, or monotonicity needs to be adopted. For example, we could use a GCI to state that all persons having an uncle who is a father also have a cousin:[2]

$$\mathsf{Person} \sqcap \exists \mathsf{uncle}.\mathsf{Father} \sqsubseteq \exists \mathsf{cousin}.\mathsf{Person}$$

Since the concept definition $A \equiv C$ can be rewritten as the pair of GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$, GCIs strictly generalize acyclic TBoxes as well as cyclic TBoxes with descriptive semantics. It should be noted that GCIs are the terminological formalism that is usually supported by modern description logic systems.

We now discuss the relation between terminological formalisms and modal logic. In the case of descriptive semantics, there is a close relationship to the universal modality: let $\mathcal{T}$ be a set of GCIs and U the universal role, i.e. $U^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for all interpretations $\mathcal{I}$. Then we can translate $\mathcal{T}$ into a concept $C_{\mathcal{T}}$ by setting

$$C_{\mathcal{T}} := \forall U. \bigsqcap_{D \sqsubseteq E \in \mathcal{T}} \neg D \sqcup E.$$

Then we have the following: if an interpretation $\mathcal{I}$ is a model of $\mathcal{T}$, then $C_{\mathcal{T}}^{\mathcal{I}} := \Delta^{\mathcal{I}}$; and if $C_{\mathcal{T}}^{\mathcal{I}} \neq \emptyset$, then $\mathcal{I}$ is a model of $\mathcal{T}$. We will see in Section 3.1 that this translation can sometimes be used to reduce reasoning with TBoxes to reasoning without TBoxes.

[2] This could be modelled in an even better way using role value maps, c.f. Section 6.

$\mathsf{Logician(DAVID)}$	$\mathsf{supervisor(DONALD, VAUGHAN)}$
$\mathsf{supervisor(VAUGHAN, DAVID)}$	$\mathsf{(Man \sqcap \exists child.Woman)(DONALD)}$

Figure 1.4. An example ABox formulated in $\mathcal{ALC}$.

In a weaker sense, we can also do the converse translation, i.e. simulate the universal modality using GCIs—see Section 2.2.1 of [112] for more details.

In the case of fixpoint semantics, Schild [140] observed that there is a direct correspondence between TBoxes and (an alternation-free fragment of) Vardi and Wolper's version of the propositional μ-calculus [155]. In contrast to the standard μ-calculus as proposed by Kozen [104], this variant provides for multiple fixpoints that correspond to constructing fixpoints for all defined concepts of a TBox simultaneously.

Finally, there is an intimate connection between our notion of definitorial TBoxes and the Beth definability property as known from modal logic [72]. Roughly, a description logic has the Beth definability property if and only if every TBox that is definitorial under descriptive semantics is equivalent to an acyclic TBox (see also [29]).

2.3 Assertional Formalisms

Apart from the concept language and the terminological formalism, there is one more important ingredient to description logics. This is the *assertional formalism*, which allows to describe (a snapshot of) the world by means of individuals populating the world, conceptual memberships of individuals, and roles relating individuals. The combination of a TBox and an ABox is commonly called a *knowledge base*. Assume that a countably infinite supply of individual names, usually denoted by a, b, c, is available. An *ABox (assertional box)* is a finite set of assertions of the form

$$\begin{array}{rl} C(a) & \text{(concept assertion)} \\ r(a,b) & \text{(role assertion)} \end{array}$$

where a and b are individual names, C is a concept description, and r is a role description. An example of an ABox is given in Figure 1.4. We us all-uppercase words to denote concrete individual names.

To assign a semantics to ABoxes, we have to extend interpretations to individual names: interpretations $\mathcal{I}$ are now required to map, additionally, every individual name a to a domain element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. Usually, the *unique name assumption (UNA)* is adopted, which requires that different individual names are mapped to distinct domain elements, i.e., $a \neq b$ implies $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. The interpretation $\mathcal{I}$ *satisfies* the concept assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and it *satisfies* the role assertion $r(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. An interpretation is a *model* of an ABox $\mathcal{A}$ if it satisfies all assertions in $\mathcal{A}$. Often, we are interested in models of an ABox $\mathcal{A}$ *w.r.t. a TBox* $\mathcal{T}$, i.e. common models of $\mathcal{A}$ and $\mathcal{T}$.

There is an obvious connection between ABoxes and nominals which provides a link between ABoxes and modal logic: if a concept language providing for nominals, conjunction, and existential restrictions is used, then we can simulate an ABox $\mathcal{A}$ using the concept description

$$C_{\mathcal{A}} := \bigsqcap_{D(a)\in\mathcal{A}} \exists u.(a \sqcap D) \sqcap \bigsqcap_{r(a,b)\in\mathcal{A}} \exists u.(a \sqcap \exists r.b)$$

where u is a role name not used in $\mathcal{A}$, and we assume that, for each individual name, there exists a nominal of the same name.[3] This is a simulation in the sense that every model for $C_{\mathcal{A}}$ is a model for $\mathcal{A}$, and every model $\mathcal{I}$ for $\mathcal{A}$ can be extended to a model for $C_{\mathcal{A}}$ by setting $u^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. However, nominals are strictly more expressive than ABoxes. For example, this is reflected by the complexity of reasoning in $\mathcal{ALCI}$, the extension of $\mathcal{ALC}$ with inverse roles: while reasoning in $\mathcal{ALCI}$ with ABoxes but without TBoxes is PSpace-complete, reasoning in $\mathcal{ALCI}$ extended with nominals is ExpTime-complete.[4] The latter is due to the possibility of defining "spy-points" in $\mathcal{ALCI}$ with nominals (see Chapter 14), which is not possible using ABoxes.

3 STANDARD DESCRIPTION LOGIC INFERENCES

Reasoning has always been a major emphasis of description logic research. The main purpose of reasoning in DLs is to explicate knowledge that is contained only implicitly in a given concept description, TBox, or ABox. This inferencing capability can be used by applications to infer new knowledge when needed, and it helps knowledge engineers to construct and structure complex knowledge bases. In this section, we introduce the inference problems for description logics that have direct counterparts in modal logic. Because these inference problems have played an important rôle since the very beginnings of description logic, they are often referred to as "standard inference problems"—in contrast to the more recent "non-standard inference problems" that are discussed in Section 5. We also give a brief survey of the most important results and techniques concerning the decidability and computational complexity of the standard inference problems. In doing so, we concentrate on description logics that have close counterparts in modal logics and defer the treatment of logics that are less common from the modal logic perspective to Section 6. Our discussion of results and techniques will be brief as these or very similar issues are covered in more detail in other chapters of this handbook.

3.1 Terminological Reasoning

The inference problems introduced here operate on concept descriptions and TBoxes, without reference to ABoxes. The basic such inference problems are the following:

Satisfiability. A concept description C is *satisfiable* with respect to a TBox $\mathcal{T}$ if there exists a common model of C and $\mathcal{T}$.

Subsumption. A concept description C is *subsumed* by a concept description D with respect to a TBox $\mathcal{T}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$ (written $C \sqsubseteq_{\mathcal{T}} D$).

In both cases, we simply drop the reference "with respect to $\mathcal{T}$" (and the index $\mathcal{T}$ from $C \sqsubseteq_{\mathcal{T}} D$) if we are interested in reasoning w.r.t. the empty TBox. In this case, we also talk about reasoning *with concept descriptions.* Intuitively, satisfiability is important to

[3] The additional role can be omitted if the "$@_a$" operator of hybrid logic is used, c.f. Chapter 14.
[4] With "reasoning" we refer to ABox consistency, c.f. Section 3.2.

(automatically) verify whether a concept description makes sense from a logical perspective, i.e., whether it is contradictory in itself or to a given TBox. Satisfiability also plays an important rôle because many other inference problems can be reduced to it. Subsumption can be used to check whether a concept D is more general than a concept C, i.e., whether each instance of C also is an instance of D. For example, the concept name Parent is subsumed by the concept description $\mathsf{Man} \sqcup \mathsf{Woman}$ w.r.t. the TBox shown in Figure 1.3: by the semantics, every Parent is also a Man or a Woman. As we will discuss in more detail later, subsumption defines a hierarchy of the concept names occurring in a TBox w.r.t. their generality. There are some additional terminological inference problems such as the equivalence of concept descriptions: C and D are *equivalent* with respect to a TBox $\mathcal{T}$ (written $C \equiv_{\mathcal{T}} D$) iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$. We will not consider such additional inference problems in this chapter since they can clearly be reduced to subsumption and satisfiability in a trivial way.

There is a straightforward connection between satisfiability and subsumption: if a concept language provides for negation and conjunction, we can polynomially reduce subsumption to unsatisfiability: $C \sqsubseteq_{\mathcal{T}} D$ if and only $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{T}$. We can also do the converse reduction if the concept language provides for (or can express) the bottom concept: C is satisfiable w.r.t. $\mathcal{T}$ if and only if $C \not\sqsubseteq_{\mathcal{T}} \bot$. Because of this close connection, many description logic systems concentrate on providing algorithms for solving satisfiability, and treat subsumption by means of the above reduction.[5] Another important group of reductions is concerned with reducing satisfiability with respect to TBoxes to satisfiability w.r.t. the empty TBox. Whether such a reduction can be done depends on the concept language and the chosen TBox formalism. Here we discuss the two most important cases.

Eliminating acyclic TBoxes. As already mentioned in Section 2.2, acyclic TBoxes merely define abbreviations for compound concept descriptions. This suggests the following reduction: to decide whether a concept description C is satisfiable w.r.t. the acyclic TBox $\mathcal{T}$, first expand $\mathcal{T}$ (c.f. Section 2.2), then replace all defined concepts in C according to their definition in the expansion of $\mathcal{T}$, and finally decide satisfiability of the resulting concept description without reference to a TBox. As observed by Nebel [128], this reduction may yield an exponential blowup even for the concept language $\mathcal{FL}_0$ that only provides for the concept constructors conjunction and value restriction. For example, expanding the following TBox of size $\mathcal{O}(n)$ yields a TBox of size 2^n:

$$\begin{array}{rcl} C_1 & \equiv & \forall r_1.C_0 \sqcap \forall r_2.C_0 \\ & \vdots & \\ C_n & \equiv & \forall r_1.C_{n-1} \sqcap \forall r_2.C_{n-1} \end{array}$$

This exponential blowup can sometimes be avoided by devising satisfiability algorithms that explicitly take acyclic TBoxes into account. For example, satisfiability of $\mathcal{ALC}$ concept descriptions w.r.t. acyclic TBoxes is PSpace-complete, and without TBoxes this problem is of exactly the same complexity [142, 110]. However this is not always the case: in Sections 4 and 6, we will discuss DLs for which reasoning w.r.t. acyclic TBoxes is considerably more difficult than reasoning without them.

Eliminating GCIs. In several expressive description logics, it is possible to reduce satisfiability w.r.t. GCIs to satisfiability without reference to GCIs. Two examples are the

[5] An important exception are sub-Boolean DLs that do not provide for general negation; c.f. Section 4.

description logics $\mathcal{ALC}_{reg}$ and $\mathcal{SHOIQ}$. It is not difficult to prove that, in $\mathcal{ALC}_{reg}$, a concept description C is satisfiable w.r.t. $\mathcal{T}$ if, and only if, $C \sqcap \forall(r_1 \sqcup \cdots \sqcup r_k)^*.C_\mathcal{T}$ is satisfiable, where $r_1, \ldots, r_k$ are the role names used in C and $\mathcal{T}$, and $C_\mathcal{T}$ is defined at the end of Section 2.2. Similarly, it has been observed in [94] that, in $\mathcal{SHOIQ}$, a concept description C is satisfiable w.r.t. a TBox $\mathcal{T}$ and a role hierarchy $\mathcal{H}$[6] if and only if $C \sqcap \forall r.C_\mathcal{T}$ is satisfiable w.r.t. the role hierarchy $\mathcal{H} \cup \{r_1 \sqsubseteq r, \ldots, r_k \sqsubseteq r\}$, where $r_1, \ldots, r_k$ are the role names used in C and $\mathcal{T}$, and r is a transitive role not occurring in C and $\mathcal{T}$. Reductions like the ones sketched above are often called *internalizations* of TBoxes, and have first been proposed in [11]. However, implemented reasoning systems usually treat GCIs in an explicit way for efficiency reasons [90, 99].

We now give a brief survey of the results and techniques for terminological reasoning. The main driving force behind the research on DL reasoning is the following trade-off between expressivity and computational complexity: on the one hand, non-trivial applications require a high expressivity of the concept language and of the terminological and assertional formalism; on the other hand, applications need an implementation of DL inference algorithms in an actual knowledge representation system that exhibits an acceptable run-time behavior on "realistic" inputs, i.e. on inputs that stem from an application and have not been artificially crafted to make reasoning hard.

It is generally agreed upon that an implemented DL system should be based on algorithms that are sound, complete, and terminating, i.e., decidability of the relevant inference problems is indispensable. Fortunately, satisfiability and subsumption is indeed decidable for almost all possible combinations of concept language and TBox formalism that we have introduced up to this point, including $\mathcal{ALC}$ with cyclic TBoxes and fixpoint semantics [140], $\mathcal{ALC}_{\mathrm{reg}}$ with GCIs [54], and $\mathcal{SHOIQ}$ with GCIs [150].[7] However, decidability of reasoning is usually only a necessary, but not a sufficient condition for the usefulness of a description logic. Additionally, it is important that the computational complexity of reasoning is within acceptable bounds, and that there exist *practical* reasoning algorithms, i.e. algorithms that have the potential of being implemented in a system that behaves well on realistic inputs as demanded above.

The general opinion on the (worst-case) complexity that is acceptable has changed dramatically over time. Historically, the early times of DL research have been concentrating on identifying formalisms for which reasoning is tractable, i.e. can be performed in polynomial time.[8] Obviously, demanding tractability means that we cannot include all Boolean operators in the concept language, and thus are in the realm of sub-Boolean DLs. The complexity of satisfiability and subsumption in this family of DLs is laid out in detail in Section 4, ranging from tractable to ExpTime-complete depending on the choice of constructors and TBox formalism. Around 1990, the $\mathcal{KRIS}$ system showed that tableau algorithms for satisfiability and subsumption in $\mathcal{ALC}$ w.r.t. acyclic TBoxes, two PSpace-complete inference problems, can be implemented in a system with acceptable run-time behavior on realistic inputs [17]. A step towards even more expressive DLs has been made around 1997 by Ian Horrocks and his FaCT system, which originally implemented satisfiability and subsumption for an ExpTime-complete fragment of

[6] C is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{H}$ if there is a model $\mathcal{I}$ of C and $\mathcal{T}$ with $r^\mathcal{I} \subseteq s^\mathcal{I}$ for all $r \sqsubseteq s \in \mathcal{H}$.

[7] An exception is satisfiability of $\mathcal{SHOIQ}$-concepts w.r.t. TBoxes with least fixpoint semantics, which was shown to be undecidable by Bonatti [35].

[8] Curiously, it was found later that reasoning in the description logic supported by the very first description logic system KL-ONE is undecidable [141]—c.f. Section 5.

$\mathcal{SHOIQ}$ with GCIs. The complexity of most description logics extending $\mathcal{ALC}$ is between PSpace-complete and NExpTime-complete. Some important landmarks are the following:

- satisfiability of $\mathcal{ALC}$ concept descriptions without reference to TBoxes is PSpace-complete [142]; this also holds in the presence of acyclic TBoxes [110];
- satisfiability of $\mathcal{ALC}$ concept descriptions w.r.t. cyclic TBoxes or GCIs is ExpTime-complete; this holds for fixpoint semantics as well as for descriptive semantics [138, 140];
- satisfiability of $\mathcal{SHOIQ}$ concept descriptions w.r.t. GCIs is NExpTime-complete, with the lower bound applying to all extensions of $\mathcal{ALC}$ that provide for (qualifying or non-qualifying) number restrictions, inverse roles, and nominals [150].

All these bounds transfer to subsumption with the exception of the last one, where NExpTime-completeness of satisfiability flips to co-NExpTime-completeness of subsumption. It should be mentioned that the exact meaning of an "acceptable run-time behavior" of course also depends on the concrete application at hand. As argued e.g. in [51], there are applications that require "real" tractability and therefore research in tractable DLs is an ongoing endeavour [51, 46, 10]. Since our survey of the complexity of reasoning in DLs is by no means exhaustive, we refer the interested reader to [61] for more information on the complexity of DLs.

The issue of practicability is not only related to computational complexity, but also to the techniques that are used to obtain decision procedures for DL reasoning. A large number of such techniques have been proposed and investigated. For sub-Boolean DLs, so-called "structural algorithms" play the most important role, and we describe them in detail in Section 4. For $\mathcal{ALC}$ and its many extensions, the following approaches are most important: tableau algorithms [30], reduction techniques [54], automata-based approaches [122, 50], and resolution calculi [101, 100]. With respect to practicability, tableau algorithms are the most successful approach so far: they proved to be amenable to a number of powerful optimization techniques (see Chapter 4 and [92]), and highly-optimized implementations of tableau algorithms in DL systems have performed extraordinarily well in system comparisons. As a result, nowadays almost all state-of-the-art DL reasoners, such as FaCT and RACER [91, 79], are based on tableau algorithms.

From the perspective of modal logic, satisfiability of concept descriptions clearly corresponds to standard formula satisfiability, whereas a subsumption $C \sqsubseteq D$ corresponds to the validity of the implication $C \rightarrow D$. For this reason, the discussion of tableau- and resolution-based algorithms for modal logics provided in Chapter 4 of this handbook applies to description logics as well, and we omit further details.

3.2 Assertional Reasoning

The inference problems discussed in this section operate on knowledge bases, i.e. on pairs $(\mathcal{A}, \mathcal{T})$ with $\mathcal{A}$ an ABox and $\mathcal{T}$ a TBox. The fundamental inference problems are the following:

Consistency. An ABox $\mathcal{A}$ is *consistent* w.r.t. a TBox $\mathcal{T}$ if there exists a common model of $\mathcal{A}$ and $\mathcal{T}$.

Instance Checking. An individual name a in an ABox $\mathcal{A}$ is an *instance* of a concept description C w.r.t. a TBox $\mathcal{T}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$ (denoted with $\mathcal{A} \models_{\mathcal{T}} C(a)$).

Consistency is the ABox-analogue of satisfiability, i.e., it can be used to check whether a given knowledge base is contradictory. The purpose of instance checking is also obvious: it is used to derive concept memberships of individuals that are not stated explicitly. For example, the individual DONALD in the ABox shown in Figure 1.4 is an instance of Father w.r.t. the TBox in Figure 1.3.

As in the previous section, there is a close connection between the two fundamental inference problems if certain Boolean constructors are available. First, consistency can be polynomially reduced to (non-)instance checking if the bottom concept is available: an ABox $\mathcal{A}$ is consistent w.r.t. a TBox $\mathcal{T}$ if and only if $\mathcal{A} \not\models_{\mathcal{T}} \bot(a)$ with a an arbitrary individual name. And second, we can do the converse reduction if full negation is available: $\mathcal{A} \models_{\mathcal{T}} C(a)$ if and only if $\mathcal{A} \cup \{\neg C(a)\}$ is inconsistent w.r.t. $\mathcal{T}$. It is sometimes also possible to eliminate acyclic TBoxes and GCIs as discussed in the previous section. In the presence of nominals, it is possible to polynomially reduce consistency (and thus also instance checking) to the satisfiability of concept descriptions using the simulation sketched at the end of Section 2.3.

The techniques used to devise decision procedures for consistency and instance checking are essentially the same as those employed for concept satisfiability and subsumption. In the case of tableau algorithms, there are two approaches for reasoning with ABoxes: first, ABox consistency can sometimes be reduced to concept satisfiability using the *pre-completion* technique described in [87]; and second, tableau algorithms can be extended to treat ABoxes in a direct way (see e.g. [86, 15, 77]). Regarding practicability, it should be noted that some optimization techniques fail or become significantly more complex in the presence of ABoxes.

Concerning the decidability and computational complexity of assertional reasoning, one should distinguish sub-Boolean DLs from $\mathcal{ALC}$ and its extensions. In the sub-Boolean case, there are some description logics for which instance checking is harder than concept subsumption (see Section 4.3). If all Boolean constructors are available, the complexity of instance checking coincides with the complexity of subsumption for all such description logics investigated so far. For example, instance checking in $\mathcal{ALC}$ ABoxes without reference to TBoxes is known to be PSPACE-complete [18], and instance checking in $\mathcal{ALC}$ ABoxes w.r.t. GCIs is EXPTIME-complete [54]—just as the corresponding cases of subsumption.

3.3 Compound Inference Problems

Some of the most important inference problems in DLs are of a compound nature in the sense that, in principle, they can be reduced to multiple invocations of the more basic inference problems mentioned above. However, when the goal is to achieve an efficient implementation, it is vital to consider compound inferences as first-class citizens [13]. Here we discuss the three most important such problems.

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

Realization. Given an ABox $\mathcal{A}$, a TBox $\mathcal{T}$, and an individual name a, compute the set $R_{\mathcal{A},\mathcal{T}}(a)$ of those concept names A that are used in $\mathcal{T}$, satisfy $\mathcal{A} \models_{\mathcal{T}} A(a)$, and are minimal with this property w.r.t. the subsumption relation "$\sqsubseteq_\mathcal{T}$".

Retrieval. Given an ABox $\mathcal{A}$, a TBox $\mathcal{T}$, and a concept C, compute the set $I_{\mathcal{A},\mathcal{T}}(C)$ of individual names a used in $\mathcal{A}$ and satisfying $\mathcal{A} \models_\mathcal{T} C(a)$.

Compound inferences are a very important interface to description logics reasoners and are offered by almost all systems. The purpose of classification is to construct a hierarchy of concept names w.r.t. their generality, with more general concepts higher up in the hierarchy: B is above A if and only if $A \sqsubseteq_\mathcal{T} B$. Such a hierarchy can then be presented to a knowledge engineer for browsing and structuring the TBox. Realization also facilitates browsing and understanding of the knowledge base, and is a precursor to certain operations on knowledge bases that presuppose knowledge of the concept memberships of individuals. The main use of retrieval is database-like querying of description logic knowledge bases: in some applications, it is natural to define ABoxes with a huge number of individual names, and to query such ABoxes like a database with deductive capabilities [78].

By definition, compound inferences can be reduced to more basic inference problems. For classification, we may simply check whether $A \sqsubseteq_\mathcal{T} B$ for all concept names A, B used in $\mathcal{T}$. In the case of realization, we can obviously just use multiple invocations of instance checking and subsumption. Similarly, multiple instance checks suffice to get a naïve implementation of retrieval. However, basic inferences such as subsumption and instance checking are potentially very costly, and thus it is vital for DL reasoners to replace these "brute force" methods of compound inferences by more subtle approaches.

