Matching in Description Logics with Existential Restrictions

Franz Baader and Ralf Küsters
LuFg Theoretical Computer Science, RWTH Aachen
Ahornstraße 55, 52074 Aachen, Germany
email: {baader,kuesters}@informatik.rwth-aachen.de

Abstract

Matching of concepts against patterns is a new inference task in Description Logics, which was originally motivated by applications of the CLASSIC system. Consequently, the work on this problem was until now mostly concerned with sublanguages of the CLASSIC language, which does not allow for existential restrictions.

This paper extends the existing work on matching in two directions. On the one hand, the question of what are the most "interesting" solutions of matching problems is explored in more detail. On the other hand, for languages with existential restrictions both, the complexity of deciding the solvability of matching problems and the complexity of actually computing sets of "interesting" matchers are determined. The results show that existential restrictions make these computational tasks more complex. Whereas for sublanguages of CLASSIC both problems could be solved in polynomial time, this is no longer possible for languages with existential restrictions.

1 Introduction

In description logics (DLs), the standard inference problems, like the subsumption and the instance problem, are now well-investigated. More recently, new types of inference problems have been introduced and investigated, like computing the least common subsumer of concepts (Cohen et al., 1992; Baader et al., 1999b) and matching concept descriptions against patterns.

This paper is concerned with the latter of these two inference problems, which has originally been introduced in (Borgida and McGuinness, 1996; McGuinness, 1996) to help filter out the unimportant aspects of large concepts appearing in knowledge bases of the CLASSIC DL system (Brachman et al., 1991; Borgida et al., 1989). More recently, matching (as well as the more general problem of unification) has been proposed as a tool for detecting redundancies in knowledge bases (Baader and Narendran, 1998) and to support the integration of knowledge bases by prompting possible interschema assertions to the integrator (Borgida and Küsters, 1999).

All three applications have in common that one wants to search the knowledge base for concepts having a certain (not completely specified) form. This "form" can be expressed with the help of so-called *concept patterns*, i.e., concept descriptions containing variables (which stand for descriptions). For example, assume that we want to find concepts that are concerned with individuals having a son and a daughter sharing some characteristic. This can be expressed by the pattern $D := \exists \mathsf{has\text{-}child}.(\mathsf{Male} \sqcap X) \sqcap \exists \mathsf{has\text{-}child}.(\mathsf{Female} \sqcap X)$, where X is a variable standing for the common characteristic. The concept description $C := \exists \mathsf{has\text{-}child}.(\mathsf{Tall} \sqcap \mathsf{Male}) \sqcap \exists \mathsf{has\text{-}child}.(\mathsf{Tall} \sqcap \mathsf{Female})$ matches this pattern in the sense that, if we replace the variable X by the description Tall, the pattern becomes *equivalent* to the description. Thus, the substitution $\sigma := \{X \mapsto \mathsf{Tall}\}$ is a *matcher modulo equivalence* of the matching problem $C \equiv^? D$. Note that not only the fact that there is a matcher is of interest, but also the matcher itself, since it tells us what is the common characteristic of the son and the daughter.

Looking for such an exact match (called *matching modulo equivalence* in the following) is not always appropriate, though. In our example, using matching modulo equivalence means that all the additional characteristics of the son and daughter mentioned in the

concept must be common to both. Thus, the description $C' := \exists$has-child.(Tall $\sqcap$ Male $\sqcap$ Talkative) $\sqcap$ $\exists$has-child.(Tall $\sqcap$ Female $\sqcap$ Quiet) does not match the pattern modulo equivalence. *Matching modulo subsumption* only requires that, after the replacement, the pattern subsumes the description. Thus, the substitution σ from above is a *matcher modulo subsumption* of the matching problem $C' \sqsubseteq^? D$.

Previous results on matching in DLs were mostly concerned with sublanguages of the CLASSIC description language, which does not allow for existential restrictions of the kind used in our example. A polynomial-time algorithm for computing matchers modulo subsumption for a rather expressive DL was introduced in (Borgida and McGuinness, 1996). The main drawback of this algorithm is that it requires the concept patterns to be in structural normal form, and thus it cannot handle arbitrary matching problems. In addition, the algorithm is incomplete, i.e., it does not always find a matcher, even if one exists. For the DL $\mathcal{ALN}$, a polynomial-time algorithm for matching modulo subsumption and equivalence was presented in (Baader et al., 1999a). This algorithm is complete and it applies to arbitrary patterns.

Motivated by an application in chemical process engineering (Baader and Sattler, 1996), which requires existential restrictions, the main purpose of this paper is to investigate matching in DLs allowing for existential restrictions. We will show that existential restrictions make matching more complex in two respects. First, whereas matching in the DLs considered in (Borgida and McGuinness, 1996; Baader et al., 1999a) is polynomial, even deciding the existence of matchers is an NP-complete problem in the presence of existential restrictions (Section 3). Second, the algorithms described in (Borgida and McGuinness, 1996; Baader et al., 1999a) always compute the least matcher (w.r.t. subsumption of substitutions; see the definition of $\sqsubseteq_s$ in Section 4) of the given matching problem. For languages with existential restrictions, such a unique least matcher need not exist. However, the set of minimal matchers is finite (though possibly exponential in the size of the matching problem), and we will show how to compute this set (Section 5). It has turned out, however, that the minimal matchers are not necessarily the most interesting ones since they may contain certain redundancies. Thus, one also needs a kind of post-processing step that removes these redundancies (Section 6). Since giving an answer to the question of what are good sets of matchers is a crucial and non-trivial problem, which has not been explored satisfactorily so far, we will treat this question in a separate section (Section 4). A more detailed presentation and complete proofs of all the results stated in this paper can be found in (Baader and Küsters, 1999).

Syntax	Semantics
$\top$	Δ
$C \sqcap D$	$C^I \cap D^I$
$\exists r.C$	$\{x \in \Delta \mid \exists y : (x,y) \in r^I \wedge y \in C^I\}$
$\forall r.C$	$\{x \in \Delta \mid \forall y : (x,y) \in r^I \rightarrow y \in C^I\}$
$\neg P$, $P \in N_C$	$\Delta \setminus P^I$
$\bot$	$\emptyset$

Table 1: Syntax and semantics of concept descriptions.

2 Preliminaries

Concept descriptions are inductively defined with the help of a set of concept *constructors*, starting with a set N_C of *concept names* and a set N_R of *role names*. In this paper, we consider concept descriptions built from the constructors shown in Table 1. In the description logic $\mathcal{EL}$, concept descriptions are formed using the constructors top-concept ($\top$), conjunction ($C \sqcap D$), and existential restriction ($\exists r.C$). The description logic $\mathcal{ALE}$ additionally provides us with value restrictions ($\forall r.C$), primitive negation ($\neg P$), and the bottom-concept ($\bot$).

As usual, the semantics of concept descriptions is defined in terms of an *interpretation* $\mathcal{I} = (\Delta, \cdot^I)$. The domain Δ of $\mathcal{I}$ is a non-empty set and the interpretation function $\cdot^I$ maps each concept name $P \in N_C$ to a set $P^I \subseteq \Delta$ and each role name $r \in N_R$ to a binary relation $r^I \subseteq \Delta \times \Delta$. The extension of $\cdot^I$ to arbitrary concept descriptions is defined inductively, as shown in the second column of Table 1.

One of the most important traditional inference services provided by DL systems is computing the subsumption hierarchy. The concept description C is *subsumed* by the description D ($C \sqsubseteq D$) iff $C^I \subseteq D^I$ holds for all interpretations $\mathcal{I}$. The concept descriptions C and D are *equivalent* ($C \equiv D$) iff they subsume each other.

In order to define concept patterns, we additionally need a set $\mathcal{X}$ of concept variables, which we assume to be disjoint from $N_C \cup N_R$. Informally, an $\mathcal{ALE}$-concept pattern is an $\mathcal{ALE}$-concept description over the concept names $N_C \cup \mathcal{X}$ and the role names N_R, with the only exception that primitive negation must not be applied to variables.

