RWTH
LTCS–Report

Aachen University of Technology
Research group for
Theoretical Computer Science

Matching Concept Descriptions with Existential Restrictions

Franz Baader and Ralf Küsters

LTCS-Report 99-07

RWTH Aachen
LuFg Theoretische Informatik
http://www-lti.informatik.rwth-aachen.de

Ahornstr. 55
52074 Aachen
Germany

Matching Concept Descriptions with Existential Restrictions

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

Abstract

Matching of concepts with variables (concept patterns) is a relatively new operation that has been introduced in the context of description logics, originally to help filter out unimportant aspects of large concepts appearing in industrial-strength knowledge bases. Previous work has concentrated on (sub-)languages of CLASSIC, which in particular do not allow for existential restrictions. In this work, we present sound and complete decision algorithms for the solvability of matching problems and for computing sets of matchers for matching problems in description logics with existential restrictions.

1 Introduction

Knowledge representation systems based on Description Logic (DL systems) can be used to represent the terminological knowledge of an application domain in a structured and formally well-understood way. In such systems, the important notions of the domain can be described by *concept descriptions*, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept constructors provided by the Description Logic language (DL language) of the system. The atomic concepts and the concept descriptions represent sets of individuals, whereas roles represent binary relations between individuals. For example, using the atomic concept Woman and the atomic role child, the concept of all *women having only daughters* (i.e., women such that all their children are again women) can be represented by the concept description

$$\mathsf{Woman} \sqcap \forall \mathsf{child}.\mathsf{Woman}.$$

DL systems provide their users with various inference capabilities that allow them to deduce implicit knowledge from the explicitly represented knowledge. For instance, the *subsumption* algorithm allows one to determine subconcept-superconcept relationships: C is subsumed by D ($C \sqsubseteq D$) if and only if all instances of C are also instances of D, i.e., the first description is always interpreted as a subset of the second description. For example, the concept descrip-

tion Woman obviously subsumes the concept description Woman⊓∀child.Woman. With the help of the subsumption algorithm, a newly introduced concept description can automatically be placed at the correct position in the hierarchy of the already existing concept descriptions. Two concept descriptions C, D are *equivalent* ($C \equiv D$) if and only if they subsume each other, i.e., if and only if they always represent the same set of individuals. For example, the descriptions Woman ⊓ ∀child.Woman and (∀child.Woman) ⊓ Woman are equivalent since ⊓ is interpreted as set intersection, which is commutative.

The traditional inference problems for DL systems (like subsumption) are now well-investigated, which means that algorithms are available for solving the subsumption problem and related inference problems in a great variety of DL languages of differing expressive power.

It has turned out, however, that building and maintaining large DL knowledge bases requires additional support in the form of inferences that have not been considered in the DL literature until very recently. The present paper is concerned with one such new inference service, namely, *matching* of concept descriptions.

Matching of Description Logics concepts has been introduced in [6, 10] as a declarative approach for selective viewing of components of complex concept and object descriptions. In projects based on the DL system CLASSIC [7], where such pruning of descritptions was required (e.g., for explaining results provided by the more traditioinal inference services [11, 10]), the resulting description was approximately an order of magnitude smaller than the unpruned one. In small applications such as [12], this actually saved 3-5 pages of printout; in larger applications such as [14, 13] it might save up to 30 pages.

Given a concept pattern D (i.e., a concept description containing variables) and a concept description C without variables, a *matching problem modulo subsumption*, $C \sqsubseteq^? D$, asks for a substitution σ (of the variables by concept descriptions) such that $C \sqsubseteq \sigma(D)$. More precisely, one is interested in a matcher σ such that the instance $\sigma(D)$ of D is as small as possible, i.e., σ should satisfy the property that there does not exist a substitution δ such that $C \sqsubseteq \delta(D) \sqsubset \sigma(D)$. In other words, one is interested in a minimal matcher with respect two the following ordering on matchers: For two matchers σ, τ of the matching problem $C \sqsubseteq^? D$, we say that σ i-subsumes τ ($\sigma \sqsubseteq_i \tau$, instance subsumes) iff $\sigma(D) \sqsubseteq \tau(D)$. A minimal matcher w.r.t. $\sqsubseteq_i$ is called *i-minimal*. For example, the i-minimal matcher of the pattern

$$D := \forall \text{research-interests}.X$$

against the description

$$C := \forall \text{pets}.\text{Cat} \sqcap \forall \text{research-interests}.\text{AI} \sqcap \forall \text{hobbies}.\text{Gardening}$$

assigns AI to the variable X, and thus finds the scientific interests (in this case Artificial Intelligence) described in the concept. (The concept pattern can be thought of as a "format statement", describing what information is to be

displayed (or explained), if the pattern matches successfully against a specific concept. If there is no match, nothing is displayed.)

Another motivation for matching comes from an application in chemical process engineering [4]. In this application, the DL system is used to support the design of a large terminology of concepts describing parts of chemical plants as well as processes that take place in these plants. Since several knowledge engineers are involved in defining new concepts, and since this knowledge acquisition process takes rather long (several years), it happens that the same (intuitive) concept is introduced several times, often with slightly different descriptions. Our goal was to use the reasoning capabilities of the DL system (in particular, testing for equivalence of concept terms) to support avoiding this kind of redundancy. However, 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. For example, assume that one knowledge engineer wants to defined a device that has two parts, namely, a thermounit and a reactor with Cooling-Jacket, by the concept description

$$D := \mathsf{Device} \sqcap \exists \mathsf{has\text{-}part}.\mathsf{TU} \sqcap \exists \mathsf{has\text{-}part}.(\mathsf{Reactor\text{-}with\text{-}CJ}).$$

A second engineer might have already defined this notion in a somewhat more fine-grained way as

$$\begin{aligned} C \quad := \quad & \mathsf{Device} \sqcap \exists \mathsf{has\text{-}part}.\mathsf{Thermounit} \sqcap \\ & \exists \mathsf{has\text{-}part}.(\mathsf{Reactor} \sqcap \exists \mathsf{connected\text{-}to}.\mathsf{Cooling\text{-}Jacket}). \end{aligned}$$

Since C and D are not equivalent, the first knowledge engineer would not realize that D was already defined by simply employing an equivalence test. However, by declaring the concept names Reactor-with-CJ and TU as concept variables, the engineer could now ask for a matcher of the matching problem $C \equiv^? D$. This problem is called *matching modulo equivalence* and, analogously to matching modulo subsumption, a matcher σ of this problem must satisfy $C \equiv \sigma(D)$. In our example, the engineer would obtain two possilbe matchers: one matcher maps TU onto Thermounit and Reactor-with-CJ onto Reactor$\sqcap\exists$connected-to.Cooling-Jacket and the other one maps TU onto Reactor$\sqcap$ $\exists$connected-to.Cooling-Jacket and Reactor-with-CJ onto Thermounit. The first matcher then indicates that C and D intuitively describe the same notion.

Note that since σ is a matcher of $C \sqsubseteq^? D$ iff it is one of $C \equiv^? C \sqcap D$, matching modulo subsumption is a special case of matching modulo equivalence.

A polynomial-time algorithm for computing an i-minimal matcher of a matching problem modulo subsumption for a rather expressive DL (extending $\mathcal{ALN}$ by existential restrictions and some other operators, where $\mathcal{ALN}$ allows for the top-concept ($\top$), bottom-concept ($\bot$), conjunction ($\sqcap$), atomic negation ($\neg A$ for concept names A), and value restriction ($\forall r.C$)) was introduced in [6]. The main drawback of this algorithm is that it requires the concept pattern to be in structural normal form, and thus it cannot handle arbitrary matching problems. In

addition, due to an incomplete treatment of the top- and the bottom-concepts, it does not always find a matcher, even if one exists.

For the DL $\mathcal{ALN}$, a polynomial-time matching algorithm that applies to arbitrary matching problems and always computes an i-minimal matcher (if the problem is solvable at all) was presented in [1]. To be more precise, this algorithm solves matching problems modulo equivalence, and moreover, the matcher σ computed by the algorithm is the *least* matcher w.r.t. so called *s-subsumption* $\sqsubseteq_s$ of substitutions, where $\sigma \sqsubseteq_s \tau$ iff $\sigma(X) \sqsubseteq \tau(X)$ for all variables X. Note that the least matcher is also i-minimal since $\sigma \sqsubseteq_s \tau$ implies $\sigma(D) \sqsubseteq \tau(D)$.

The purpose of this work is to transfer these results to DLs allowing for existential restrictions ($\exists R.C$), which are needed in many applications, e.g., process engineering. In order to get a feel for the new problems caused by existential restrictions, we start with the small DL $\mathcal{EL}$, which allows for the constructors top-concept, conjunction, and existential restriction. The results obtained for $\mathcal{EL}$ are then extended to the DL $\mathcal{ALE}$, which additionally allows for the constructors bottom-concept, atomic negation, and value restriction.

In contrast to the case for $\mathcal{ALN}$, solvable $\mathcal{EL}$-matching problems and $\mathcal{ALE}$-matching problems need not have a unique s-minimal (i.e., minimal w.r.t. $\sqsubseteq_s$) or i-minimal matcher, as illustrated by the following example: the $\mathcal{EL}$-matching problem $\exists R.A \sqcap \exists R.B \sqsubseteq^? \exists R.X$ has two (s-incomparable) s-minimal solutions $\sigma := \{X \mapsto A\}$ and $\tau := \{X \mapsto B\}$ leading to two (i-incomparable) instances $\sigma(\exists R.X) = \exists R.A$ and $\tau(\exists R.X) = \exists R.B$ of the pattern. (Note that $\{X \mapsto A \sqcap B\}$ is not a solution of the matching problem.) The example presented above for the chemical process engineering application shows that for matching modulo equivalence there might exist more than one s-minimal matcher as well.

The matching algorithm proposed in this work computes a finite *s-complete* set of matchers, i.e., a set $\mathcal{C}$ of solutions of the given matching problem such that, for every matcher τ of the problem, there exists $\sigma \in \mathcal{C}$ with $\sigma \sqsubseteq_s \tau$. Such a set contains all s-minimal and i-minimal matchers, which—given such a set—can be found using the subsumption algorithms for $\mathcal{EL}$ and $\mathcal{ALE}$.

In the following section, we will formally introduce the notions needed in this work. In particular, we will define matching and state fundamental properties. As already mentioned, we then present the matching algorithm and complexity results for $\mathcal{EL}$, which are extended to $\mathcal{ALE}$ in the subsequent section. Finally, we highlite the differences between matching in $\mathcal{ALN}$ and matching in $\mathcal{ALE}$ and comment on future work.

2 Preliminaries

In this section, we introduce the basic notions used in this work and state some fundamental properties for matching in $\mathcal{ALE}$ which have already been presented in [1] for the language $\mathcal{ALN}$. In addition, we will argue that to describe the solutions of a matching problem we need the notion of a so-called complete set of matchers.