To illustrate how compound inferences can be implemented in a more efficient way, we exemplarily consider classification. Here, the aim is to minimize the number of subsumption tests, of which the naïve approach performs n^2 many with n the number of concept names in $\mathcal{T}$. The common strategies for achieving this minimization are described and evaluated by Baader et al. in [13]. Although Baader et al. restrict themselves to acyclic TBoxes, the proposed strategies can also be used for cyclic ones. In general, two kinds of optimizations can be distinguished. Firstly, classification can be conceived as an abstract combinatorial problem on partial orders: compute a complete representation of a partial ordering by making as few as possible comparisons. This quite general problem is also considered in non-DL contexts, see e.g. [69]. Secondly, we can take into account the structure of concept descriptions to reveal obvious subsumption relationships and to control the order in which concepts are added to the hierarchy. In the following, we assume that the restriction of "$\sqsubseteq_\mathcal{T}$" to the concepts names of $\mathcal{T}$ is represented as a Hasse diagram, i.e. as a directed acyclic graph (DAG) such that

- nodes are sets of concept names that are pair-wise equivalent w.r.t. $\mathcal{T}$;
- two nodes S_1, S_2 are connected by an edge if every $A_2 \in S_2$ is a *direct subsumer* of every $A_1 \in S_1$, i.e., we have (i) $A_1 \sqsubseteq_\mathcal{T} A_2$ and (ii) $A_1 \sqsubseteq_\mathcal{T} B \sqsubseteq_\mathcal{T} A_2$ implies $B \equiv_\mathcal{T} A_1$ or $B \equiv_\mathcal{T} A_2$ for all concept names B in $\mathcal{T}$.

To this diagram, we henceforth refer as the *(concept) hierarchy.* We assume that the hierarchy always contains a top node whose label includes $\top$, and a bottom node whose label includes $\bot$.[9] In DL TBoxes originating from applications, the concept hierarchy is usually not too deep, i.e., the represented order has short chains and long antichains.

[9]In case of unsatisfiable TBoxes, the top node and bottom node coincide.

One way to compute the concept hierarchy with only few subsumption tests is to use an incremental algorithm [13]: we start with a hierarchy containing only $\top$ and $\bot$, and then repeatedly place additional concept names at the appropriate position in the (growing) hierarchy. The placing of a new concept name A consists of two phases: a *top search* phase computing the direct subsumers of A that are already contained in the hierarchy, and a *bottom search* phase computing the set of all concept names that are already contained in the hierarchy, and of which A is a direct subsumer. Obviously, knowledge of these two sets allows us to place A appropriately. Due to the transitivity of the subsumption relation, the top search phase is best implemented as a top down search, whereas a bottom up approach is appropriate for the bottom search phase. Additionally, failed tests can be propagated down the hierarchy in the top search phase: if $A \not\sqsubseteq_{\mathcal{T}} B$ and B' is below B in the hierarchy (implying $B' \sqsubseteq_{\mathcal{T}} B$), then it follows immediately that $A \not\sqsubseteq_{\mathcal{T}} B'$. Analogously to propagation in the top search phase, successful tests can be propagated up the hierarchy in the bottom search phase. Finally, it is possible to use information gained in the top search phase to speed up the bottom search phase, and vice versa (see [13] for details).

Using the structure of concepts, we can additionally avoid subsumption tests in a straightforward way: if we find a concept definition $A \equiv C$ with C a conjunction having as one of its conjuncts a concept name B, then $A \sqsubseteq_{\mathcal{T}} B$ holds trivially. In this case, B is a *told subsumer* of A. Of course, if B is a defined concept, it can have told subsumers as well, and these (and their told subsumers, etc.) can also be viewed as told subsumers of A. The information about the told subsumers can be propagated down the hierarchy before starting the top search phase. To take full advantage of this idea, it is advisable to classify concepts in *definition order*. This means that a concept is not classified until all of its told subsumers are classified.

These optimizations typically reduce the number of necessary subsumption tests to a small fraction of n^2 (see [13] for details). Most techniques sketched here can be used in the same or a slightly modified form if sets of GCIs are used instead of TBoxes. Of course, it is (at least) equally important to optimize the subsumption test itself. More on this issue can be found in Chapter 4.

4 SUB-BOOLEAN DESCRIPTION LOGICS

As mentioned in the introduction, the DLs used in the first DL systems did not allow for all Boolean operators. Usually, these DLs provided for conjunction, value-restriction, and number restriction, and some other special constructors, but existential restriction, disjunction and full negation were not available. In some of these formalisms, disjointness statements between concept names or atomic negation (i.e., negation restricted to concept names) were allowed.

This restriction to sub-Boolean logics was, on the one hand, due to the origins of DLs. These formalisms were not primarily seen as logics (where the inclusion of at least proposition logic is natural), but as knowledge representation formalisms in the spirit of semantic networks and frames, though equipped with a formal semantics. Graph-based formalisms like semantic networks usually favor a conjunctive point of view since conjunction corresponds to just drawing several things in the same picture, whereas expressing disjunction and negation would require special conventions (like drawing a box around the parts that are negated, as in conceptual graphs [147]), which easily destroy the readability of such graphical representations.

On the other hand, the restriction to sub-Boolean DLs was motivated by the goal of designing representation formalisms with tractable (i.e., polynomial-time decidable) inference problems, which would be precluded by the presence of all Boolean operators. The first paper addressing the trade-off between expressiveness and tractability of reasoning in the context of DL was [109], where it was shown that a seemingly minor extension of the description language can make the subsumption problem intractable. This work triggered an extensive investigation of the borderline between tractability and intractability of reasoning in sub-Boolean DLs [127, 145, 63, 62, 40, 65].[10] In *Section 4.1*, we give a brief review of these results. We also sketch in more detail polynomial-time subsumption algorithms for the DL $\mathcal{FL}_0$ (which allows for conjunction and value restriction only) and some of its extensions. The results mentioned until now were all restricted to extensions of $\mathcal{FL}_0$. The reason was that until the late 1990ies, both conjunction and value restriction were assumed to be indispensable for a DL. For conjunction this indeed appears to be the case since one usually wants to require several properties simultaneously when defining a concept. In order to obtain more than just a fragment of propositional logic, one also needs at least one constructor involving roles. However, instead of value restrictions one could also use existential restrictions. In fact, there are large DL-based medical terminologies [134, 148] that employ existential restrictions rather than value restrictions. The complexity of reasoning in DLs extending $\mathcal{EL}$, which allows for conjunction, existential restriction, and the top-concept, is less well-investigated than for extensions of $\mathcal{FL}_0$. We will briefly review the results in [84], where the complexity of the satisfiability problem in all fragments of $\mathcal{ALC}$, including ones that do not extend $\mathcal{FL}_0$, is investigated. In addition, we sketch a polynomial-time subsumption algorithms for $\mathcal{EL}$.

All the complexity results mentioned until now are concerned with satisfiability and subsumption of concept descriptions. If one considers reasoning w.r.t. a TBox, then the complexity may increase drastically, even for acyclic TBoxes, which do not increase the expressive power. The first such result is due to Nebel [128], who showed that the subsumption problem w.r.t. acyclic TBoxes is coNP-complete for the DL $\mathcal{FL}_0$. Recall that subsumption of $\mathcal{FL}_0$ concept descriptions is polynomial. If one allows for cyclic TBoxes, then subsumption in $\mathcal{FL}_0$ becomes PSpace-complete, and in the presence of GCIs it becomes even ExpTime-complete. In contrast, the subsumption problem in $\mathcal{EL}$ remains polynomial in the presence of acyclic or cyclic TBoxes and GCIs. These results for reasoning w.r.t. TBoxes in sub-Boolean DLs will be described in more detail in *Section 4.2*.

Not only TBox reasoning, but also ABox reasoning, may be harder than reasoning with concept descriptions. *Section 4.3* gives an example, due to A. Schaerf [137], of a sub-Boolean DL where the instance problem is harder than the subsumption problem.

Finally, *Section 4.4* reviews bisimulation characterizations for various sub-Boolean DLs due to de Rijke and Kurtonina [105], which can be used to characterize the expressive power of these DLs.

[10] We have not included [64] in this list since the polynomiality results for subsumption claimed there turned out to be incorrect (see [66] for details).

Symbol	Syntax	$\mathcal{ALN}$	$\mathcal{ALE}$	$\mathcal{ALU}$	$\mathcal{ALUN}$	$\mathcal{ALEN}$	$\mathcal{ALC}$	$\mathcal{ALCN}$
$\mathcal{E}$	$\exists r.C$		x			x	x	x
$\mathcal{U}$	$C \sqcup D$			x	x		x	x
$\mathcal{N}$	$(\leqslant n\,r), (\geqslant n\,r)$	x			x	x		x

Figure 1.5. The $\mathcal{AL}$ family of DLs.

4.1 Reasoning with concept descriptions in sub-Boolean DLs

Donini et al. [65] start their investigation of the complexity of sub-Boolean DLs with the DL $\mathcal{AL}$, whose concept descriptions are formed according to the following syntax rule:

$$C, D \longrightarrow A \mid \top \mid \bot \mid \neg A \mid C \sqcap D \mid \forall r.C \mid \exists r.\top$$

The difference to $\mathcal{ALC}$ is that the application of negation is restricted to concept names (atomic negation) and that in existential restrictions only the top concept may occur (restricted existential quantification).

In the following, we consider the extensions of $\mathcal{AL}$ by subsets of the following set of constructors: (full) existential restriction, number restrictions, and disjunction.[11] This yields the 7 different extensions of $\mathcal{AL}$ shown in Figure 1.5. Note that adding both existential restriction and disjunction to $\mathcal{AL}$ yields $\mathcal{ALC}$. This is due to the presence of atomic negation in $\mathcal{AL}$. In fact, by using de Morgan's law, the duality of the quantifiers, and the removal of double negation, any $\mathcal{ALC}$ concept description can be transformed into an equivalent one that employs only atomic negation.

Figure 1.6 gives a complete classification of the DLs belonging to the $\mathcal{AL}$ family regarding the worst-case complexity of subsumption and (un)satisfiability of concept descriptions. Except for the case of $\mathcal{ALEN}$, these results are shown in [65]. PSPACE-hardness of $\mathcal{ALEN}$ was shown by Hemaspaandra [84].

Before trying to explain some of these results, let us first point out that subsumption and unsatisfiability are in general not trivially interreducible in sub-Boolean DLs. We have seen that

$$\begin{array}{rll} C \text{ is unsatisfiable} & \text{iff} & C \sqsubseteq \bot, \\ C \sqsubseteq D & \text{iff} & \neg C \sqcap D \text{ is unsatisfiable.} \end{array}$$

Since $\bot$ is available in the DLs of the $\mathcal{AL}$ family, unsatisfiability can be reduced in this way to subsumption, and thus subsumption is at least as hard as unsatisfiability, and unsatisfiability is at least as easy as subsumption. However, for a DL strictly below $\mathcal{ALC}$, $\neg C \sqcap D$ usually does not belong to this DL even if C and D do. Nevertheless, the results summarized in Figure 1.6 show that, for the $\mathcal{AL}$ family, the complexities of unsatisfiability and of subsumption coincide. In general, this need not be the case. For example, the extension $\mathcal{ALNI}$ of $\mathcal{ALN}$ by inverse roles has a polynomial-time (un)satisfiability problem, but the subsumption problem is coNP-hard [66]. In very simple DLs such as $\mathcal{FL}_0$ and $\mathcal{EL}$, satisfiability is even trivial (in contrast to subsumption) since there are no unsatisfiable concepts We exemplarily discuss at least one DL for each of the complexity classes appearing in Figure 1.6. Satisfiability in intractable DLs is treated in Section 4.1 and subsumption in tractable DLs such as $\mathcal{FL}_0$ and $\mathcal{ALN}$ is discussed in Section 4.1.

[11] Donini et al. [65] actually consider a somewhat larger family of DLs, where also intersection of roles is available as a constructor.

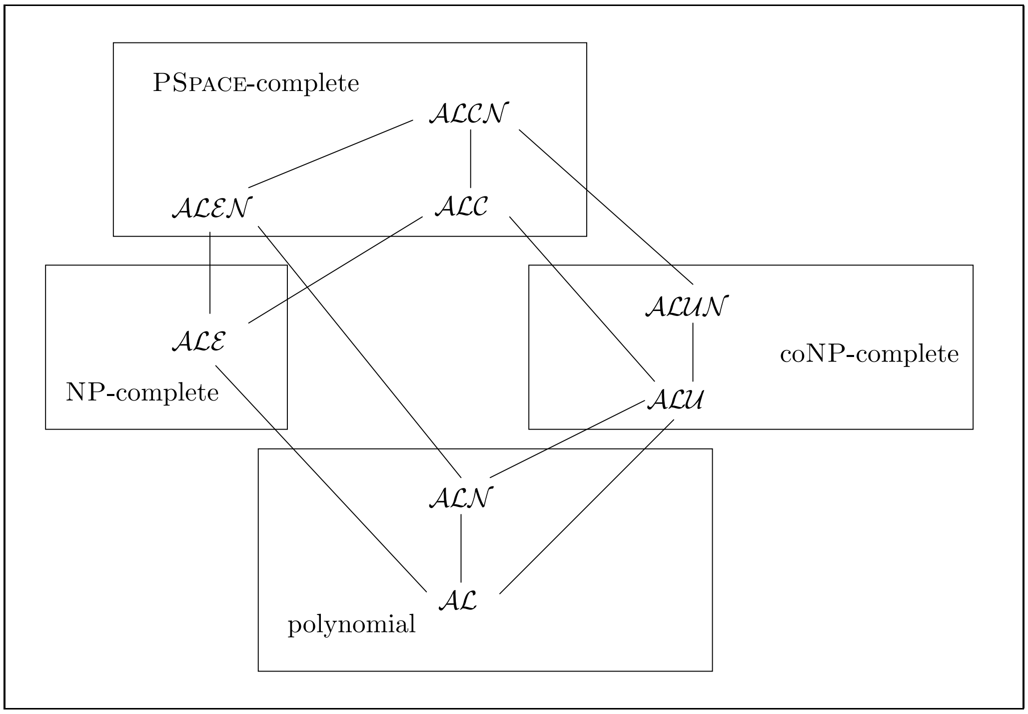

Figure 1.6. The complexity of unsatisfiability and subsumption for the $\mathcal{AL}$ family of DLs.

In Section 4.1 we review the results from [84] on satisfiability in other sub-Boolean DLs, and in Section 4.1 we sketch a polynomial-time subsumption algorithm for $\mathcal{EL}$ concept description.

(Un)satisfiability in $\mathcal{ALC}$, $\mathcal{ALU}$, and $\mathcal{ALE}$

One way of deciding satisfiability in $\mathcal{ALC}$ is to use a tableau algorithm, as described in more detail in Chapter 4 for the modal logic equivalent K_n of $\mathcal{ALC}$. This algorithm tries to generate a finite, tree-shaped model for a given input concept description C. This tree model is of depth linear in the size of C, but may still be exponentially large due to the branching in the tree model. There are two sources of complexity for this approach: first, the potentially exponential size of the model that must be generated, and second the non-deterministic treatment of disjunction when trying to generate the model. In order to stay within PSpace, one generates one branch of the tree at a time, whereas non-determinism is harmless since it is well-known that NPSpace = PSpace.

If we restrict this approach to $\mathcal{ALU}$, then it is easy to see that the tree models generated by a tableau-based algorithm are of polynomial size. Thus, to decide satisfiability within NP (and thus unsatisfiability within coNP) one can simply guess an interpretation of polynomial size, and check whether it is a model. NP-hardness is trivial since $\mathcal{ALU}$ contains full propositional logic.

In the case of $\mathcal{ALE}$, the model generated by a tableau-based algorithm may still be exponentially large, but the process of generating it is deterministic. In order to check unsatisfiability within NP (and thus satisfiability within coNP), one can guess one path through the potential model, and then check whether it must satisfy contradictory constraints. To be more precise, instead of trying to generate successors for all existential restrictions, one non-deterministically chooses the ones that actually lead to a contradiction. Since the paths are linear in the size of the input description, this leads to an NP-algorithm for unsatisfiability.

Showing the lower complexity bound for $\mathcal{ALE}$ is less trivial than for $\mathcal{ALU}$. To show that unsatisfiability of concept descriptions in $\mathcal{ALE}$ is NP-complete, we sketch the reduction from set traversal given in [62]. An instance of the *set traversal problem* is given by a finite collection $\mathcal{M} = \{M_1, \ldots, M_m\}$ of finite sets of positive integers. A *set traversal* is a finite set of positive integers N such that $N \cap M_\ell$ is a singleton set for all $\ell, 1 \leq \ell \leq m$. NP-hardness of the existence of a set traversal is an immediate consequence of the fact that monotone ONE-IN-THREE 3SAT [73] is a special case of this problem.

Let $\mathcal{M} = \{M_1, \ldots, M_m\}$ be an instance of the set traversal problem, and assume without loss of generality that the numbers occurring in the sets are the ones from 1 to n for some positive integer n. We define the corresponding $\mathcal{ALE}$ concept description as

$$C_{\mathcal{M}} := C_1 \sqcap \ldots \sqcap C_n \sqcap D,$$

where

$$C_\ell := Q_{1,\ell} r.Q_{2,\ell} r. \cdots Q_{m,\ell} r.\ Q_{1,\ell} r.Q_{2,\ell} r. \cdots Q_{m,\ell} r.\ \top$$

such that

$$Q_{i,\ell} = \begin{cases} \exists & \text{if } \ell \in M_i, \\ \forall & \text{if } \ell \notin M_i, \end{cases}$$

and D is the nesting of $2m$ value restrictions followed by $\bot$, i.e.,

$$D := \underbrace{\forall r. \cdots \forall r.}_{2m \text{ times}} \bot.$$

As an example, consider the two instances

$$\mathcal{M} := \{\{1,3,5\}, \{2,4\}, \{4,5\}\} \qquad \text{and} \qquad \mathcal{M}' := \{\{1,3\}, \{2,4\}, \{4,5\}\}$$

of the set traversal problem. Then the corresponding $\mathcal{ALE}$ concept descriptions look as follows:

$$\begin{array}{ll}
C_1 = \exists\mathbf{r}.\forall r.\forall r.\ \exists\mathbf{r}.\forall r.\forall .r\ \top, & C_1' = \exists\mathbf{r}.\forall r.\forall r.\ \exists\mathbf{r}.\forall r.\forall .r\ \top, \\
C_2 = \forall r.\exists\mathbf{r}.\forall r.\ \forall r.\exists\mathbf{r}.\forall r.\ \top, & C_2' = \forall r.\exists\mathbf{r}.\forall r.\ \forall r.\exists\mathbf{r}.\forall r.\ \top, \\
C_3 = \exists\mathbf{r}.\forall r.\forall r.\ \exists\mathbf{r}.\forall r.\forall r.\ \top, & C_3' = \exists\mathbf{r}.\forall r.\forall r.\ \exists\mathbf{r}.\forall r.\forall r.\ \top, \\
C_4 = \forall r.\exists\mathbf{r}.\exists\mathbf{r}.\ \forall r.\exists\mathbf{r}.\exists\mathbf{r}.\ \top, & C_4' = \forall r.\exists\mathbf{r}.\exists\mathbf{r}.\ \forall r.\exists\mathbf{r}.\exists\mathbf{r}.\ \top, \\
C_5 = \exists\mathbf{r}.\forall r.\exists\mathbf{r}.\ \exists\mathbf{r}.\forall r.\exists\mathbf{r}.\ \top, & C_5' = \forall r.\forall r.\exists\mathbf{r}.\ \forall r.\forall r.\exists\mathbf{r}.\ \top, \\
D = \forall r.\forall r.\forall r.\ \forall r.\forall r.\forall r.\ \bot, & D' = \forall r.\forall r.\forall r.\ \forall r.\forall r.\forall r.\ \bot.
\end{array}$$

One can view the quantifier prefixes as a matrix, where the rows correspond to the elements of the sets, whereas the columns correspond to the sets (written twice). The existential quantifier indicates that the element belongs to the respective set. For example, the first existential quantifier in C_5 expresses that 5 belongs to the first set of $\mathcal{M}$, whereas the universal quantifier at the same position in C'_5 says that 5 does not belong to the first set of $\mathcal{M}'$.

In [62], it is shown that $C_{\mathcal{M}}$ is unsatisfiable iff $\mathcal{M}$ has a set traversal. We illustrate the connection between unsatisfiability of $C_{\mathcal{M}}$ and the existence of a set traversal for $\mathcal{M}$ on our example. For $C_{\mathcal{M}}$ to be unsatisfiable, the conjunction of the concept descriptions C_i must enforce an r-path of length $2m$, which then clashes with the $\bot$ in D. The set $\{1, 2, 5\}$ is a set traversal for $\mathcal{M}'$. This implies that $C'_1 \sqcap C'_2 \sqcap C'_5$ enforces a path of length 6. In fact, C'_1 starts with an existential restriction, and C'_2 and C'_5 with value restrictions. Thus, there is an r-successor of the initial element that must satisfy the conjunction of the three concept description C''_1, C''_2, C''_5 that are respectively obtained from C'_1, C'_2, C'_5 by removing the first quantifier. Now C''_2 starts with an existential quantifier, and the other two with value restrictions. Thus, we can continue with the corresponding r-successor. In general, the properties of a set traversal ensure that each time we have one existential restriction, whereas all the others are value restrictions (and thus the remaining parts of the concept descriptions are propagated to the r-successor required by the existential restriction).

It remains to explain why we have to encode the sets twice. Again, we illustrate this on our example. It is easy to see that the collection $\mathcal{M} := \{\{1, 3, 5\}, \{2, 4\}, \{4, 5\}\}$ does not have a set traversal. Nevertheless, if we consider the simpler reduction concept $\widehat{C}_{\mathcal{M}} := \widehat{C}_1 \sqcap \widehat{C}_2 \sqcap \widehat{C}_3 \sqcap \widehat{C}_4 \sqcap \widehat{C}_5 \sqcap \widehat{D}$ where

$$
\begin{aligned}
\widehat{C}_1 &= \exists\mathbf{r}.\forall r.\forall.r\ \top,\\
\widehat{C}_2 &= \forall r.\exists\mathbf{r}.\forall r.\ \top,\\
\widehat{C}_3 &= \exists\mathbf{r}.\forall r.\forall r.\ \top,\\
\widehat{C}_4 &= \forall r.\exists\mathbf{r}.\exists\mathbf{r}.\ \top,\\
\widehat{C}_5 &= \exists\mathbf{r}.\forall r.\exists\mathbf{r}.\ \top,\\
\widehat{D} &= \forall r.\forall r.\forall r.\ \bot,
\end{aligned}
$$

then $\widehat{C}_{\mathcal{M}}$ is unsatisfiable. In fact, $\widehat{C}_4 \sqcap \widehat{C}_5$ enforce a path of length 3. The corresponding set $\{4, 5\}$ is not a set traversal since its intersection with $M_3 = \{4, 5\}$ is not a singleton. For the shorter reduction concept $\widehat{C}_{\mathcal{M}}$, the fact that $\widehat{C}_4$ and $\widehat{C}_5$ have an existential restriction in the same row is irrelevant. However, for the correct longer reduction concept $C_{\mathcal{M}}$ this means that, for one of the two, the remaining part is missing in the second round.

Subsumption in $\mathcal{FL}_0$ and $\mathcal{ALN}$

Subsumption in $\mathcal{ALN}$ can be decided in polynomial time using a *structural subsumption algorithm*, i.e., an algorithm that normalizes the descriptions to be tested for subsumption, and then compares the syntactic structure of the normal forms. For simplicity, we first explain the ideas underlying this approach for the small DL $\mathcal{FL}_0$, which allows for conjunction ($C \sqcap D$) and value restriction ($\forall r.C$) only. Subsequently, we show how the bottom concept ($\bot$), atomic negation ($\neg A$), and number restrictions ($\leqslant n\, r$ and $\geqslant n\, r$) can be handled.