Definition 1 *The set of all* $\mathcal{ALE}$-concept patterns *over* N_C, N_R, *and* $\mathcal{X}$ *is inductively defined as follows:*

- *Every concept variable $X \in \mathcal{X}$ is a pattern.*
- *Every $\mathcal{ALE}$-concept description over N_C and N_R is a pattern.*
- *If D_1 and D_2 are concept patterns, then $D_1 \sqcap D_2$ is a concept pattern.*
- *If D is a concept pattern and r is a role name, then $\forall r.D$ and $\exists r.D$ are concept patterns.*

The notion of a pattern (and also the notions "substitution" and "matching problem" introduced below) can be restricted to $\mathcal{EL}$ in the obvious way. For example, if X, Y are concept variables, r a role name, and A, B concept names, then $D := A \sqcap X \sqcap \exists r.(B \sqcap Y)$ is both an $\mathcal{EL}$- and $\mathcal{ALE}$-concept pattern, but $\neg X$ is neither an $\mathcal{ALE}$- nor an $\mathcal{EL}$-concept pattern.

A *substitution* σ is a mapping from $\mathcal{X}$ into the set of all $\mathcal{ALE}$-concept descriptions. This mapping is extended to concept patterns in the usual way, i.e.,

- $\sigma(C) := C$ for all $\mathcal{ALE}$-concept descriptions C,
- $\sigma(D_1 \sqcap D_2) := \sigma(D_1) \sqcap \sigma(D_2)$,
- $\sigma(\forall r.D) := \forall r.\sigma(D)$ and $\sigma(\exists r.D) := \exists r.\sigma(D)$.

For example, if we apply the substitution $\sigma := \{X \mapsto A \sqcap B, Y \mapsto A\}$ to the pattern D from above, we obtain the description $\sigma(D) = A \sqcap A \sqcap B \sqcap \exists r.(B \sqcap A)$. The result of applying a substitution to an $\mathcal{ALE}$-concept pattern is always an $\mathcal{ALE}$-concept description. Note that this would no longer be the case if negation were allowed in front of concept variables. In fact, $\sigma(\neg X) = \neg(A \sqcap B)$ cannot be expressed in $\mathcal{ALE}$.

Definition 2 *Let C be an $\mathcal{ALE}$-concept description and D an $\mathcal{ALE}$-concept pattern. Then, $C \equiv^? D$ is an* $\mathcal{ALE}$-matching problem modulo equivalence *and $C \sqsubseteq^? D$ is an* $\mathcal{ALE}$-matching problem modulo subsumption. *The substitution σ is a* matcher *of $C \equiv^? D$ iff $C \equiv \sigma(D)$, and it is a* matcher *of $C \sqsubseteq^? D$ iff $C \sqsubseteq \sigma(D)$.*

3 Complexity of the decision problem

In this section, we are interested in the complexity of deciding whether a given matching problem has a matcher or not. Our results are summarized in Table 2. The first and the second row of the table refer to matching modulo subsumption and matching modulo equivalence, respectively.

	$\mathcal{EL}$	$\mathcal{ALE}$
subsumption	P	NP-complete
equivalence	NP-complete	NP-complete

Table 2: Deciding solvability of matching problems

First, note that patterns are not required to contain variables. Consequently, matching modulo subsumption (equivalence) is at least as hard as subsumption (equivalence). Thus, NP-completeness of subsumption in $\mathcal{ALE}$ (Donini et al., 1992) yields the hardness part of the second column of Table 2. Second, for the languages $\mathcal{ALE}$ and $\mathcal{EL}$, matching modulo subsumption can be reduced to subsumption: $C \sqsubseteq^? D$ has a matcher iff $\sigma_\top := \{X \mapsto \top \mid X \in \mathcal{X}\}$ is a matcher of $C \sqsubseteq^? D$. Thus, the known complexity results for subsumption in $\mathcal{ALE}$ and $\mathcal{EL}$ (Donini et al., 1992; Baader et al., 1999b) complete the first row of Table 2. Third, NP-hardness of matching modulo equivalence for $\mathcal{EL}$ can be shown by a reduction of SAT. The reduction in (Baader and Küsters, 1999) uses only a fixed number of role names. Here, we give a simpler reduction for which, however, the number of role names grows with the formula.

Lemma 3 *Deciding the solvability of matching problems modulo equivalence in $\mathcal{EL}$ is NP-hard.*

Proof. Let $\phi = p_1 \wedge \cdots \wedge p_m$ be a propositional formula in conjunctive normal form and let $\{x_1, \ldots, x_n\}$ 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:

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

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

In order to encode ϕ, we introduce a concept pattern D_{p_i} for each conjunct p_i. For example, if $p_i = x_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_4$, then $D_{p_i} := X_1 \sqcap \overline{X}_2 \sqcap X_3 \sqcap \overline{X}_4 \sqcap B$. The whole formula is then represented by the matching problem $C_\phi \equiv^? D_\phi$, where

$$\begin{aligned} C_\phi &:= \exists s_1.(A \sqcap B) \sqcap \cdots \sqcap \exists s_m.(A \sqcap B), \\ D_\phi &:= \exists s_1.D_{p_1} \sqcap \cdots \sqcap \exists s_m.D_{p_m}. \end{aligned}$$

This matching problem ensures that, among all the variables in D_{p_i}, at least one must be replaced by A. This corresponds to the fact that, within one conjunct p_i, there must be at least one literal that evaluates to *true*. Note that we need the concept B in D_{p_i} to cover the case where all variables in D_{p_i} are substituted with A.

We combine the two matching problems introduced above into a single problem $C_n \sqcap C_\phi \equiv^? D_n \sqcap D_\phi$. It is easy to verify that ϕ is satisfiable iff this matching problem is solvable. ∎

It remains to show that matching modulo equivalence in $\mathcal{EL}$ and $\mathcal{ALE}$ can in fact be decided in nondeterministic polynomial time. This is an easy consequence of the following lemma.

Lemma 4 *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.*

The lemma (together with the known complexity results for subsumption) shows that the following can be realized in NP: "guess" a substitution satisfying the given size bound, and then test whether it is a matcher.

For $\mathcal{EL}$, the polynomial bound stated in the lemma can be derived from our algorithm for computing so-called s-co-complete sets of matchers presented in Section 6. At the end of Section 6, we will also comment on the (quite involved) proof of Lemma 4 for $\mathcal{ALE}$.

4 Solutions of matching problems

In this section and in Section 5.1, we will use the $\mathcal{EL}$-concept description C^1_{ex} and the pattern D^1_{ex} shown in Figure 1 as our running example. (The graphical representation will be explained later on.)

It is easy to see that the substitution $\sigma_\top$ is a matcher of $C^1_{\text{ex}} \sqsubseteq^? D^1_{\text{ex}}$, and thus this matching problem modulo subsumption is indeed solvable. However, the matcher $\sigma_\top$ is obviously not an interesting one. We are interested in matchers that bring us as close as possible to the description C^1_{ex}. In this sense, the matcher $\sigma_1 := \{X \mapsto \mathsf{W} \sqcap \exists \mathsf{hc}.\mathsf{W}, Y \mapsto \mathsf{W}\}$ is better than $\sigma_\top$, but still not optimal. In fact, $\sigma_2 := \{X \mapsto \mathsf{W} \sqcap \exists \mathsf{hc}.\mathsf{W} \sqcap \exists \mathsf{hc}.(\mathsf{W} \sqcap \mathsf{P}), Y \mapsto \mathsf{W} \sqcap \mathsf{D}\}$ is better than σ_1 since it satisfies $C^1_{\text{ex}} \equiv \sigma_2(D^1_{\text{ex}}) \sqsubset \sigma_1(D^1_{\text{ex}})$.