Construct name	Syntax	Semantics
primitive concept $P \in N_C$	P	$P^I \subseteq \Delta$
top-concept	$\top$	Δ
conjunction	$C \sqcap D$	$C^I \cap D^I$
existential restr. for $r \in N_R$	$\exists r.C$	$\{x \in \Delta \mid \exists y : (x,y) \in r^I \wedge y \in C^I\}$
value restr. for $r \in N_R$	$\forall r.C$	$\{x \in \Delta \mid \forall y : (x,y) \in r^I \rightarrow y \in C^I\}$
primitive negation, $P \in N_C$	$\neg P$	$\Delta \setminus P^I$
bottom-concept	$\bot$	$\emptyset$

Table 1: Syntax and semantics of concept descriptions.

The Language $\mathcal{ALE}$ and Sublanguages

Concept descriptions are inductively defined with the help of a set of *constructors*, starting with a set N_C of *concept names* and a set N_R of *role names*. The constructors determine the expressive power of the DL. In this work, 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 $\mathcal{ALE}$ allows for all the constructors shown in Table 1. In the following, we refer to concept descriptions in the languages $\mathcal{EL}$ or $\mathcal{ALE}$ by $\mathcal{EL}$- and $\mathcal{ALE}$-concept descriptions, respectively.

The semantics of a concept description is defined in terms of an *interpretation* $I = (\Delta, \cdot^I)$. The domain Δ of I is a non-empty set of individuals 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 inductively defined, as shown in the third column of Table 1.

Subsumption, Equivalence, and Least Common Subsumer

One of the most important traditional inference services provided by DL systems is computing the subsumption hierarchy.

Definition 1 Let C, D be concept descriptions.

- D *subsumes* C (for short $C \sqsubseteq D$) iff $C^I \subseteq D^I$ for all interpretations I.
- C is *equivalent* to D (for short $C \equiv D$) iff $C \sqsubseteq D$ and $D \sqsubseteq C$, i.e., $C^I = D^I$ for all interpretations I.
- D *strictly subsumes* C (for short $C \sqsubset D$) iff $C \sqsubseteq D$ and $C \not\equiv D$.

As shown in [8], deciding subsumption of $\mathcal{ALE}$-concept descriptions is NP-complete. In $\mathcal{EL}$ subsumption can be decided in time polynomial in the size of the concept descriptions [3, 2].

As it turns out, in order to solve a matching problem we need to be able to compute the *least common subsumer* (lcs) of concept descriptions.

Definition 2 Let $C_1, \ldots, C_n$ and C be concept descriptions in a DL $\mathcal{L}$. The concept description C is a *least common subsumer* (lcs) of $C_1, \ldots, C_n$ (for short $C = lcs(C_1, \ldots, C_n)$) iff

1. $C_i \sqsubseteq C$ for all $1 \leq i \leq n$, and
2. C is the least concept description with this property, i.e., if C' is a concept description satisfying $C_i \sqsubseteq C'$ for all $1 \leq i \leq n$, then $C \sqsubseteq C'$.

Matching — Introduction and Fundamental Properties

We first present the different notions needed for matching. Then we argue that for $\mathcal{ALE}$, unlike $\mathcal{ALN}$, a *set* of "interesting" matchers has to be computed in contrast to only one matcher.

In order to define matching of concept descriptions, we must introduce the notion of a concept pattern and of substitutions operating on patterns. For this purpose, we introduce an additional set of symbols $\mathcal{X}$ (concept variables), which is disjoint from $N_C \cup N_R$.

Definition 3 The set of all $\mathcal{ALE}$-concept patterns over N_C, N_R, $\mathcal{X}$ is inductively defined as follows:

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

Concept patterns for sublanguages of $\mathcal{ALE}$ are defined analogously. Later on, we will need a certain normalform of concept patterns. We therefore have to define equivalence of concept patterns: Two concept patterns C, D are called *equivalent* ($C \equiv D$ for short) if and only if C and D are equivalent concept descriptions where variables are considered to be concept names.

The following notions can be restricted to sublanguages of $\mathcal{ALE}$ as well. 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 obvious way, i.e.,

- $\sigma(E) := E$ for all $E \in N_C$,
- $\sigma(\top) := \top$ and $\sigma(\bot) := \bot$,
- $\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D)$,
- $\sigma(\forall R.C) := \forall R.\sigma(C)$ and $\sigma(\exists R.C) := \exists R.\sigma(C)$, $R \in N_R$.

For example, applying the substitution $\sigma := \{X \mapsto E \sqcap \forall R.E,\ Y \mapsto F\}$ to the pattern $X \sqcap Y \sqcap \forall R.X$ yields the description $E \sqcap (\forall R.E) \sqcap F \sqcap \forall R.(E \sqcap \forall R.E)$.

Obviously, the result of applying a substitution to an $\mathcal{ALE}$-concept pattern is an $\mathcal{ALE}$-concept description.

For $\mathcal{ALE}$, and more generally, as already mentioned in [1], for any description language in which variables in patterns may only occur in the scope of "monotonic" opertors, one can easily show

Lemma 4 Let D be a $\mathcal{ALE}$-concept pattern and let σ, τ be two substitutions such that $\sigma(X) \sqsubseteq \tau(X)$ for all variables X occurring in D. Then, $\sigma(D) \sqsubseteq \tau(D)$.

Definition 5 An *$\mathcal{ALE}$-matching problem* is of the form $C \equiv^? D$ where C is an $\mathcal{ALE}$-concept description and D is an $\mathcal{ALE}$-concept pattern. A *solution* or *matcher* of this problem is a substitution σ such that $C \equiv \sigma(D)$. A matching problem is said to be *solvable* if there exists a solution.

Instead of a single matching problem, we may also consider a finite system $\{C_1 \equiv^? D_1, \ldots, C_m \equiv^? D_m\}$ of such problems. The substitution σ is a solution of this system if and only if it is a solution of all the matching problems $C_i \equiv^? D_i$ contained in the system. However, as already stated in [1] for the language $\mathcal{ALN}$, solving systems of matching problems can be reduced (in linear time) to solving a single matching problem. The proof of this result can be extended to $\mathcal{ALE}$ where instead of value-restrictions one can also use existential restrictions to simulate a set of matching problems.

Lemma 6 Let $R_1, \ldots, R_m$ be distinct atomic roles. Then σ solves the system $\{C_1 \equiv^? D_1, \ldots, C_m \equiv^? D_m\}$ if and only if it solves the single matching problem

$$\forall R_1.C_1 \sqcap \cdots \sqcap \forall R_m.C_m \equiv^? \forall R_1.D_1 \sqcap \cdots \sqcap \forall R_m.D_m.$$

Consequently, we may (without loss of generality) restrict our attention to single matching problems.

In [6, 10, 1] a different type of matching problems has been considered. We will refer to those problems as *matching problems modulo subsumption* in order to distinguish them from the *matching problems modulo equivalence* introduced above.

Definition 7 A *matching problem modulo subsumption* is of the form $C \sqsubseteq^? D$ where C is a concept description and D is a pattern. A solution of this problem is a substitution σ satisfying $C \sqsubseteq \sigma(D)$.

For any description language allowing conjunction of concepts, matching modulo subsumption can be reduced (in linear time) to matching modulo equivalence:

Lemma 8 The substitution σ solves the matching problem $C \sqsubseteq^? D$ if and only if it solves $C \equiv^? C \sqcap D$.

For $\mathcal{ALE}$, and, as mentioned in [1], more generally for any description language in which variables in patterns may only occur in the scope of "monotonic" operators, solvability of matching problems modulo subsumption can be reduced to subsumption:

Lemma 9 Let $C \sqsubseteq^? D$ be a matching problem modulo subsumption in $\mathcal{ALE}$, and let $\sigma_\top$ be the substitution that replaces each variable by $\top$. Then $C \sqsubseteq^? D$ has a solution if and only if $C \sqsubseteq \sigma_\top(D)$.

Thus, solvability of matching problems modulo subsumption in $\mathcal{ALE}$ is not an interesting new problem. However, this changes if one is not interested in an arbitrary solution of the matching problem $C \sqsubseteq^? D$, but rather in computing "minimal" solutions.

The notion "minimal" depends on the ordering chosen for solutions. In this work, we consider two different quasi-orderings, i.e., orderings that are reflexive and transitiv.

In unification theory (see e.g., [5]), the same situation occurs: one is interested in a set of minimal unifiers for a given unification problem. The orderings on unifiers used in unification theory are quasi-orderings as well. It has turned out that so-called minimal complete sets exactly represent the set of minimal solutions we are interested in.

Before returning to matching, we first present the abstract notions used for describing sets of solutions introduced in unifications theory (for details see, e.g., [5]).

In the following let $\sqsubseteq_q$ be a quasi-ordering on a set S (of solutions). Let $\equiv_q$ be the equivalence relation induced by $\sqsubseteq_q$, i.e., for all $x, y \in S$, $x \equiv_q y$ if and only if $x \sqsubseteq_q y$ and $y \sqsubseteq_q x$. The strict ordering $\sqsubset_q$ of $\sqsubseteq_q$ is defined as usual: $x \sqsubset y$ if and only if $x \sqsubseteq_q y$ and $x \not\equiv_q y$.

Definition 10 An element $s \in S$ is called *q-minimal* if and only if for all $s' \in S$, $s' \sqsubseteq_q s$ implies $s' \equiv_q s$.

A minimal complete set of solutions, mentioned above, is now defined as follows:

Definition 11 A subset $\mathcal{C} \subseteq S$ of S is called *q-complete* if and only if for all elements $s \in S$ there exists an element $s' \in \mathcal{C}$ such that $s' \sqsubseteq_q s$. Furthermore, $\mathcal{C}$ is called *miminal q-complete* if and only if $\mathcal{C}$ is complete and any two distinct elements in $\mathcal{C}$ are incomparable, i.e., for all $s, s' \in \mathcal{C}$, $s \sqsubseteq_q s'$ implies $s = s'$.

Later on we will need the following lemma in order to compare two quasi-orderings.

Lemma 12 If $\sqsubseteq_q$ and $\sqsubseteq_p$ are two quasi-orderings over S, then $\sqsubseteq_q \subseteq \sqsubseteq_p$ implies that every q-complete set is also p-complete.