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

$$A_1 \sqcap \ldots \sqcap A_m \sqcap \forall r_1.C_1 \sqcap \ldots \sqcap \forall r_n.C_n,$$

where $A_1, \ldots, A_m$ are distinct concept names, $r_1, \ldots, r_n$ are *distinct* role names, and $C_1, \ldots, C_n$ are $\mathcal{FL}_0$ concept descriptions in normal form. It is easy to see that any description can be transformed into an equivalent one in normal form, using associativity, commutativity and idempotence of $\sqcap$, and the fact that the descriptions $\forall r.(C \sqcap D)$ and $(\forall r.C) \sqcap (\forall r.D)$ are equivalent. Now, let

$$C \equiv A_1 \sqcap \ldots \sqcap A_m \sqcap \forall r_1.C_1 \sqcap \ldots \sqcap \forall r_n.C_n \quad \text{and} \quad D \equiv B_1 \sqcap \ldots \sqcap B_k \sqcap \forall s_1.D_1 \sqcap \ldots \sqcap \forall s_\ell.D_\ell$$

respectively be the normal forms of the $\mathcal{FL}_0$ concept descriptions C and D. Then $C \sqsubseteq D$ iff the following two conditions hold:

1. for all $i, 1 \leq i \leq k$, there exists $j, 1 \leq j \leq m$ such that $B_i = A_j$.
2. For all $i, 1 \leq i \leq \ell$, there exists $j, 1 \leq j \leq n$ such that $s_i = r_j$ and $C_j \sqsubseteq D_i$.

It is easy to see that this characterization of subsumption is sound (i.e., the "if" direction of the equivalence holds) and complete (i.e., the "only-if" direction of the equivalence holds as well). This characterization yields an obvious recursive algorithm for computing subsumption. This algorithm can easily be shown to be of polynomial time complexity: in Condition 2, there is at most one subsumption test per role name occurring in C and D since all role names in $r_1, \ldots, r_n$ and all role names in $s_1, \ldots, s_\ell$ are distinct.

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

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

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

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

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

Finally, if we consider the language $\mathcal{ALN}$, the additional presence of number restrictions leads to a new type of conflict. On the one hand, as in the case of atomic negation, number restrictions may be conflicting with each other (e.g., $\geqslant 2\,r$ and $\leqslant 1\,r$). On the other hand, at-least restrictions $\geqslant n\,r$ for $n \geq 1$ are in conflict with value restrictions $\forall r.\bot$ that prohibit role successors. When computing the normal form, one can again treat number restrictions like concept names, and then take care of the new types of conflicts by introducing $\bot$ and using it for normalization as described above. During the structural comparison of normal forms, one must also take into account inherent subsumption relationships between number restrictions (e.g., $\geqslant n\,r \sqsubseteq \geqslant m\,r$ iff $n \geq m$). A more detailed description of a structural subsumption algorithm working on a graph-like data structure for a DL extending $\mathcal{ALN}$ can be found in [40].

Satisfiability in other sub-Boolean DLs

Until now, we have only considered sub-Boolean DLs that extend $\mathcal{AL}$. Hemaspaandra [84] looks at all possible combinations of the constructors

$$\top,\quad \bot,\quad C \sqcap D,\quad C \sqcup D,\quad \neg A,\quad \neg C,\quad \forall r.C,\quad \exists r.C,$$

and shows that there are only four possibilities for the complexity of the *satisfiability* problem:[12] P, NP-complete, coNP-complete, and PSPACE-complete:

- DLs that contain a complete basis for $\mathcal{ALC}$ have a PSPACE-complete satisfiability problem.
- DLs that contain a complete basis for propositional logic, but not for $\mathcal{ALC}$, have an NP-complete satisfiability problem.
- The DL $\mathcal{ALE}$ and its sublanguages where $\bot$, $\top$, or both are disallowed, have a coNP-complete satisfiability problem.
- The DL defined by the constructors $\bot$, $C \sqcap D$, $C \sqcup D$, $\forall r.C$, $\exists r.C$ has a PSPACE-complete satisfiability problem.
- All other DLs obtained as a combination of the above constructors have a polynomial satisfiability problem.

Subsumption in $\mathcal{EL}$

Recall that $\mathcal{EL}$ is defined by the constructors top concept ($\top$), conjunction ($C \sqcap D$), and existential restriction ($\exists r.C$). We show that subsumption of $\mathcal{EL}$ concept descriptions can be decided in polynomial time by reducing the subsumption problem to a combinatorial problem on trees. Any $\mathcal{EL}$ concept description C can be represented as a tree $\mathcal{G}_C$ whose edges are labeled with role names and whose nodes are labeled with sets of primitive concepts (where the empty set stands for $\top$). For example, the $\mathcal{EL}$ concept descriptions

$$\begin{aligned} C &:= P \sqcap \exists r.(\exists r.(P \sqcap Q) \sqcap \exists s.Q) \sqcap \exists r.(P \sqcap \exists s.P) \\ D &:= \exists r.(\exists r.P \sqcap \exists s.Q) \sqcap \exists r.P \end{aligned}$$

yield the $\mathcal{EL}$ description trees $\mathcal{G}_C$ and $\mathcal{G}_D$ depicted in Figure 1.7 (see [25] for a formal definition of the translation between $\mathcal{EL}$ concept descriptions and $\mathcal{EL}$ description trees).

[12] Unfortunately, the complexity of subsumption is not considered in [84].

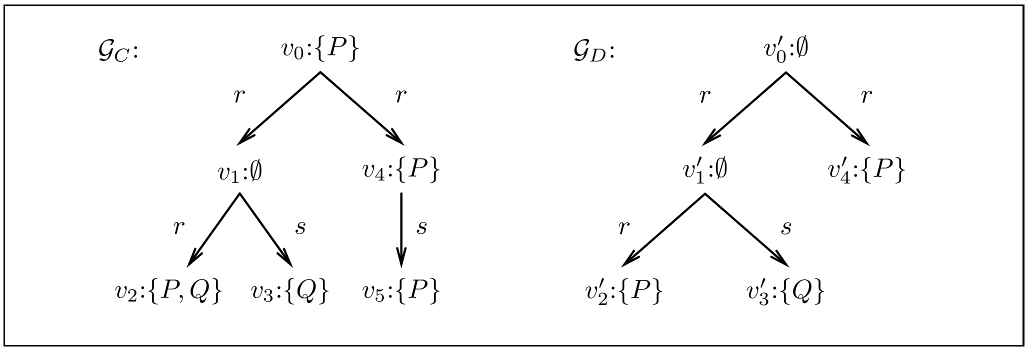

Figure 1.7. Two $\mathcal{EL}$ description trees.

Let C, D be $\mathcal{EL}$ concept descriptions. A *simulation* from $\mathcal{G}_D$ to $\mathcal{G}_C$ is a binary relation Z between the nodes of $\mathcal{G}_D$ and the nodes of $\mathcal{G}_C$ such that

1. $(v', v) \in Z$ implies that the label of v' is contained in the label of v;
2. $(v', v) \in Z$ implies that, for every r-successor u' of v' there is an r-successor u of v such that $(u', u) \in Z$.

Subsumption between $\mathcal{EL}$ concept descriptions corresponds to the existence of a simulation relation[13] between the corresponding trees: if v_0 is the root of $\mathcal{G}_C$ and v'_0 the root of $\mathcal{G}_D$, then we have

$$C \sqsubseteq D \quad \text{iff} \quad \text{there is a simulation } Z \text{ from } \mathcal{G}_D \text{ to } \mathcal{G}_C \text{ such that } (v'_0, v_0) \in Z.$$

In our example, we have $C \sqsubseteq D$ since the following relation Z is a simulation from $\mathcal{G}_D$ to $\mathcal{G}_C$:

$$Z := \{(v'_0, v_0),\ (v'_1, v_1),\ (v'_2, v_2),\ (v'_3, v_3),\ (v'_4, v_4)\}.$$

The definition of a simulation suggests the following top-down algorithm for constructing a simulation Z containing the tuple consisting of the roots (v'_0, v_0). First, put (v'_0, v_0) into Z and check whether the first property of a simulation (containment of labels) is satisfied for this tuple. If not, then stop with failure. Otherwise, try to extend Z by guessing pairs of successors of v'_0 and v_0, respectively, such that the second property of a simulation is satisfied for the pair (v'_0, v_0). Then continue the process with these new pairs. Since there are different ways of pairing off the successors, this algorithm is non-deterministic, and thus it does not yield a deterministic polynomial-time subsumption algorithm.

Fortunately, one can do better. One can compute the largest simulation between two trees (and actually also between two graphs with distinguished "root" nodes) by starting with all pairs of nodes, and then successively removing pairs that violate the first condition in the definition of a simulation or the second one (w.r.t. the current relation). It is easy to see that this procedure terminates after polynomially many steps with the largest simulation $\widehat{Z}$ from $\mathcal{G}_C$ to $\mathcal{G}_D$ (see [85] for a more efficient algorithm). We have $C \sqsubseteq D$ iff the pair consisting of the root nodes has not been removed, i.e., belongs to $\widehat{Z}$ (see [9] for more details).

[13] In [25], subsumption was actually characterized by the existence of a homomorphism, i.e., a simulation that is a total function. However, it is easy to see that, in case of *trees*, the existence of a simulation implies the existence of a homomorphism.

4.2 TBox reasoning in sub-Boolean DLs

As mentioned in Section 3, reasoning w.r.t. acyclic TBoxes can be reduced to reasoning on concept descriptions by expanding the definitions. Unfortunately, expansion may lead to an exponential blow-up of the descriptions. Is this due to the inherent complexity of reasoning with TBoxes, or can this exponential increase in the complexity be avoided? It turns out that the answer to this question depends on which sub-Boolean DL we consider.[14]

For the DL $\mathcal{FL}_0$, subsumption of concept descriptions is polynomial, whereas subsumption w.r.t. acyclic TBoxes is coNP-complete, subsumption w.r.t. cyclic TBoxes is PSpace-complete, and subsumption w.r.t. GCIs in ExpTime-complete. In contrast, for the DL $\mathcal{EL}$, subsumption remains polynomial even w.r.t. GCIs.

TBox reasoning in $\mathcal{FL}_0$

We start by describing an alternative approach for showing that subsumption of $\mathcal{FL}_0$ concept descriptions can be decided in polynomial time. In Section 4.1, the equivalence $\forall r.C \sqcap \forall r.D \equiv \forall r.(C \sqcap D)$ was used as a rewrite rule from left to right in order to compute the *structural subsumption normal form* of $\mathcal{FL}_0$ concept descriptions. If we use this rule in the opposite direction, we obtain a different normal form, which is called *concept-centered normal form* in [24], since it groups the concept descriptions w.r.t. concept names (and not w.r.t. role names, as the structural subsumption normal form does). Using this rule, any $\mathcal{FL}_0$ concept description can be transformed into an equivalent description that is a conjunction of descriptions of the form $\forall r_1. \cdots \forall r_m.A$ for $m \geq 0$ (not necessarily distinct) role names $r_1, \ldots, r_m$ and a concept name A. We abbreviate $\forall r_1. \cdots \forall r_m.A$ by $\forall r_1 \ldots r_m.A$, where $r_1 \ldots r_m$ is viewed as a word over the alphabet Σ of all role names. In addition, instead of $\forall w_1.A \sqcap \ldots \sqcap \forall w_\ell.A$ we write $\forall L.A$ where $L := \{w_1, \ldots, w_\ell\}$ is a finite set of words over Σ. The term $\forall \emptyset.A$ is considered to be equivalent to the top concept $\top$, which means that it can be added to a conjunction without changing the meaning of the concept. Using these abbreviations, any pair of $\mathcal{FL}_0$ concept descriptions C, D containing the concept names $A_1, \ldots, A_k$ can be rewritten as

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

where U_i, V_i are finite sets of words over the alphabet of all role names. This normal form provides us with the following *characterization of subsumption* of $\mathcal{FL}_0$ concept descriptions [28]:

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

Since the size of the concept-based normal forms is polynomial in the size of the original descriptions, and since the inclusion tests $U_i \supseteq V_i$ can also be realized in polynomial time, this yields a polynomial-time decision procedure for subsumption in $\mathcal{FL}_0$. In fact, as shown in [24], the structural subsumption algorithm for $\mathcal{FL}_0$ can be seen as a special implementation of these inclusion tests.

This characterization of subsumption via inclusion of finite sets of words can be extended to cyclic TBoxes with greatest fixpoint semantics as follows. A given TBox $\mathcal{T}$ can

[14] The same is true for propositionally closed DLs (see Section 6).

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

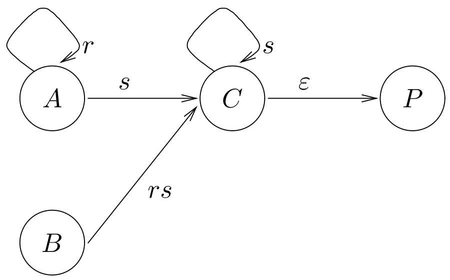

Figure 1.8. A cyclic $\mathcal{FL}_0$ TBox and the corresponding automaton.

be translated into a finite automaton[15] $\mathcal{A}_\mathcal{T}$ whose states are the concept names occurring in $\mathcal{T}$ and whose transitions are induced by the value restrictions occurring in $\mathcal{T}$ (see Fig. 1.8 for an example and [5] for the formal definition). For a defined concept A and a primitive concept P in $\mathcal{T}$, the language $L_{\mathcal{A}_\mathcal{T}}(A, P)$ is the set of all words labeling paths in $\mathcal{A}_\mathcal{T}$ from A to P. The languages $L_{\mathcal{A}_\mathcal{T}}(A, P)$ represent all the value restrictions that must be satisfied by instances of the concept A. With this intuition in mind, it should not be surprising that subsumption w.r.t. cyclic $\mathcal{FL}_0$ TBoxes can be characterized in terms of inclusion of regular languages represented by automata. Indeed, the following *characterizes subsumption w.r.t. greatest fixpoint semantics*:

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

In the example of Fig. 1.8, we have $L_{\mathcal{A}_\mathcal{T}}(A, P) = r^*ss^* \supset rss^* = L_{\mathcal{A}_\mathcal{T}}(B, P)$, and thus $A \sqsubseteq_\mathcal{T} B$, but not $B \sqsubseteq_\mathcal{T} A$.

Obviously, the languages $L_{\mathcal{A}_\mathcal{T}}(A, P)$ are regular, and any regular language can be obtained as such a language. Since inclusion of regular languages is a PSpace-complete problem [73], this shows that subsumption w.r.t. cyclic $\mathcal{FL}_0$ TBoxes with greatest fixpoint semantics is PSpace-complete [5]. The same complexity can be shown for subsumption in cyclic $\mathcal{FL}_0$ TBoxes interpreted with least fixpoint semantics or with descriptive semantics [5, 102]. In addition, the PSpace-completeness result can be extended to the DL $\mathcal{ALN}$ [106].

For an acyclic $\mathcal{FL}_0$ TBox $\mathcal{T}$, the automaton $\mathcal{A}_\mathcal{T}$ is acyclic as well. Since inclusion of languages accepted by acyclic finite automata is coNP-complete [73] and subsumption w.r.t. greatest fixpoint semantics coincides with subsumption w.r.t. descriptive semantics in the case of acyclic TBoxes, this proves Nebel's result that subsumption w.r.t. acyclic $\mathcal{FL}_0$-TBoxes is coNP-complete [128]. Thus, for $\mathcal{FL}_0$, even the presence of acyclic TBoxes increases the complexity of the subsumption problem.

Finally, ExpTime-hardness of subsumption in $\mathcal{FL}_0$ w.r.t. GCIs was shown in [10]. The ExpTime-upper bound follows from the fact that subsumption in $\mathcal{ALC}$ w.r.t. GCIs is in ExpTime [138].

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

$B \equiv P \sqcap \exists r.C,$
$C \equiv Q \sqcap \exists r.D, \quad D \equiv P \sqcap \exists r.C,$
$A \equiv P \sqcap Q \sqcap \exists r.A', \quad A' \equiv Q \sqcap P \sqcap \exists r.D.$

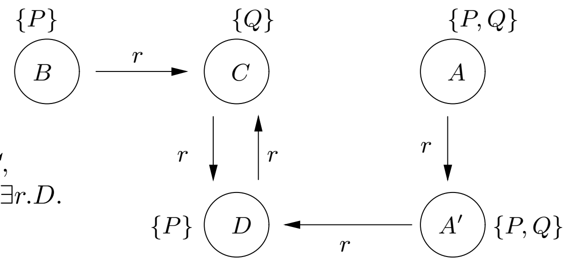

Figure 1.9. A normalized $\mathcal{EL}$ TBox and the corresponding description graph.

TBox reasoning in $\mathcal{EL}$

The approach for deciding subsumption between $\mathcal{EL}$ concept descriptions sketched in Section 4.1 can be extended to $\mathcal{EL}$ TBoxes [9]. In fact, one can show that any $\mathcal{EL}$ TBox can be transformed in polynomial time into an equivalent *normalized TBox* whose definitions are of the form

$$A \equiv P_1 \sqcap \ldots \sqcap P_m \sqcap \exists r_1.B_1 \sqcap \ldots \sqcap \exists r_\ell.B_\ell,$$

where $P_1, \ldots, P_m$ are primitive concepts, $r_1, \ldots, r_\ell$ roles, and $B_1, \ldots, B_\ell$ defined concepts. Any normalized $\mathcal{EL}$ TBox $\mathcal{T}$ can then be transformed into an $\mathcal{EL}$ *description graph* $\mathcal{G}_\mathcal{T}$ whose nodes are the defined concepts of $\mathcal{T}$. If A is a defined concept whose definition in $\mathcal{T}$ is of the normalized form shown above, then A has label $\{P_1, \ldots, P_m\}$, and is the source of the edges $(A, r_1, B_1), \ldots, (A, r_\ell, B_\ell)$ (see Figure 1.9 for an example).

Subsumption w.r.t. greatest fixpoint semantics corresponds to the existence of an appropriate simulation on $\mathcal{G}_\mathcal{T}$ [9]: if A, B are defined concepts in $\mathcal{T}$, then

$$A \sqsubseteq_{gfp,\mathcal{T}} B \quad \text{iff} \quad \text{there is a simulation } Z \text{ from } \mathcal{G}_\mathcal{T} \text{ to } \mathcal{G}_\mathcal{T} \text{ such } (B, A) \in Z.$$

Since the algorithm for computing the largest simulation sketched in Section 4.1 also works on graphs, this shows that subsumption w.r.t. $\mathcal{EL}$ TBoxes interpreted with greatest fixpoint semantics can be decided in polynomial time. In [9], the same result is also shown for descriptive and least fixpoint semantics. As a special case we have that subsumption w.r.t. acyclic $\mathcal{EL}$ TBoxes is polynomial.

In [46], it is shown that subsumption in $\mathcal{EL}$ remains polynomial even in the presence of GCIs, and in [10] this result is extended to the DL $\mathcal{EL}^{++}$, which extends $\mathcal{EL}$ by the bottom concept, nominals, a restricted form of concrete domains, and a restricted form of role-value maps.[16]

The polynomial-time subsumption algorithms for $\mathcal{EL}$ and $\mathcal{EL}^{++}$ actually classify the given set of GCIs $\mathcal{T}$, i.e., they simultaneously compute all subsumption relationships between the concept names occurring in $\mathcal{T}$. In the following, we sketch an algorithm for $\mathcal{EL}$. This algorithm proceeds in four steps:

1. Normalize the set of GCIs.

2. Translate the normalized set of GCIs into a graph.

[16] Concrete domains and role-value maps will be introduced in Section 6. Adding unrestricted concrete domains or role-value maps to $\mathcal{EL}$ with GCIs would cause undecidability of subsumption [10, 8].

3. Complete the graph using completion rules.
4. Read off the subsumption relationships from the normalized graph.

A set of $\mathcal{EL}$ GCIs is *normalized* iff it only contains GCIs of the following form:

$$A_1 \sqcap A_2 \sqsubseteq B, \quad A \sqsubseteq \exists r.B, \quad \exists r.A \sqsubseteq B,$$

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

$$\begin{array}{rl}
\exists r.A \sqcap \exists r.\exists s.A \sqsubseteq A \sqcap \exists r.\top & \leadsto \exists r.A \sqsubseteq B_1,\ B_1 \sqcap \exists r.\exists s.A \sqsubseteq A \sqcap \exists r.\top \\
& \leadsto \exists r.A \sqsubseteq B_1,\ \exists r.\exists s.A \sqsubseteq B_2,\ B_1 \sqcap B_2 \sqsubseteq A \sqcap \exists r.\top \\
& \leadsto \exists r.A \sqsubseteq B_1,\ \exists s.A \sqsubseteq B_3,\ \exists r.B_3 \sqsubseteq B_2,\ B_1 \sqcap B_2 \sqsubseteq A \sqcap \exists r.\top \\
& \leadsto \exists r.A \sqsubseteq B_1,\ \exists s.A \sqsubseteq B_3,\ \exists r.B_3 \sqsubseteq B_2,\ B_1 \sqcap B_2 \sqsubseteq A,\ B_1 \sqcap B_2 \sqsubseteq \exists r.\top
\end{array}$$

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

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

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

The label sets are supposed to satisfy the following *invariants*:

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

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

The labels of nodes and edges are then extended by applying the rules of Figure 1.10. Note that such a rule is only applied if it really extends a label set. It is easy to see that these rules preserve the above invariants. For example, consider the (most complicated) rule (R3). Obviously, $\exists r.B_1 \sqsubseteq A_1 \in \mathcal{T}$ implies $\exists r.B_1 \sqsubseteq_\mathcal{T} A_1$, and the assumption that the invariants are satisfied before applying the rule yields $B \sqsubseteq_\mathcal{T} B_1$ and $A \sqsubseteq_\mathcal{T} \exists r.B$. The subsumption relationship $B \sqsubseteq_\mathcal{T} B_1$ obviously implies $\exists r.B \sqsubseteq_\mathcal{T} \exists r.B_1$. By applying transitivity of the subsumption relation $\sqsubseteq_\mathcal{T}$, we thus obtain $A \sqsubseteq_\mathcal{T} A_1$.

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

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

Figure 1.10. The completion rules for subsumption in $\mathcal{EL}$ w.r.t. GCIs.

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

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

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

More details can be found in [46, 10].

4.3 ABox reasoning in sub-Boolean DLs

In [67], the complexity of instance checking in DLs of the $\mathcal{AL}$ family is investigated. With one exception, the complexity[17] of instance checking coincides with the complexity of subsumption. This one exception is $\mathcal{ALE}$, where the subsumption problem is NP-complete, whereas instance checking is PSPACE-complete. In the following, we sketch the PSPACE-hardness proof given in [67]. It depends on the PSPACE-hardness proof of satisfiability in $\mathcal{ALC}$ given in [143], which works by a reduction[18] from Quantified Boolean Formulae (QBF), whose validity problem is known to be PSPACE-complete [73]. A given QBF φ is translated in polynomial time into an $\mathcal{ALC}$ concept description C_φ such that φ is valid iff C_φ is satisfiable. For the purpose of sketching the PSPACE-hardness proof from [67], it is not really necessary to know what a QBF is and how the reduction concept C_φ is defined in detail. The first important observation made in [67] is that C_φ is equivalent to a concept description of the form

$$D \sqcap \neg D_1 \sqcap \ldots \sqcap \neg D_n$$

[17] To be more precise, the "combined complexity" of $\mathcal{A} \models C(a)$, i.e., w.r.t. both the size of $\mathcal{A}$ and the size of C.