We formalize this intuition with the help of the following precedence ordering on matchers. For a given matching problem $C \sqsubseteq^? D$ and two matchers σ, τ we define

$$\sigma \sqsubseteq_i \tau \text{ iff } \sigma(D) \sqsubseteq \tau(D).$$

Here "i" stands for "instance". Two matchers σ, τ are *i-equivalent* ($\sigma \equiv_i \tau$) iff $\sigma \sqsubseteq_i \tau$ and $\tau \sqsubseteq_i \sigma$. A matcher σ is called *i-minimal* iff, for every matcher τ, $\tau \sqsubseteq_i \sigma$ implies $\tau \equiv_i \sigma$. We are interested in *computing i-minimal matchers*; more precisely, we want to obtain at least one i-minimal matcher for each of the minimal i-equivalence classes (i.e., i-equivalence classes of i-minimal matchers). Since an i-equivalence class usually contains more than one matcher, the question is which ones to prefer.

In (Baader et al., 1999a), it is shown that a given $\mathcal{ALN}$-matching problem always has a unique minimal i-equivalence class, and that this class is the class of the least matcher w.r.t. the ordering

$$\sigma \sqsubseteq_s \tau \text{ iff } \sigma(X) \sqsubseteq \tau(X) \text{ for all } X \in \mathcal{X},$$

where "s" stands for "substitution". The matcher σ is a *least matcher w.r.t.* $\sqsubseteq_s$ iff $\sigma \sqsubseteq_s \tau$ for all matchers τ. The notions s-minimal, s-equivalent, etc. are defined in the obvious way.

For $\mathcal{EL}$ and $\mathcal{ALE}$, things are quite different. As illustrated by the example $\exists r.A \sqcap \exists r.B \sqsubseteq^? \exists r.X$, a given matching problem may have several non-equivalent i-minimal (s-minimal) matchers: the substitutions $\{X \mapsto A\}$ and $\{X \mapsto B\}$ are both i- and s-minimal, and they are obviously neither i- nor s-equivalent. It can be shown (Baader and Küsters, 1999) that the set of all s-minimal matchers (up to s-equivalence) also contains all i-minimal matchers (up to i-equivalence). However, the s-minimal matchers are usually not the best representatives of their i-equivalence class.

In our running example, σ_2 is a least and therefore i-minimal matcher. Nevertheless, it is not the one we really want to compute since it contains redundancies, i.e., expressions that are not really necessary for obtaining the instance $\sigma_2(D^1_{\text{ex}})$ (modulo equivalence). In fact, σ_2 contains two different kinds of redundancies. First, the existential restriction $\exists \mathsf{hc}.\mathsf{W}$ in $\sigma_2(X)$ is redundant since removing it still yields a concept description equivalent to $\sigma_2(X)$. Second, W in $\sigma_2(Y)$ is redundant in that the substitution obtained by deleting W from $\sigma_2(Y)$ still yields the same instance of D^1_{ex} (although the substitution obtained this way is no longer s-equivalent to σ_2). In our example, the only i-minimal matcher (modulo associativity and commutativity of concept conjunction) that is free of redundancies in this sense is $\sigma_3 := \{X \mapsto \mathsf{W} \sqcap \exists \mathsf{hc}.(\mathsf{W} \sqcap \mathsf{P}), Y \mapsto \mathsf{D}\}$.

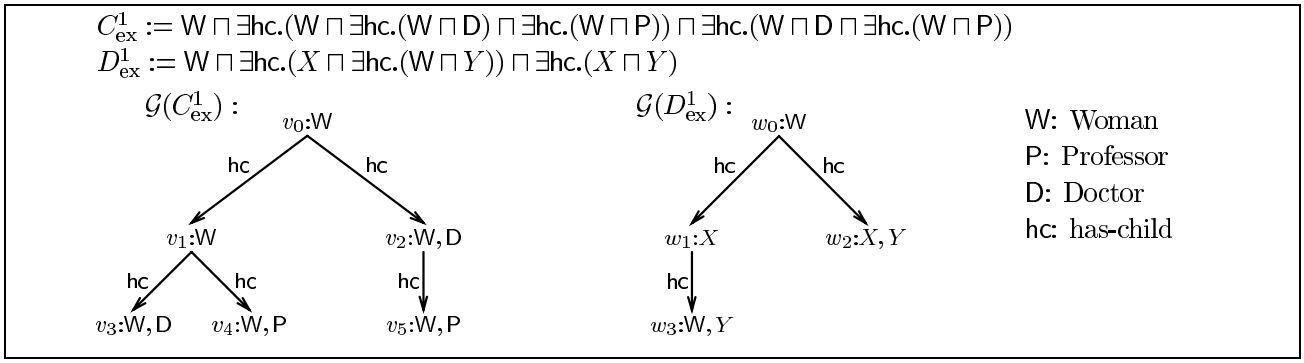

Figure 1: $\mathcal{EL}$-concept description and pattern, and their $\mathcal{EL}$-description trees.

We want to *compute i-minimal matchers that are reduced*, i.e., free of redundancies. It remains to formalize the notion "reduced" more rigorously. For this purpose, we need the notion of a subdescription.

Definition 5 *For an $\mathcal{ALE}$-concept description C, the $\mathcal{ALE}$-concept description $\widehat{C}$ is a* subdescription *of C ($\widehat{C} \preceq_d C$) iff* (i) $\widehat{C} = C$; *or* (ii) $\widehat{C} = \bot$; *or* (iii) *$\widehat{C}$ is obtained from C by removing some (negated) concept names, value restrictions, or existential restrictions on the top-level of C, and for all remaining value/existential restrictions $\forall r.E/\exists r.E$ replacing E by a subdescription of E.*

Note that, if everything is removed from C, then $\widehat{C} = \top$. For $\mathcal{EL}$, subdescriptions are defined analogously; clearly, (ii) must be removed for $\mathcal{EL}$. The concept description $\widehat{C}$ is a *strict subdescription* of C iff $\widehat{C} \preceq_d C$ and $\widehat{C} \neq C$.

Definition 6 *The $\mathcal{ALE}$-concept description C is called* reduced *iff there does not exist a strict subdescription of C that is equivalent to C.*

For example, $\sigma_3(X)$ is reduced, whereas $\sigma_2(X)$ is not. Reduced concept descriptions are unique, provided that they are also in $\forall$-normal form. An $\mathcal{ALE}$-concept description C is in *$\forall$-normal form* iff the $\forall$-rule $\forall r.E \sqcap \forall r.F \longrightarrow \forall r.(E \sqcap F)$ cannot be applied to C. Clearly, every concept description can easily be transformed (in polynomial time) into its $\forall$-normal form by exhaustively applying the $\forall$-rule.

Lemma 7 *(Baader and Küsters, 1999) Equivalent and reduced $\mathcal{ALE}$-concept descriptions in $\forall$-normal form are equal up to associativity and commutativity of concept conjunction.*

The ordering $\preceq_d$ can be extended to substitutions in the obvious way:

$$\sigma \preceq_d \tau \text{ iff } \sigma(X) \preceq_d \tau(X) \text{ for all } X \in \mathcal{X}.$$

The matcher σ of $C \sqsubseteq^? D$ is called *reduced* iff it is a d-minimal matcher (i.e., minimal w.r.t. $\preceq_d$). Note that, given a reduced matcher σ, every concept description $\sigma(X)$ is reduced. However, as illustrated by our running example (removal of W in $\sigma_2(Y)$), just replacing the descriptions $\sigma(X)$ by equivalent reduced descriptions does not necessarily yield a reduced matcher.

To sum up, given a matching problem $C \sqsubseteq^? D$, we want to compute matchers that are i-minimal and reduced. It should be noted that a given i-equivalence class of matchers may contain different reduced matchers. Since reduced and s-equivalent matchers are equal up to associativity and commutativity of conjunction, it is, however, sufficient to compute the reduced matchers up to s-equivalence.