PROOF. Let $\mathcal{C}$ be a q-complete set and let $s \in S$. Thus, there exists an element $s' \in \mathcal{C}$ such that $s' \sqsubseteq_q s$. We know that then $s' \sqsubseteq_p s$ which shows that $\mathcal{C}$ is p-complete. ∎

In order to obtain a characterization of minimal q-complete sets, we now consider q-equivalence classes over S. For an element $s \in S$, its q-equivalence class is defined as usual as $[s]_q := \{s' \in S \mid s \equiv_q s'\}$. Then, $\overline{S} := \{[s]_q \mid s \in S\}$ denotes the set of q-equivalence classes of S. The ordering $\sqsubseteq_q$ can be extended to $\overline{S}$ as follows: $[s]_q \leq_q [s']_q$ iff $s \sqsubseteq_q s'$. Now, $\leq_q$ is a partial ordering over $\overline{S}$. The notions $\leq_q$-*minimal*, $\leq_q$-complete, and minimal $\leq_q$-complete are defined analogously to Definition 10 and Definition 11. As shown in [5], a minimal $\leq_q$-complete set can equivalently be defined as a complete set which is minimal among all complete sets with respect to set inclusion.

The following lemma describes the connection between minimal $\leq_q$-complete sets and the set of all $\leq_q$-minimal elements in $\overline{S}$ [5].

Lemma 13 Let M be the set of $\leq_q$-minimal elements of $\overline{S}$. Then:

1. If $\mathcal{C} \subseteq \overline{S}$ is a minimal $\leq_q$-complete set, then $\mathcal{C} = M$.
2. If M is $\leq_q$-complete, then it is minimal $\leq_q$-complete.

This means that if an minimal $\leq_q$-complete set exists then this set is exactly the set of $\leq_q$-minimal elements. And on the other hand, if the set of $\leq_q$-minimal elements is not complete then there is no minimal $\leq_q$-complete set.

An easy consequence of this lemma is the following theorem [5]:

Theorem 14 Let M be the set of all $\leq_q$-minimal elements of $\overline{S}$. If $\mathcal{C}$ is a minimial q-complete set over S, then $M = \{[s]_q \mid s \in \mathcal{C}\}$. Conversely, if M is $\leq_q$-complete, then any set of representatives of M is a minimal q-complete set over S.

From this theorem one can conclude that there exists a minimal q-complete set over S if and only if M is $\leq_q$-complete. Furthermore, the minimal q-complete sets in S are unique up to q-equivalence.

We now can apply the results stated above to matching. If, for a given matching problem, S is the set of matchers of this problem and $\sqsubseteq_q$ is a quasi-ordering on these matchers, then Theorem 14 shows that a minimal q-complete set defined in Definition 11 exactly represents the q-minimal solutions of the matching problem.

In the remainder of this section we proceed as follows: We introduce the two different quasi-orderings, mentioned above, for solutions of a matching problem $C \sqsubseteq^? D$. Then we will argue that for both orderings a minimal complete set of solutions might contain more than one (minimal) solution. Along the way we will compare the two orderings defined for solutions and single out the one that is most suitable to describe all "interesting" solutions.

Roughly speaking, solutions can be compared according to the instances $\sigma(D)$ which they induce or by the substitutions themselves. In the following definition we formally define these orderings.

Definition 15 Let $C \sqsubseteq^? D$ be a matching problem. And let σ and τ be solutions of this problem. Then we define

1. σ is *s-subsumed* ("s" for "substitution") by τ ($\sigma \sqsubseteq_s \tau$) if and only if $\sigma(X) \sqsubseteq \tau(X)$ for all variables $X \in \mathcal{X}$ where $\mathcal{X}$ is the set of variables in D.[1]

2. σ is *i-subsumed* ("i" for "instance") by τ ($\sigma \sqsubseteq_i \tau$) if and only if $\sigma(D) \sqsubseteq \tau(D)$.

The notions s-equivalence, s-minimal, and (minimal) s-complete for $\sqsubseteq_s$ as well as i-equivalence, i-minimal, and (minimal) i-complete for $\sqsubseteq_i$ are defined as for the quasi-ordering $\sqsubseteq_q$.

Some examples of matching problems shall now illustrate the relationship between the different orderings.

The first example shows that an s-minimal solution is not necessarily i-minimal.

Example 16 For the matching problem

$$\exists R.A \sqcap \exists R.B \quad \sqsubseteq^? \quad \exists R.X \sqcap \exists R.Y$$

the matcher $\sigma := \{X \mapsto A, Y \mapsto A\}$ is s-minimal, but not i-minimal since with $\tau := \{X \mapsto A, Y \mapsto B\}$ it is $\tau \sqsubset_i \sigma$.

In addition, i-minimality does not imply s-minimality as shown in the next example.

Example 17 For the matching problem

$$\exists R.A \quad \sqsubseteq^? \quad \exists R.A \sqcap \exists R.X$$

the matcher $\sigma := \{X \mapsto \top\}$ is i-minimal, but not s-minimal since $\tau := \{X \mapsto A\}$ is a solution of this problem as well, and moreover, $\tau \sqsubset \sigma$.

As already mentioned in the introduction, for the orderings of choice there might be different, i.e., uncomparable, minimal solutions for a matching problem which implies that minimal complete sets might contain more than one solution.

In Example 16, the matcher $\sigma := \{X \mapsto A, Y \mapsto B\}$ and $\tau := \{X \mapsto B, Y \mapsto A\}$ are both s-minimal, but they are not s-equivalent and therefore uncomparable.

Furthermore, although $\sigma := \{X \mapsto A\}$ and $\tau := \{X \mapsto B\}$ are i-minimal solutions for the matching problem $\exists R.A \sqcap \exists R.B \sqsubseteq^? \exists R.X$, they are not i-equivalent.

We have seen that neither an s-minimal solution is i-minimal nor vice versa. As for complete sets however, an s-complete set is always i-complete: By Lemma 4, we can conclude $\sqsubseteq_s \subseteq \sqsubseteq_i$. Then, Lemma 12 ensures that every s-complete set

[1] For a given set $\mathcal{X}$ of variables this ordering can be extended to arbitrary substitutions.

is also i-complete. The converse direction does in general not hold: for the matching problem of Example 16 the solutions $\sigma := \{X \mapsto A, Y \mapsto B\}$ and $\tau := \{X \mapsto B, Y \mapsto A\}$ form an i-complete set. However, this set is not s-complete since for the solution $\delta := \{X \mapsto A, Y \mapsto A\}$ we have $\sigma \not\sqsubseteq_s \delta$ and $\tau \not\sqsubseteq_s \delta$.

According to Theorem 14 and the above observation a (minimal) s-complete set contains the minimal solutions with respect to both orderings, i.e., s- and i-minimal solutions. On the other hand, an i-complete set does not necessarily contain all solutions one might expect. For the matching problem in Example 16, the singleton set only consisting of the solution $\{X \mapsto A, Y \mapsto B\}$ is i-complete. However, one would also expect $\{X \mapsto B, Y \mapsto A\}$ to be an element of the complete set.

To sum up, among the orderings of choice (minimal) s-complete sets seem to be most suitable to represent all minimal solutions. In the following sections, we will specify an algorithm computing an s-complete set for a matching problem. To be more precise, our algorithm solves a more general problem. It computes an s-complete set of matchers for matching modulo equivalence, i.e., a problem of the form $C \equiv^? D$ where s-completeness is defined analogously to matching problems modulo subsumption. By Lemma 8, the set of matchers for $C \sqsubseteq^? D$ coincides with the set of matchers for $C \equiv^? C \sqcap D$. Therefore, an s-complete for $C \equiv^? C \sqcap D$ is also s-complete for $C \sqsubseteq^? D$. A minimal s-complete set can then be obtained by iteratively eliminating all matchers σ of the s-complete set such that there is a matcher τ, $\tau \neq \sigma$, in the set with $\tau \sqsubseteq_s \sigma$. In the same way, one can derive a (minimal) i-complete set from an s-complete set since every s-complete set is i-complete as well.

In the next section, we present the matching algorithm for the language $\mathcal{EL}$. We then extend the algorithm to $\mathcal{ALE}$ in the subsequent section. The reason for presenting the results for the sublanguage $\mathcal{EL}$ of $\mathcal{ALE}$ is chiefly to illustrate the main idea behind the matching algorithm, since $\mathcal{ALE}$ requires more details. However, decidability results for matching in one language do not necessarily carry over to sublanguages, which distinguishes non-standard inferences, like matching, from standard inferences like subsumption and consistency.

3 Matching in $\mathcal{EL}$

As our matching algorithm is mainly based on the characterization of subsumption in terms of homomorphisms between so-called description trees and on the results for lcs presented in [3], we first shall briefly sum up those notions and results. In [3], only concept descriptions were considered. We need to extend some of the notions to concept patterns as well.

3.1 $\mathcal{EL}$-description trees

$\mathcal{EL}$-description trees are used to represent $\mathcal{EL}$-concept descriptions and patterns.

Definition 18 An *$\mathcal{EL}$-description tree* is a tree of the form $\mathcal{G} = (V, E, v_0, \ell)$ with root v_0 where

- the edges $vrw \in E$ (called *$\exists$-edges*) are labeled with primitive roles r from N_R, and
- the nodes $v \in V$ are labeled with sets $\ell(v)$ of concept names and concept variables, i.e., subsets of $N_C \cup \mathcal{X}$.

The empty label corresponds to the top-concept.

In the sequel, $\mathcal{G}(v)$, where v is a node in $\mathcal{G}$, denotes the subtree of $\mathcal{G}$ with root v.

Intuitively, an $\mathcal{EL}$-description tree is merely a graphical representation of the syntax of the concept description/pattern. In order to translate concept descriptions/patterns into description trees, we need the following notion. The *role depth* of an ($\mathcal{EL}$-, $\mathcal{ALE}$-) concept description/pattern C (for short $\mathsf{depth}(C)$) is inductively defined by

- $depth(\bot) = depth(\top) = depth(P) = depth(\neg P) = depth(X) := 0$;
- $depth(C \sqcap D) := max(depth(C), depth(D))$;
- $depth(\forall r.C) = depth(\exists r.C) := depth(C) + 1$.

Every $\mathcal{EL}$-concept description/pattern C can be written (modulo equivalence) as $C \equiv P_1 \sqcap \ldots \sqcap P_n \sqcap \exists r_1.C_1 \sqcap \ldots \sqcap \exists r_m.C_m$ with $P_i \in N_C \cup \mathcal{X} \cup \{\top\}$. Such a concept description can now be translated into an $\mathcal{EL}$-description tree $\mathcal{G}_C = (V, E, v_0, \ell)$ by induction on the role depth of C as follows.

If $\mathsf{depth}(C) = 0$ **then** $V := \{v_0\}$, $E := \emptyset$, and $\ell(v_0) := \{P_1, \ldots, P_n\} \setminus \{\top\}$.