[18] Note that this reduction actually differs from the one usually employed in modal logic to show PSPACE-hardness of K [81].

where $D, D_1, \ldots, D_n$ are $\mathcal{ALE}$ concept descriptions whose size is polynomially related to the size of C_φ. Thus, it remains to be shown that satisfiability of concept descriptions of this form can be reduced in polynomial time to instance checking in $\mathcal{ALE}$.

Assume that $\mathcal{ALE}$ concept descriptions $D, D_1, \ldots, D_n$ are given. Let q be a new role and E_n a new concept name. We define $\mathcal{ALE}$ concept descriptions $E_0, \ldots, E_{n-1}$ as follows:

$$E_i := \exists q.(D_{i+1} \sqcap E_{i+1}) \quad \text{for } i = 0, \ldots, n-1.$$

The ABox $\mathcal{A}$ is defined as

$$\mathcal{A} \;:=\; \{\ q(a,a),\ q(a,b),\quad D_1(a),\ \ldots,\ D_n(a),\quad E_1(b),\ \ldots,\ E_n(b),\quad D(b)\ \}$$

In [67], it is shown that

$$D \sqcap \neg D_1 \sqcap \ldots \sqcap \neg D_n \text{ is satisfiable} \quad \text{iff} \quad \mathcal{A} \not\models E_0(a).$$

In fact, assume that $\mathcal{A} \not\models E_0(a)$. Then there is a model $\mathcal{I}$ of $\mathcal{A}$ such that

$$a^\mathcal{I} \in (\neg E_0)^\mathcal{I} = (\forall q.(\neg D_1 \sqcup \neg E_1))^\mathcal{I}.$$

Thus $(a^\mathcal{I}, a^\mathcal{I}) \in q^\mathcal{I}, a^\mathcal{I} \in D_1^\mathcal{I}$ and $(a^\mathcal{I}, b^\mathcal{I}) \in q^\mathcal{I}, b^\mathcal{I} \in E_1$ imply that $a^\mathcal{I} \in (\neg E_1)^\mathcal{I}$ and $b^\mathcal{I} \in (\neg D_1)^\mathcal{I}$. We can now apply the same argument to $a^\mathcal{I} \in (\neg E_1)^\mathcal{I} = (\forall q.(\neg D_2 \sqcup \neg E_2))^\mathcal{I}$, etc. In the end, we obtain that $b^\mathcal{I} \in (\neg D_i)^\mathcal{I}$ for $i = 1, \ldots, n$, and since we also have $b^\mathcal{I} \in D^\mathcal{I}$ this shows that $D \sqcap \neg D_1 \sqcap \ldots \sqcap \neg D_n$ is satisfiable.

Conversely, it is easy to see that a model of $D \sqcap \neg D_1 \sqcap \ldots \sqcap \neg D_n$ can be used to construct a model of $\mathcal{A}$ in which a does not belong to E_0.

4.4 Bi-simulation characterizations of sub-Boolean DLs

As noted before, concept descriptions of the $\mathcal{AL}$ family (and also of many other DLs) can be translated into first-order formulae with one free variable. Thus, any such DL $\mathcal{L}$ yields a fragment $FO_\mathcal{L}$ of first-order predicate logic, which consists of those formulae with one free variable that are equivalent to the first-order translation of an $\mathcal{L}$ concept description. These fragments can be used to compare the *expressive power*[19] of DLs.

We say that a DL $\mathcal{L}_2$ is *strictly more expressive* than a DL $\mathcal{L}_1$ ($\mathcal{L}_1 \prec \mathcal{L}_2$) iff $FO_{\mathcal{L}_1} \subset FO_{\mathcal{L}_2}$, i.e., every first-order translation of an $\mathcal{L}_1$ concept description is equivalent to the first-order translation of an $\mathcal{L}_2$ concept description, and there is an $\mathcal{L}_2$ concept description whose translation is not equivalent to any translation of an $\mathcal{L}_1$ concept description.

Usually, the inclusion between two fragments $FO_{\mathcal{L}_1}$ and $FO_{\mathcal{L}_2}$ is relatively easy to show. However, how can one show that such an inclusion is strict? One way of doing this is to use an appropriate bisimulation characterization of the first-order fragments. For example, it is well-know that the fragment $FO_\mathcal{ALC}$ consists of those first-order formulae that are preserved under bisimulation (see Chapter 1). This can be used to show that $\mathcal{ALC} \prec \mathcal{ALCN}$ by giving an example of an $\mathcal{ALCN}$ concept description that is not preserved under bisimulation (see [105]).

In [105], the bisimulation characterization of the first-order fragment corresponding to $\mathcal{ALC}$ is adapted to various sub-Boolean DLs, and then used to compare their expressive power. Here, we only sketch the characterization of $FO_\mathcal{AL}$.

[19] The definition of the expressive power of DLs obtained this way is used in [105]; it is weaker than the one defined in [4] since it does not allow one to extend the vocabulary.

First, we must introduce some notation. If X, Y are subsets of a set Δ, and R is a binary relation on Δ, then we define

$$\begin{array}{lll} X\,R^{\uparrow}\,Y & \text{iff} & \text{for all } d \in X \text{ there is } e \in Y \text{ such that } (d,e) \in R, \\ X\,R^{\downarrow}\,Y & \text{iff} & \text{for all } e \in Y \text{ there is } d \in X \text{ such that } (d,e) \in R. \end{array}$$

In order to explain the intuition underlying this definition from a DL point of view, assume that r is a role, C, D are concept descriptions, and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is an interpretation, and define $R := r^{\mathcal{I}}$, $X := C^{\mathcal{I}}$, and $Y := D^{\mathcal{I}}$. Then $X\,R^{\uparrow}\,Y$ means that $C^{\mathcal{I}} \subseteq (\exists r.D)^{\mathcal{I}}$, and $X\,R^{\downarrow}\,Y$ means that $D^{\mathcal{I}} \subseteq (\exists r^{-}.C)^{\mathcal{I}}$.

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ be two interpretations. An *$\mathcal{AL}$-simulation* between $\mathcal{I}$ and $\mathcal{J}$ is a non-empty relation $Z \subseteq 2^{\Delta^{\mathcal{I}}} \times \Delta^{\mathcal{J}}$ such that the following three conditions are satisfied:

1. If $(X_1, d_2) \in Z$, then $X_1 \subseteq A^{\mathcal{I}}$ implies $d_2 \in A^{\mathcal{J}}$ and $X_1 \subseteq (\neg A)^{\mathcal{I}}$ implies $d_2 \in (\neg A)^{\mathcal{J}}$ for all concept names A.
2. For every role name r, if $(X_1, d_2) \in Z$ and $X_1\,(r^{\mathcal{I}})^{\uparrow}\,Y_1$, then there is an $e_2 \in \Delta^{\mathcal{J}}$ such that $(d_2, e_2) \in r^{\mathcal{I}}$.
3. For every role name r, if $(X_1, d_2) \in Z$ and $(d_2, e_2) \in r^{\mathcal{J}}$, then there is an $Y_1 \subseteq \Delta^{\mathcal{I}}$ such that $X_1\,(r^{\mathcal{I}})^{\downarrow}\,Y_1$ and $(Y_1, e_2) \in Z$.

Intuitively, the fact that $\mathcal{AL}$-simulations are binary relations between *subsets* of $\Delta^{\mathcal{I}}$ and $\Delta^{\mathcal{J}}$ makes sure that disjunction (which is not available in $\mathcal{AL}$) is not preserved. The first clause of the definition ensures that concept names and negated concept names (but not full negation) are preserved. The second clause ensures preservation of restricted existential restrictions $\exists r.\top$, but not of full existential restrictions $\exists r.C$ (since we do not require $(Y_1, e_2) \in Z$). The third clause ensures that value restrictions are preserved.

The first order formula $\alpha(x)$ is *preserved under $\mathcal{AL}$-simulations* iff for all interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ and all $\mathcal{AL}$-simulations Z between $\mathcal{I}$ and $\mathcal{J}$, we have:

$$(X, d_2) \in Z \text{ and } \mathcal{I} \models \alpha(d_1) \text{ for all } d_1 \in X \quad \text{implies} \quad \mathcal{I} \models \alpha(d_2).$$

In [105], it is shown that $FO_{\mathcal{AL}}$ consists of those first-order formulae that are preserved under $\mathcal{AL}$-simulations.

This result can be used to show that $\mathcal{ALU}$ is strictly more expressive than $\mathcal{AL}$. In fact, the formula $A(x) \vee B(x)$ obtained by translating the $\mathcal{ALU}$ concept description $A \sqcup B$ into first-order logic is not preserved under $\mathcal{AL}$-simulations. To see this, let $\mathcal{I}$ be the interpretation consisting of two elements d_1, e_1, where d_1 belongs to A and e_1 belongs to B, and let $\mathcal{J}$ be the interpretation consisting of the element d_2, which belongs neither to A nor to B. It is easy to see that $Z := \{(\{d_1, e_1\}, d_2)\}$ is an $\mathcal{AL}$-simulation between $\mathcal{I}$ and $\mathcal{J}$. However, $(\{d_1, e_1\}, d_2) \in Z$ and both d_1 and e_1 satisfy $A(x) \vee B(x)$, but d_2 does *not* satisfy $A(x) \vee B(x)$.

5 NON-STANDARD INFERENCES

After motivating the need for non-standard inferences in DLs and illustrating some of them by examples, we give formal definitions of the most important non-standard inferences considered until now, and review the existing results.

5.1 Motivation

All DL systems provide their users with standard inference services like computing the subsumption hierarchy, testing ABox consistency, and instance checking. These inferences are not only useful when working with "finished" knowledge bases, they can also support the knowledge engineer while building a knowledge base, by pointing out inconsistencies and unwanted consequences. They can help the knowledge engineer to check whether a concept definition makes sense, but they provide no support for actually coming up with a first version of the definition. The non-standard inferences introduced in this section can be used to overcome this deficit, basically by providing two ways of reusing "old" knowledge when defining new one: (i) constructing concepts by generalizing from examples, and (ii) constructing concepts by modifying "similar" ones.

The first approach was introduced as *bottom-up construction* of description logic knowledge bases in [20, 25]. Instead of defining the relevant concepts of an application domain from scratch, this methodology allows the user to give typical examples of individuals belonging to the concept to be defined. These individuals are then generalized to a concept by first computing the most specific concept (msc) of each individual (i.e., the least concept description w.r.t. subsumption in the available description language that has this individual as an instance), and then computing the least common subsumer (lcs) of these concepts (i.e., the least concept description w.r.t. subsumption in the available description language that subsumes all these concepts). The knowledge engineer can then use the computed concept as a starting point for the concept definition. As a simple example, assume that the knowledge engineer has already defined the concept of a man and a woman as

$$\mathsf{Man} \equiv \mathsf{Human} \sqcap \mathsf{Male} \quad \text{and} \quad \mathsf{Woman} \equiv \mathsf{Human} \sqcap \mathsf{Female},$$

and now wants to define the concept of a parent, but does not know how to do this within the available DL (which we assume to be $\mathcal{EL}$ in this example). However, the available ABox

$$\begin{array}{lll} \mathsf{Man}(\mathsf{JACK}), & \mathsf{child}(\mathsf{JACK}, \mathsf{CAROLINE}), & \mathsf{Woman}(\mathsf{CAROLINE}), \\ \mathsf{Woman}(\mathsf{JACKIE}), & \mathsf{child}(\mathsf{JACKIE}, \mathsf{JOHN}), & \mathsf{Man}(\mathsf{JOHN}), \end{array}$$

contains the individuals JACK and JACKIE, of whom the knowledge engineer knows that they are parents. The most specific concepts of JACK and JACKIE in the given ABox are

$$\mathsf{Man} \sqcap \exists \mathsf{child}.\mathsf{Woman} \quad \text{and} \quad \mathsf{Woman} \sqcap \exists \mathsf{child}.\mathsf{Man},$$

respectively, and the least common subsumer (in $\mathcal{EL}$) of these two concepts w.r.t. the definitions of Man and Woman is

$$\mathsf{Human} \sqcap \exists \mathsf{child}.\mathsf{Human},$$

which looks like a good starting point for a definition of parent.

In contrast to standard inferences like subsumption and instance checking, the output of the non-standard inferences we have mentioned until now (computing the msc and the lcs) is a concept description rather than a yes/no answer. In such a setting, it is important that the returned descriptions are as readable and comprehensible as possible. Unfortunately, the descriptions that are produced by the known algorithms for computing the lcs and the msc do not satisfy this requirement. The reason is that – like most algorithms for the standard inference problems – these algorithms work on expanded concept descriptions, i.e., concept descriptions that do not contain names defined in the underlying TBox. Consequently, the descriptions that the algorithms produce also do not use defined concepts, which makes them in many cases large and hard to read and comprehend.[20] This problem can be overcome by *rewriting* the resulting concept w.r.t. the given TBox. Informally, the problem of rewriting a concept given a terminology can be stated as follows: given an acyclic TBox $\mathcal{T}$ and a concept description C that does not contain concept names defined in $\mathcal{T}$, can this description be rewritten into an equivalent shorter description E by using (some of) the names defined in $\mathcal{T}$? For example, w.r.t. the TBox in Figure 1.3, the concept description

$$\mathsf{Person} \sqcap \forall \mathsf{child}.\mathsf{Female} \sqcap \exists \mathsf{child}.\top \sqcap \forall \mathsf{child}.\mathsf{Person}$$

can be rewritten to the equivalent concept $\mathsf{Parent} \sqcap \forall \mathsf{child}.\mathsf{Woman}$.

Rewriting w.r.t. a TBox is just one instance of a more general rewriting framework, which will be introduced below. Another instance of this framework is *approximation*, where one tries to express a concept description C_1 defined in one DL $\mathcal{L}_1$ by a concept description C_2 expressed in another DL $\mathcal{L}_2$. If $\mathcal{L}_1$ is strictly more expressive than $\mathcal{L}_2$, then it is not always possible to find a concept description C_2 that is equivalent to C_1. In this case, one can try to approximate C_1 by an $\mathcal{L}_2$ concept description that is as "close" as possible to C_1, for example by trying to find an $\mathcal{L}_2$ concept description that subsumes C_1 and is minimal w.r.t. subsumption. One possible application for such an inference is translating knowledge bases from the language employed by one system into the language employed by another system.

In order to apply the second approach of constructing concepts by modifying existing ones, one must first find the right candidates for modification. One way of doing this is to give a partial description of the concept to be defined as a concept pattern (i.e., a concept description containing variables standing for concept descriptions), and then look for concept descriptions that match this pattern. For example, the pattern

$$\mathsf{Man} \sqcap \exists \mathsf{child}.(\mathsf{Man} \sqcap X) \sqcap \exists \mathsf{spouse}.(\mathsf{Woman} \sqcap X)$$

looks for descriptions of classes of men whose wives and sons share some characteristic. An example of a concept description *matching* this pattern is $\mathsf{Man} \sqcap \exists \mathsf{child}.(\mathsf{Man} \sqcap \mathsf{Tall}) \sqcap \exists \mathsf{spouse}.(\mathsf{Woman} \sqcap \mathsf{Tall})$.

Unification is a generalization of matching where both concepts may contain variables. The main motivation for introducing unification in DLs was to avoid redundancies in knowledge bases that are built by several knowledge engineers over a long time period. In this setting, it frequently happens that the same (intuitive) concept is introduced several times, often with slightly differing descriptions. Testing for equivalence of concepts is not always sufficient to find out whether, for a given concept description, there already exists another concept description in the knowledge base describing the same notion. As an example, lets us ask whether the following two $\mathcal{FL}_0$ concept descriptions might denote

[20]In the above example, this means that the definitions of Man and Woman are expanded before applying the lcs algorithm. If Human also had a definition, then it would also be expanded, and instead of the concept description containing Human shown above, the algorithm would return its expanded version.

the same (intuitive) concept:

$$\forall\mathsf{child}.\forall\mathsf{child}.\mathsf{Rich} \sqcap \forall\mathsf{child}.\mathsf{Rmr} \quad \text{and} \quad \mathsf{Acr} \sqcap \forall\mathsf{child}.\mathsf{Acr} \sqcap \forall\mathsf{child}.\forall\mathsf{spouse}.\mathsf{Rich}.$$

The answer is yes, since replacing the concept name Rmr by the description $\mathsf{Rich} \sqcap \forall\mathsf{spouse}.\mathsf{Rich}$ and Acr by $\forall\mathsf{child}.\mathsf{Rich}$ yields the descriptions

$$\begin{array}{l} \forall\mathsf{child}.\forall\mathsf{child}.\mathsf{Rich} \sqcap \forall\mathsf{child}.(\mathsf{Rich} \sqcap \forall\mathsf{spouse}.\mathsf{Rich}), \\ \forall\mathsf{child}.\mathsf{Rich} \sqcap \forall\mathsf{child}.\forall\mathsf{child}.\mathsf{Rich} \sqcap \forall\mathsf{child}.\forall\mathsf{spouse}.\mathsf{Rich}, \end{array}$$

which are obviously equivalent. Thus, under the assumption that Rmr stands for "Rich and married rich" and Acr for "All children are rich", we can conclude that both descriptions are meant to express the concept "All grandchildren are rich and all children are rich and married rich". This connection between the two description can be found by a unification algorithm if we declare Rmr and Acr to be variables. Of course, unifiability does not necessarily mean that the concept descriptions are meant to represent the same concept. Unifiability only suggests that there is a possible connection: the final decision must be taken by the knowledge engineer.

5.2 Least common subsumers and most specific concepts

Intuitively, the least common subsumer of a given collection of concept descriptions is a description that represents the properties that all the elements of the collection have in common. More formally, it is the most specific concept description that subsumes the given descriptions. What this most specific description looks like, whether it really captures the intuition of representing the properties common to the input descriptions, and whether it exists at all strongly depends on the DL under consideration.

Let $\mathcal{L}$ be a DL. A concept description E of $\mathcal{L}$ is a *least common subsumer* (lcs) of the concept descriptions $C_1, \ldots, C_n$ in $\mathcal{L}$ ($lcs_{\mathcal{L}}(C_1, \ldots, C_n)$ for short) iff it satisfies

1. $C_i \sqsubseteq E$ for all $i = 1, \ldots, n$, and
2. E is the least $\mathcal{L}$ concept description with this property, i.e., if E' is an $\mathcal{L}$ concept description satisfying $C_i \sqsubseteq E'$ for all $i = 1, \ldots, n$, then $E \sqsubseteq E'$.

As an easy consequence of this definition, the lcs is unique up to equivalence, which justifies talking about *the* lcs. In addition, the n-ary lcs as defined above can be reduced to the binary lcs (the case where $n = 2$). Indeed, it is easy to see that $lcs_{\mathcal{L}}(C_1, \ldots, C_n) \equiv lcs_{\mathcal{L}}(C_1, \ldots, lcs_{\mathcal{L}}(C_{n-1}, C_n) \cdots)$. Thus, it is enough to devise algorithms for computing the binary lcs.

It should be noted, however, that the lcs need not always exist. This can have different reasons: (a) there may not exist a concept description in $\mathcal{L}$ satisfying (i) of the definition (i.e., subsuming $C_1, \ldots, C_n$); (b) there may be several subsumption incomparable minimal concept descriptions satisfying (i) of the definition; (c) there may be an infinite chain of more and more specific descriptions satisfying (i) of the definition. Obviously, (a) cannot occur for DLs containing the top concept. It is easy to see that, for DLs allowing for conjunction of descriptions, (b) cannot occur. An example for a DL exhibiting behavior (c) can be found in [6], where the lcs is defined w.r.t. a cyclic TBox.

It is also clear that in DLs allowing for disjunction, the lcs of $C_1, \ldots, C_n$ is their disjunction $C_1 \sqcup \ldots \sqcup C_n$. In this case, the lcs is not really of interest. Instead of

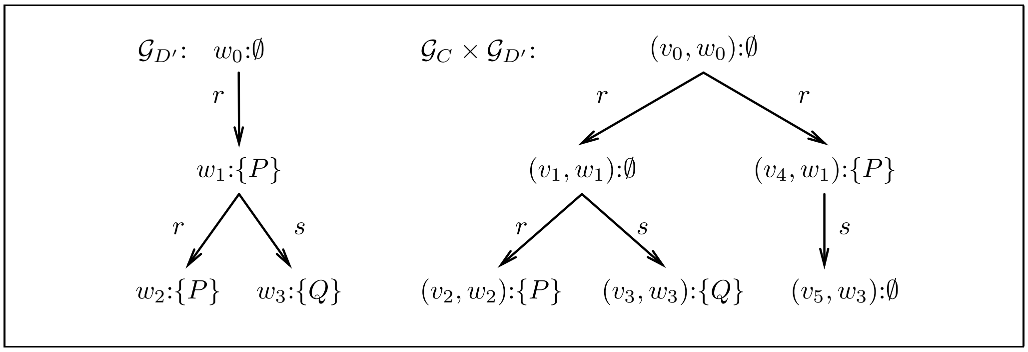

Figure 1.11. The product of $\mathcal{EL}$ description trees.

extracting properties common to $C_1, \ldots, C_n$, it just gives their disjunction, which does not provide us with new information. Thus, it only makes sense to look at the lcs in sub-Boolean DLs.

For DLs whose expressive power lies between $\mathcal{FL}_0$ and $\mathcal{ALN}$, one can use the characterization of subsumption via finite languages over the alphabet of the role names (see Subsection 4.2) to compute the lcs. Recall that any pair of $\mathcal{FL}_0$ concept descriptions C, D containing the concept names $A_1, \ldots, A_k$ can be written as

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

where U_i, V_i are finite sets of words over the alphabet of all role names, and that $C \sqsubseteq D$ iff $U_i \supseteq V_i$ for $i = 1, \ldots, k$. As an easy consequence of this characterization we obtain that the lcs E of C, D is of the form

$$E \equiv \forall(U_1 \cap V_1).A_1 \sqcap \ldots \sqcap \forall(U_k \cap V_k).A_k.$$

Using the language-based characterization of subsumption in $\mathcal{ALN}$ [106], this approach for computing the lcs by language intersection can be extended to $\mathcal{ALN}$ [20], but this involves the use of certain infinite regular languages.

For DLs with existential restrictions, the characterization of subsumption via the existence of certain simulation relations between description trees (see Subsection 4.2) implies that the lcs corresponds to the product of the description trees [25]. The *product* $\mathcal{G}_C \times \mathcal{G}_D$ of two $\mathcal{EL}$ description trees $\mathcal{G}_C$ and $\mathcal{G}_D$ is defined by induction on the depth of the trees. Its root is the pair (v_0, w_0) consisting of the roots of $\mathcal{G}_C$ and $\mathcal{G}_D$, and the label of (v_0, w_0) is the intersection of the labels of v_0 and w_0. For each r-successor v of v_0 in $\mathcal{G}_C$ and w of w_0 in $\mathcal{G}_D$, we obtain an r-successor (v, w) of (v_0, w_0) in $\mathcal{G}_C \times \mathcal{G}_D$ that is the root of the product of the subtree of $\mathcal{G}_C$ with root v and the subtree of $\mathcal{G}_D$ with root w.

As an example, the product of the description tree $\mathcal{G}_C$ shown in Figure 1.7 and the description tree $\mathcal{G}_{D'}$ shown in Figure 1.11 is depicted on the right-hand side of Figure 1.11. Thus, the lcs in $\mathcal{EL}$ of the concept descriptions

$$C := P \sqcap \exists r.(\exists r.(P \sqcap Q) \sqcap \exists s.Q) \sqcap \exists r.(P \sqcap \exists s.P) \quad \text{and} \quad D' := \exists r.(P \sqcap \exists r.P \sqcap \exists s.Q)$$

is $lcs_{\mathcal{EL}}(C, D') \equiv \exists r.(\exists r.P \sqcap \exists s.Q) \sqcap \exists r.(P \sqcap \exists s.\top)$.

