Matching under Side Conditions in Description Logics *

Franz Baader and **Sebastian Brandt**
Theoretical Computer Science, RWTH Aachen
email: {baader,sbrandt}@cs.rwth-aachen.de

Ralf Küsters
Institute for Computer Science and Applied Mathematics, University of Kiel
email: kuesters@ti.informatik.uni-kiel.de

Abstract

Whereas matching in Description Logics is now relatively well-investigated, there are only very few formal results on matching under additional side conditions, though these side conditions were already present in the original paper by Borgida and McGuinness introducing matching in DLs. The present paper closes this gap for the DL $\mathcal{ALN}$ and its sublanguages: matching under subsumption conditions remains polynomial, while strict subsumption conditions increase the complexity to NP.

1 Introduction

The traditional inference problems (like subsumption) in description logics (DLs) are now well-investigated, which means that there exist complexity results and algorithms for a great variety of DLs of differing expressive power [Donini *et al.*, 1996] as well as optimized implementations of the algorithms for expressive DLs [Horrocks, 1998]. In contrast, matching concepts against patterns is a relatively new inference problem in DLs, 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 system [Brachmann *et al.*, 1991]. 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 [Borgida and Küsters, 2000].

All three applications have in common that one wants to search a large 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. For example, the pattern $D := X \sqcap \forall \mathsf{child}.(Y \sqcap \mathsf{Female})$ looks for concepts that restrict the child role to fillers that are Female, such as the concept $C := (\geq 1\, \mathsf{child}) \sqcap \forall \mathsf{child}.(\mathsf{Female} \sqcap \mathsf{Rich})$. In fact, applying the substitution $\sigma := \{X \mapsto (\geq 1\, \mathsf{child}),\ Y \mapsto \mathsf{Rich}\}$ to the pattern D yields a concept equivalent to C, i.e., σ is a solution (matcher) of the matching problem $C \equiv^? D$.

This type of matching problems has been investigated in detail for sublanguages of the DLs $\mathcal{ALN}$ and $\mathcal{ALE}$ in [Baader *et al.*, 1999] and [Baader and Küsters, 2000], respectively. In particular, it was shown that, for sublanguages of $\mathcal{ALN}$, solvable matching problems always have a least matcher (w.r.t. subsumption), which can be computed in polynomial time. For sublanguages of $\mathcal{ALE}$, deciding solvability of matching problems modulo equivalence is already NP-complete.

In [Borgida and McGuinness, 1996; McGuinness, 1996], the expressivity of matching problems was further enhanced by allowing for additional *side conditions* on the variables (through the as-construct): a (strict) subsumption condition is of the form $X \sqsubseteq^? E$ ($X \sqsubset^? E$) where X is a variable and E a pattern, and it restricts the matchers to substitutions satisfying $\sigma(X) \sqsubseteq \sigma(E)$ ($\sigma(X) \sqsubset \sigma(E)$). Using a subsumption condition, the matching problem of the above example can be written more intuitively as $X \sqcap \forall \mathsf{child}.Z \equiv^? (\geq 1\, \mathsf{child}) \sqcap \forall \mathsf{child}.(\mathsf{Female} \sqcap \mathsf{Rich})$ under the subsumption condition $Z \sqsubseteq^? \mathsf{Female}$. One result of this paper is that also more complex sets of subsumption conditions do not extend the expressive power of matching problems (see below). However, they are often more convenient to state. In contrast, strict subsumption conditions cannot always be simulated by pure matching problems. They can, e.g., be used to avoid trivial matches. For example, the pattern $D' := X \sqcap \forall \mathsf{child}.Y$ matches every concept since $\forall \mathsf{child}.\top \equiv \top$ (where the top concept $\top$ stands for the set of all individuals). The additional strict subsumption condition $Y \sqsubset^? \top$ ensures that we can only match concepts with a real restriction on child.

The first (rather restricted) formal results on matching under side conditions were given in [Baader *et al.*, 1999]: it was shown that matching under strict subsumption conditions in the small DL $\mathcal{FL}_0$ is already NP-hard, and that matching under so-called acyclic subsumption conditions can be reduced to matching without side conditions. However, [Baader *et al.*, 1999] does not give a complexity upper bound for matching under strict subsumption conditions and the reduction for acyclic subsumption conditions given there is exponential.

This paper investigates in detail matching under side conditions in sublanguages of $\mathcal{ALN}$. We will show that matching under subsumption conditions can be reduced in polynomial time to matching without side conditions. In particular,

*This work has been partially supported by the Deutsche Forschungsgemeinschaft, DFG Project BA 1122/4-1.

this implies that solvable matching problems under subsumption conditions in sublanguages of $\mathcal{ALN}$ always have a least matcher, which can be computed in polynomial time. For acyclic strict subsumption conditions, matching is shown to be NP-complete in the sublanguages $\mathcal{FL}_\bot$ and $\mathcal{FL}_\neg$ of $\mathcal{ALN}$.