If $\mathsf{depth}(C) > 0$ **then** for $1 \leq i \leq m$, let $\mathcal{G}_i = (V_i, E_i, v_{0i}, \ell_i)$ be the inductively defined description tree corresponding to C_i where, w.l.o.g., the V_i are pairwise disjoint. Then

- $V := \{v_0\} \cup \bigcup_{1 \leq i \leq m} V_i$,
- $E := \{v_0 r_i v_{0i} \mid 1 \leq i \leq m\} \cup \bigcup_{1 \leq i \leq m} E_i$,
- $\ell(v) := \begin{cases} \{P_1, \ldots, P_n\} \setminus \{\top\}, & v = v_0 \\ \ell_i(v), & v \in V_i, 1 \leq i \leq m \end{cases}$

Example 19 The $\mathcal{EL}$-concept description

$$C := P \sqcap \exists r.(\exists r.(P \sqcap Q) \sqcap \exists s.Q) \sqcap \exists r.(P \sqcap \exists s.P)$$

yields the tree $\mathcal{G}_C$ depicted in Figure 1, a).

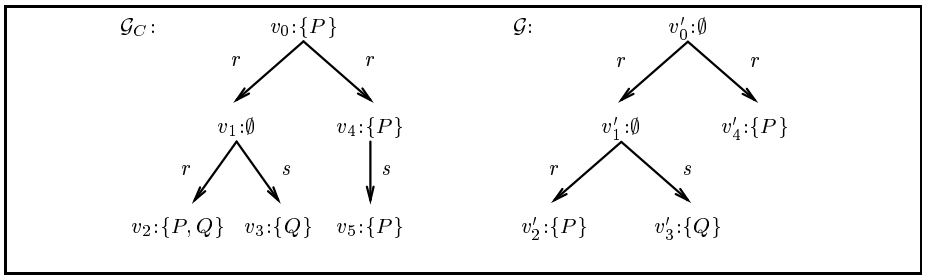

Figure 1: $\mathcal{EL}$-description trees.

Conversely, every $\mathcal{EL}$-description tree $\mathcal{G} = (V, E, v_0, \ell)$ without variables in its labels can be translated into an $\mathcal{EL}$-concept description $C_\mathcal{G}$ by induction on the depth of $\mathcal{G}$.[2]

If $\mathsf{depth}(\mathcal{G}) = 0$ **then** $V = \{v_0\}$, $E = \emptyset$. If $\ell(v_0) = \emptyset$, then $C_\mathcal{G} := \top$; otherwise, we have $\ell(v_0) = \{P_1, \ldots, P_n\}$, $n \geq 1$ and define $C_\mathcal{G} := P_1 \sqcap \ldots \sqcap P_n$.

If $\mathsf{depth}(\mathcal{G}) > 0$ **then** let $\ell(v_0) = \{P_1, \ldots, P_n\}$, $n \geq 0$, and let $\{v_1, \ldots, v_m\}$ be the set of all successors of v_0 with $v_0 r_i v_i \in E$ for some $r_i \in N_R$, $1 \leq i \leq m$. Further, let $C_1, \ldots, C_m$ denote the inductively defined $\mathcal{EL}$-concept descriptions corresponding to the subtrees of $\mathcal{G}$ with root v_i, $1 \leq i \leq m$. We define $C_\mathcal{G} := P_1 \sqcap \ldots \sqcap P_n \sqcap \exists r_1.C_1 \sqcap \ldots \sqcap \exists r_m.C_m$.

Note that only for a leaf $v \in V$ the empty label is translated into the top-concept. For a node $v \in V$ with $\ell(v) = \emptyset$ that is not a leaf, the empty set is not translated into the top-concept: the concept description corresponding to the subtree with root v only consists of existential restrictions induced by the direct successors of v.

Example 20 The $\mathcal{EL}$-description tree $\mathcal{G}$ in Figure 1, b) yields the $\mathcal{EL}$-concept description

$$C_\mathcal{G} = \exists r.(\exists r.P \sqcap \exists s.Q) \sqcap \exists r.P.$$

The *semantics* of an $\mathcal{EL}$-description tree $\mathcal{G}$ without variables in its labels is given by the semantics of the corresponding $\mathcal{EL}$-concept description, i.e., for an interpretation $I = (\Delta, \cdot^I)$ we define $\mathcal{G}^I := C_\mathcal{G}^I$.

The translations of $\mathcal{EL}$-concept descriptions and $\mathcal{EL}$-description trees into one another preserve semantics in the sense that $C \equiv C_{\mathcal{G}_C}$.

3.2 Homomorphisms, Subsumption, and lcs in $\mathcal{EL}$

In [3], homomorphisms between description trees have been employed to characterize subsumption between concept descriptions. Moreover, such characterizations have been used to describe the lcs of concept descriptions as product

[2] The *depth* of a description tree $\mathcal{G}$ is defined as the length of the longest path in $\mathcal{G}$. Since it directly corresponds to the depth of the corresponding concept description, it is also denoted by $\mathsf{depth}(\mathcal{G})$.

of descriptions trees. In the next section, we will see that homomorphisms are also crucial for the matching algorithm we propose.

Definition 21
Let $\mathcal{G} = (V_G, E_G, v_0, \ell_G)$ and $\mathcal{H} = (V_H, E_H, w_0, \ell_H)$ be $\mathcal{EL}$-description trees. A mapping $\varphi : V_H \longrightarrow V_G$ is a *homomorphism* from $\mathcal{H}$ into $\mathcal{G}$ iff the following conditions are satisfied:

1. $\varphi(w_0) = v_0$,
2. $(\ell_H(v) \setminus \mathcal{X}) \subseteq \ell_G(\varphi(v))$ for all $v \in V_H$, and
3. $\varphi(v) r \varphi(w) \in E_G$ for all $vrw \in E_H$.

Now, subsumption between $\mathcal{EL}$-concept descriptions can be characterized in terms of homomorphisms between their corresponding $\mathcal{EL}$-description trees [3]:

Theorem 22 Let C, D be $\mathcal{EL}$-concept descriptions and $\mathcal{G}_C, \mathcal{G}_D$ the corresponding description trees. Then $C \sqsubseteq D$ iff there exists a homomorphism from $\mathcal{G}_D$ into $\mathcal{G}_C$.

Example 23 (Example 19 continued)
Consider the $\mathcal{EL}$-description trees depicted in Figure 1. We have $C \sqsubseteq C_{\mathcal{G}}$, because mapping v_i' onto v_i for $0 \leq i \leq 4$ yields a homomorphism from $\mathcal{G}$ to $\mathcal{G}_C$.

Empolying Theorem 22, one can show that the lcs of concept descriptions C and D corresponds to the product of the descriptions trees $\mathcal{G}_C$ and $\mathcal{G}_D$. For specifying our matching algorithm we need the following results, which can easily be derived from results in [3]:

Proposition 24 The size of the lcs of a sequence $C_1, \ldots, C_n$ of $\mathcal{EL}$-concept descriptions may grow exponential in the size of the sequence. The lcs of a sequence of concept descriptions can be computed in time exponential in the size of the sequence.

3.3 The $\mathcal{EL}$-Matching Algorithm

As motivated in Section 2, we are interested in an s-complete set of matchers for a matching problem of the form $C \equiv^? D$ where C is an $\mathcal{EL}$-concept description and D is an $\mathcal{EL}$-concept pattern.

The matching algorithm described in Fig. 2 first tries to construct substitutions σ such that $C \sqsubseteq \sigma(D)$, i.e., there is a homomorphism from $\mathcal{G}_{\sigma(D)}$ into $\mathcal{G}_C$. In a second step, it checks which of the computed substitutions really solve the matching problem, i.e., also satisfies $C \sqsupseteq \sigma(D)$. (Obviously, for a matching problem modulo subsumption, this second step can be dispensed with.) The first step 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 $\sigma(X)$ as $C_{\mathcal{G}_C(\varphi(w))}$ where w is such that X occurs

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

Compute $\mathcal{G}_C$ and $\mathcal{G}_D = (V, E, w_0, \ell)$
$\mathcal{C} := \emptyset$
For all homomorphisms φ from $\mathcal{G}_D$ into $\mathcal{G}_C$
  Define σ by $\sigma(X) := lcs(C_{\mathcal{G}_C(\varphi(w))} \mid X \in \ell(w))$
    for all variables X in D
  If $C \sqsupseteq \sigma(D)$ then $\mathcal{C} := \mathcal{C} \cup \{\sigma\}$

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

in the label of w. Since there may exist several nodes w with this property, we take the lcs of the corresponding subconcepts 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. $\sqsubseteq_s$. An algorithm for computing the lcs of $\mathcal{EL}$-concepts has been described in [3].

Before proving the soundness of our matching algorithm, we illustrate the algorithm by the following example.

Example 25 Let the concept description C and the concept pattern D be defined as follows:

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

The corresponding description trees are depicted in Figure 3. There are exactly two homomorphisms from $\mathcal{G}_D$ into $\mathcal{G}_C$. Both homomorphisms must map w_i onto v_i for $i = 0, 1, 2$. Only w_3 can be mapped onto two different nodes, namely, v_3 or v_4; let φ_1 be the homomorphism which maps w_3 on v_3, i.e, $\varphi_1(w_3) = v_3$, and let φ_2 be the one with $\varphi_2(w_3) = v_4$. Thus, the algorithm depicted in Figure 2 computes two substitutions, namely, one with respect to φ_1 and the other one for φ_2. More precisely, for φ_1 the substitution σ_1 is computed as follows:

$$\begin{aligned} \sigma_1(X) &= lcs(P \sqcap \exists r.P \sqcap \exists r.Q, P \sqcap Q \sqcap \exists r.P) \\ &\equiv \exists r.P \sqcap P \\ \sigma_1(Y) &= lcs(P \sqcap Q \sqcap \exists r.P, P) \\ &\equiv P \end{aligned}$$

The subsequent subsumption test C
$sqsupseteq\sigma_1(D)$ in the matching algorithm fails such that σ_1 is not added to $\mathcal{C}$. For φ_2 the algorithms yields

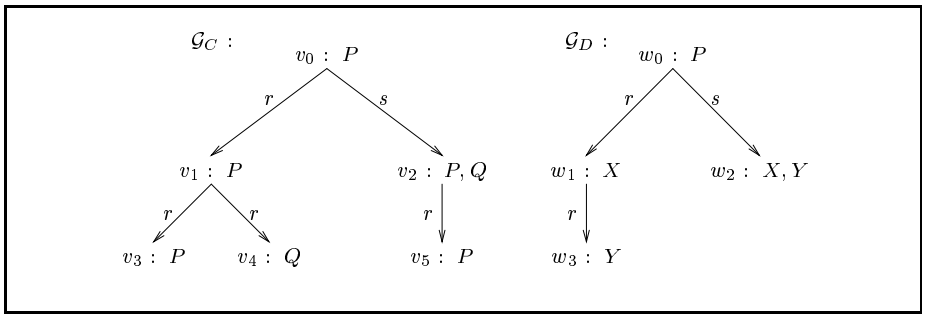

Figure 3: $\mathcal{EL}$-description trees.

$$\begin{aligned}
\sigma_2(X) &= lcs(P \sqcap \exists r.P \sqcap \exists r.Q, P \sqcap Q \sqcap \exists r.P) \\
&\equiv \exists r.P \sqcap P \\
\sigma_2(Y) &= lcs(P \sqcap Q \sqcap \exists r.P, Q) \\
&\equiv Q
\end{aligned}$$

In this case, the subsumption test $C \equiv \sigma_2(D)$ succeeds and $\mathcal{C} = \{\sigma_2\}$ is the output of the matching algorithm.