This approach of computing the lcs as a product of description trees can be extended to $\mathcal{ALE}$ [25] and to $\mathcal{ALEN}$ [108]. The main difference is that the concept descriptions must be normalized appropriately before building the description trees.

Now, we come to the formal definition of the most specific concept. Let $\mathcal{L}$ be a DL. The $\mathcal{L}$ concept description E is the *most specific concept* (msc) in $\mathcal{L}$ of the individual a in the $\mathcal{L}$ ABox $\mathcal{A}$ ($msc_{\mathcal{L}}(a)$ for short) iff

1. $\mathcal{A} \models E(a)$, and
2. E is the least concept satisfying (i), i.e., if E' is an $\mathcal{L}$ concept description satisfying $\mathcal{A} \models E'(a)$ then $E \sqsubseteq E'$.

As with the lcs, the msc is unique up to equivalence, if it exists. In contrast to the lcs, which usually exists for standard DLs, the msc does not always exist in $\mathcal{EL}$, $\mathcal{ALN}$, and $\mathcal{ALE}$. This is due to the presence of so-called role cycles in the ABox. For example, w.r.t. the ABox

$$\{\mathsf{loves}(\mathsf{NARCIS}, \mathsf{NARCIS}),\ \mathsf{Vain}(\mathsf{NARCIS})\},$$

the individual NARCIS does not have an msc in $\mathcal{EL}$. In fact, assume that E is the msc of NARCIS. Then E has a finite role depth, i.e., a finite maximal number of nestings of existential restrictions. If this role depth is smaller than n, then E is not subsumed by the $\mathcal{EL}$ concept description

$$E' := \underbrace{\exists\mathsf{loves}.\cdots\exists\mathsf{loves}.}_{n \text{ times}}\mathsf{Vain},$$

in spite of the fact that NARCIS is an instance of E'. The same example works for $\mathcal{ALE}$, and a similar one can be given for $\mathcal{ALN}$ [20].

One way to overcome this problem is to allow for cyclic TBoxes interpreted with greatest fixpoint semantics. In the above example, the defined concept $\mathsf{Narcis} \equiv \mathsf{Vain} \sqcap \exists\mathsf{loves}.\mathsf{Narcis}$ is then an msc of the individual NARCIS. In order to employ this approach in the bottom-up construction of DL knowledge bases, one must allow these knowledge bases to contain cyclic definitions. Thus, also the subsumption problem and the problem of computing the lcs must be solved w.r.t. cyclic definitions interpreted with greatest fixpoint semantics. In [106, 20], this is done for $\mathcal{ALN}$, and in [9, 7] for $\mathcal{EL}$. The appropriate treatment of cyclic TBoxes in $\mathcal{ALE}$ is still an open problem.

Another possibility is to approximate the msc by restricting the attention to concept descriptions whose role depth is bounded by a fixed number k [53, 107].

5.3 Matching and unification

Concept patterns are concept descriptions in which *concept variables* (usually denoted by X, Y, etc.) may occur in place of concept names. The main difference between concept names and concept variables is that the latter can be replaced by concept descriptions when applying a substitution.

For example, $D := P \sqcap X \sqcap \forall r.(Y \sqcap \forall r.X)$ is a concept pattern containing the concept variables X and Y. By applying the substitution $\sigma := \{X \mapsto Q,\ Y \mapsto \forall r.P\}$ to it, we obtain the concept description

$$\sigma(D) = P \sqcap Q \sqcap \forall r.(\forall r.P \sqcap \forall r.Q).$$

Let $\mathcal{L}$ be a DL. An $\mathcal{L}$ *unification problem* is of the form

$$C_1 \equiv^? D_1, \ \ldots, \ C_n \equiv^? D_n,$$

where $C_1, \ldots, D_n$ are $\mathcal{L}$ concept patterns. A *unifier* of this problem is a substitution σ such that $\sigma(C_i) \equiv \sigma(D_i)$ for $i = 1, \ldots, n$.

For unification, the only results available until now are for the small DL $\mathcal{FL}_0$ and its extension $\mathcal{FL}_{reg}$ by the role constructors union, composition, and reflexive-transitive closure. In [28], it is shown that deciding unifiability of $\mathcal{FL}_0$-patterns is an ExpTime-complete problem, and in [22] this result is extended to $\mathcal{FL}_{reg}$ and it's extension with $\perp$. In the following, we sketch how the results for unification in $\mathcal{FL}_0$ can be obtained. As shown in [28], we can without loss of generality restrict the attention to unification problems consisting of a single equation $C \equiv^? D$. Using the language-based normal form of $\mathcal{FL}_0$ concept descriptions, we can write the patterns C, D in the form

$$\begin{aligned} C &\equiv \forall S_{0,1}.A_1 \sqcap \ldots \sqcap \forall S_{0,k}.A_k \sqcap \forall S_1.X_1 \sqcap \ldots \sqcap \forall S_n.X_n, \\ D &\equiv \forall T_{0,1}.A_1 \sqcap \ldots \sqcap \forall T_{0,k}.A_k \sqcap \forall T_1.X_1 \sqcap \ldots \sqcap \forall T_n.X_n, \end{aligned}$$

where $A_1, \ldots, A_k$ are the concept names and $X_1, \ldots, X_n$ the concept variables occurring in C, D, and $S_{0,i}, S_j, T_{0,i}, T_j$ $(i = 1, \ldots, k, j = 1, \ldots, n)$ are finite sets of words over the alphabet of all role names. In [28], it is shown that $C \equiv^? D$ has a unifier iff for all $i = 1, \ldots, k$, the *linear language equation*

$$S_{0,i} \cup S_1 X_{1,i} \cup \cdots \cup S_n X_{n,i} = T_{0,i} \cup T_1 X_{1,i} \cup \cdots \cup T_n X_{n,i}$$

has a solution, i.e., we can substitute the variables $X_{j,i}$ by finite languages such that the equation holds. Note that this is not a system of k equations that must be solved simultaneously: since they do not share variables, each of these equations can be solved separately.

Let us illustrate the connection between $\mathcal{FL}_0$ unification problems and linear language equations by a simple example. The normal forms of the concept patterns

$$C := \forall r.(A_1 \sqcap \forall r.A_2) \sqcap \forall r.\forall s.X_1 \quad \text{and} \quad D := \forall r.\forall s.(\forall s.A_1 \sqcap \forall r.A_2) \sqcap \forall r.X_1 \sqcap \forall r.\forall r.A_2$$

are

$$C \equiv \forall\{r\}.A_1 \sqcap \forall\{rr\}.A_2 \sqcap \forall\{rs\}.X_1 \quad \text{and} \quad D \equiv \forall\{rss\}.A_1 \sqcap \forall\{rsr, rr\}.A_2 \sqcap \forall\{r\}.X_1.$$

Thus, the unification problem $C \equiv^? D$ leads to the two linear language equations

$$\begin{aligned} \{r\} \cup \{rs\}X_{1,1} &= \{rss\} \cup \{r\}X_{1,1}, \\ \{rr\} \cup \{rs\}X_{1,2} &= \{rsr, rr\} \cup \{r\}X_{1,2}. \end{aligned}$$

The first equation (the one for A_1) has $X_{1,1} = \{\varepsilon, s\}$ as a solution, and the second (the one for A_2) has $X_{1,2} = \{r\}$ as a solution. These two solutions yield the following unifier of $C \equiv^? D$:

$$\{X_1 \mapsto A_1 \sqcap \forall s.A_1 \sqcap \forall r.A_2\}.$$

By an exponential time reduction to the emptiness problem of top-down automata on finite trees it is shown in [28] that solvability of linear language equations of the form introduced above can be decided in exponential time. ExpTime-hardness is shown by a reduction from the intersection emptiness problem for deterministic top-down tree automata. This shows that solvability of $\mathcal{FL}_0$ unification problems is an ExpTime-complete problem. In [22], these results are extended to $\mathcal{FL}_{reg}$. Basically, instead of linear language equations over finite sets, one obtains linear language equations over regular sets, and uses automata working on infinite trees to solve them.

An extension of these results to more expressive DLs, such as $\mathcal{ALC}$, appears to be very hard. This is supported by the fact that research on unification in modal logics has also not yet produced results on unification in K. In modal logic, unification can be seen as a special case of testing for the admissibility of an inference rule (see Chapter 8 for more details and references). The rule

$$\frac{\alpha_1(x_1,\ldots,x_n),\ldots,\alpha_m(x_1,\ldots,x_n)}{\beta(x_1,\ldots,x_n)}$$

is called *admissible* in a logic $\mathcal{L}$ iff every substitution of the x_i by formulae making $\alpha_1,\ldots,\alpha_m$ valid also makes β valid. If $\mathcal{L}$ is consistent, then the rule $\alpha(x_1,\ldots,x_n)/\bot$ is admissible iff $\alpha(x_1,\ldots,x_n) \equiv^? \top$ is not unifiable. If the logic is propositionally closed, then all unification problems can be brought into this form. Consequently, decidability of the admissible inference rule problem (e.g., in $\mathsf{K4}$, Grz) implies decidability of the unification problem.

Kracht also shows in Chapter 8 that admissibility of inference rules can be reduced to unification if every unification problem has a computable finite complete set of unifiers (see [32] for the relevant definitions from unification theory). Using Ghilardi's results that unification in $\mathsf{K4}$, $\mathsf{S4}$, and intuitionistic logic is finitary in this sense [76, 75], this shows that admissibility of inference rules is decidable for intuitionistic logic. It should be noted however, that these result consider only elementary unification, i.e., unification without free constants. In the DL setting introduced above, this means that they do not allow for concept names in concept descriptions (only concept variables). Also note that unification in $\mathcal{FL}_0$ is not finitary [2]. For the modal logic K, decidability of both admissibility of inference rules and unification are open problems (and generally assumed to be very hard).

Matching can be seen as a special case of unification where the left-hand sides of the unification problem are concept descriptions, i.e., the concept descriptions C_i in $C_i \equiv^? D_i$ do not contain variables. For DLs that are propositionally closed, unification can be reduced to matching. Indeed, it is easy to see that the equation $C \equiv^? D$ has the same solutions as $\top \equiv^? (C \sqcap D) \sqcup (\neg C \sqcap \neg D)$. Thus, for $\mathcal{ALC}$, matching is as hard as unification. For sub-Boolean DLs, matching can be significantly easier than unification (see below).

In [39], a different notion of matching, called *matching modulo subsumption*, was introduced.[21] In this setting, a matching problem is of the form $C \sqsubseteq^? D$ where C is a concept description and D a concept pattern. A *matcher* is then a substitution σ such that $C \sqsubseteq \sigma(D)$. Since $C \sqsubseteq \sigma(D)$ iff $C \sqcap \sigma(D) \equiv C$, and $C \sqcap \sigma(D) = \sigma(C) \sqcap \sigma(D) = \sigma(C \sqcap D)$, this matching problem modulo subsumption can be reduced to the following matching problem modulo equivalence: $C \equiv^? C \sqcap D$.

However, in many cases, matching modulo subsumption is simpler than matching modulo equivalence since it can be reduced to the subsumption problem. This is the case

[21]In the following, we call matching problems of the form $C \equiv^? D$ matching problems *modulo equivalence* to distinguish them from matching problems modulo subsumption.

for DLs that allow for $\top$ and where all constructors are *monotonic*, i.e., replacing their arguments by larger ones w.r.t. subsumption yields a larger description. An example of a monotonic constructor is conjunction: if $C \sqsubseteq C'$ and $D \sqsubseteq D'$, then $C \sqcap D \sqsubseteq C' \sqcap D'$. Other examples are value and existential restrictions as well as disjunction. For such a DL, $C \sqsubseteq^? D$ has a matcher iff the substitution $\sigma_\top$ that replaces all variables by $\top$ is a matcher, i.e., if $C \sqsubseteq \sigma_\top(D)$. In fact, monotonicity of the constructors implies that $\sigma(D) \sqsubseteq \sigma_\top(D)$ holds for all substitutions σ, and thus whenever there is a matcher σ, then $\sigma_\top$ is also a matcher. Also note that matching cannot be simpler than subsumption since the matching problem $C \sqsubseteq^? D$ where D does not contain variables has a solution iff $C \sqsubseteq D$.

In the context of matching modulo subsumption, one is, however, usually not interested in arbitrary solution, and in particular not in the trivial largest one $\sigma_\top$, but rather in *minimal* ones, i.e., in matchers σ of $C \sqsubseteq^? D$ such that there does not exist another substitution δ such that $C \sqsubseteq \delta(D) \sqsubset \sigma(D)$ (see [39] for a motivation). Computing minimal matcher may again be harder than simply testing whether the trivial solution candidate $\sigma_\top$ is indeed a matcher.

In [28], the language-based approach for unification is used to show that solvability of matching problems modulo equivalence (and thus also modulo subsumption) in $\mathcal{FL}_0$ can be decided in polynomial time, and that minimal matchers of matching problems modulo subsumption are unique up to equivalence and can be computed in polynomial time. In [23], this result is extended to $\mathcal{ALN}$.[22] Matching in $\mathcal{EL}$ and $\mathcal{ALE}$ is considered in [21]. For both DLs, matching modulo equivalence is NP-complete. As explained above, the complexity of matching modulo subsumption coincides with the complexity of the subsumption problem, i.e., it is polynomial for $\mathcal{EL}$ and NP-complete for $\mathcal{ALE}$. In the following, we consider the complexity of matching modulo equivalence in $\mathcal{EL}$ and $\mathcal{ALE}$ in some more detail. NP-hardness of matching in $\mathcal{ALE}$ is an immediate consequence of NP-hardness of subsumption in $\mathcal{ALE}$.

NP*-hardness of matching modulo equivalence in* $\mathcal{EL}$ is shown in [21] by a reduction from SAT. Let $\phi = \varphi_1 \wedge \cdots \wedge \varphi_m$ be a propositional formula in conjunctive normal form and let $\{p_1, \ldots, p_n\}$ be the propositional variables of this problem. For these variables, we introduce the concept variables $\{X_1, \ldots, X_n, \overline{X}_1, \ldots, \overline{X}_n\}$. Furthermore, we need concept names A and B as well as role names $r_1, \ldots, r_n$ and $s_1, \ldots, s_m$. First, we specify a matching problem $C_n \equiv^? D_n$ that encodes the truth values of the n propositional variables:

$$\begin{array}{rcl} C_n & := & \exists r_1.A \sqcap \exists r_1.B \sqcap \ldots \sqcap \exists r_n.A \sqcap \exists r_n.B \\ D_n & := & \exists r_1.X_1 \sqcap \exists r_1.\overline{X}_1 \sqcap \ldots \sqcap \exists r_n.X_n \sqcap \exists r_n.\overline{X}_n. \end{array}$$

The matchers of this problem are exactly the substitutions that replace X_i by A and $\overline{X}_i$ by B (corresponding to $p_i = \mathsf{true}$), or vice versa (corresponding to $p_i = \mathsf{false}$).

In order to encode ϕ, we introduce a concept pattern D_{φ_i} for each clause φ_i. For example, if $\varphi_i = p_1 \vee \neg p_2 \vee p_3 \vee \neg p_4$, then $D_{\varphi_i} := X_1 \sqcap \overline{X}_2 \sqcap X_3 \sqcap \overline{X}_4 \sqcap B$. The whole formula is then represented by the matching problem $C_\phi \equiv^? D_\phi$, where

$$C_\phi := \exists s_1.(A \sqcap B) \sqcap \ldots \sqcap \exists s_m.(A \sqcap B) \quad \text{and} \quad D_\phi := \exists s_1.D_{\varphi_1} \sqcap \ldots \sqcap \exists s_m.D_{\varphi_m}.$$

This matching problem ensures that, among all the variables in D_{φ_i}, at least one must be replaced by A. This corresponds to the fact that, within one clause φ_i, there must

[22] In the presence of atomic negation one defines patterns such atomic negation may not be applied to variables, and thus atomic negation does not destroy the monotonicity property introduced above.

$$C := \mathsf{W} \sqcap \exists \mathsf{c}.(\mathsf{W} \sqcap \exists \mathsf{c}.(\mathsf{W} \sqcap \mathsf{D}) \sqcap \exists \mathsf{c}.(\mathsf{W} \sqcap \mathsf{P})) \sqcap \exists \mathsf{c}.(\mathsf{W} \sqcap \mathsf{D} \sqcap \exists \mathsf{c}.(\mathsf{W} \sqcap \mathsf{P}))$$
$$D := \mathsf{W} \sqcap \exists \mathsf{c}.(X \sqcap \exists \mathsf{c}.(\mathsf{W} \sqcap Y)) \sqcap \exists \mathsf{c}.(X \sqcap Y)$$

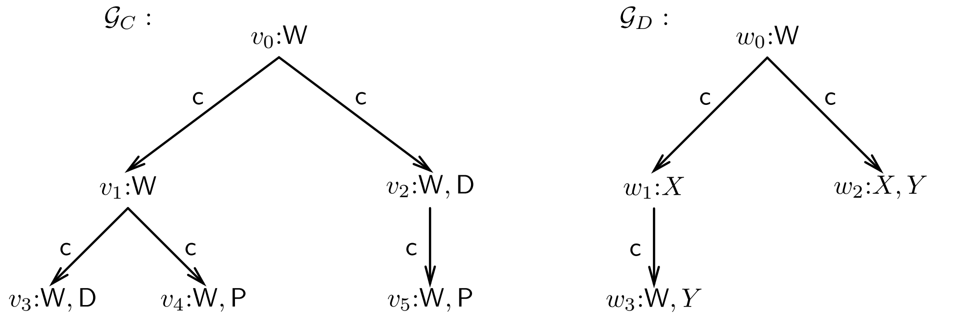

Figure 1.12. An $\mathcal{EL}$ concept description and an $\mathcal{EL}$ concept pattern, and the corresponding description trees.

be at least one literal that evaluates to true. Note that we need the concept B in D_{φ_i} to cover the case where all variables in D_{φ_i} are substituted with A. If we combine the two matching problems introduced above into a single problem $C_n \sqcap C_\phi \equiv^? D_n \sqcap D_\phi$, then it is easy to verify that ϕ is satisfiable iff this matching problem is solvable.

Membership in NP *for matching modulo equivalence in $\mathcal{EL}$ and $\mathcal{ALE}$* is an easy consequence of the following two (non-trivial) facts [21]. If an $\mathcal{EL}$ or $\mathcal{ALE}$ matching problem modulo equivalence has a matcher, then it has one of size polynomially bounded by the size of the problem. Furthermore, this matcher uses only concept and role names already contained in the matching problem. Thus, one can simply guess a substitution satisfying the given size bound, and then test (in P for $\mathcal{EL}$ and in NP for $\mathcal{ALE}$) whether it is a matcher.

Of course, this NP-algorithm for testing solvability of a matching problem does not yield a practical algorithm for actually computing matchers. A more practical *algorithm that computes all minimal matchers* of $\mathcal{EL}$ and $\mathcal{ALE}$ matching problems *modulo subsumption* is based on the characterization of subsumption through the existence of a homomorphism (i.e., a simulation relation that is a function) between the corresponding description trees [25]. As an example, consider the $\mathcal{EL}$ matching problem $C \sqsubseteq^? D$ for the concept description C and the concept pattern D depicted in Figure 1.12. Readers not liking such abstract examples may read W as Woman, D as Doctor, P as Professor, and c as child. Thus, the pattern describes concepts consisting of women that have (i) a child satisfying some property X and having a female child satisfying some property Y, and (ii) a child satisfying both X and Y.

When considering homomorphisms between the description trees of a concept pattern and a concept description, we simply ignore the concept variables, i.e., the inclusion condition between the labels does not take variables into account. In our example, there are six homomorphisms from $\mathcal{G}_D$ into $\mathcal{G}_C$. We consider the ones mapping w_i onto v_i for $i = 0, 1, 2$, and w_3 onto v_3 or w_3 onto v_4, which we denote by h_1 and h_2, respectively.

The matching algorithm described in [21] tries to construct substitutions τ such that $C \sqsubseteq \tau(D)$, i.e., there is a homomorphism from $\mathcal{G}_{\tau(D)}$ into $\mathcal{G}_C$. This is achieved by first computing all homomorphisms from $\mathcal{G}_D$ into $\mathcal{G}_C$. Assume that the node w in $\mathcal{G}_D$, whose label contains X, is mapped onto the note v of $\mathcal{G}_C$. The idea is then to substitute X with the concept description corresponding to the subtree of $\mathcal{G}_C$ starting with the node v. We will denote this description by C_v in the following. The remaining problem is that a variable X may occur more than once in D, and thus nodes containing X may be mapped to several nodes in $\mathcal{G}_C$. Thus, we cannot simply define $\tau(X)$ as $C_{h(w)}$ where w is such that X occurs in the label of w. Since there may exist several nodes w with this property, we take the least common subsumer of the corresponding parts of C. The reason for taking the *least* common subsumer is that we want to compute minimal matchers.

In our example, the homomorphism h_1 yields the substitution τ_1:

$$\tau_1(X) := lcs(C_{v_1}, C_{v_2}) \equiv \mathsf{W} \sqcap \exists \mathsf{c}.(\mathsf{W} \sqcap \mathsf{P}), \quad \tau_1(Y) := lcs(C_{v_2}, C_{v_3}) \equiv \mathsf{W} \sqcap \mathsf{D},$$

whereas h_2 yields the substitution τ_2:

$$\tau_2(X) := lcs(C_{v_1}, C_{v_2}) \equiv \mathsf{W} \sqcap \exists \mathsf{c}.(\mathsf{W} \sqcap \mathsf{P}), \quad \tau_2(Y) := lcs(C_{v_2}, C_{v_4}) \equiv \mathsf{W}.$$

The algorithm is guaranteed to compute all minimal matchers, but may also compute some non-minimal ones, which must be removed in a post-processing step. In our example, the substitution τ_1 is a minimal matcher, but τ_2 is not minimal. In general, a given matching problem modulo subsumption may have exponentially many inequivalent minimal matchers, and the size of these minimal matchers may also be exponential in the size of the matching problem [21].

5.4 Rewriting and approximation

In [26], a very general framework for rewriting in DLs is introduced, which has several interesting instances. In order to introduce this framework, we fix a set N_R of role names and a set N_P of primitive concept names. Now, let $\mathcal{L}_s$, $\mathcal{L}_d$, and $\mathcal{L}_t$ be three DLs (the source-, destination, and TBox-DL, respectively). A *rewriting problem* is given by

- an $\mathcal{L}_t$ TBox $\mathcal{T}$ containing only role names from N_R and primitive concepts from N_P; the set of defined concepts occurring in $\mathcal{T}$ is denoted by N_D;
- an $\mathcal{L}_s$ concept description C using only the names from N_R and N_P;
- a binary relation ρ between $\mathcal{L}_s$ concept descriptions and $\mathcal{L}_d$ concept descriptions.

An $\mathcal{L}_d$ *rewriting of* C *using* $\mathcal{T}$ is an $\mathcal{L}_d$ concept description E built using role names from N_R and concept names from $N_P \cup N_D$ such that $C \rho E$. Given an appropriate ordering $\preceq$ on $\mathcal{L}_d$ concept descriptions, a rewriting E is called $\preceq$-*minimal* iff there does not exist a rewriting E' such that $E' \prec E$.