2 Description logics

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 $\mathcal{FL}_0$, concept descriptions are formed using the constructors top-concept ($\top$), conjunction ($C \sqcap D$), and value restriction ($\forall r.C$). The description logic $\mathcal{FL}_\bot$ additionally provides us with the bottom concept ($\bot$), and $\mathcal{FL}_\neg$ also allows for primitive negation ($\neg P$). Finally, $\mathcal{ALN}$ extends $\mathcal{FL}_\neg$ with number restrictions ($\geq n\ r$) and ($\leq n\ r$) (see Table 1).

As usual, the semantics of concept descriptions is defined in terms of an *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$. The domain $\Delta^\mathcal{I}$ of $\mathcal{I}$ is a non-empty set and the interpretation function $\cdot^\mathcal{I}$ maps each concept name $P \in N_C$ to a set $P^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and each role name $r \in N_R$ to a binary relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The extension of $\cdot^\mathcal{I}$ 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 $C^\mathcal{I} \subseteq D^\mathcal{I}$ holds for all interpretations $\mathcal{I}$; C and D are *equivalent* ($C \equiv D$) iff they subsume each other; C is *strictly subsumed* by D ($C \sqsubset D$) iff $C \sqsubseteq D$ and $C \not\equiv D$. For all DLs listed in Table 1, subsumption can be decided in polynomial time using a structural subsumption algorithm [Borgida and Patel-Schneider, 1994].

Matching in description logics

In order to define concept patterns, we additionally need a set N_X of concept variables, which we assume to be disjoint from $N_C \cup N_R$. Informally, an $\mathcal{ALN}$-concept pattern is an $\mathcal{ALN}$-concept description over the concept names $N_C \cup N_X$ and the role names N_R, with the only exception that primitive negation must not be applied to variables. More formally, *concept patterns* (denoted D, D') are defined using the following syntax rules:

$$D, D' \longrightarrow X \mid C \mid D \sqcap D' \mid \forall r.D,$$

where $X \in N_X$, $r \in N_R$, and C is an $\mathcal{ALN}$-concept description. For example, if X, Y are concept variables, r a role name, and A, B concept names, then $D := A \sqcap X \sqcap \forall r.(B \sqcap Y)$ is an $\mathcal{ALN}$-concept pattern, but $\neg X$ is *not*. The notion of a pattern (and also the notions "substitution" and "matching problem" introduced below) can be restricted to sublanguages of $\mathcal{ALN}$ in the obvious way.

A *substitution* σ is a mapping from N_X into the set of all $\mathcal{ALN}$-concept descriptions. This mapping is extended to concept patterns in the usual way by replacing the occurrences of the variables X in the pattern by the corresponding concept description $\sigma(X)$. 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 $\sigma(D) = A \sqcap A \sqcap B \sqcap \forall r.(B \sqcap A)$. The result of applying a substitution to an $\mathcal{ALN}$-concept pattern is always an $\mathcal{ALN}$-concept description. Note that this would no longer be the case if negation were allowed in front of concept variables.

Subsumption can be extended to substitutions as follows: the substitution σ is subsumed by the substitution τ ($\sigma \sqsubseteq \tau$) iff $\sigma(X) \sqsubseteq \tau(X)$ for all variables $X \in N_X$.

Definition 1 *Let C be an $\mathcal{ALN}$-concept description and D an $\mathcal{ALN}$-concept pattern. Then, $C \equiv^? D$ is an $\mathcal{ALN}$-*matching problem*. The substitution σ is a* solution *(*matcher*) of $C \equiv^? D$ iff $C \equiv \sigma(D)$.*

In the following, we will abbreviate a matching problem of the form $C \equiv^? C \sqcap D$ as $C \sqsubseteq^? D$. This notation is justified by the fact that σ solves $C \equiv^? C \sqcap D$ iff $C \sqsubseteq \sigma(D)$.

A matching problem can either be viewed as a *decision problem*, where one asks whether the problem is solvable, or as a *computation problem*, where one asks for actual matchers of this problem (if any). Although the computation problem is usually the more interesting one, the decision problem can serve as a starting point for the complexity analysis. In general, matching problems may have several (even an infinite number of) solutions, and thus the question arises which matcher to compute. Following [Borgida and McGuinness, 1996; Baader *et al.*, 1999] we will here concentrate on the problem of computing a least matcher (w.r.t. the ordering $\sqsubseteq$ on substitutions).

Instead of a single matching problem, we may also consider finite systems $\{C_1 \equiv^? D_1, \ldots, C_m \equiv^? D_m\}$ of such problems, which must be solved simultaneously. As shown in [Baader *et al.*, 1999], solving such a system can, however, be reduced to solving the single matching problem

$$\forall r_1.C_1 \sqcap \cdots \sqcap \forall r_m.C_m \equiv^? \forall r_1.D_1 \sqcap \cdots \sqcap \forall r_m.D_m$$

where the r_i are pairwise distinct role names.

The following theorem summarizes the results obtained in [Baader and Narendran, 1998; Baader *et al.*, 1999] for matching in (sublanguages of) $\mathcal{ALN}$:

Theorem 2 *Let $\mathcal{L} \in \{\mathcal{FL}_0, \mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}\}$. Then there exists a polynomial time matching algorithm that computes the least matcher of a given system of $\mathcal{L}$-matching problems, if this system has a solution, and returns "fail" otherwise.*

Obviously, this implies that the existence of a solution can be decided in polynomial time. In the sequel, let $\text{MATCH}_\mathcal{L}$ denote an algorithm according to Theorem 2.