3.4 Soundness of the $\mathcal{EL}$-Matching Algorithm

For the set $\mathcal{C}$ computed by the matching algorithm (Figure 2) we have to verify two properties. First, we have to show that the substitutions in $\mathcal{C}$ are matchers for the given matching problem. Second, we hav to prove that $\mathcal{C}$ is s-complete, i.e., for every matcher σ' of $C \equiv^? D$ we have to verify that there is a matcher $\sigma \in \mathcal{C}$ such that $\sigma \sqsubseteq_s \sigma'$. We break the proof down in two lemmas. The first lemma will also be useful to specify an optimized algorithm for matching problems modulo subsumption. It says that for every substitution σ computed by the algorithm it is $C \sqsubseteq \sigma(D)$.

Lemma 26 Let φ be a homomorphism from $\mathcal{G}_D = (V, E, r, \ell)$ into $\mathcal{G}_C = (V', E', r', \ell')$ and let σ be the corresponding substitution as specified by the matching algorithm in Figure 2. Then, $C \sqsubseteq \sigma(D)$.

Proof. Let X be a variable in D. Then for every $w \in V$ with $X \in \ell(w)$ there exists a node $v \in V'$ with $\varphi(w) = v$. By the definition of σ it follows $C_{\mathcal{G}_C(v)} \sqsubseteq \sigma(X)$. Using that $C_{\mathcal{G}_C(v)} \equiv \mathcal{G}_C(v)$, according to Theorem 22 there exists a homomorphism from $\mathcal{G}_{\sigma(X)}$ into $\mathcal{G}_C(v)$.

Obviously, one obtains the description tree $\mathcal{G}_{\sigma(D)}$ by extending $\mathcal{G}_D$ for every node w and every variable X in the label of w by (a new instance of) $\mathcal{G}_{\sigma(X)}$ where the root of $\mathcal{G}_{\sigma(X)}$ and the node w are identified. To the label of w the

concept names in the label of the root of $\mathcal{G}_{\sigma(X)}$ are added. Furthermore, X is deleted from the label of w. Note that the new label of w is still a subset of the label of v. For this reason, φ can be extended to an homomorphism from $\mathcal{G}_{\sigma(D)}$ into $\mathcal{G}_C$. By Theorem 22, this shows $C \sqsubseteq \sigma(D)$. ∎

In the sequel, we will refer to the process, described in the above proof, of replacing the variables X in a description tree $\mathcal{G}_D$ by instances of $\mathcal{G}_{\sigma(X)}$ as *instantiating* $\mathcal{G}_D$ *by* $\mathcal{G}_{\sigma(X)}$ for all variables X in D.

Now, let σ' be a matcher for $C \equiv^? D$. This implies $C \sqsubseteq \sigma'(D)$. By Theorem 22, there is a homomorphism φ' from $\mathcal{G}_{\sigma(D')}$ into $\mathcal{G}_C$. When deleting the variables in $\mathcal{G}_D$ then $\mathcal{G}_D$ is a subtree of $\mathcal{G}_{\sigma'(D)}$. Thus, restricting φ' to the nodes of $\mathcal{G}_D$ yields a homomorphism φ from $\mathcal{G}_D$ into $\mathcal{G}_C$. For all variables X in D let $\sigma(X) := lcs(C_{\mathcal{G}_C(\varphi(w))} \mid X \in \ell(w))$ be defined as specified by the matching algorithm in Figure 2.

Lemma 27 It is $\sigma \sqsubseteq_s \sigma'$.

Proof. We have to verify $\sigma(X) \sqsubseteq \sigma'(X)$ for every variable X in D. Let X be a variable in D, $\mathcal{G}_D = (V, E, r, \ell)$, and $w \in V$ such that $X \in \ell(w)$ and $\varphi(w) = v$. Restricting φ' to the description tree $\mathcal{G}_{\sigma'(D)}(w)$ provides us with a homomorphism from $\mathcal{G}_{\sigma'(D)}(w)$ into $\mathcal{G}_C(v)$. Since $X \in \ell(w)$, $\mathcal{G}_{\sigma'(D)}(w)$ contains a subtree corresponding to $\sigma'(X)$. Consequently, there is a homomorphism from $\mathcal{G}_{\sigma'(X)}$ into $\mathcal{G}_C(v)$ which shows $C_{\mathcal{G}_C(v)} \sqsubseteq \sigma'(X)$. Since this is true for all $v \in V(\varphi, X)$, we can conclude $\sigma(X) \sqsubseteq \sigma'(X)$. Thus, $\sigma \sqsubseteq_s \sigma'$. ∎

With these two lemm at hand the soundness of the matching algorithm can be derived as follows: If $\sigma \in \mathcal{C}$ is a substitution computed by the matching algorithm then by Lemma 26 we know $C \sqsubseteq \sigma(D)$. But then, the subsumption test $C \sqsupseteq \sigma(D)$ in the matching algorithm ensures $C \equiv \sigma(D)$ which shows that σ is a matcher of the matching problem $C \equiv^? D$.

Now, let σ' be some matcher for $C \equiv^? D$ and let σ be defined as specified above Lemma 27. From Lemma 27 we know $\sigma \sqsubseteq_s \sigma'$. Thus, by Lemma 4 we can conclude $\sigma(D) \sqsubseteq \sigma'(D)$. According to the definition, σ is computed by the matching algorithm in Figure 2. But then, Lemma 26 implies $C \sqsubseteq \sigma(D)$. Thus, we have $\sigma(D) \sqsubseteq \sigma'(D) \equiv C \sqsubseteq \sigma(D)$ which means $C \equiv \sigma(D)$. Hence, σ belongs to the set $\mathcal{C}$ computed by the matching algorithm. Furthmore, the fact that $\sigma \sqsubseteq_s \sigma'$ shows that $\mathcal{C}$ is s-complete.

3.5 Complexity of Matching in $\mathcal{EL}$

We first show, by means of an example, that an s-complete set of matchers might grow exponential in the size of the matching problem.

Example 28 Let

$$\exists r.A \sqcap \exists r.B \equiv^? \exists r.X_1 \sqcap \cdots \sqcap \exists r.X_n$$

be a $\mathcal{EL}$-matching problems where A, B are concept names and X_i, $1 \leq i \leq n$, are concept variables.

For a word $w = a_1 \cdots a_n \in \{A, B\}^n$ we define $\sigma_w(X_i) := a_i$ for every $1 \leq i \leq n$. Obviously, for a word w that contains both A and B, σ_w is a matcher of the above matching problem. Now, let $\mathcal{C}$ be an s-complete set of this matching problem. Then, there exists a $\sigma \in \mathcal{C}$ such that $\sigma \sqsubseteq_s \sigma_w$. Using Theorem 22, it is not hard to see $\sigma \equiv_s \sigma_w$. Furthermore, for $w \neq w'$, $w, w' \in \{A, B\}^n$, it is easy to verify that $\sigma_w \not\sqsubseteq_s \sigma_{w'}$ and $\sigma_{w'} \not\sqsubseteq_s \sigma_w$. Now observe that the number of words in $\{A, B\}^n$ with at least one A and one B is exponential in n. As a result, $\mathcal{C}$ must be exponential in n.

As an immediate consequence of the example we obtain

Proposition 29 The cardinality of a (minimal) s-complete set of matchers might grow exponentially in the size of the corresponding $\mathcal{EL}$-matching problem.

On the other hand, using the matching algorithm we can state an upper bound for the size of an s-complete set. The size of such a set is defined as follows: Since a concept pattern D contains only a finite number of variables, a matcher can be represented as a finite set of tuples where the first component of a tuple contains a variable and the second component contains the concept description which is assigned to this variable by means of the matcher. Then, the *size of a matcher* can be defined in terms of the sum of the size of the tuples. In addition, the *size of a set of matchers* is defined by the sum of the size of the matcher in the set.

Now, note that the number of mappings from a description tree $\mathcal{G}_D$ into $\mathcal{G}_C$ is exponential in the size of the description trees. Since the size of these trees is linear in the size of the matching problem $C \equiv^? D$ we can conclude that an s-complete set of matcher computed by our matching algorithm is at most exponential. Furthermore, by Proposition 24 the lcs of a sequence of $\mathcal{EL}$-concept descriptions can be computed in time exponential in the size of the sequence. Thus, the size of every substitution computed by the matching algorithm is at most exponential in the size of the matching problem. Finally, as mentioned in Section 2, from an s-complete set one can obtain a minimal s-complete (i-complete) set by iteratively eliminating certain solutions from the s-complete set. These results can be summarized as follows:

Corollary 30 For every $\mathcal{EL}$-matching problem there exists a (minimal) s-complete (i-complete) set with size at most exponential in the size of the matching problem.

Subsumption of $\mathcal{EL}$-concept descriptions can be decided in time polynomial in the size of the given concept descriptions [3]. As already mentioned, the size of a substitution σ computed by our matching algorithm is at most exponential in the size of the matching problem. Thus, $C \sqsupseteq \sigma(D)$ can be decided in time exponential in the size of the matching problem $C \equiv^? D$. Now, using Proposition 24, it is easy to see that the loop body of the algorithm in Figure 2 runs

in exponential time. As already mentioned, there exists only an exponential number of mappings from $\mathcal{G}_D$ into $\mathcal{G}_C$. For these mappings it can be decided in time polynomial in the size of the matching problem if they are homomorphisms. Consequently, our matching algorithm runs in time exponential in the size of the matchng problem. Moreover, for a given s-complete set one can obtain a minimal s-complete (i-complete) set in time polynomial in the size of the s-complete set using the method mentioned in Section 2.

Corollary 31 A (minimal) s-complete set of matchers for an $\mathcal{EL}$-matching problem can be computed in time exponential in the size of the matching problem.

Because of Lemma 8, Corollary 31 also holds true for matching modulo subsumption. However, in case of matching modulo subsumption we can dispense with the test $C \sqsupseteq \sigma(D)$ but rather add σ to $\mathcal{C}$ right away. On the other hand, this has no impact on the complexity of computing an s-complete set for a matching problem modulo subsumption, which is still exponential.

Deciding Solvability of Matching in $\mathcal{EL}$

Lemma 9 ensures that the solvability of an $\mathcal{EL}$-matching problem modulo subsumption can be decided in time polynomial in the size of the matching problem.