To illustrate the use of this general framework by examples, we consider two of its instances in more detail: the *minimal rewriting problem* and the *approximation problem*.

Minimal rewriting

This is the instance of the framework where (i) all three DLs are the same language $\mathcal{L}$; (ii) the TBox $\mathcal{T}$ is acyclic; (iii) the binary relation ρ corresponds to equivalence w.r.t. the TBox; and (iv) $\mathcal{L}$ concept descriptions are ordered by size, i.e., $E \preceq E'$ iff $|E| \leq |E'|$. The size $|E|$ of a concept description E is defined to be the number of occurrences of concept and role names in E.

In order to determine the complexity of the minimal rewriting problem, Baader et al. [26] first analyse the *decision problem* induced by this optimization problem for a given DL $\mathcal{L}$: given an $\mathcal{L}$ concept description C, an acyclic $\mathcal{L}$ TBox $\mathcal{T}$, and a nonnegative integer κ, does there exist an $\mathcal{L}$ rewriting E of C using $\mathcal{T}$ such that $|E| \leq \kappa$? Since this decision problem can obviously be reduced to the problem of computing a minimal rewriting of C using $\mathcal{T}$, hardness results for the decision problem carry over to the optimization problem.

For $\mathcal{ALC}$, this decision problem is PSpace-hard since the PSpace-complete subsumption problem can be reduced to it. Indeed, let C, D be two $\mathcal{ALC}$ concept descriptions, and A, P_1, P_2 three different concept names not occurring in C, D. Then $C \sqsubseteq D$ iff there exists a minimal rewriting of size 1 of the $\mathcal{ALC}$ concept description $P_1 \sqcap P_2 \sqcap C$ using the TBox $\mathcal{T} := \{A \equiv P_1 \sqcap P_2 \sqcap C \sqcap D\}$ [26]. The two concept names P_1 and P_2 are introduced to ensure that the size of the concept description to be rewritten is strictly larger than the size of A.

However, subsumption is not the only source of complexity for the minimal rewriting problem. In fact, even for the small DL $\mathcal{FL}_0$, for which subsumption of concept descriptions and w.r.t. expanded TBoxes is polynomial, the rewriting problem (using an expanded TBox) is NP-hard. This is shown in [26] by a reduction from the set cover problem.

For an arbitrary DL $\mathcal{L}$, the minimal rewriting decision problem can obviously be decided by a non-deterministic polynomial time algorithm that uses an oracle for subsumption. This algorithm just guesses an $\mathcal{L}$ concept description over the available vocabulary and of size at most κ, and then checks whether this description is equivalent to the input description modulo the TBox. For $\mathcal{ALC}$, this shows that the minimal rewriting decision problem is PSpace-complete. It can also be used to show that the problem is NP-complete for $\mathcal{FL}_0$ (see [26] for details).

Let us now come to the problem of actually *computing minimal rewritings*. The hardness results mentioned above imply that computing one minimal rewriting is already a hard problem. In addition, the following simple example shows that the number of minimal rewritings of a concept description C using a TBox $\mathcal{T}$ can be exponential in the size of C and $\mathcal{T}$.

For a nonnegative integer n, let $C_n := P_1 \sqcap \ldots \sqcap P_n$ and $\mathcal{T}_n := \{A_i \equiv P_i \mid 1 \leq i \leq n\}$. For each vector $\mathbf{i} = (i_1, \ldots, i_n) \in \{0,1\}^n$, we define

$$E_{\mathbf{i}} := \bigsqcap_{1 \leq j \leq n, i_j = 0} P_j \sqcap \bigsqcap_{1 \leq j \leq n, i_j = 1} A_j.$$

Obviously, for all $\mathbf{i} \in \{0,1\}^n$, $E_{\mathbf{i}}$ is a rewriting of C_n of size $|E_{\mathbf{i}}| = n = |C_n|$. Furthermore, it is easy to see that there does not exist a smaller rewriting of C_n using $\mathcal{T}_n$. Hence, there exists an exponential number of different minimal rewritings of C_n using $\mathcal{T}_n$.

A *naïve algorithm* for computing one minimal rewriting would enumerate all concept descriptions E of size $k = 1$, then $k = 2$, etc., until a rewriting E_0 of C using $\mathcal{T}$ is encountered. By construction, this rewriting is minimal, and since C is a rewriting of itself, one need not consider sizes larger than $|C|$. If one is interested in computing all minimal rewritings, it remains to enumerate all concept descriptions of size $|E_0|$, and test for each of them whether they are equivalent to C modulo $\mathcal{T}$. Obviously, this algorithm is not practical.

The main ideas underlying the more practical algorithm described in [26] is the following. For a given input description C, one splits the computation of rewritings into two steps:

- Compute an *extension* C^* of C. Such an extension is obtained from C by conjoining defined concepts at some positions of C while making sure that $C \equiv_{\mathcal{T}} C^*$ holds.
- Compute a *reduction* $\widehat{C}$ of C^*. Such a reduction is obtained from C^* by removing certain parts of C^* while making sure that $C^* \equiv_{\mathcal{T}} \widehat{C}$ holds.

The exact definitions of the right notions of extension and reduction depend, of course, on the DL under consideration. In [26], these definitions are given for $\mathcal{ALE}$. It is shown that the algorithm obtained this way computes only writings of the input description, and that all minimal rewritings are among the computed rewritings. In addition, [26] describes a more efficient heuristic algorithm that is not guaranteed to find *minimal* rewritings, but behaves quite well in practice. Basically, this algorithm uses a greedy strategy in the extension step, i.e., it conjoins as many defined concepts as possible to each position of C.

Approximation

This is the instance of the framework where (i) $\mathcal{T}$ is empty, and thus $\mathcal{L}_t$ is irrelevant; (ii) both ρ and $\preceq$ are the subsumption relation $\sqsubseteq$. In this case, we talk about approximation rather than rewriting. Given two DLs $\mathcal{L}_s$ and $\mathcal{L}_d$, an $\mathcal{L}_d$ *approximation* of an $\mathcal{L}_s$ concept description C is thus an $\mathcal{L}_d$ concept description D such that $C \sqsubseteq D$ and D is minimal (w.r.t. subsumption) with this property.

The case where $\mathcal{L}_s = \mathcal{ALC}$ and $\mathcal{L}_d = \mathcal{ALE}$ is investigated in [48]. Recall that the only difference between $\mathcal{ALC}$ and $\mathcal{ALE}$ is that disjunction is disallowed in $\mathcal{ALE}$ concept descriptions.[23] If C_1, C_2 are $\mathcal{ALE}$ concept descriptions, then it is easy to see that the approximation of the $\mathcal{ALC}$ concept description $C_1 \sqcup C_2$ by an $\mathcal{ALE}$ concept description is $lcs_{\mathcal{ALE}}(C_1, C_2)$. This suggests the following approach for approximating an $\mathcal{ALC}$ concept description C by an $\mathcal{ALE}$ concept description: just replace every disjunction in C by an application of the lcs operation. The following example demonstrates that this approach is too naïve: let $C := (\forall r.B \sqcup (\exists r.B \sqcap \forall r.A)) \sqcap \exists r.A$. If we replace the disjunction by an lcs operation and then compute the lcs, we obtain the $\mathcal{ALE}$ concept description

$$lcs_{\mathcal{ALE}}(\forall r.B, (\exists r.B \sqcap \forall r.A)) \sqcap \exists r.A \equiv \top \sqcap \exists r.A \equiv \exists r.A.$$

However, this concept description is too general. It is easy to see that $C \sqsubseteq \exists r.(A \sqcap B) \sqsubset \exists r.A$. In fact, $\exists r.(A \sqcap B)$ is the correct approximation.

In order to overcome this problem, the $\mathcal{ALC}$ concept description has to be transformed into an appropriate normal form. Basically, this normal form is obtained by distributing

[23] Here we assume without loss of generality that all $\mathcal{ALC}$ concept descriptions are in negation normal form where negation occurs only in front of concept names.

conjunctions over disjunctions, and by applying the rule $\forall r.C \sqcap \forall r.D \to \forall r.(C \sqcap D)$. For the example from above, the normal form is

$$C \equiv (\forall r.B \sqcap \exists r.A) \sqcup (\exists r.B \sqcap \forall r.A \sqcap \exists r.A),$$

and $lcs_{\mathcal{ALE}}(\forall r.B \sqcap \exists r.A, \exists r.B \sqcap \forall r.A \sqcap \exists r.A) = \exists r.(A \sqcap B)$.

However, even for $\mathcal{ALC}$ concept descriptions in this normal form, one cannot simply replace disjunction by the lcs operation to obtain their $\mathcal{ALE}$ approximation. Consider the $\mathcal{ALC}$ concept description $C' = \exists r.A \sqcap \exists r.B \sqcap \forall r.(\neg A \sqcup \neg B)$. If we simply replace the disjunction by the lcs, then we obtain $\exists r.A \sqcap \exists r.B \sqcap \forall r.\top \equiv \exists r.A \sqcap \exists r.B$. However, C' is also subsumed by the more specific $\mathcal{ALE}$ concept description $\exists r.(A \sqcap \neg B) \sqcap \exists r.(B \sqcap \neg A)$. This problem can be overcome by also propagating value restrictions onto existential restrictions. An approximation algorithms based on these ideas is described in [48]. It is shown that every $\mathcal{ALC}$ concept description has an $\mathcal{ALE}$ approximation, and this approximation is unique up to equivalence, i.e., there is always a least approximation. However, the size of the approximation may grow exponentially with the size of the input description. The algorithm for computing the approximation given in [48] runs in doubly-exponential time, and it is not clear whether this time bound can be improved. In [47], these results are extended to the approximation of $\mathcal{ALCN}$ concepts descriptions by $\mathcal{ALEN}$ concept descriptions.

6 NON-STANDARD EXPRESSIVITY

As discussed in Section 2, many expressive means of description logics have a counterpart in modal logic. In this section, we discuss two expressive means that are important for DLs, but lack a direct modal counterpart: concrete domains and role value maps.

6.1 Concrete Domains

The purpose of concrete domains is to enable the definition of concept descriptions with reference to concrete qualities of real-world objects such as their age, weight, temperature, and spatial extension. For example, we may define a teenager as a human whose age is between 10 and 19, or formulate a GCI stating that the age of a child is always smaller than the age of its parents. Representing concrete qualities and constraints of this form is necessary in almost all applications of description logics, such as reasoning about the semantic web [19] and about conceptual database models [114]. For this reason, even early DL systems such as MESON [68] and CLASSIC [44] addressed the issue of representing concrete qualities. However, these early approaches were of a rather ad hoc nature. The first approach that was fully (and formally) integrated with a description logic was presented by Baader and Hanschke [14], who proposed to extend the description logic $\mathcal{ALC}$ with so-called concrete domains.

Definitions

A *concrete domain* $\mathcal{D}$ is a pair $(\Delta^{\mathcal{D}}, \Phi^{\mathcal{D}})$ consisting of a non-empty set $\Delta^{\mathcal{D}}$ and a collection $\Phi^{\mathcal{D}}$ of *predicates names* such that each predicate $P \in \Phi^{\mathcal{D}}$ is equipped with an arity n and a fixed extension $P^{\mathcal{D}} \subseteq (\Delta^{\mathcal{D}})^n$. Slightly abusing notation, we will sometimes refer to the set $\Delta^{\mathcal{D}}$ as the concrete domain. In contrast, the domain $\Delta^{\mathcal{I}}$ of interpretations $\mathcal{I}$ will be called the *abstract domain*. For many application areas, the most interesting concrete domains are numerical ones. A typical numerical concrete domain is $\mathsf{Q} = (\mathbb{Q}, \Phi^{\mathsf{Q}})$, where $\mathbb{Q}$ denotes the rational numbers, and Φ^{Q} is comprised of the following predicates:

- unary predicates P_q for each $P \in \{<, \leq, =, \neq, \geq, >\}$ and each $q \in \mathbb{Q}$ with $(P_q)^{\mathsf{Q}} = \{q' \in \mathbb{Q} \mid q' \; P \; q\}$;
- binary predicates $<, \leq, =, \neq, \geq, >$ with the obvious extensions;
- a ternary predicate $+$ with $(+)^{\mathsf{Q}} = \{(q, q', q'') \in \mathbb{Q}^3 \mid q + q' = q''\}$.

By integrating a concrete domain $\mathcal{D}$ into $\mathcal{ALC}$, we obtain the basic description logic with concrete domains $\mathcal{ALC}(\mathcal{D})$. More precisely, $\mathcal{ALC}(\mathcal{D})$ is obtained from $\mathcal{ALC}$ by augmenting it with

- *abstract features*: a new sort of roles that is interpreted as a partial function from $\Delta^{\mathcal{I}}$ to $\Delta^{\mathcal{I}}$; abstract features can be used inside value restrictions and existential restrictions;
- *concrete features*: a new sort of roles that is interpreted as a partial function from the abstract domain $\Delta^{\mathcal{I}}$ into the concrete domain $\Delta^{\mathcal{D}}$; concrete features can *not* be used inside value restrictions and existential restrictions;
- a new concept constructor $P(u_1, \ldots, u_n)$, where $P \in \Phi^{\mathcal{D}}$ is a predicate of arity n, and each u_i is an expression $f_1 \circ \cdots \circ f_k \circ g$ with $f_1, \ldots, f_k$ ($k \geq 0$) abstract features and g a concrete feature. In the following, such expressions will be called *concrete paths*. The semantics of the new constructor is

  $$P(u_1, \ldots, u_n)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \exists x_1, \ldots, x_n \in \Delta^{\mathcal{D}} : u_i^{\mathcal{I}}(d) = x_i \text{ for } 1 \leq i \leq n \text{ and } (x_1, \ldots, x_n) \in P^{\mathcal{D}}\},$$

  where the interpretation $u^{\mathcal{I}}$ of a concrete path $u = f_1 \circ \cdots \circ f_k \circ g$ is defined as the partial function that maps $d \in \Delta^{\mathcal{I}}$ to $g^{\mathcal{I}}(f_k^{\mathcal{I}} \cdots (f_1^{\mathcal{I}}(d)) \cdots)$.

Using the concrete domain Q, the teenagers mentioned above can now be defined as

$$\mathsf{Teenager} \equiv \mathsf{Human} \sqcap >_9(\mathsf{age}) \sqcap <_{20}(\mathsf{age})$$

where age is a concrete feature. Similarly, the constraint saying that the age of children is smaller than the age of their parents can be formulated as

$$\top \sqsubseteq <(\mathsf{age}, \mathsf{mother} \circ \mathsf{age}) \sqcap <(\mathsf{age}, \mathsf{father} \circ \mathsf{age}),$$

where mother and father are abstract features.

There is a slight difference between the logic $\mathcal{ALC}(\mathcal{D})$ as defined here and the original version introduced in [14]: Baader and Hanschke's variant uses only a single type of feature whose interpretation is a partial function from $\Delta^{\mathcal{I}}$ to $\Delta^{\mathcal{I}} \cup \Delta^{\mathcal{D}}$. Thus, this type of feature combines our abstract and concrete features into one sort. In the literature, both versions of $\mathcal{ALC}(\mathcal{D})$ are considered. All results discussed in this section hold for both versions. Also note that the assumption that $\mathcal{ALC}$ is extended with only *one* concrete domain can be made without loss of generality, as it is shown in [14] that multiple concrete domains can be combined into a single one. In the world of modal logic, the closest relatives to DLs with concrete domains are linear temporal logics with constraints, see for example [33, 60].

Basic Results

When considering a description logic that is equipped with concrete domains, it is most desirable to obtain decidability results and complexity bounds that do not depend on a particular concrete domain, but rather apply to a class of concrete domains that is as large as possible. The first (decidability) result in this spirit was given by Baader and Hanschke in their original paper. Their result concerns the satisfiability of $\mathcal{ALC}(\mathcal{D})$ concept descriptions, where the concrete domain $\mathcal{D}$ is only required to satisfy some weak conditions. These conditions are derived from the fact that any satisfiability algorithm not committing itself to a particular concrete domain must call some concrete domain reasoner as a subprocedure via a well-defined interface. This observation leads to the notion of *admissibility*.

Let $\mathcal{D}$ be a concrete domain and V a countably infinite set of variables. A $\mathcal{D}$-*conjunction* is a predicate conjunction of the form

$$c = \bigwedge_{i<k} P_i(x_0^{(i)}, \ldots, x_{n_i}^{(i)})$$

where $P_i \in \Phi^{\mathcal{D}}$ is an n_i-ary predicate for each $i < k$ and the $x_j^{(i)}$ are variables from V. A $\mathcal{D}$-conjunction c is *satisfiable* iff there exists a function δ mapping the variables in c to elements of $\Delta^{\mathcal{D}}$ such that $(\delta(x_0^{(i)}), \ldots, \delta(x_{n_i}^{(i)})) \in P_i^{\mathcal{D}}$ for each $i < k$. We say that the concrete domain $\mathcal{D}$ is *admissible* iff

1. its set of predicates is closed under negation[24] and contains a name $\top_{\mathcal{D}}$ for $\Delta^{\mathcal{D}}$, and
2. the satisfiability of $\mathcal{D}$-conjunctions is decidable.

We refer to the satisfiability of $\mathcal{D}$-conjunctions as $\mathcal{D}$-*satisfiability*. Property 1 of admissibility has to be satisfied since $\mathcal{ALC}(\mathcal{D})$ provides for negation: for example, the concept description $C := {=}(g_1, g_1)) \sqcap {=}(g_2, g_2) \sqcap \neg{<}(g_1, g_2)$ is such that $d \in C^{\mathcal{I}}$ implies $g_1^{\mathcal{I}}(d) \geq g_2^{\mathcal{I}}(d)$ without explicitly using the "$\geq$" predicate,[25] and such information must be conveyed to the concrete domain reasoner. Note that the concrete domain Q presented above can easily be extended to satisfy Property 1 of admissibility: simply add predicates $\top_{\mathsf{Q}}$, $\bot_{\mathsf{Q}}$, and $\overline{+}$ (i.e., the negation of "+") with the obvious extensions. Let Q^a denote the extended version of Q. By using a reduction to linear programming, it is straightforward to show that Q^a-satisfiability is decidable in polynomial time [115], and thus Q^a is admissible.

The basic decidability result for $\mathcal{ALC}(\mathcal{D})$ given by Baader and Hanschke states that satisfiability (and thus also subsumption) of $\mathcal{ALC}(\mathcal{D})$-concept descriptions is decidable if $\mathcal{D}$ is admissible [14]. The complexity of this problem has been analyzed by Lutz [113], who proved PSpace-completeness under the assumption that $\mathcal{D}$ is admissible and $\mathcal{D}$-satisfiability is in PSpace. Thus if $\mathcal{D}$-satisfiability is in PSpace, then adding concrete domains to $\mathcal{ALC}$ does not increase the complexity of reasoning. Since Q^a-satisfiability is a polynomial time problem, we obtain PSpace-completeness for the instance $\mathcal{ALC}(\mathsf{Q}^a)$ of $\mathcal{ALC}(\mathcal{D})$. A discussion of the complexity of $\mathcal{D}$-satisfiability for a variety of numerical, temporal, and spatial concrete domains can be found in [115, 112].

[24] i.e., for each $P \in \Phi^{\mathcal{D}}$ of arity n, we find a $\overline{P} \in \Phi^{\mathcal{D}}$ with $\overline{P}^{\mathcal{D}} = (\Delta^{\mathcal{D}})^n \setminus P^{\mathcal{D}}$.

[25] The first two conjuncts are needed to ensure that $g_1^{\mathcal{I}}(d)$ and $g_2^{\mathcal{I}}(d)$ are actually defined.

When TBoxes are admitted, the complexity of reasoning increases drastically. We first consider acyclic TBoxes. As discussed in Section 2.2, satisfiability and subsumption w.r.t. acyclic TBoxes can be reduced to satisfiability w.r.t. the empty TBox using expansion. Thus, we obtain decidability of $\mathcal{ALC}(\mathcal{D})$ reasoning w.r.t. acyclic TBoxes if $\mathcal{D}$ is admissible. Since expansion is worst-case exponential, we also obtain an ExpSpace upper bound if $\mathcal{D}$-satisfiability is in PSpace. In the case of $\mathcal{ALC}$ without concrete domains, this upper bound can be improved to a PSpace one. Quite surprisingly, we can only push down the ExpSpace upper bound to a NExpTime one in the case of $\mathcal{ALC}(\mathcal{D})$: as proved in [117], there exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in PTime and satisfiability in $\mathcal{ALC}(\mathcal{D})$ w.r.t. acyclic TBoxes is NExpTime-hard. A matching upper bound states that satisfiability in $\mathcal{ALC}(\mathcal{D})$ w.r.t. acyclic TBoxes is in NExpTime if $\mathcal{D}$ is admissible and $\mathcal{D}$-satisfiability is in NP [117]. For subsumption, this yields analogous co-NExpTime bounds.

The jump in complexity from PSpace-complete to NExpTime-complete that is induced by adding acyclic TBoxes to $\mathcal{ALC}(\mathcal{D})$ is due to the fact that this addition increases the succinctness of $\mathcal{ALC}(\mathcal{D})$ (but not the expressivity). The NExpTime lower bound has been proved by reduction of a NExpTime-complete variant of the Post Correspondence Problem (PCP). As we cannot describe the reduction in full detail here, we sketch only how it makes use of the succinctness of acyclic TBoxes. The key observation is that it is possible to devise an acyclic TBox of size $O(k)$ that enforces models (of the concept name L_0) to contain a binary tree of depth k such that left successors are reachable via the abstract feature ℓ and right successors are reachable via the abstract feature r:

$$L_0 \equiv \exists \ell.L_1 \sqcap \exists r.L_1, \quad \ldots, \quad L_{k-1} \equiv \exists \ell.L_k \sqcap \exists r.L_k.$$

Without TBoxes, such a tree can only be enforced with a concept of length exponential in k. For the reduction, we add concept definitions expressing that the (exponentially many) leaves of the tree are connected via a chain of concrete domain predicates. For example, if we augment the above TBox with the following concept definitions and consider models of the conjunction $L_0 \sqcap C_0$, then we enforce that each leaf has a smaller number stored in the concrete feature g than all leaves that are to the right of it:

$$\begin{array}{rcl}
C_0 & \equiv & <(\ell r^{k-1} g, r\ell^{k-1} g) \sqcap \forall \ell.C_1 \sqcap \forall r.C_1, \\
& \vdots & \\
C_{k-2} & \equiv & <(\ell r g, r \ell g) \sqcap \forall \ell.C_{k-1} \sqcap \forall r.C_{k-1}, \\
C_{k-1} & \equiv & <(\ell g, r g).
\end{array}$$

Intuitively, the exponentially long chain of concrete domain predicates connecting the leaves can now be used to simulate the exponentially time-bounded computation of a Turing machine, or to talk about the concatenation of words in a PCP.