Our approach for computing i-minimal and reduced matchers of $C \sqsubseteq^? D$ proceeds in two steps (which we consider in more detail in the next two sections):

1. Compute the set of all i-minimal matchers of $C \sqsubseteq^? D$ up to i-equivalence (i.e., one matcher for each i-equivalence class).
2. For each i-minimal matcher σ computed in the first step, compute the d-minimal matchers up to s-equivalence of $\sigma(D) \equiv^? D$.

Of course, if we are interested in matching modulo equivalence in the first place, we just apply the second step to $C \equiv^? D$.

5 Computing i-minimal matchers

In this section, we show how to compute the set of all i-minimal matchers up to i-equivalence for a given matching problem $C \sqsubseteq^? D$. In fact, the algorithms for $\mathcal{EL}$ and $\mathcal{ALE}$ that we will present below solve a slightly different problem: they compute so-called s-complete sets of matchers.

Definition 8 *A set of matchers is called* s-complete *iff it contains (at least) all s-minimal matchers up to s-equivalence. It is called* minimal s-complete*, if it consists of one representative of every s-equivalence class of the s-minimal matchers.*

Analogously, one can define (minimal) i-complete sets. A simple proof shows the following relationship between s- and i-complete sets.

Lemma 9 *Every s-complete set is also i-complete.*

Given an s-complete set, one can therefore in a post-processing step use the subsumption algorithm (for $\mathcal{EL}$ or $\mathcal{ALE}$) to determine a minimal i-complete set.

Mainly for didactic reasons, we present the algorithm for computing s-complete sets of matchers both for $\mathcal{EL}$ and $\mathcal{ALE}$. In general, an algorithm for a given DL does not necessarily work for its sublanguages since the set of potential matchers changes. In this particular case, however, the algorithm for $\mathcal{ALE}$ applied to $\mathcal{EL}$-matching problems only yields matchers in $\mathcal{EL}$.

5.1 Computing s-complete sets in $\mathcal{EL}$

The algorithm for computing s-complete sets of matchers in $\mathcal{EL}$ is based on a characterization of subsumption between $\mathcal{EL}$-concepts via homomorphisms between the corresponding description trees. This characterization has been introduced in (Baader et al., 1999b) for the purpose of computing the least common subsumer (lcs) of $\mathcal{EL}$-concepts. Before introducing the matching algorithm, we briefly recall the characterization of subsumption.

Characterizing subsumption in $\mathcal{EL}$

Definition 10 *An* $\mathcal{EL}$-description tree *is a tree of the form* $\mathcal{G} = (V, E, v_0, \ell)$ *where*

- V *is a finite set of* nodes*;*
- $E \subseteq V \times N_R \times V$ *is a finite set of* edges *labeled with role names* r *(*$\exists$*-edges);*
- $v_0 \in V$ *is the* root *of* $\mathcal{G}$*;*
- ℓ *is a* labeling function *mapping the nodes in* V *to finite subsets of* N_C*. The empty label corresponds to the top-concept.*

The *$\mathcal{EL}$-description tree* $\mathcal{G}(C)$ corresponding to the $\mathcal{EL}$-concept description C simply reflects the syntactic structure of the description. For example, the description tree corresponding to the $\mathcal{EL}$-concept description C^1_{ex} of our example is depicted on the left-hand side of Figure 1. For an $\mathcal{EL}$-concept description C and a node v in the corresponding description tree $\mathcal{G}(C)$, we denote the part of C corresponding to v by C_v. In our example, we have, for instance, $C^1_{\text{ex},v_1} = \mathsf{W} \sqcap \exists \mathsf{hc}.(\mathsf{W} \sqcap \mathsf{D}) \sqcap \exists \mathsf{hc}.(\mathsf{W} \sqcap \mathsf{P})$.

Definition 11 *A mapping* $\varphi : V_H \longrightarrow V_G$ *from an* $\mathcal{EL}$*-description tree* $\mathcal{H} = (V_H, E_H, w_0, \ell_H)$ *to an* $\mathcal{EL}$*-description tree* $\mathcal{G} = (V_G, E_G, v_0, \ell_G)$ *is called* homomorphism *iff the following conditions are satisfied:*

1. $\varphi(w_0) = v_0$,
2. *for all* $v \in V_H$ *we have* $\ell_H(v) \subseteq \ell_G(\varphi(v))$,
3. *for all* $vrw \in E_H$*, we have* $\varphi(v) r \varphi(w) \in E_G$.

Now, subsumption can be characterized as follows:

Lemma 12 *(Baader et al., 1999b) Given two* $\mathcal{EL}$*-concept descriptions* C, D*, we have* $C \sqsubseteq D$ *iff there exists a homomorphism from* $\mathcal{G}(D)$ *into* $\mathcal{G}(C)$*.*

The $\mathcal{EL}$-matching algorithm

In order to employ this lemma for deriving the $\mathcal{EL}$-matching algorithm, we need to generalize the notions introduced above to concept patterns. $\mathcal{EL}$-description trees can be extended to concept patterns by simply treating variables like concept names. For example, the concept pattern D^1_{ex} in the example yields the description tree depicted on the right-hand side of Figure 1. When *extending* the notion of a *homomorphism to description trees representing concept patterns*, we simply ignore the concept variables, i.e., the second condition must hold only for non-variable concept names.

In our example, there are six homomorphisms from $\mathcal{G}(D^1_{\text{ex}})$ into $\mathcal{G}(C^1_{\text{ex}})$. 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 φ_1 and φ_2, respectively.

Input: $\mathcal{EL}$-matching problem $C \sqsubseteq^? D$.
Output: s-complete set $\mathcal{C}$ for $C \sqsubseteq^? D$.

$\mathcal{C} := \emptyset$;
For all homomorphisms φ from
  $\mathcal{G}(D) = (V, E, v_0, \ell)$ into $\mathcal{G}(C)$ do
    Define τ by $\tau(X) := lcs\{C_{\varphi(v)} \mid X \in \ell(v)\}$
      for all variables X in D;
    $\mathcal{C} := \mathcal{C} \cup \{\tau\}$;

Figure 2: The $\mathcal{EL}$-matching algorithm

The matching algorithm described in Figure 2 constructs 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 $\mathcal{G}(C)$. The remaining problem is that a variable X may occur more than once in D. Thus, we cannot simply define $\tau(X)$ as $C_{\varphi(v)}$ where v is such that X occurs in the label of v. Since there may exist several nodes v 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 substitutions that are as small as possible w.r.t. $\sqsubseteq_s$. Recall that E is the *least common subsumer* (lcs) of $E_1, \ldots, E_n$ ($lcs(E_1, \ldots, E_n)$ for short) iff (i) E subsumes $E_1, \ldots, E_n$ and (ii) E is the least concept description w.r.t. subsumption that satisfies (i), i.e., for every concept description E', if $E' \sqsupseteq E_1, \ldots, E_n$, then $E' \sqsupseteq E$. Algorithms for computing the lcs of $\mathcal{EL}$- and $\mathcal{ALE}$-concept descriptions have been described in (Baader et al., 1999b).