Since an s-complete set of matchers of a matching problem is empty if and only if the matching problem has no solution the matching algorithm shown in Figure 2 can be employed to decide the solvability of a matching problem modulo equivalence. This provides us with an exponential-time decision algorithm. However, we strongly conjecture that there is also an NP algorithm. We now show that the problem is NP-hard.

We show NP-hardness by reducing SAT [9] to deciding the matching problem modulo equivalence.

Let $\phi = p_1 \wedge \cdots \wedge p_m$ be a propositional formulae 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 the role names r, r', s, s'. We first show that one can specify a matching problem such that either X_i must be substituted by A (corresponding to $true$) and $\overline{X}_i$ by B or vice versa. This corresponds to assigning $true$ or $false$ to x_i. Such a matching problem can be written in terms of the following concept descriptions/patterns:

$$\begin{aligned} C_0 &:= \top \\ C_{k+1} &:= \exists r'.A \sqcap \exists r'.B \sqcap \exists r.C_k \\ D_0 &:= \top \\ D_{k+1} &:= \exists r'.X_{k+1} \sqcap \exists r'.\overline{X}_{k+1} \sqcap \exists r.D_k \end{aligned}$$

Now, the matching problem can be stated as $C_n \equiv^? D_n$. It is easy to see that the matcher of that problem must substitute either X_i by A and $\overline{X}_i$ by B or vice versa.

In order to decode ϕ we translate a conjunct p_i, $1 \leq i \leq m$, into a concept pattern D_{p_i} as follows. For example, if $p_i = x_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_4$ then $D_{p_i} := X_1 \sqcap \overline{X}_2 \sqcap X_3 \sqcap \overline{X}_4 \sqcap B$. We now have to decode the evaluation of a formulae:

$$\begin{aligned} C'_0 &:= \top \\ C'_{k+1} &:= \exists s'.(A \sqcap B) \sqcap \exists s.C'_k \\ D'_0 &:= \top \\ D'_{k+1} &:= \exists s'.D_{p_{k+1}} \sqcap \exists s.D'_k \end{aligned}$$

The matching problem $C'_m \equiv^? D'_m$ ensures that among the variables in D_{p_i} there must be at least one variable substituted by A. This corresponds to the fact that within one conjunct p_i there must be at least one propositional variable that evaluates to *true*. Note that we need the concept B in D_{p_i} because otherwise if all variables in D_{p_i} were substituted by A then C'_m and D'_m would not be equivalent.

We combine the two parts of the reduction in the final matching problem as follows:

$$\begin{aligned} C_\phi &:= C_n \sqcap C'_m \\ D_\phi &:= D_n \sqcap D'_m \end{aligned}$$

It is easy to verify that ϕ is satisfiable if and only if the matching problem $C_\phi \equiv^? D_\phi$ is solvable since a truth assignment can be directly translated into a substitution and vice versa. This proves

Lemma 32 Deciding the solvability of an $\mathcal{EL}$-matching problem modulo equivalence is NP-hard and can be decided in time exponential in the size of the matching problem.

4 Extending the results to $\mathcal{ALE}$

The main idea of the matching algorithm for $\mathcal{ALE}$ is the same as for $\mathcal{EL}$, i.e., governed by homomorphisms from the description tree of the concept pattern D into the description tree of the concept description C one defines certain substitutions and, in case of matching modulo equivalence, checks if these substitutions are solutions of the matching problem. However, as it is illustrated by the following example, we now need to compute complete sets for a set of so-called $\top$-patterns of the original pattern D, which are obtained from D by replacing certain variables by $\top$.

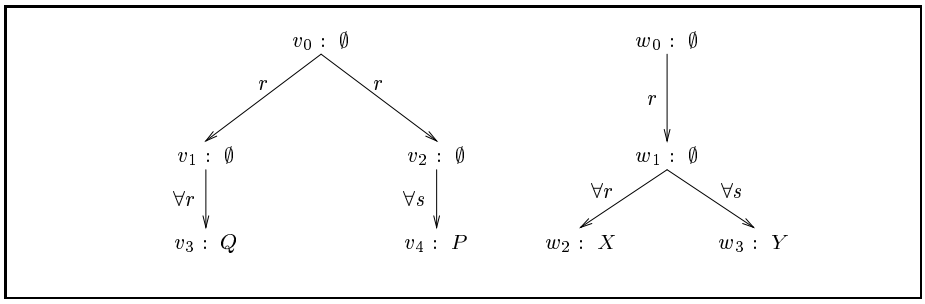

Figure 4: The description trees for C and D.

Example 33 Let

$$C := (\exists r.\forall r.Q) \sqcap (\exists r.\forall s.P) \quad \sqsubseteq^? \quad \exists r.(\forall r.X \sqcap \forall s.Y) =: D$$

be an $\mathcal{ALE}$-matching problem. The description trees for C and D are depicted in Figure 4 where, in addition to $\exists$-edge, $\forall$-edges are introduced to represent value-restrictions. Obviously, $\sigma := \{X \mapsto Q, Y \mapsto \top\}$ and $\tau := \{X \mapsto \top, Y \mapsto P\}$ are solutions for the given matching problem. However, there is no homomorphism from the tree for D into the one for C: The node w_1 can be mapped either on v_1 or 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 example shows that there need not to be a homomorphism between a description tree corresponding to a pattern and a description tree for a concept description, although the matching problem is solvable. On the other hand, as shown in [3], subsumption of $\mathcal{ALE}$-concept descriptions can be characterized in terms of homomorphisms. Still, this requires the normalization of both the subsumee and the subsumer. For characterizing the subsumption of concept descriptions in terms of homomorphisms, it is necessary to apply the following normalization rule to the subsumer:[3]

$$\forall r.\top \quad \longrightarrow \quad \top$$

But as far as matching is concerned, a normalization of the concept pattern (subsumer) would depend on the substitutions for the variables, which we do not no in advance. So there is no unique way to normalize a concept pattern. We illustrate this problem by Example 33.

Both instances $\sigma(D)$, $\tau(D)$ of D are not normalized according to the normalization rule since they contain the subconcepts $\forall s.\top$ and $\forall r.\top$, respectively.

[3] For the subsumee additional normalization rules are necessary, which we will present in Section 4.1.

The description tree for the *normalized* concept description $\sigma(D)$ would not include the node w_3 and the $\forall s$-edge leading to it. In this case, there is a homomorphism from this description tree into the one for C. Analogously, for $\tau(D)$, w_2 would be deleted, which again allows to define a homomorphism.

As a result, in order to compute matchers using homormophisms one has to consider a *set* of so-called $\top$-patterns of D. These patterns are obtained by replacing a subset of variables in D by $\top$. Furthermore, before turning such a $\top$-pattern into a description tree one has to normalize the pattern using the normalization rule stated above. Referring to the example, we get the following normalized $\top$-patterns for D: D, $\exists r.(\forall r.X)$, $\exists r.(\forall s.Y)$, and $\exists r.\top$. Matching these patterns against C our matching algorithm 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 for $C \sqsubseteq^? D$.

Another way to overcome the problem shown in Example 33, is to introduce *partial* homomorphisms instead of total homomorphism. The idea is that these partial homomorphisms allow to leave the image of certain nodes of a description tree for D undefined if there is no corresponding node in C. In Example 33, this means that the image of w_2 and w_3 might be undefined. We will not pursue this approach in the sequel, however.

In the following sections, we shall extend the notion of homomorphism, the characterization of subsumption, and the results for lcs to $\mathcal{ALE}$ as they have been proved in [3]. Then we will exploit the ideas presented in this section to specify a matching algorithm for $\mathcal{ALE}$.

4.1 Subsumption and Lcs in $\mathcal{ALE}$

First, we have to define the notion of $\mathcal{ALE}$-description trees formally.

Definition 34 An *$\mathcal{ALE}$-description tree* is a tree of the form $\mathcal{G} = (V, E, v_0, \ell)$ with root v_0 where

- the edges in E are labeled with primitive roles r from N_R ($\exists$-edges) or with $\forall r$ for some $r \in N_R$ ($\forall$-edges), and
- the nodes $v \in V$ are labeled with sets $\ell(v) = \{P_1, \ldots, P_n\}$ where each P_i, $1 \leq i \leq n$, is of one of the following forms: $P_i \in N_C \cup \mathcal{X}$, $P_i = \neg P$ for some $P \in N_C$, or $P_i = \bot$.

The empty label corresponds to the top-concept.

Just as for $\mathcal{EL}$, an $\mathcal{ALE}$-concept description/pattern C can be tranlated into an $\mathcal{ALE}$-description tree $\mathcal{G}(C)$ (see [3] for details). As an example, consider the $\mathcal{ALE}$-concept description

$$C \quad := \quad \forall r.(\exists s.Q \sqcap \forall s.\neg Q) \sqcap \exists s.(P \sqcap \exists r.Q).$$

The corresponding description tree $\mathcal{G}(C)$ is depicted in Figure 5.

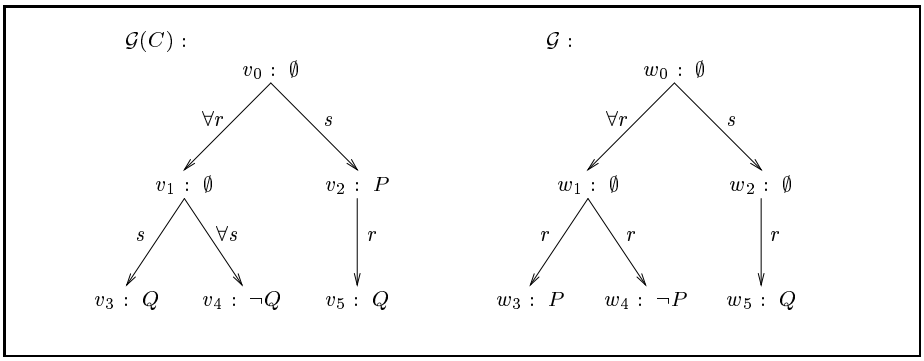

Figure 5: The description trees for $\mathcal{G}(C)$ and $\mathcal{G}$.

On the other hand, every $\mathcal{ALE}$-description tree $\mathcal{G}$ without variables in its labels can be translated into an $\mathcal{ALE}$-concept description $C_{\mathcal{G}}$ ([3] contains a formal translation). The description graph $\mathcal{G}$ in Figure 5 yields the $\mathcal{ALE}$-concept description

$$D \quad := \quad C_{\mathcal{G}} \quad = \quad (\forall r.(\exists r.P \sqcap \exists r.\neg P)) \sqcap (\exists s.\exists r.Q)$$

Just as for $\mathcal{EL}$, the semantics of $\mathcal{ALE}$-description trees $\mathcal{G}$ without variables in their labels is defined by the semantics of their corresponding concept descriptions $C_{\mathcal{G}}$, i.e., $\mathcal{G}^I := C_{\mathcal{G}}^I$ for an interpretation I. Again, it is easy to see that the translation of concept descriptions and description trees in one another preserves semantics, i.e., $C \equiv C_{\mathcal{G}_C}$.