We now consider reasoning in $\mathcal{ALC}(\mathcal{D})$ with respect to GCIs, starting with a closely related result: let $\mathcal{ALC}^+(\mathcal{D})$ be the extension of $\mathcal{ALC}(\mathcal{D})$ with a transitive closure operator on roles and abstract features. Baader and Hanschke [16] prove that reasoning in $\mathcal{ALC}^+(\mathsf{R})$ w.r.t. the empty TBox is undecidable, where R is the concrete domain of real numbers with predicates based on Tarski algebra [149]. Their proof can easily be adapted to reasoning in $\mathcal{ALC}(\mathsf{R})$ w.r.t. GCIs, which is thus also undecidable. This adaptation is performed in [117], where a more general result is obtained: satisfiability (and thus also subsumption) in $\mathcal{ALC}(\mathcal{D})$ w.r.t. GCIs is undecidable if the concrete domain $\mathcal{D}$ satisfies

$\mathbb{N} \subseteq \Delta^{\mathcal{D}}$, and $\Phi^{\mathcal{D}}$ provides for a unary predicate for equality with 0, a binary equality predicate, and a binary predicate for incrementation. Thus, reasoning in $\mathcal{ALC}(\mathsf{Q})$ w.r.t. GCIs is undecidable since, in Q, incrementation can be expressed using the predicates "$=_1$" and "$+$". There are two ways for overcoming this rather disappointing result: either use a less powerful concrete domain constructor or very carefully choose the concrete domain.

The first approach was adopted, among others, by Möller et al. [80] and by Horrocks and Sattler [95, 130]. The imposed restriction on the concrete domain constructor usually is to allow only concrete features inside the concrete domain constructor instead of concrete paths of arbitrary length. In the following, the variant of $\mathcal{ALC}(\mathcal{D})$ obtained by this restriction will be called *path-free*. A particular form of path-freeness is to admit only unary predicates as proposed in [95]: in this case, reasoning in $\mathcal{ALC}(\mathcal{D})$ can be reduced to reasoning in path-free $\mathcal{ALC}(\mathcal{D})$ by replacing each concept description $P(f_1 \circ \cdots \circ f_k \circ g)$ with the equivalent path-free one $\exists f_1.\exists f_2.\cdots.\exists f_k.P(g)$. In [78] and [130], it is shown that reasoning in $\mathcal{SHN}(\mathcal{D})$ and $\mathcal{SHOQ}(\mathcal{D})$, the extensions of two expressive fragments of $\mathcal{SHOIQ}$ by concrete domains, is decidable w.r.t. GCIs if path-freeness is assumed and the concrete domain $\mathcal{D}$ is admissible. A more general result has been obtained in Section 5.3 of [27], where it is shown that any description logic $\mathcal{L}$ such that (i) satisfiability in $\mathcal{L}$ w.r.t. GCIs is decidable and (ii) $\mathcal{L}$'s class of interpretations is closed under disjoint unions (see [27] for details) can be extended with the path-free variant of the concrete domain constructor without losing decidability—provided that the concrete domain is admissible. Indeed, the "harmlessness" of the path-free concrete domain constructor is not very surprising since dropping concrete paths deprives concrete domains of most of their expressive power. For this reason, the complexity of reasoning w.r.t. GCIs in a DL incorporating path-free concrete domains is often not harder than the corresponding problem without concrete domains (if it dominates the complexity of $\mathcal{D}$-satisfiability). For example, in Section 2.4.1 of [112], it is shown that satisfiability in path-free $\mathcal{ALC}(\mathcal{D})$ w.r.t. GCIs is ExpTime-complete if $\mathcal{D}$ is admissible and $\mathcal{D}$-satisfiability is in ExpTime.

The second approach to overcome undecidability of $\mathcal{ALC}(\mathcal{D})$ with GCIs is to keep the original version of the concrete domain constructor and identify concrete domains that do not destroy decidability of reasoning if combined with GCIs. The first positive result following this route was established in [116], where a concrete domain C (for *c*omparison) is considered that is based on the rational numbers $\mathbb{Q} = \Delta^{\mathsf{C}}$, and provides for the binary predicates $<, \leq, =, \neq, \geq$, and $>$. It is shown that satisfiability (and thus also subsumption) in $\mathcal{ALC}(\mathsf{C})$ w.r.t. GCIs is ExpTime-complete, and that an analogous result holds for an interval-based temporal concrete domain. In [111], these results are further improved: first, the concrete domain C is extended to C^+, which additionally admits unary predicates $=_q$ for each $q \in \mathbb{Q}$; and second, the description logic is extended from $\mathcal{ALC}(\mathcal{D})$ to $\mathcal{SHIQ}(\mathcal{D})$, i.e. $\mathcal{SHOIQ}$ without nominals, but with the concrete domain C^+. For this extended logic, an ExpTime result analogous to the one stated above is established. A more general result is proved in [120], where a property of concrete domains is identified that is sufficient for decidability of $\mathcal{ALC}(\mathcal{D})$ with GCIs: a concrete domain $\mathcal{D}$ is called *ω-admissible* if it satisfies all of the following:

- $\mathcal{D}$ has only binary predicates;
- $\mathcal{D}$ has compactness: an infinite $\mathcal{D}$-conjunction is satisfiable if and only if every finite sub-conjunction is satisfiable;

- $\mathcal{D}$ has the patchwork property, which is defined as follows. A $\mathcal{D}$-conjunction c is *complete* iff, for all variables x, y occurring in c, c contains exactly one conjunct $P(x, y)$ and exactly one conjunct $P(y, x)$. Then, $\mathcal{D}$ has the *patchwork property* if the union of two satisfiable and complete $\mathcal{D}$-conjunctions that agree w.r.t. the conjuncts $P(x, y)$ and $P(y, x)$ w.r.t. all shared variables x, y is satisfiable.

The concrete domain C and the temporal concrete domain based on time intervals considered in [116] are ω-admissible. Additionally, it is shown in [120] that a spatial concrete domain based on the topological RCC8 relations is also ω-admissible, and therefore the corresponding incarnation of $\mathcal{ALC}(\mathcal{D})$ is decidable with GCIs. The result in [120] for DLs with ω-admissible concrete domains is established using a tableau algorithm and does not yield tight upper complexity bounds.

Roles in the Concrete Domain Constructor

The reader may wonder why the concrete domain constructor is introduced along with abstract features, instead of admitting normal roles in concrete paths. Indeed, this variant of concrete domains has also been considered by Hanschke [82]: a *concrete role path* is an expression $r_1 \circ \cdots \circ r_k \circ g$ with $r_1, \ldots, r_k$ roles and g a concrete feature. Then we may extend $\mathcal{ALC}$ with the concept constructors $\forall R_1, \ldots, R_k.P$ and $\exists R_1, \ldots, R_k.P$, where $R_1, \ldots, R_k$ are concrete role paths and $P \in \Phi^{\mathcal{D}}$ is a predicate of arity k. The semantics of these constructors is as follows:

$$
\begin{array}{lcl}
(\forall R_1, \ldots, R_k.P)^{\mathcal{I}} & := & \{d \in \Delta^{\mathcal{I}} \mid \forall x_1, \ldots, x_k \in \Delta^{\mathcal{D}} : \\
& & \quad (d, x_i) \in R_i^{\mathcal{I}} \text{ for } 1 \leq i \leq k \text{ implies } (x_1, \ldots, x_k) \in P^{\mathcal{D}}\} \\
(\exists R_1, \ldots, R_k.P)^{\mathcal{I}} & := & \{d \in \Delta^{\mathcal{I}} \mid \exists x_1, \ldots, x_k \in \Delta^{\mathcal{D}} : \\
& & \quad (d, x_i) \in R_i^{\mathcal{I}} \text{ for } 1 \leq i \leq k \text{ and } (x_1, \ldots, x_k) \in P^{\mathcal{D}}\}
\end{array}
$$

where the interpretation $R^{\mathcal{I}}$ of concrete role paths R is defined in the obvious way through relational composition.[26] The resulting DL is called $\mathcal{ALCP}(\mathcal{D})$. Reasoning with $\mathcal{ALCP}(\mathcal{D})$ concept descriptions has been proved to be decidable in [82]. When investigating the complexity of $\mathcal{ALCP}(\mathcal{D})$, it becomes clear that the restriction to abstract features inside the concrete domain constructor has computational advantages: it is shown in [117] that there exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in PTime and satisfiability of $\mathcal{ALCP}(\mathcal{D})$ concept descriptions is NExpTime-hard. Again, a matching upper bound is obtained for the case where $\mathcal{D}$-satisfiability is in NP. This should be contrasted with the PSpace-completeness of satisfiability of concept descriptions in $\mathcal{ALC}(\mathcal{D})$.

We have seen that both the generalized concrete domain constructor and acyclic TBoxes are seemingly moderate extension of $\mathcal{ALC}(\mathcal{D})$ that make reasoning considerably harder. Other such extensions include inverse roles, role conjunction, nominals, and a concrete domain *role* constructor [117, 118]. Thus, the PSpace upper bound of $\mathcal{ALC}(\mathcal{D})$ is not robust w.r.t. extensions of the language.

Uniqueness Constraints and Functional Dependencies

Uniqueness constraints (sometimes also called *identification constraints* and *keys*) and functional dependencies play an important role in the database area, and are also useful

[26] In $\mathcal{ALC}(\mathcal{D})$ with only functional roles inside the concrete domain constructor, the universal version of this constructor can be defined in terms of the existential one (see e.g. [113]).

in connection with concrete domains [118, 119].[27] Say, for example, that there exists a concrete feature socnum associating humans with their social security number. Then, if a human is American, this person should be *uniquely* identified by this number: no other instance of the concept name American should have the same value of the concrete feature socnum. This corresponds to a uniqueness constraint. As another example, we may want to enforce that all books having the same ISBN number share the same title. This is a functional dependency, i.e. the value of the ISBN number determines the title in a functional way. It is not a uniqueness constraint since different books may have the same ISBN number (say, two different copies of the "Handbook of Modal Logic").

In the following, we concentrate on uniqueness constraints. A *key box* is a finite set of *uniqueness constraints* $(u_1, \ldots, u_n \text{ keyfor } C)$, where $u_1, \ldots, u_n$ are concrete paths and C is a concept description. An interpretation $\mathcal{I}$ *satisfies* $(u_1, \ldots, u_n \text{ keyfor } C)$ if, for all $d, e \in C^{\mathcal{I}}$,

$$u_i^{\mathcal{I}}(d) = u_i^{\mathcal{I}}(e) \text{ for } 1 \leq i \leq n \quad \text{ implies } \quad d = e,$$

and it is a *model* of a key box $\mathcal{K}$ if it satisfies all uniqueness constraints in $\mathcal{K}$. In the presence of key boxes, we are interested in the satisfiability of a concept description w.r.t. a TBox and a key box, i.e. in joint models of all three components (and similarly for subsumption).

It is interesting to note that there is a close relationship between nominals and key boxes. For example, if used together with the uniqueness constraint $(g \text{ keyfor } \top)$, the $\mathcal{ALC}(\mathsf{Q})$ concept description $\exists g.{=_q}$ behaves similar to a nominal for each $q \in \mathbb{Q}$: it is interpreted by a set of cardinality at most one. Key boxes are strong enough to render reasoning in $\mathcal{ALC}(\mathcal{D})$ undecidable [118]: satisfiability of $\mathcal{ALC}(\mathcal{D})$ concept descriptions w.r.t. key boxes is undecidable (even without TBoxes) if the concrete domain $\mathcal{D}$ satisfies $\mathbb{N} \subseteq \Delta^{\mathcal{D}}$ and $\Phi^{\mathcal{D}}$ provides a unary predicate for equality with 0, a binary equality predicate, and a binary predicate for incrementation. Note that this result is similar to the undecidability of satisfiability in $\mathcal{ALC}(\mathcal{D})$ w.r.t. GCIs.

Decidability can be regained by allowing only Boolean combinations of concept names inside uniqueness constraints. Key boxes satisfying this property are called *Boolean*. Even w.r.t. Boolean key boxes, reasoning is much harder than reasoning without key boxes: there exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in PTIME and satisfiability of $\mathcal{ALC}(\mathcal{D})$ concept descriptions w.r.t. Boolean key boxes is NEXPTIME-hard [118]. This high complexity cannot even be reduced if paths are restricted to length one inside $\mathcal{ALC}(\mathcal{D})$ concept descriptions and key boxes (*path-freeness*). The matching upper bound relies on a modified notion of admissibility, called *key-admissibility*. Roughly spoken, a concrete domain is key-admissible if it is admissible and provides for a binary equality predicate. The original definition given in [118] is slightly more general, but too complex to be repeated here. In [118] it is shown that satisfiability of $\mathcal{ALC}(\mathcal{D})$ concept descriptions w.r.t. Boolean key boxes is in NEXPTIME if $\mathcal{D}$ is key-admissible and $\mathcal{D}$-satisfiability is in NP. In the same work, also a more powerful DL with concrete domains, $\mathcal{SHOQ}(\mathcal{D})$, is extended with key boxes, and a decidability result for the path-free case is established (where $\mathcal{D}$ is required to be key-admissible, but key boxes are not expected to be Boolean).

In the case of functional dependencies, quite similar results can be established. We refer to [119] for more details.

[27] They can also be used in description logics without concrete domains, c.f. [41, 49, 103, 151].

Aggregation

Aggregation is a useful mechanism available in many expressive conceptual modelling formalisms such as database schema languages and query languages. The use of aggregation in the context of concrete domains has been proposed in [31]. As an example, consider the following description of a process and its subprocesses:

$$\mathsf{Process} \sqcap >_0(\mathsf{duration}) \sqcap \forall \mathsf{subproc}.(\mathsf{Process} \sqcap >_0(\mathsf{duration})).$$

The aggregation function "sum" is needed if we want to express that the duration of the mother process is identical to the sum of the durations of its subprocesses (of which there may be arbitrarily many).

A *concrete domain with aggregation* is a concrete domain that, additionally, provides for a set of aggregation functions $\mathsf{agg}(\mathcal{D})$, where each $\Gamma \in \mathsf{agg}(\mathcal{D})$ is associated with a partial function $\Gamma^{\mathcal{D}}$ from the set of finite multisets over $\Delta^{\mathcal{D}}$ into $\Delta^{\mathcal{D}}$. To distinguish concrete domains with aggregation from those without, we denote the former with Σ. Typical aggregation functions are min, max, sum, count, and $\mathsf{average}$. The set of $\mathcal{ALC}(\Sigma)$ concept descriptions is now defined in the same way as $\mathcal{ALC}(\mathcal{D})$ concept descriptions, except that *aggregated features* may be be used in place of concrete features, where an aggregated feature is an expression $\Gamma(r \circ g)$ with r a role, g a concrete feature, and Γ an aggregation function from Σ. The semantics of aggregated features is defined via multisets: for each interpretation $\mathcal{I}$ and each $d \in \Delta^{\mathcal{I}}$ such that the set $\{e \mid (d,e) \in r^{\mathcal{I}}\}$ is finite, we use $M_d^{r \circ g}$ to denote the multiset that, for each $z \in \Delta^{\mathcal{D}}$, contains z exactly $|\{e \mid (d,e) \in r^{\mathcal{I}} \text{ and } g^{\mathcal{I}}(e) = z\}|$ times. The semantics of aggregated features is now defined as follows:

$$(\Gamma(r \circ g))^{\mathcal{I}}(d) := \begin{cases} \Gamma^{\Sigma}(M_d^{r \circ g}) & \text{if } \{e \mid (d,e) \in r^{\mathcal{I}}\} \text{ is finite} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Returning to the initial example, we can now express the fact that the duration of the mother process is identical to the sum of the durations of all its subprocesses by writing $=(\mathsf{duration}, \mathsf{sum}(\mathsf{subproc} \circ \mathsf{duration}))$. The investigations performed by Baader and Sattler [31] reveal that the expressive power provided by aggregation functions is hard to tame in order to obtain a decidable formalism: for concrete domains with aggregation Σ where (i) $\mathbb{N} \subseteq \Delta^{\Sigma}$, (ii) Φ^{Σ} contains a (unary) predicate for equality with 1 and a (binary) equality predicate, and (iii) $\mathsf{agg}(\Sigma)$ contains min, max, and sum, satisfiability of $\mathcal{ALC}(\Sigma)$ concept descriptions is undecidable. This lower bound applies even if we admit only conjunction, the $\forall r.C$ constructor, and the concrete domain constructor, but drop all other concept constructors. Rather strong measures have to be taken to regain decidability: either, we have to drop the $\forall r.C$ constructor from the language, thus obtaining a sub-Boolean DL, or we have to confine ourselves to "well-behaved" aggregation functions such as min and max of which there exist only very few. More details can be found in [31].

6.2 Role Value Maps

Role value maps are a family of concept constructors that were available in the first description logic system, KL-ONE [45], and have since then been considered in several variations. The original and most powerful variant of role value maps has later been found to cause undecidability, even if used with quite weak (sub-Boolean) description logics such as the one available in the KL-ONE system.

To define role value maps, we must introduce the notion of a *path*, i.e., a composition $r_1 \circ \cdots \circ r_n$ of role names. If R and S are paths, then the expression $(R \subseteq S)$ is a *containment role value map* and $(R = S)$ is an *equality role value map*. To extend $\mathcal{ALC}$ with role value maps, we admit them as additional concept constructors. The resulting description logic is denoted $\mathcal{ALC}^{\mathsf{rvm}}$. The semantics of the additional concept constructors is as follows:

$$\begin{array}{rcl} (R \subseteq S)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \forall e.(d,e) \in R^{\mathcal{I}} \text{ implies } (d,e) \in S^{\mathcal{I}}\}, \\ (R = S)^{\mathcal{I}} & = & \{e \in \Delta^{\mathcal{I}} \mid \forall e.(d,e) \in R^{\mathcal{I}} \text{ iff } (d,e) \in S^{\mathcal{I}}\}, \end{array}$$

where the interpretation $R^{\mathcal{I}}$ of paths R is defined in the obvious way through relational composition. For example, the concept description $\mathsf{Person} \sqcap (\mathsf{child} \circ \mathsf{friend} \subseteq \mathsf{knows})$ describes persons knowing all the friends of their children.

Though there appears to be no direct modal counterpart to role value maps as a concept constructor, there is a connection to *modal reduction principles* as discussed in Chapter 7. Modal reduction principles (MRPs) are axioms of the form $Mp \rightarrow Np$, where M and N are sequences of modal operators $\Box_i$ and $\Diamond_j$ [152]. We call an MRP *box-only* if M and N are non-empty sequences of box operators. There is a close correspondence between normal modal logics axiomatized by box-only MRPs and $\mathcal{ALC}^{\mathsf{rvm}}$: it follows from Sahlqvist's completeness theorem that normal modal logics axiomatized by a box-only MRP $\varphi = \Box_{i_1} \cdots \Box_{i_n} p \rightarrow \Box_{j_1} \cdots \Box j_m p$ are characterized by the class of frames that validates φ; moreover, it is a routine task to show that the same class of frames is determined by all models of the GCI $\top \sqsubseteq (r_{i_1} \circ \cdots \circ r_{i_n} \subseteq s_{j_1} \circ \cdots \circ s_{j_m})$. There is also a close connection between role value maps and so-called grammar logics [58], which will be discussed in more detail below.

Undecidability

Reasoning in the first description logic system KL-ONE was initially believed to be in PTime. However, in 1989 Schmidt-Schauß was able to show that it is undecidable, identifying role value maps as the main culprit [141]. More precisely, Schmidt-Schauß proves that, even in the description logic $\mathcal{FL}_0^{\mathsf{rvm}}$ providing only for the constructors conjunction, value restriction, and role value maps, subsumption w.r.t. the empty TBox is undecidable.

The proof of Schmidt-Schauß uses a reduction of the word problem for groups. We present here a slight variation that reduces the word problem for semigroups [36].[28] For simplicity, we first show undecidability of $\mathcal{FL}_0^{\mathsf{rvm}}$ w.r.t. GCIs, and then eliminate the GCIs from the reduction. A finitely presented semigroup S is given in the form of defining identities $s_1 = t_1, \ldots, s_m = t_m$, where the s_i and t_i are words over some finite alphabet Σ. Then the word problem is to decide, given S and words s and t, whether $s = t$ holds in S, i.e., whether the identity $s = t$ can be derived from the defining identities of S and the usual axioms for semigroups. For the reduction, we view the symbols $r_1, \ldots, r_n$ of Σ

[28] The reduction of Schmidt-Schauß yields a slightly stronger result since it applies also to the case where we have only equality role value maps, but no containment role value maps.

as role names and construct a set of GCIs $\mathcal{T}_S$ and a concept description $D_{s,t}$ as follows:

$$\begin{array}{rcl}\mathcal{T}_S & := & \{\top \sqsubseteq (s_1 = t_1) \sqcap \cdots \sqcap (s_m = t_m)\} \\ D_{s,t} & := & (s = t)\end{array}$$

Then it is not hard to show that $\top \sqsubseteq_{\mathcal{T}_S} D_{s,t}$ if and only if $s = t$ can be derived from the defining identities of S. This yields undecidability of subsumption in $\mathcal{FL}_0^{\mathsf{rvm}}$ w.r.t. GCIs. To get rid of GCIs, we can internalize them (c.f. Section 3.1). More precisely, the subsumption $\top \sqsubseteq_{\mathcal{T}_S} D_{s,t}$ holds if and only if $C_S \sqsubseteq D_{s,t}$, where C_S is defines as follows:

$$C_S \quad := \quad \bigsqcap_{r \in \Sigma} \big((r \subseteq u) \sqcap (u \circ u \subseteq u)\big) \sqcap \forall u.((s_1 = t_1) \sqcap \cdots \sqcap (s_m = t_m))$$

where u is a role name that does not occur in Σ. It follows that subsumption in $\mathcal{FL}_0^{\mathsf{rvm}}$ is undecidable also without GCIs.

There is a related result in modal logic that should be mentioned: Shehtman proved in 1982 that there exists a set Γ of box-only modal reduction principles such that, in the normal modal logic axiomatized by Γ, satisfiability is undecidable [146]. By what was said above about the connection between MRPs and role value maps, and since GCIs can be internalized in $\mathcal{ALC}^{\mathsf{rvm}}$ similar to what was done above in the case of $\mathcal{FL}_0^{\mathsf{rvm}}$, it is obvious that this gives a proof of undecidability of reasoning in $\mathcal{ALC}^{\mathsf{rvm}}$ without TBoxes and GCIs.

To avoid undecidability, two approaches have been considered: first, role value maps have been weakened into feature agreements and feature disagreements, which have a similar semantics but are restricted to paths comprised only of functional roles; and second, the original role value maps have been used for paths of a syntactically restricted form. In the next section, we describe the first approach. Syntactic restrictions on paths are discussed subsequently in Section 6.2.

Feature Agreements

A *feature path* is a composition $f_1 \circ \cdots \circ f_n$ of abstract features as introduced in Section 6.1. If u and v are feature paths, then the expression $(u = v)$ is a *feature agreement*, and $(u \neq v)$ is a *feature disagreement*. The description logic $\mathcal{ALCF}$ is obtained from $\mathcal{ALC}^{\mathsf{rvm}}$ by replacing role value maps with feature (dis)agreements. The semantics of the new concept constructors is:

$$\begin{array}{rcl}(u = v)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \exists e.(d,e) \in u^{\mathcal{I}} \text{ and } (d,e) \in v^{\mathcal{I}}\} \\ (u \neq v)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \exists e, e'.(d,e) \in u^{\mathcal{I}}, (d,e') \in v^{\mathcal{I}}, \text{ and } e \neq e'\}\end{array}$$

In the literature, feature agreements are sometimes called the *same-as* constructor, e.g. in their incarnation in the CLASSIC system [44]. The restriction of role value maps to feature paths has an impairing effect on the usefulness of feature (dis)agreements. For example, the concept description

$$\mathsf{Person} \sqcap (\mathsf{child} \circ \mathsf{friend} \subseteq \mathsf{knows})$$

cannot be expressed in $\mathcal{ALCF}$ since child and knows should not be forced to be functional. However, feature (dis)agreements can still be usefully employed, as illustrated by the following concept definition:

$$\mathsf{ParentsMarried} \equiv (\mathsf{mother} \circ \mathsf{married\text{-}to} = \mathsf{father})$$

The main advantage of feature agreements over role value maps lies in their computational properties. Indeed, Hollunder and Nutt [89] show that satisfiability (and thus also subsumption) of $\mathcal{ALCF}$ concept descriptions is decidable and PSPACE-complete (see also [113]).