In our example, the homomorphism φ_1 yields the substitution τ_1:

$$\begin{aligned} \tau_1(X) &:= lcs\{C^1_{\mathsf{ex},v_1},\, C^1_{\mathsf{ex},v_2}\} \equiv \mathsf{W} \sqcap \exists \mathsf{hc}.(\mathsf{W} \sqcap \mathsf{P}), \\ \tau_1(Y) &:= lcs\{C^1_{\mathsf{ex},v_2},\, C^1_{\mathsf{ex},v_3}\} \equiv \mathsf{W} \sqcap \mathsf{D}, \end{aligned}$$

whereas φ_2 yields the substitution τ_2:

$$\begin{aligned} \tau_2(X) &:= lcs\{C^1_{\mathsf{ex},v_1},\, C^1_{\mathsf{ex},v_2}\} \equiv \mathsf{W} \sqcap \exists \mathsf{hc}.(\mathsf{W} \sqcap \mathsf{P}), \\ \tau_2(Y) &:= lcs\{C^1_{\mathsf{ex},v_2},\, C^1_{\mathsf{ex},v_4}\} \equiv \mathsf{W}. \end{aligned}$$

The substitution τ_1 is an i-minimal matcher, but τ_2 is neither i-minimal nor s-minimal. Therefore, τ_2 will be removed in the post-processing step when extracting a *minimal* i-complete set from the computed s-complete one. By applying Lemma 12, it is easy to show:

Theorem 13 *(Baader and Küsters, 1999) The algorithm described in Figure 2 always computes an s-complete set of matchers for a given $\mathcal{EL}$-matching problem modulo subsumption.*

5.2 Computing s-complete sets in $\mathcal{ALE}$

The algorithm for computing s-complete sets for $\mathcal{ALE}$-matching problems modulo subsumption is similar to the one for $\mathcal{EL}$. However, due to inconsistent concepts expressible in $\mathcal{ALE}$ and the interaction between existential and value restrictions, things become more complicated.

As before, the algorithm is based on the characterization of subsumption via homomorphisms between description trees, which we briefly recall in the following (see (Baader et al., 1999b) for more details).

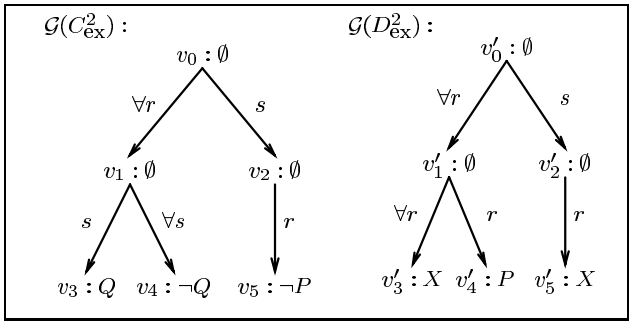

Figure 3: $\mathcal{ALE}$-description trees.

Characterizing subsumption in $\mathcal{ALE}$

The notion of $\mathcal{EL}$-description trees is generalized to $\mathcal{ALE}$ in a straightforward manner: (i) In addition to $\exists$-edges labeled with role names r, $\mathcal{ALE}$-description trees may also contain $\forall$-edges labeled $\forall r$; (ii) beside concept names and variables, labels of nodes in $\mathcal{ALE}$-description trees may also contain negated concept names as well as the bottom-concept.

Again, any $\mathcal{ALE}$-concept description/pattern C can be translated into a corresponding $\mathcal{ALE}$-description tree $\mathcal{G}(C)$. For example, the description trees $\mathcal{G}(C^2_{\mathsf{ex}})$ and $\mathcal{G}(D^2_{\mathsf{ex}})$ corresponding to the $\mathcal{ALE}$-concept description C^2_{ex} and the concept pattern D^2_{ex} defined as

$$\begin{aligned} C^2_{\mathsf{ex}} &:= \forall r.(\exists s.Q \sqcap \forall s.\neg Q) \sqcap \exists s.\exists r.\neg P, \\ D^2_{\mathsf{ex}} &:= \forall r.(\forall r.X \sqcap \exists r.P) \sqcap \exists s.\exists r.X \end{aligned}$$

are depicted in Figure 3.

In order to extend the characterization of subsumption from $\mathcal{EL}$ to $\mathcal{ALE}$, the notion of a homomorphism must be extended and concept descriptions need to be normalized before turning them into description trees.

Obviously, a homomorphism between $\mathcal{ALE}$-description trees must distinguish between $\forall$- and $\exists$-edges. More important, a homomorphism must be allowed to map a node and all its successors onto an inconsistent node, i.e., a node whose label contains $\bot$.

Definition 14 *A mapping $\varphi : V_H \longrightarrow V_G$ from an $\mathcal{ALE}$-description tree $\mathcal{H} = (V_H, E_H, w_0, \ell_H)$ to an $\mathcal{ALE}$-description tree $\mathcal{G} = (V_G, E_G, v_0, \ell_G)$ is called* homomorphism *iff the following conditions are satisfied:*

1. *$\varphi(w_0) = v_0$,*
2. *for all $v \in V_H$ we have $\ell_H(v) \subseteq \ell_G(\varphi(v))$ or $\bot \in \ell_G(\varphi(v))$,*
3. *for all $vrw \in E_H$, either $\varphi(v)r\varphi(w) \in E_G$, or $\varphi(v) = \varphi(w)$ and $\bot \in \ell_G(\varphi(v))$, and*

4. *for all* $v\forall r w \in E_H$, *either* $\varphi(v)\forall r\varphi(w) \in E_G$, *or* $\varphi(v) = \varphi(w)$ *and* $\bot \in \ell_G(\varphi(v))$.

The main purpose of the normalization rules on $\mathcal{ALE}$-concept descriptions (see below) is to make implicitly inconsistent parts of C explicit, and to propagate value restrictions onto existential restrictions and other value restrictions.

Definition 15 *Let* E, F *be two* $\mathcal{ALE}$*-concept descriptions,* $r \in N_R$*, and* $P \in N_C$*. The* $\mathcal{ALE}$-normalization rules *are defined as follows*

$$\begin{array}{rcl} P \sqcap \neg P,\ \exists r.\bot,\ E \sqcap \bot & \longrightarrow & \bot, \\ \forall r.E \sqcap \exists r.F & \longrightarrow & \forall r.E \sqcap \exists r.(E \sqcap F), \\ \forall r.E \sqcap \forall r.F & \longrightarrow & \forall r.(E \sqcap F), \\ \forall r.\top & \longrightarrow & \top. \end{array}$$

A concept description C *is called* normalized *if none of the normalization rules can be applied to some part of* C.

The rules should be read modulo commutativity of conjunction; e.g., $\exists r.E \sqcap \forall r.F$ is also normalized to $\exists r.(E \sqcap F) \sqcap \forall r.F$. An unnormalized concept description C can be normalized by exhaustively applying the normalization rules in C. The resulting (normalized) concept description is called *normal form of* C. Since each normalization rule preserves equivalence, the normal form of C is equivalent to C. The $\mathcal{ALE}$-description tree corresponding to the normal form of C is denoted by $\mathcal{G}_C$.

If only the rule $\forall r.\top \longrightarrow \top$ is exhaustively applied to a concept description C, then the resulting concept description is called $\top$*-normal form of* C, and the corresponding tree is denoted by $\mathcal{G}_C^\top$.

Now, subsumption can be characterized in terms of homomorphisms as follows:

Lemma 16 *(Baader et al., 1999b) Let* C, D *be* $\mathcal{ALE}$*-concept descriptions. Then,* $C \sqsubseteq D$ *iff there exists a homomorphism from* $\mathcal{G}_D^\top$ *to* $\mathcal{G}_C$.

It should be noted that the theorem stated in (Baader et al., 1999b) requires a homomorphism from $\mathcal{G}_D$ instead of $\mathcal{G}_D^\top$. However, a closer look at the proof in (Baader et al., 1998) reveals that $\top$-normalization of the subsumer is sufficient. This will be important in the following when we use Lemma 16 as basis for the $\mathcal{ALE}$-matching algorithm. In this context, the subsumer is an instantiation of the pattern D, and thus, normalizing the subsumer $\sigma(D)$ depends on the substitution σ, which is not known in advance. Therefore, it is crucial that only very simple normalizations of the subsumer are required.