As shown in [3], in order to characterize subsumption of $\mathcal{ALE}$-concept descriptions in terms of homomorphisms between the corresponding description trees, the concept descriptions need to be normalized before translating them into description trees.

Definition 35 Let E, F be two $\mathcal{ALE}$-concept descriptions and $r \in N_R$ a primitive role. The *$\mathcal{ALE}$-normalization rules* are defined as follows

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

A concept descripton C is called *normalized* if none of the above normalization rules can be applied to some subconcept 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 to subconcepts of 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. We refer to $\mathcal{G}_C$ as the description tree corresponding to the normal form of 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. We refer to $\mathcal{G}_C^\top$ as the description tree corresponding to the $\top$-normal form of C.

Obviously, a homomorphism between $\mathcal{ALE}$-description trees must take $\forall$-edges into account, which are dealt with like $\exists$-edges. In addition, homomorphisms as introduced for $\mathcal{EL}$-description trees are extended by allowing them to map a node and all nodes in its subtree onto an inconsistent node, i.e., a node labeled $\{\bot\}$. Formally, a homomorphism is defined as follows:

Definition 36 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* if and only if the following conditions are satisfied:

1. $\varphi(w_0) = v_0$,
2. for all $v \in V_H$ we have $(\ell_H(v) \setminus \mathcal{X}) \subseteq \ell_G(\varphi(v))$ or $\ell_G(\varphi(v)) = \{\bot\}$,
3. for all $vrw \in E_H$, either $\varphi(v)r\varphi(w) \in E_G$, or $\varphi(v) = \varphi(w)$ and $\ell_G(\varphi(v)) = \{\bot\}$, and
4. for all $v\forall rw \in E_H$, either $\varphi(v)\forall r\varphi(w) \in E_G$, or $\varphi(v) = \varphi(w)$ and $\ell_G(\varphi(v)) = \{\bot\}$.

Now, subsumption can be characterized in terms of homomorphisms [3]:

Theorem 37 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 [3] requires a homomorphism from $\mathcal{G}_D$ instead of $\mathcal{G}_D^\top$. However, the proof reveals that $\top$-normalization of the subsumer is sufficient. Furthermore, we can conclude from the proof of Theorem 37:

Remark 38 The existence of a homomorphism from a description tree $\mathcal{G}$ into $\mathcal{H}$ implies $\mathcal{H} \sqsubseteq \mathcal{G}$, i.e., for all interpretations I it is $\mathcal{H}^I \subseteq \mathcal{G}^I$. This means that the if-direction of Theorem 37 does not require normalization.

We illustrate the theorem by means of the concept description C and D introduced above. The normal form of C is $\forall r.\bot \sqcap \exists s.(P \sqcap \exists r.Q)$; D is already in $\top$-normal form. A homomorphism from $\mathcal{G}_D^\top$ into $\mathcal{G}_C$ is depicted in Figure 6. By Theorem 37 we can conclude $C \sqsubseteq D$. Observe, however, that there is no homomorphism from $\mathcal{G}(D)$ into $\mathcal{G}(C)$. This shows that the only-if direction of Theorem 37 requires normalizing the concept descriptions before translating them into description trees.

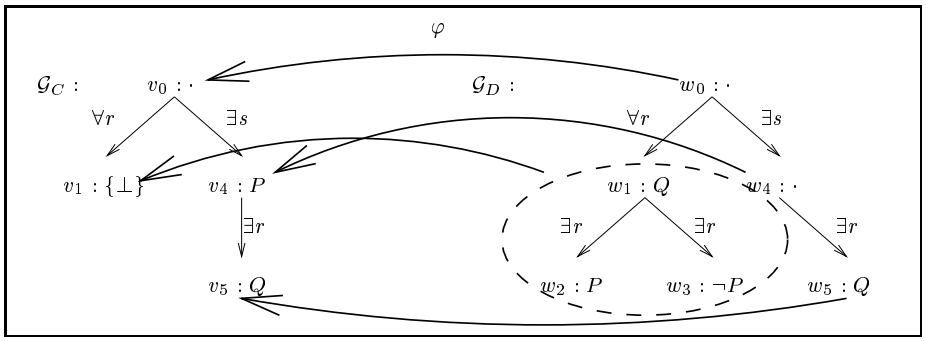

Figure 6: Subsumption for $\mathcal{ALE}$.

Just as for $\mathcal{EL}$, the lcs of $\mathcal{ALE}$-concept descriptions can be represented by the product of their description trees. In [3], the following complexity results regarding the lcs have been shown:[4]

Proposition 39 The size of the lcs of a sequence $C_1, \ldots, C_n$ of $\mathcal{ALE}$-concept descriptions may grow exponential in the size of the sequence. The lcs of a sequence of concept descriptions can be computed in time exponential in the size of the sequence.

4.2 The $\mathcal{ALE}$-Matching Algorithm

As mentioned above, the idea of the matching algorithm for $\mathcal{ALE}$ is to consider a set of "$\top$-pattern" D' for D and computing sets of solutions for $C \equiv^? D'$. We shall show that the union of these sets provides us with an s-complete set of solutions for $C \equiv^? D$.

Definition 40 A concept pattern D' is called $\top$-*pattern* of D if D' is obtained from D by substituting a subset of variables in D by $\top$.

Using this notion the $\mathcal{ALE}$-matching algorithm can be specified as depicted in Figure 7. We already illustrated the algorithm by the example discussed above.

4.3 Soundness of the $\mathcal{ALE}$-Matching Algorithm

The proof will proceed similarly to the one for $\mathcal{EL}$ in Section 3.4 and just as for $\mathcal{EL}$, it is based on two lemmata.

The first lemma says that for all substitutions σ computed by the algorithm in Figure 7, it is $C \sqsubseteq \sigma(D)$. This will also have a major impact on the complexity of matching problems modulo subsumption.

[4] Actually, only complexity results for the lcs of two concept descriptions are contained in that paper. However, the proofs allow us to simply extend the results to sequences of concept descriptions.

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

$\mathcal{C} := \emptyset$
For all $\top$-pattern D' of D do
  For all homomorphisms φ from $\mathcal{G}^{\top}_{D'}$ into $\mathcal{G}_C$
    Define σ by
      $\sigma(X) := lcs(C_{\mathcal{G}_C(\varphi(w))} \mid X \in \ell(w))$ for all variables X in D' and
      $\sigma(X) := \top$ for all variables X in D not in D'
    If $C \sqsupseteq \sigma(D')$ then $\mathcal{C} := \mathcal{C} \cup \{\sigma\}$

Figure 7: The $\mathcal{ALE}$-matching algorithm.

Lemma 41 Let D' be a $\top$-pattern of D, φ be a homomorphism from $\mathcal{G}^{\top}_{D'}$ into $\mathcal{G}_C$, and let σ be a substitution computed by the matching algorithm in Figure 7 w.r.t. D' and φ. Then, $C \sqsubseteq \sigma(D)$.

Proof. We first show that $C \sqsubseteq \sigma(D')$. Let X be a variable in D'. Then for every node w in D' with X in its label $\ell(w)$ there exists a node v in $\mathcal{G}_C$ with $\varphi(w) = v$. By the definition of σ it follows $C_{\mathcal{G}_C(v)} \sqsubseteq \sigma(X)$. Using that $C_{\mathcal{G}_C(v)} \equiv \mathcal{G}_C(v)$, according to Theorem 37 there exists a homomorphism from $\mathcal{G}^{\top}_{\sigma(X)}$ into $\mathcal{G}_C(v)$.

For this reason, analogously to Lemma 26, one can extend φ to a homomorphism from $\mathcal{G}$ into $\mathcal{G}_C$ where $\mathcal{G}$ is defined as the instantiation of $\mathcal{G}^{\top}_{D'}$ by $\mathcal{G}^{\top}_{\sigma(X)}$ for all variables X in D' (see Section 3.4 for instantiation). It is easy to see that $\mathcal{G} \equiv \sigma(D')$. Therefore, by Remark 38 we can conclude $C \sqsubseteq \sigma(D')$.

Now, from the definition of a $\top$-pattern and since $\sigma(Y) = \top$ for all variables in D that are not in D' we know $\sigma(D) \equiv \sigma(D')$. This shows $C \sqsubseteq \sigma(D)$ and completes the proof of the lemma. ∎

Analogously to Lemma 27 we now show

Lemma 42 If σ' is a matcher for $C \equiv^? D$, then there exists a matcher σ in the set S of matchers computed by the matching algorithm given in Figure 7 with $\sigma \sqsubseteq_s \sigma'$.

Proof. We know $C \sqsubseteq \sigma'(D)$. Let $T := \{X \mid \sigma'(X) \equiv \top\}$ be a subset of variables in D. Furthermore, let D' be the $\top$-pattern of D where the variables in T are substituted by $\top$. We can conclude $\sigma'(D) \equiv \sigma'(D')$. Thus, $C \sqsubseteq \sigma'(D')$. Theorem 37 implies that there exists a homomorphism φ' from $\mathcal{G}^{\top}_{\sigma'(D')}$ into $\mathcal{G}_C$.

Now, let $\mathcal{G}$ be the instantiation of $\mathcal{G}^{\top}_{D'}$ by $\mathcal{G}^{\top}_{\sigma(X)}$ for variables X in D' where the notion of instantiation is defined as in Section 3.4. We claim that $\mathcal{G}$ is isomorphic to $\mathcal{G}^{\top}_{\sigma'(D')}$, i.e., equal up to renaming of nodes:

Computing $\mathcal{G}_E^\top$ for some concept pattern E corresponds to iteratively deleting the nodes (and edges) in $\mathcal{G}(E)$ which are connected with their direct predecessors by means of an $\forall$-edge, which labels are empty, and which have no outgoing edges ($\forall$- or $\exists$-edges).

Now, obviously, the description tree $\mathcal{G}(\sigma'(D'))$ is the instantiation of $\mathcal{G}(D')$ by $\mathcal{G}(\sigma'(X))$. By definition of D', we know that $\sigma'(X) \not\equiv \top$ for all variables X in D'. Thus, the nodes in the subtree $\mathcal{G}(D')$ of $\mathcal{G}(\sigma'(D'))$ containing variables are not deleted when $\top$-normalizing $\mathcal{G}(\sigma'(D'))$. For this reason, one can obtain $\mathcal{G}^\top_{\sigma'(D')}$ by first $\top$-normalizing $\mathcal{G}(D')$, which yields $\mathcal{G}^\top_{D'}$, and then instantiating $\mathcal{G}^\top_{D'}$ by $\mathcal{G}^\top_{\sigma'(X)}$. This shows that $\mathcal{G}$ is isomophic to $\mathcal{G}^\top_{\sigma'(D')}$.