When further expressive means are added to $\mathcal{ALCF}$, the computational complexity often gets dramatically worse. In this respect, feature (dis)agreements resemble concrete domains: the PSPACE upper bound of the basic description logic with feature (dis)agreements $\mathcal{ALCF}$ is rather fragile w.r.t. extensions of the language. An important example for this behaviour are TBoxes. As shown in [110], satisfiability in $\mathcal{ALCF}$ w.r.t. acyclic TBoxes is NEXPTIME-complete. The result is established by reduction of a NEXPTIME-complete variant of the domino problem, and exploits the succinctness gained by introducing acyclic TBoxes similar to what is discussed in Section 6.1 in the context of concrete domains. When cyclic TBoxes or GCIs are admitted, satisfiability and subsumption in $\mathcal{ALCF}$ even become undecidable [11], which can be shown by a reduction of the word problem for groups. Other extensions of $\mathcal{ALCF}$ that make reasoning harder include intersection of roles, inverse roles, and transitive closure of functional roles. In the first case, satisfiability of concept descriptions becomes NEXPTIME-complete, while it is undecidable in the latter two cases [11, 112].

Restricted Paths in Role Value Maps

In this section, we consider syntactic restrictions on paths inside role value maps. There are some quite drastic restrictions that are easily seen to regain decidability of reasoning in $\mathcal{ALC}^{\mathsf{rvm}}$:

- Only allow paths of length one. In this case, reasoning in $\mathcal{ALC}^{\mathsf{rvm}}$ can be reduced to reasoning in $\mathcal{ALC}^{\neg,\cap,\cup}$, the extension of $\mathcal{ALC}$ with Boolean role constructors: simply replace $(r = s)$ with $(r \subseteq s) \sqcap (s \subseteq r)$, and $(r \subseteq s)$ with $\forall(r \cap \neg s).\bot$. Since satisfiability and subsumption in $\mathcal{ALC}^{\neg,\cap,\cup}$ w.r.t. GCIs is known to be decidable [122, 121], so is the restricted version of $\mathcal{ALC}^{\mathsf{rvm}}$. Note that there is also a close connection to role hierarchies: a role inclusion $r \sqsubseteq s$ as used in a role hierarchy can be simulated using the concept equation $\top \equiv (r \subseteq s)$ in $\mathcal{ALC}^{\mathsf{rvm}}$.

- Only admit role value maps of the form $(r \circ r \subseteq r)$. Clearly, we obtain a localized variant of transitive roles. It is straightforward to adapt the standard techniques for dealing with (globally) transitive roles [135, 94] to show that, in this variant of $\mathcal{ALC}^{\mathsf{rvm}}$, satisfiability and subsumption w.r.t. GCIs are decidable. It appears to be an open problem whether admitting single-role role value maps of the form $(r^n \subseteq r^m)$, with $n, m \in \mathbb{N}$ and r^n denoting the n-fold composition of r, yields a decidable variant of $\mathcal{ALC}^{\mathsf{rvm}}$.

More powerful decidable fragments of $\mathcal{ALC}^{\mathsf{rvm}}$ can be obtained by restricting paths in a less strict way. This is done by Horrocks and Sattler [96] in their work on complex role inclusion axioms (RIAs), and in the closely related area of grammar logics [58, 34, 59]. In both cases, role value maps are *not* considered to be concept constructors, but rather they are *global*, similar to the role inclusions in a role hierarchy, i.e., the role value map must hold for every element of the interpretation domain. In the following, we consider $\mathcal{ALC}$ concept descriptions and assume the presence of a *role box*, i.e. a finite set of role value maps $(R \subseteq S)$. An interpretation $\mathcal{I}$ is a *model* of a role box $\mathcal{R}$ if it satisfies $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

for all $(R \subseteq S) \in \mathcal{R}$. In this setting, we are interested in satisfiability and subsumption w.r.t. TBoxes/GCIs and role boxes, i.e., in common models of all three inputs.

Translated to this terminology, the idea of grammar logic is to view a role value map $(r_1 \circ \cdots \circ r_k \subseteq s_1 \circ \cdots \circ s_\ell)$ as a production rule $s_1 \cdots s_\ell \to r_1 \cdots r_k$, and a role box as a formal grammar [58]. Here, the mapping from role names to terminal and non-terminal symbols is arbitrary, but fixed. From the description logic perspective, the most relevant results from grammar logic are the following. First, Demri shows that satisfiability and subsumption of $\mathcal{ALC}$ concept descriptions is ExpTime-complete if only role boxes corresponding to left-linear or right-linear grammars are admitted [59]. In role boxes of this form, we can express properties such as "the enemies of my friends are my enemies":

$$(\mathsf{friend} \circ \mathsf{enemy} \subseteq \mathsf{enemy}).$$

Note that this result captures the case where paths are required to be of length at most one, but not the case $(r \circ r \subseteq r)$. Second, Baldoni et al. show that satisfiability of $\mathcal{ALC}$ concept descriptions is undecidable if role boxes corresponding to context-free grammars are admitted [34]. This result was later strengthened by Demri to linear grammars [59].

Horrocks and Sattler consider the extension of $\mathcal{SHIQ}$ with global role value maps of the form $(r \subseteq s)$, $(r \circ s \subseteq r)$, and $(s \circ r \subseteq r)$, where r, s is a role name or the inverse of a role name [96]. For example, the statement about enemies of friends from above can be strengthened by additionally saying that the friends of my enemies are my enemies:

$$\begin{array}{lcl}(\mathsf{friend} \circ \mathsf{enemy} & \subseteq & \mathsf{enemy}) \\ (\mathsf{enemy} \circ \mathsf{friend} & \subseteq & \mathsf{enemy}).\end{array}$$

Note that this role box does not correspond to a left-linear or right-linear grammar. Role value maps of this form are of particular interest since they allow to describe the propagation of properties, e.g. along part-whole relations: "the owner of a whole is the owner of all parts" can be written as $(\mathsf{part\text{-}of} \circ \mathsf{owner} \subseteq \mathsf{owner})$. Let us call Horrocks and Sattler's variant of role value maps *HS-RVMs*. Horrocks and Sattler obtain the following results: first, reasoning in the extension of $\mathcal{SHIQ}$ with HS-RVMs is undecidable in the general case. An inspection of the proof shows that undecidability already arises in $\mathcal{ALC}$ extended with inverse roles, number restrictions of the form $(\leqslant 1\ r)$, GCIs, and HS-RVMs. Decidability of plain $\mathcal{ALC}$ extended with HS-RVMs (and possibly GCIs) appears to be an open problem. Second, satisfiability and subsumption in $\mathcal{SHIQ}$ w.r.t. GCIs and role boxes becomes decidable if we admit only HS-RVMs and *acyclic* role boxes. Acyclicity of role boxes is defined similar to the TBox case: a role r *directly affects* a role s if $r \neq s$ and there is an HS-RVM with (i) r appearing on the left-hand side (possibly inside a composition) and s appearing on the right-hand side, or (ii) the inverse of r appearing on the left-hand side and the inverse of s appearing on the right-hand side. *Affects* is the transitive closure of "directly affects". Then a role box is acyclic if no role affects itself. Observe that the above example about friends and enemies is acyclic.

There are several other restrictions of paths that can be considered. An interesting example is to admit only role value maps $(R \subseteq S)$ with R and S paths of equal length. This restriction has been investigated by Molitor [126], but the decidability status of $\mathcal{ALC}^{\mathsf{rvm}}$ under this restriction is, as of now, unknown.

BIBLIOGRAPHY

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

[2] Franz Baader. Unification in commutative theories. *J. of Symbolic Computation*, 8:479–497, 1989.

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

[4] Franz Baader. A formal definition for the expressive power of terminological knowledge representation languages. *J. of Logic and Computation*, 6:33–54, 1996.

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

[6] Franz Baader. Computing the least common subsumer in the description logic $\mathcal{EL}$ w.r.t. terminological cycles with descriptive semantics. In *Proceedings of the 11th International Conference on Conceptual Structures, ICCS 2003*, volume 2746 of *Lecture Notes in Artificial Intelligence*, pages 117–130. Springer-Verlag, 2003.

[7] Franz Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 319–324. Morgan Kaufmann, 2003.

[8] Franz Baader. Restricted role-value-maps in a description logic with existential restrictions and terminological cycles. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, CEUR-WS, 2003.

[9] Franz Baader. Terminological cycles in a description logic with existential restrictions. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 325–330. Morgan Kaufmann, 2003.

[10] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*. Morgan Kaufmann, Los Altos, 2005.

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

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

[13] Franz Baader, Enrico Franconi, Bernhard Hollunder, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

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

[15] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. Technical Report RR-91-10, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 1991.

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

[17] Franz Baader and Bernhard Hollunder. $\mathcal{KRIS}$: $\mathcal{K}$nowledge $\mathcal{R}$epresentation and $\mathcal{I}$nference $\mathcal{S}$ystem. *SIGART Bull.*, 2(3):8–14, 1991.

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

[19] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. In Dieter Hutter and Werner Stephan, editors, *Festschrift in honor of Jörg Siekmann*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003.

[20] Franz Baader and Ralf Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In *Proc. of the 22nd German Annual Conf. on Artificial Intelligence (KI'98)*, volume 1504 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1998.

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

[22] Franz Baader and Ralf Küsters. Unification in a description logic with transitive closure of roles. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, (LPAR 2001)*, Lecture Notes in Artificial Intelligence, Havana, Cuba, 2001. Springer-Verlag.

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

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

[25] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 96–101, 1999.

[26] Franz Baader, Ralf Küsters, and Ralf Molitor. Rewriting concepts using terminologies. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 297–308, 2000.

[27] Franz Baader, Carsten Lutz, Holger Sturm, and Frank Wolter. Fusions of description logics and abstract description systems. *Journal of Artificial Intelligence Research (JAIR)*, 16:1–58, 2002.

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

[29] Franz Baader and Werner Nutt. Basic description logics. In *[12]*, pages 43–95. 2003.

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

[31] Franz Baader and Ulrike Sattler. Description logics with aggregates and concrete domains. *Information Systems*, 28(8):979–1004, 2003.

[32] Franz Baader and Wayne Snyder. Unification theory. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning, Volume I*, pages 447–533. Elsevier Science Publishers, 2001.

[33] Philippe Balbiani and Jean-Francois Condotta. Computational complexity of propositional linear temporal logics based on qualitative spatial or temporal reasoning. In *Proc. of the 4th Int. Workshop on Frontiers of Combining Systems (FroCoS 2002)*, number 2309 in Lecture Notes in Artificial Intelligence, pages 162–176. Springer-Verlag, 2002.

[34] Matteo Baldoni, Laura Giordano, and Alberto Martelli. A tableau calculus for multimodal logics and some (un)decidability results. In *Proc. of the Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'98)*, volume 1397 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1998.

[35] Piero A. Bonatti. On the undecidability of description and dynamic logics with recursion and counting. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 331–336. Morgan Kaufmann, Los Altos, 2003.

[36] William W. Boone. The word problem. *Ann. of Mathematics*, 2(70):207–265, 1959.

[37] Alexander Borgida. On the relationship between description logics and predicate logic queries. Technical Report LCSR-TR-295-A, Rutgers University, New Brunswick (NJ, USA), 1992.

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

[39] Alexander Borgida and Deborah L. McGuinness. Asking queries about frames. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 340–349, 1996.

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

[41] Alexander Borgida and Grant E. Weddell. Adding uniqueness constraints to description logics (preliminary report). In *Proc. of the 5th Int. Conf. on Deductive and Object-Oriented Databases (DOOD'97)*, pages 85–102, 1997.

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

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

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

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

[46] Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, gci axioms, and—what else? In *Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI 2004)*, 2004.

[47] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin Turhan. Approximating $\mathcal{ALCN}$-concept descriptions. In *Proc. of the 2002 Description Logic Workshop (DL 2002)*, 2002.

[48] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin Turhan. Approximation and difference in description logics. In D. Fensel, F. Giunchiglia, D. McGuiness, and M.-A. Williams, editors, *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, pages 203–214, San Francisco, CA, 2002. Morgan Kaufman.

[49] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification constraints and functional dependencies in description logics. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 155–160, 2001.

[50] Diego Calvanese, Giuseppe de Giacomo, and Maurizio Lenzerini. 2-ATAs make DLs easy. In *Proc. of the 2002 Description Logic Workshop (DL 2002)*, 2002.

[51] Diego Calvanese, Giuseppe de Giacomo, Maurizio Lenzerini, Riccardo Rosati, and Guido Vetere. DL-Lite: Practical reasoning in rich DLs. In Ralf Möller and Volker Haarslev, editors, *Proceedings of the International Workshop in Description Logics 2004 (DL2004)*, volume 104 of *CEUR-WS (http://ceur-ws.org/)*, 2004.

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

[53] William W. Cohen and Haym Hirsh. Learning the CLASSIC description logics: Theoretical and experimental results. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 121–133, 1994.

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

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

[56] Giuseppe De Giacomo and Maurizio Lenzerini. What's in an aggregate: Foundations for description logics with tuples and sets. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI'95)*, pages 801–807, 1995.

[57] Giuseppe De Giacomo and Maurizio Lenzerini. TBox and ABox reasoning in expressive description logics. In Luigia C. Aiello, John Doyle, and Stuart C. Shapiro, editors, *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 316–327. Morgan Kaufmann, Los Altos, 1996.

[58] Luis Farinas del Cerro and Martti Penttonen. Grammar logics. *Logique et Analyse*, 121-122:123–134, 1988.

[59] Stephane Demri. The complexity of regularity in grammar logics and related modal logics. *J. of Logic and Computation*, 11(6), 2001.

[60] Stephane Demri. LTL over integer periodicity constraints. In I. Walukiewicz, editor, *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'04)*, number 2987 in Lecture Notes in Computer Science, pages 121–135, Barcelona, Spain, 2004. Springer-Verlag.

[61] Francesco Donini. Complexity of reasoning. In *[12]*, pages 96–136. 2003.

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

[63] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 151–162. Morgan Kaufmann, Los Altos, 1991.

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

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

[66] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. Tractability and intractability in description logics. Available at ftp://ftp.dis.uniroma1.it/pub/ai/papers/dlnn99d.ps.gz, 1999.

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

[68] Jürgen Edelmann and Bernd Owsnicki-Klewe. Data models in knowledge representation systems: A case study. In C. R. Rollinger and W. Horn, editors, *GWAI-86 and 2. Österreichische Artificial-Intelligence-Tagung*, volume 124 of *Informatik-Fachberichte*, pages 69–74. Springer-Verlag, 1986.

[69] Ulrich Faigle and György Turán. Sorting and recognition problems for ordered sets. *SIAM J. on Computing*, 17(1), 1988.

[70] Maurizio Fattorosi-Barnaba and F. de Caro. Graded modalities I. *Studia Logica*, 44(2):197–221, 1985.

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

[72] Dov Gabbay and Larisa Maksimova. *Interpolation and Definability: Modal and Intuitionistic Logic.* Oxford University Press, 2005.

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

[74] George Gargov and Valentin Goranko. Modal logic with names. *J. of Philosophical Logic*, 22:607–636, 1993.

[75] Silvio Ghilardi. Unification in intuitionistic logic. *J. of Symbolic Logic*, 64:859–880, 1999.

[76] Silvio Ghilardi. Best solving modal equations. *Ann. Pure Appl. Logic*, 102(3):183–198, 2000.

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

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

[79] Volker Haarslev and Ralf Möller. RACER system description. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, 2001.

[80] Volker Haarslev, Ralf Möller, and Michael Wessel. The description logic $\mathcal{ALCNH}_{R^+}$ extended with concrete domains: A practically motivated approach. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 29–44. Springer-Verlag, 2001.

[81] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

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

[83] Patrick J. Hayes. The logic of frames. In D. Metzing, editor, *Frame Conceptions and Text Understanding*, pages 46–61. Walter de Gruyter and Co., 1979. Republished in [43].

[84] Edith Hemaspaandra. The complexity of poor man's logic. *J. of Logic and Computation*, 11(4):609–622, 2001.

[85] Monika R. Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *36th Annual Symposium on Foundations of Computer Science*, pages 453–462, Milwaukee, Wisconsin, 1995. IEEE Computer Society Press.

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

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

[88] Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 335–346, 1991.

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

[90] Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics.* PhD thesis, University of Manchester, 1997.

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

[92] Ian Horrocks. Implementation and optimization techniques. In *[12]*, pages 306–346. 2003.

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

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

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

[96] Ian Horrocks and Ulrike Sattler. Decidability of SHIQ with complex role inclusion axioms. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 343–348. Morgan Kaufmann, Los Altos, 2003.

[97] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, Edinburgh (UK), 2005. Morgan Kaufmann, Los Altos.

[98] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 161–180. Springer-Verlag, 1999.

[99] Ian Horrocks and Stephan Tobies. Reasoning with axioms: Theory and practice. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*. Morgan Kaufmann, Los Altos, 2000.

[100] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing SHIQ-description logic to disjunctive datalog programs. In *Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004)*. Morgan Kaufmann, Los Altos, 2004.

[101] Ullrich Hustadt and Renate A. Schmidt. Issues of decidability for description logics in the framework of resolution. In R. Caterra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logic*, volume 1761 of *Lecture Notes in Artificial Intelligence*, pages 191–205. Springer-Verlag, 2000.

[102] Yevgeny Kazakov and Hans de Nivelle. Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*. CEUR Electronic Workshop Proceedings, http://CEUR-WS.org/Vol-81/, 2003.

[103] Vitaliy L. Khizder, David Toman, and Grant E. Weddell. On decidability and complexity of description logics with uniqueness constraints. In *Proc. of the 8th Int. Conf. on Database Theory (ICDT 2001)*, 2001.

[104] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[105] Natasha Kurtonina and Maarten de Rijke. Expressiveness of concept expressions in first-order description logics. *Artificial Intelligence*, 107(2):303–333, 1999.

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

[107] Ralf Küsters and Ralf Molitor. Approximating most specific concepts in description logics with existential restrictions. In Franz Baader, Gerd Brewka, and Thomas Eiter, editors, *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI 2001)*, volume 2174 of *Lecture Notes in Artificial Intelligence*, pages 33–47, Vienna, Austria, 2001. Springer-Verlag.

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

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

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

[111] Carsten Lutz. Adding numbers to the $\mathcal{SHIQ}$ description logic—First results. In *Proc. of the 8th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2002)*, pages 191–202. Morgan Kaufmann, Los Altos, 2002.

[112] Carsten Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2002.

[113] Carsten Lutz. PSpace reasoning with the description logic $\mathcal{ALCF}(\mathcal{D})$. *Logic Journal of the IGPL*, 10(5):535–568, 2002.

[114] Carsten Lutz. Reasoning about entity relationship diagrams with complex attribute dependencies. In *Proc. of the 2002 Description Logic Workshop (DL 2002)*, 2002.

[115] Carsten Lutz. Description logics with concrete domains—a survey. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Advances in Modal Logics Volume 4*, pages 265–296. King's College Publications, 2003.

[116] Carsten Lutz. Combining interval-based temporal reasoning with general TBoxes. *Artificial Intelligence*, 152(2):235–274, 2004.

[117] Carsten Lutz. NExpTime-complete description logics with concrete domains. *ACM Transactions on Computational Logic*, 5(4):669–705, 2004.

[118] Carsten Lutz, Carlos Areces, Ian Horrocks, and Ulrike Sattler. Keys, nominals, and concrete domains. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 349–354. Morgan Kaufmann, Los Altos, 2003.

[119] Carsten Lutz and Maja Milicic. Description logics with concrete domains and functional dependencies. In *Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI 2004)*, 2004.

[120] Carsten Lutz and Maja Milicic. A tableau algorithm for description logics with concrete domains and GCIs. In *Proc. of the Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX 2005)*, number 3702 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 2005.

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

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

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

[124] Eric Mays, Robert Dionne, and Robert Weida. K-REP system overview. *SIGART Bull.*, 2(3), 1991.

[125] Marvin Minsky. A framework for representing knowledge. In J. Haugeland, editor, *Mind Design*. The MIT Press, 1981. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [43].

[126] Ralf Molitor. Konsistenz von Wissensbasen in Beschreibungslogiken mit Rollenoperatoren. Diploma thesis, RWTH Aachen, Germany, 1997.

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

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

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

[130] Jeff Z. Pan and Ian Horrocks. Reasoning in the $\mathcal{SHOQ}(\mathcal{D}_n)$ description logic. In Ian Horrocks and Sergio Tessaris, editors, *Proceedings of the International Workshop in Description Logics 2002 (DL2002)*, volume 53 of *CEUR-WS (http://ceur-ws.org/)*, pages 53–62, 2002.

[131] Peter F. Patel-Schneider and Ian Horrocks. DLP and FaCT. In Neil V. Murray, editor, *Proc. of the Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'99)*, volume 1617 of *Lecture Notes in Artificial Intelligence*, pages 19–23. Springer-Verlag, 1999.

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

[133] M. Ross Quillian. Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science*, 12:410–430, 1967. Republished in [43].

[134] Alan Rector and Ian Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*, Stanford, CA, 1997. AAAI Press.

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

[136] Ulrike Sattler. Description logics for the representation of aggregated objects. In W.Horn, editor, *Proc. of the 14th Eur. Conf. on Artificial Intelligence (ECAI 2000).* IOS Press, Amsterdam, 2000.

[137] Andrea Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *J. of Intelligent Information Systems*, 2:265–278, 1993.

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

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

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

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

[142] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with unions and complements. Technical Report SR-88-21, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern (Germany), 1988.

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

[144] Len K. Schubert, Randy G. Goebel, and Nicola J. Cercone. The structure and organization of a semantic net for comprehension and inference. In N. V. Findler, editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 121–175. Academic Press, 1979.

[145] Fabrizio Sebastiani and Umberto Straccia. A computationally tractable terminological logic. In *Proc. of the 3rd Scandinavian Conf. on Artificial Intelligence SCAI-91*, pages 307–315, 1991.

[146] Valentin Shehtman. Undecidable propositional calculi. *USSR Academy of Sciences*, 75:74–116, 1982. (Russian).

[147] John F. Sowa. *Conceptuals Structures: Information Processing in Mind and Machine*. Addison Wesley Publ. Co., Reading, Massachussetts, 1984.

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

[149] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, Berkeley, 1951.

[150] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation.* PhD thesis, RWTH Aachen, 2001.

[151] David Toman and Grant Weddell. On reasoning about structural equality in XML: A description logic approach. In D. Calvanese, M. Lenzerini, and R. Motwani, editors, *Proceedings of the 9th International Conference on Database Theory*, number 2572 in LNCS, pages 96–110, 2002.

[152] Johan van Benthem. Modal reduction principles. *J. of Symbolic Logic*, 41:301–312, 1976.

[153] Wiebe Van der Hoek and Maarten de Rijke. Counting objects. *J. of Logic and Computation*, 5(3):325–345, 1995.

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

[155] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. In *Proc. of the 16th ACM SIGACT Symp. on Theory of Computing (STOC'84)*, pages 446–455, 1984.