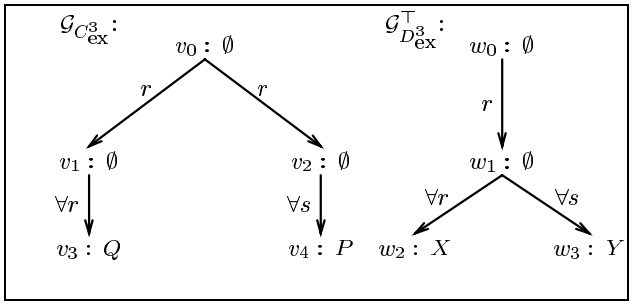

Figure 5: The description trees for C^3_{ex} and D^3_{ex}.

The $\mathcal{ALE}$-matching algorithm

Following Lemma 16, the $\mathcal{EL}$-matching algorithm is modified as follows: (i) instead of the trees $\mathcal{G}(C)$ and $\mathcal{G}(D)$, we now consider $\mathcal{G}_C$ and $\mathcal{G}_D^\top$, where the $\top$-normal form of D is obtained by treating concept variables like concept names; (ii) homomorphisms are computed with respect to Definition 14, where again variables are ignored in 2.

This straightforward extension of the $\mathcal{EL}$-matching algorithm is sufficient to solve the $\mathcal{ALE}$-matching problem $C^2_{\text{ex}} \sqsubseteq D^2_{\text{ex}}$. There exists exactly one homomorphism φ from $\mathcal{G}^\top_{D^2_{\text{ex}}}$ into $\mathcal{G}_{C^2_{\text{ex}}}$ (see Figure 4). Following the $\mathcal{EL}$-matching algorithm, φ gives rise to the matcher σ with $\sigma(X) := lcs\{\bot, \neg P\} \equiv \neg P$. The singleton set $\{\sigma\}$ is indeed an s-complete set.

However, as illustrated by the next example, this simple extension of the $\mathcal{EL}$-matching algorithm does not work in general.

Example 17 *Consider the* $\mathcal{ALE}$*-matching problem* $C^3_{ex} \sqsubseteq^? D^3_{ex}$*, where*

$$\begin{array}{rcl} C^3_{ex} & := & (\exists r.\forall r.Q) \sqcap (\exists r.\forall s.P) \\ D^3_{ex} & := & \exists r.(\forall r.X \sqcap \forall s.Y). \end{array}$$

The description trees corresponding to C^3_{ex} *and* D^3_{ex} *are depicted in Figure 5. Obviously,* $\sigma := \{X \mapsto Q, Y \mapsto \top\}$ *and* $\tau := \{X \mapsto \top, Y \mapsto P\}$ *are solutions of the matching problem. However, there is no homomorphism from* $\mathcal{G}^\top_{D^3_{ex}}$ *into* $\mathcal{G}_{C^3_{ex}}$*. Indeed, the node* w_1 *can be mapped either on* v_1 *or on* v_2*. In the former case,* w_2 *can be mapped on* v_3*, but then there is no way to map* w_3*. In the latter case,* w_3 *must be mapped on* v_4*, but then there is no node* w_2 *can be mapped on.*

The problem is that Lemma 16 requires the subsumer to be in $\top$-normal form. However, the $\top$-normal form of the instantiated concept pattern depends on the matcher, and thus cannot be computed in advance. For instance, in Example 17 the instances $\sigma(D^3_{\text{ex}})$ and

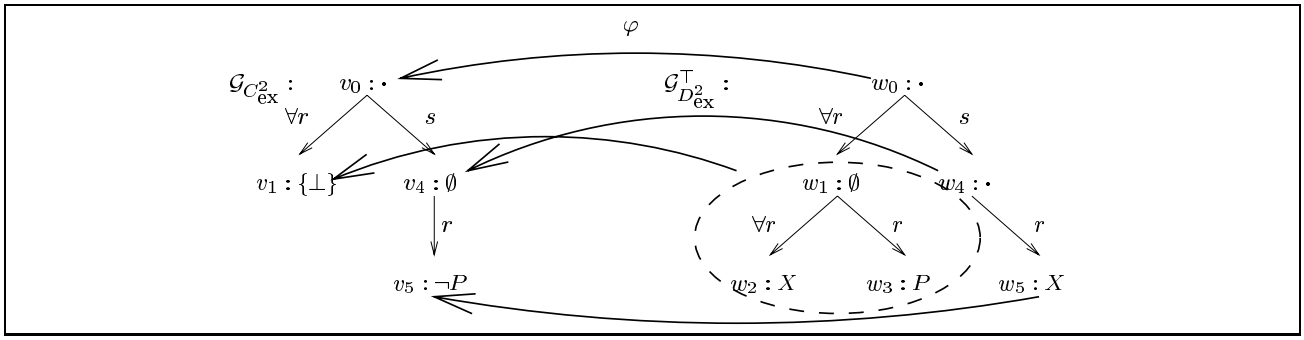

Figure 4: Subsumption for $\mathcal{ALE}$.

$\tau(D^3_{\text{ex}})$ of D^3_{ex} are not $\top$-normalized since they contain $\forall s.\top$ and $\forall r.\top$, respectively. The description tree for the $\top$-normalized concept description $\sigma(D^3_{\text{ex}})$ does not include the node w_3 and the $\forall s$-edge leading to it. Analogously, for $\tau(D)$, w_2 would be deleted.

This illustrates that the instances of a pattern are not necessarily in $\top$-normal form and that the $\top$-normal form depends on the particular instance. However, only those matchers cause problems that replace variables by the top-concept. Therefore, instead of considering homomorphisms originating from $\mathcal{G}^{\top}_D$, the algorithm computes homomorphisms from the so-called $\top$-patterns of D into $\mathcal{G}_C$.

Definition 18 *The concept pattern E is called* $\top$-pattern *of D iff E is obtained from D by replacing some of the variables in D by $\top$.*

In our example, we obtain the following $\top$-normalized $\top$-patterns for D^3_{ex}: D^3_{ex}, $\exists r.(\forall r.X)$, $\exists r.(\forall s.Y)$, and $\exists r.\top$. Matching these patterns against C^3_{ex}, the extended matching algorithm described above computes the following sets of solutions: $\emptyset$, $\{\sigma\}$, $\{\tau\}$, and $\{\{X \mapsto \top, Y \mapsto \top\}\}$. The union of these sets provides us with an s-complete set of solutions of $C^3_{\text{ex}} \sqsubseteq^? D^3_{\text{ex}}$.

The matching algorithm for $\mathcal{ALE}$ obtained from these considerations is depicted in Figure 6. Here $\ell(v)$ denotes the label of v in $\mathcal{G}^{\top}_E$, and $C^{\varphi(v)}$ stands for the concept description corresponding to the subtree of $\mathcal{G}_C$ with root $\varphi(v)$.

5.3 Complexity of computing i-complete sets

There are two different aspects to consider. First, the size of i-complete (and s-complete) sets, and second, the complexity of the algorithm for computing these sets. The following theorem shows that the size of complete sets may grow exponentially in the size of the matching problem.

Input: $\mathcal{ALE}$-matching problem $C \sqsubseteq^? D$
Output: s-complete set $\mathcal{C}$ for $C \sqsubseteq^? D$

$\mathcal{C} := \emptyset$
For all $\top$-patterns E of D do
  For all homomorphisms φ from $\mathcal{G}^{\top}_E$ into $\mathcal{G}_C$
    Define τ by
      $\tau(X) := lcs\{C^{\varphi(v)} \mid X \in \ell(v)\}$
        for all X in E and
      $\tau(X) := \top$
        for all X in D not contained in E
    $\mathcal{C} := \mathcal{C} \cup \{\tau\}$

Figure 6: The $\mathcal{ALE}$-matching algorithm.

Theorem 19 *Both, for $\mathcal{EL}$ and $\mathcal{ALE}$*

1. *the cardinality of minimal i-complete and s-complete sets of matchers and the size of the matchers in these sets are at most exponential in the size of the matching problem;*
2. *these exponential upper-bounds are tight.*

We illustrate the second part of the theorem by two examples, one for the cardinality of complete sets and one for the size of the matchers.