But then, $\mathcal{G}^\top_{D'}$ is a subtree of $\mathcal{G}^\top_{\sigma'(D')}$. Therefore, restricting φ' to $\mathcal{G}^\top_{D'}$, provides us with a homomorphism φ from $\mathcal{G}^\top_{D'}$ into $\mathcal{G}_C$.

Let σ be the substitution computed by the matching algorithm in Figure 7 w.r.t. D' and φ. It remains to show $\sigma \sqsubseteq_s \sigma'$:

If X is a variable in D but not in D' then we know $\sigma'(X) \equiv \top$. Thus, $\sigma(X) = \top \sqsubseteq \sigma'(X)$.

Now, let X be a variable in D'. There exists a node w in D' such that X is in the label $\ell(w)$ of w and $\varphi(w) = v$. When restricting φ' to $\mathcal{G}^\top_{\sigma'(D')}(w)$, one gets an homomorphism from $\mathcal{G}^\top_{\sigma'(D')}(w)$ into $\mathcal{G}_C(v)$. Since $\mathcal{G}$ is isomorphic to $\mathcal{G}^\top_{\sigma'(D')}$, we can conclude from the definition of $\mathcal{G}$ that $\mathcal{G}^\top_{\sigma'(X)}$ is a subtree of $\mathcal{G}^\top_{\sigma'(D')}(w)$. Therefore, there exists a homomorphism from $\mathcal{G}^\top_{\sigma'(X)}$ into $\mathcal{G}_C(v)$. By Theorem 37, this yields $C_{\mathcal{G}_C(v)} \sqsubseteq \sigma'(X)$. Hence, $\sigma(X) \sqsubseteq \sigma'(X)$. This completes the proof of the lemma. ∎

Note that the proof of this lemma makes heavy use of the fact that for the concept pattern D (or the $\top$-patterns D') only a "minor" normalization, namely, $\top$-normalization, is necessary, as opposed to the full normalization which is needed for C. As we will see in the next section, this has a major impact on complexity issues as well.

With Lemma 41 and Lemma 42 at hand, analogously to $\mathcal{EL}$ (Section 3.4), it can be shown that all substitutions in the set $\mathcal{C}$ computed by the matching algorithm are matchers of the given matching problem and that $\mathcal{C}$ is s-complete.

4.4 Complexity of Matching in $\mathcal{ALE}$

Just as for $\mathcal{EL}$, Example 28 shows

Proposition 43 The cardinality of a (minimal) s-complete set of matchers might grow exponential in the size of the corresponding $\mathcal{ALE}$-matching problem.

Again, as in Section 3.5, the matching algorithm can be used to show that there always exists an s-complete set of matchers of size at most expontial in the size of the matching problem. Since the size of $\mathcal{G}^\top_{D'}$ for some $\top$-pattern D' of D is linear in D and since $\mathcal{G}_C$ is at most exponential in C the number

of mappings from $\mathcal{G}_{D'}^{\top}$ into $\mathcal{G}_C$ is exponentially bounded. Furthermore, the number of $\top$-patterns of D is exponentially bounded by the size of D. As shown in [3], $lcs(C_{\mathcal{G}_C(varphi(w))} \mid X \in \ell(w))$ corresponds to the product of the description trees $\mathcal{G}_C(\varphi(w))$. Due to the fact that the number of nodes w in $\mathcal{G}_{D'}^{top}$ is linear in the size of the matching problem and the size of $\mathcal{G}_C(\varphi(w))$ is at most exponential in C the lcs can be computed in time exponential in the size of the matching problem $C \equiv^? D$. Thus, the size of every substitution computed by the matching algorithm is at most exponential in the size of the matching problem. As mentioned in Section 2, one can derive a minimal s-complete (i-complete) set from a given s-complete set. To sum up,

Corollary 44 For every $\mathcal{ALE}$-matching problem there exists a (minimal) s-complete (i-complte) set with size at most exponential in the size of the matching problem.

We now consider the complexity of the matching algorithm itself. As shown in [8], subsumption of $\mathcal{ALE}$-concept descriptions is NP-complete. Since a substitution σ computed by the matching algorithm is of size at most exponential in the size of the matching problem $C \equiv^? D$ it follows that $C \equiv^? \sigma(D)$ can be decided in non-deterministic exponential time in the size of the matching problem. Thus, it can be decided in space exponential in the size of the matching algorithm. Recalling that the number of $\top$-patterns D' of D and the number of mappings from $\mathcal{G}_{D'}^{\top}$ into $\mathcal{G}_C$ is at most exponential in the size of D and that the lcs specified in the matching algorithm can be computed in time exponential in the size of the matching problem, our matching algorithms runs in space exponential in the size of the matching problem. Finally, one can obtain a minimal s-complete (i-complete) set from the computed s-complete set in time exponential in the size of the s-complete set.

The matching algorithm can also be used to decide the solvability of a matching problem since an s-complete set of matchers of a matching problem is empty if and only if the matching problem has no solution. In addition, it is easy to see that Lemma 32 is valid for $\mathcal{ALE}$-matching problems as well. For $\mathcal{ALE}$, NP-hardness follows also immediately from the fact that deciding equivalence of $\mathcal{ALE}$-concept descriptions is NP-complete [8] and that matching modulo equivalence corresponds to testing equivalence in case D is a concept description. To sum up, we obtain the following complexity results for matching modulo equivalence in $\mathcal{ALE}$:

Corollary 45 A (minimal) s-complete (i-complete) set of matchers for an $\mathcal{ALE}$-matching problem modulo equivalence can be computed in space exponential in the size of the matching problem. Furthermore, the solvability of an $\mathcal{ALE}$-matching problem is NP-hard and can be decided in non-deterministic exponential time in the size of the matching problem.

Note that if in the matching algorithm mappings from $\mathcal{G}_{D'}$ (instead of $\mathcal{G}_{D'}^{\top}$) had to be considered then the matching algorithm would have been more complex since in this case the number of homomorphisms might be double exponential in the size of the given matching problem. Furthermore, computing the lcs

could be double exponential as well. Thus, as mentioned earlier, the fact that we only need to consider a "minor" normalization of the concept pattern has a great impact on the complexity of the matching algorithm.

It turns out that the two problems of computing an s-complete set of matchers and deciding solvability of matching modulo subsumption are less complex than those for matching modulo equivalence. The reason is that we can dispense with the subsumption test $C \sqsupseteq \sigma(D')$ when it comes to matching modulo subsumption. Furthermore, deciding matching modulo subsumption can be reduced to subsumption (cf. Lemma 9), which is NP-complete for $\mathcal{ALE}$-concept descriptions [8], and subsumption can be reduced two matching. As a result, we obtain

Corollary 46 For an $\mathcal{ALE}$-matching problem $C \sqsubseteq^? D$ a (minimal) s-complete (i-minimal) set of matchers can be computed in time exponential in the size of the matching problem. Deciding the solvability of such a problem is NP-complete.

5 Matching Value Restrictions vs. Matching Existential Restrictions

In this section, we point out the differences between matching in description logics with existential restrictions and those without existential restrictions.

In [1], the language $\mathcal{ALN}$ has been considered which allows for concept conjunction, primitive negation, number restrictions and value restrictions. In particular, this language does not allow for existential restrictions. For this language, it has been shown that for matching there is always a least solution w.r.t. the ordering $\sqsubseteq_s$ in case the problem is solvable at all. In other words, there is always an s-complete set of matcher with at most one element. On the other hand, Example 28 shows that for the languages considered in this work which allow for existential restrictions one matcher might not be sufficient. The number of elements in an s-complete set of matcher might even grow exponential in the size of the matching problem.

Furthermore, there are significant differences regarding the complexity of computing solutions for matching problems and deciding the solvability of a matching problem. In $\mathcal{ALN}$ an s-complete set, i.e., the least solution, can be computed in time polynomial in the size of the matching problem. We have seen that computing s-complete sets for languages with existential restrictions might take time exponential in the size of the matching problem. Moreover, matching in $\mathcal{ALN}$ can be decided in time polynomial in the size of the matching problem, whereas, as proved in this work, deciding matching modulo equivalence is NP-hard even for the small language $\mathcal{EL}$.

6 Conclusion and Future Work

In this work, we have seen that when adding existential restrictions to a description logics there might exist more than one minimal matcher of a matching problem. We therefore have introduced the notion of complete sets, known from unification theory, that contain all minimal solutions of a problem. In addition, compared to languages, like $\mathcal{ALN}$, which do not allow for existential restrictions the complexity of computing matchers and of deciding the solvability of a matching problem increases.

Still, both for $\mathcal{EL}$ and $\mathcal{ALE}$, the complexity of our matching algorithm is not optimal w.r.t. the known lower bounds. Thus, our short-term goal is to obtain tighter complexity bounds for matching in these DLs. We will also try to extend the results to DLs allowing for number restrictions, and—in the long run—to DLs allowing for full negation. We conjecture, however, that DLs with full negation will require techniques quite different from the ones used in this work.

References

[1] F. Baader, R. Küsters, A. Borgida, and D.L. McGuinness. Matching in description logics. *Journal of Logic and Computation*, 9, 1999.

[2] F. Baader, R. Küsters, and R. Molitor. Computing least common subsumer in description logics with existential restrictions. LTCS-Report 98-09, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1998. See http://www-lti.informatik.rwth-aachen.de/Forschung/Papers.html.

[3] F. Baader, R. Küsters, and R. Molitor. Computing least common subsumer in description logics with existential restrictions. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, 1999. To appear.

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

[5] F. Baader and W. Snyder. Unification theory. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, 1999. To appear.

[6] A. Borgida and D. L. McGuinness. 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., 1996. Morgan Kaufmann.

[7] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, L. A. Resnick, and A. Borgida. Living with classic: When and how to use a kl-one-like

language. In J. Sowa, editor, *Principles of Semantic Networks*, pages 401–456. Morgan Kaufmann, San Mateo, Calif., 1991.

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

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

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

[11] D.L. McGuinness and Alex Borgida. Explaining subsumption in description logics. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI'95*, August 1995.

[12] D.L. McGuinness, L. Alperin Resnick, and C. Isbell. Description logic in practice: A classic application. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI'95*, pages 2045–2046, August 1995. Video Presentation.

[13] D.L. McGuinness and J.R. Wright. An industrial strength description logic-based configurator platform. *IEEE Intelligent Systems*, 13(4):66–77, 1998.

[14] J.R. Wright, E.S. Weixelbaum, G.T. Vesonder, K. Brown, S.R. Palmer, J.I. Berman, and H.H. Moore. A knowledge-based configurator that supports sales, engineering, and manufacturing at at&t network systems. *AI Magazine*, 14(3):69–80, 1993.