Example 20 *Let C_n be the $\mathcal{EL}$-/$\mathcal{ALE}$-concept description*

$$C_n := \prod_{i=1}^{n} \exists r.(\prod_{j=1}^{n} \exists r.(A_i \sqcap B_j),$$

and D_n be the $\mathcal{EL}$-/$\mathcal{ALE}$-concept pattern

$$D_n := \prod_{i=1}^{n} \exists r.\exists r.X_i.$$

For the matching problem $C_n \sqsubseteq^? D_n$ and a word $w = a_1 \cdots a_n \in \{1, \ldots, n\}^n$ of length n over the alphabet $\{1, \ldots, n\}$, the substitution $\sigma_w(X_i) := A_i \sqcap B_j$ for $a_i = j$ is obviously an i-minimal and s-minimal matcher.

Furthermore, for different words $w, w' \in \{1, \ldots, n\}^n$, one obtains i-incomparable and s-incomparable matchers. Since there are n^n such words, the number of i-minimal and s-minimal matchers grows exponentially in n, and thus in the size of the matching problems $C_n \sqsubseteq^? D_n$.

The next example demonstrates that the size of a single matcher in an s-complete/i-complete set may grow exponentially in the size of the matching problem.

Example 21 *In (Baader et al., 1999b), it has been shown that there is a sequence $E_1, E_2, \ldots$ of $\mathcal{EL}$-/$\mathcal{ALE}$-concept descriptions such that the size of $lcs(E_1, \ldots, E_n)$ grows exponentially in the size of $E_1, \ldots, E_n$.*

Now, consider the $\mathcal{EL}$-/$\mathcal{ALE}$-matching problem $C'_n \sqsubseteq^? D'_n$, where $C'_n := \exists r_1.E_1 \sqcap \cdots \sqcap \exists r_n.E_n$ and $D'_n := \exists r_1.X \sqcap \cdots \sqcap \exists r_n.X$. Clearly, for an i-minimal or s-minimal matcher σ of this matching problem, $\sigma(X) \equiv lcs(E_1, \ldots, E_n)$. Thus, $\sigma(X)$ grows exponentially in the size of the matching problem.

We now turn to the complexity of the $\mathcal{ALE}$-matching algorithm and show that it runs in exponential time. Obviously, this also implies the exponential upper-bound stated in the first part of Theorem 19. Similar arguments can be employed for the $\mathcal{EL}$-matching algorithm.

First, note that the number of $\top$-patterns E of D is at most exponential in the size of D. Also, the size of $\mathcal{G}_E^\top$ for each such $\top$-pattern is linear in the size of D. As shown in (Baader et al., 1999b), the size of $\mathcal{G}_C$ is at most exponential in the size of C. Consequently, it is easy to see that the number of homomorphisms from $\mathcal{G}_E^\top$ into $\mathcal{G}_C$ is at most exponential in the size of C and D. This shows that the number of matchers σ computed by the algorithm is at most exponential in the size of the given matching problem.

It remains to be shown that each such σ can be computed in exponential time. As proved in (Baader et al., 1999b), the lcs of n concepts $C_1, \ldots, C_n$ can be computed in time bounded by the product of the sizes of the concepts C_i. Since the size of $\mathcal{G}_C$, and thus also of $C^{\varphi(v)}$, is at most exponential in C and the number of nodes v in $\mathcal{G}_E^\top$ is linear in D, it follows that each $\sigma(X)$ can be computed in time exponential in the size of C and D. Finally, the fact that D (and thus also E) contains only a linear number of variables shows that σ can be computed in exponential time. This proves

Corollary 22 *Computing i- and s-complete sets for $\mathcal{EL}$-/$\mathcal{ALE}$-matching problems modulo subsumption can be carried out in time exponential in the size of the matching problem.*

Recall that we are actually interested in computing a *minimal* i-complete set of the matching problem $C \sqsubseteq^? D$. As mentioned at the beginning of this section, given an s-complete set, one can compute a minimal i-complete set by testing subsumption between the instances $\tau(D)$ of D, where τ belongs to the s-complete set. Since the size of these instances may be exponential in the size of the matching problem, and since subsumption is polynomial in $\mathcal{EL}$ and NP-complete in $\mathcal{ALE}$, we obtain the following complexity result for computing minimal i-complete sets:

Corollary 23 *Computing minimal i-complete sets for matching modulo subsumption can be carried out in exponential time for $\mathcal{EL}$- and in exponential space for $\mathcal{ALE}$-matching problems.*

6 Computing d-minimal matchers

In order to realize the second step of the matching algorithm sketched at the end of Section 4, we must show how to compute all d-minimal (i.e., reduced) matchers up to s-equivalence of a given matching problem modulo *equivalence*. For such a matching problem $C \equiv^? D$, sets containing at least all d-minimal matchers up to s-equivalence are called *d-complete*. Such a set is called *minimal* if it contains exactly one d-minimal matcher for each s-equivalence class.

The following theorem implies that, in the worst case, algorithms computing d-complete sets need exponential time.

Theorem 24 *Let $C \equiv^? D$ be an $\mathcal{EL}$- or $\mathcal{ALE}$-matching problem modulo equivalence. Then,*

1. *the cardinality of (minimal) d-complete sets can grow exponentially in the size of the matching problem;*
2. *however, there always exists a (minimal) d-complete set such that the size of each matcher in this set is polynomially bounded.*

The first statement of the theorem is an easy consequence of the fact that every d-complete set for the $\mathcal{EL}$-/$\mathcal{ALE}$-matching problem $\exists r.A_1 \sqcap \cdots \sqcap \exists r.A_n \equiv^? \exists r.X_1 \sqcap \cdots \exists r.X_n$ contains at least an exponential number of matchers.

The non-trivial result to prove is the second part of Theorem 24 (see (Baader and Küsters, 1999)). This result yields a naïve exponential-time algorithm for

computing d-complete sets: Enumerate all substitutions up to the polynomial bound and filter out those that are not solutions of the problem or that are not d-minimal. The filtering can be realized by a polynomial time algorithm using an oracle for subsumption. Obviously, this naïve algorithm is very inefficient.

For $\mathcal{EL}$, we will sketch an improved exponential time algorithm, which significantly prunes the search space for candidate matchers. This algorithm is based on the following (non-trivial) lemma:

Lemma 25 *(Baader and Küsters, 1999) The matcher σ of the $\mathcal{EL}$-matching problem $C \equiv^? D$ is d-minimal iff it is s-maximal and $\sigma(X)$ is reduced for all variables X.*

Here s-maximality of matchers is defined w.r.t. $\sqsubseteq_s$ in the obvious way. Note that this lemma does not hold for $\mathcal{ALE}$, and thus it is not clear how to extend our approach from $\mathcal{EL}$ to $\mathcal{ALE}$.

Because of the lemma, the task of computing a d-complete set of matchers in $\mathcal{EL}$ can be split into two subtasks. First, compute an *s-co-complete* set of matchers, i.e., a set containing all s-maximal matchers up to s-equivalence. Second, for every matcher σ in the set and every variable X, compute a reduced concept description equivalent to $\sigma(X)$. In (Baader and Küsters, 1999), it is shown that, for $\mathcal{EL}$, the second task can be realized by a polynomial time algorithm. In the following, we sketch an algorithm for performing the first task (see (Baader and Küsters, 1999) for a complete description and proof of correctness). Roughly speaking, this algorithm is the dual of the one in Figure 2. The duality occurs at two places in the algorithm.

First, instead of computing substitutions τ satisfying $C \sqsubseteq \tau(D)$, we now compute substitutions τ that satisfy $C \sqsupseteq \tau(D)$. To make sure that the substitutions computed by the algorithm really solve the matching problem $C \equiv^? D$, we use the subsumption algorithm for $\mathcal{EL}$ to remove those substitutions not satisfying $C \sqsubseteq \tau(D)$. In order to obtain substitutions τ satisfying $C \sqsupseteq \tau(D)$, we now consider homomorphisms in the other direction, i.e., from $\mathcal{G}(C)$ into $\mathcal{G}(D)$. To be more precise, we consider homomorphisms ψ that are *partial* in the following sense: (i) certain nodes of $\mathcal{G}(C)$ need not be mapped onto nodes of $\mathcal{G}(D)$; and (ii) for certain nodes the inclusion condition between labels need not hold. The idea is that the parts of C that are not mapped and the labels violating the inclusion condition are covered by the concepts substituted for the variables. For this reason, a partial homomorphism implicitly associates with each variable a set of concepts that must be covered by this variable. (Note that, for a given partial homomorphism, there are different ways of associating concepts with variables.)

The second duality occurs when defining the substitution τ: in the definition of $\tau(X)$, the lcs is replaced by conjunction of the concepts associated with X. The exact definition of partial homomorphisms is such that a given partial homomorphism can always be extended to a (total) homomorphism φ from $\mathcal{G}(C)$ into $\mathcal{G}(\tau(D))$ for the constructed substitution τ. A detailed description of the algorithm can be found in (Baader and Küsters, 1999). Here, we only illustrate it by the matching problem $C^1_{\text{ex}} \equiv^? D^1_{\text{ex}}$. For instance, $\psi := \{v_0 \mapsto w_0, v_1 \mapsto w_1, v_2 \mapsto w_2, v_3 \mapsto w_3\}$ is a partial homomorphism from $\mathcal{G}(C^1_{\text{ex}})$ into $\mathcal{G}(D^1_{\text{ex}})$. Here the nodes v_4 and v_5 are not mapped by ψ. The "missing parts" can be covered by associating with X the concepts W and $\exists \mathsf{hc}.(\mathsf{W} \sqcap \mathsf{P})$, and with Y the concept D. In addition, it is not hard to verify that the substitution $\tau := \{X \mapsto \mathsf{W} \sqcap \exists \mathsf{hc}.(\mathsf{W} \sqcap \mathsf{P}), Y \mapsto \mathsf{D}\}$ also satisfies $C^1_{\text{ex}} \sqsubseteq \tau(D^1_{\text{ex}})$. Thus, τ is a matcher in the computed s-co-complete set.

A sketch of the proof of Lemma 4

It is easy to see that, by construction, the matchers in the s-co-complete set computed by the algorithm sketched above are of size polynomial in the size of the matching problem. This shows Lemma 4 for $\mathcal{EL}$, since an s-co-complete set can only be empty if the matching problem is not solvable.

For $\mathcal{ALE}$, things are more complicated. In the sequel, let σ' be an arbitrary matcher of the $\mathcal{ALE}$-matching problem $C \equiv^? D$ (where, without loss of generality, $\sigma'(X)$ is in $\forall$-normal form for every variable X). The task is to construct from σ' a new matcher σ of size polynomially bounded in the size of the matching problem. In the following, let C^r be the reduced concept description equivalent to C. Note that the size of C^r is linear in the size of C. Furthermore, let E' be the $\forall$-normal form of $\sigma'(D)$.

First, assume that C^r does not contain $\bot$. As an easy consequence of Lemma 7, we can show that there exists an injective homomorphism ψ from $\mathcal{G}(C^r)$ into $\mathcal{G}(E')$. Let $\mathcal{G}'$ be the image of $\mathcal{G}(C^r)$ under ψ. In principle, σ is obtained from σ' by removing from each description $\sigma'(X)$ all the parts that are not needed to obtain $\mathcal{G}'$. Consequently, the size of $\sigma(X)$ is bounded by the size of C^r, and thus σ is polynomial in the size of the matching problem.

It remains to be shown that σ solves the matching problem $C \equiv^? D$. Let E be the $\forall$-normal form of $\sigma(D)$.

The construction of σ ensures that ψ is still an injective homomorphism from $\mathcal{G}(C^r)$ into $\mathcal{G}(E)$. This implies that $\sigma(D) \equiv E \sqsubseteq C^r \equiv C$. Conversely, the definition of σ also implies $\sigma' \sqsubseteq_s \sigma$, and thus $C \equiv \sigma'(D) \sqsubseteq \sigma(D)$.

If C^r contains $\bot$, then each description $\sigma(X)$ must be extended by those parts of $\mathcal{G}(E')$ that contribute to inconsistencies. In (Baader and Küsters, 1999), these parts are identified and it is shown that they can be chosen such that the size of σ can still be polynomially bounded by the size of the matching problem.

7 Future work

The remaining technical challenge is to design practical algorithms for computing d-minimal matchers for $\mathcal{ALE}$, and for $\mathcal{ALN}$ and its extension to the CLASSIC description language.

We will also evaluate the usefulness of matching for removing redundancies in knowledge bases within our process engineering application (Baader and Sattler, 1996). In order to apply matching to the problem of integrating knowledge bases (Borgida and Küsters, 1999), we first need to extend the matching algorithm to an algorithm that takes schemas (i.e., certain types of inclusion axioms) into account.

References

Baader, F. and Küsters, R. (1999). Matching in Description Logics with Existential Restrictions Revisited. Tech. Report LTCS-Report 99-13, LuFg Theoretical Computer Science, RWTH Aachen, Germany. See http://www-lti.informatik.rwth-aachen.de/Forschung/Reports.html.

Baader, F., Küsters, R., Borgida, A., and McGuinness, D. (1999a). Matching in description logics. *Journal of Logic and Computation*, 9(3):411–447.

Baader, F., Küsters, R., and Molitor, R. (1998). Computing Least Common Subsumer in Description Logics with Existential Restrictions. Tech. Report LTCS-Report 98-09, LuFg Theoretical Computer Science, RWTH Aachen, Germany. See http://www-lti.informatik.rwth-aachen.de/Forschung/Reports.html.

Baader, F., Küsters, R., and Molitor, R. (1999b). Computing least common subsumer in description logics with existential restrictions. In Dean, T., editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 96–101. Morgan Kaufmann.

Baader, F. and Narendran, P. (1998). Unification of concept terms in description logics. In *Proceedings of the 13th Biennial European Conference on Artificial Intelligence (ECAI-98)*. Brighton, UK.

Baader, F. and Sattler, U. (1996). Knowledge representation in process engineering. In *Proceedings of the International Workshop on Description Logics*, Cambridge (Boston), MA, U.S.A. AAAI Press/The MIT Press.

Borgida, A., Brachman, R. J., McGuinness, D. L., and Resnick, L. A. (1989). CLASSIC: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67, Portland, OR.

Borgida, A. and Küsters, R. (1999). What's not in a name? Initial explorations of a structural approach to integrating large concept knowledge-bases. Technical Report DCS-TR-391, Rutgers University, USA. Available via ftp://ftp.cs.rutgers.edu/pub/technical-reports/.

Borgida, A. and McGuinness, D. L. (1996). Asking queries about frames. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 340–349, San Francisco, Calif. Morgan Kaufmann.

Brachman, R. J., McGuinness, D. L., Patel-Schneider, P. F., Resnick, L. A., and Borgida, A. (1991). Living with CLASSIC: When and how to use a KL-ONE-like language. In Sowa, J., editor, *Principles of Semantic Networks*, pages 401–456. Morgan Kaufmann, San Mateo, Calif.

Cohen, W., Borgida, A., and Hirsh, H. (1992). Computing least common subsumers in description logics. In Swartout, W., editor, *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 754–760, San Jose, CA. MIT Press.

Donini, F., Hollunder, B., Lenzerini, M., Marchetti, A., Nardi, D., and Nutt, W. (1992). The complexity of existential quantification in concept languages. *Artificial Intelligence*, 2–3:309–327.

McGuinness, D. (1996). *Explaining Reasoning in Description Logics*. PhD thesis, Department of Computer Science, Rutgers University. Also available as a Rutgers Technical Report LCSR-TR-277.