Matching under side conditions

In this paper, we focus on more general matching problems, those that allow for additional *side conditions*.

Definition 3 *A* subsumption condition *is of the form $X \sqsubseteq^? E$ where X is a concept variable and E is a pattern; a* strict subsumption condition *is of the form $X \sqsubset E$ where X and E are as above. A* side condition *is either a subsumption condition or a strict subsumption condition. The substitution σ satisfies the side condition $X\ \rho\ E$ for $\rho \in \{\sqsubseteq, \sqsubset\}$ iff $\sigma(X)\ \rho\ \sigma(E)$.*

A matching problem under side conditions *is a tuple $M := \langle C \equiv^? D, S\rangle$, where $C \equiv^? D$ is a matching problem and S*

Syntax	Semantics	$\mathcal{FL}_0$	$\mathcal{FL}_\bot$	$\mathcal{FL}_\neg$	$\mathcal{ALN}$
$\top$	$\Delta^\mathcal{I}$	x	x	x	x
$C \sqcap D$	$C^\mathcal{I} \cap D^\mathcal{I}$	x	x	x	x
$\forall r.C$	$\{x \in \Delta^\mathcal{I} \mid \forall y\colon (x,y) \in r^\mathcal{I} \to y \in C^\mathcal{I}\}$	x	x	x	x
$\bot$	$\emptyset$		x	x	x
$\neg P, P \in N_C$	$\Delta^\mathcal{I} \setminus P^\mathcal{I}$			x	x
$(\geq n\, r), n \in \mathbb{N}$	$\{x \in \Delta^\mathcal{I} \mid \#\{y \mid (x,y) \in r^\mathcal{I}\} \geq n\}$				x
$(\leq n\, r), n \in \mathbb{N}$	$\{x \in \Delta^\mathcal{I} \mid \#\{y \mid (x,y) \in r^\mathcal{I}\} \leq n\}$				x

Table 1: Syntax and semantics of concept descriptions.

is a finite set of side conditions. If the set S contains only subsumption conditions, then M is called matching problem under subsumption conditions. *The substitution σ is a* solution *(*matcher*) of M iff it is a matcher of $C \equiv^? D$ that satisfies every side condition in S.*

In the next section, we will restrict the attention to matching problems under subsumption conditions. Section 4 then treats matching problems under acyclic side conditions.

In order to define matching problems under acyclic side conditions, we say that a variable X *directly depends* on a variable Y in S iff S contains a side condition $X\ \rho\ E$ such that Y occurs in E. If there are $n \geq 1$ variables $X_1, \ldots, X_n$ such that X_i directly depends on X_{i+1} in S $(1 \leq i \leq n-1)$, then we say that X_1 *depends* on X_n in S. The set of side conditions S is *cyclic* iff there is a variable X that depends on itself in S; otherwise, S is *acyclic*.

3 Matching under subsumption conditions

Let $\mathcal{L}$ be one of the DLs $\mathcal{FL}_\bot, \mathcal{FL}_\neg, \mathcal{ALN}$. We present a polynomial time algorithm that, given an $\mathcal{L}$-matching problems under subsumption conditions, returns a least matcher (w.r.t. the ordering $\sqsubseteq$ on substitutions) if the problem is solvable, and "fail" otherwise. In principle, the algorithm iterates the application of $\textsc{match}_\mathcal{L}$ until a fixpoint is reached. However, the matcher computed in one step is used to modify the matching problem to be solved in the next step. Given an $\mathcal{L}$-matching problem under subsumption conditions $M := \langle C \equiv^? D, S\rangle$ and a substitution σ, we define

$$M_\sigma := \{C \equiv^? D\} \cup \{\sigma(X) \sqsubseteq^? E \mid X \sqsubseteq^? E \in S\}.$$

Recall that $\sigma(X) \sqsubseteq^? E$ abbreviates the matching problem $\sigma(X) \equiv^? \sigma(X) \sqcap E$. Thus M_σ is a system of $\mathcal{L}$-matching problems without side conditions, to which $\textsc{match}_\mathcal{L}$ can be applied.

Algorithm 4 *Let $M := \langle C \equiv^? D, S\rangle$ be an $\mathcal{L}$-matching problem under subsumption conditions. Then, the algorithm $\textsc{match}^\sqsubseteq_\mathcal{L}(M)$ works as follows:*

1. *$\sigma(X) := \bot$ for all variables X;*
2. *If $\textsc{match}_\mathcal{L}(M_\sigma)$ returns "fail", then return "fail";*
   else if $\sigma \equiv \textsc{match}_\mathcal{L}(M_\sigma)$, then return σ;
   else $\sigma := \textsc{match}_\mathcal{L}(M_\sigma)$; continue with 2.

Let σ_0 denote the substitution defined in step 1 of the algorithm, and σ_t $(t \geq 1)$ the matcher computed in the t-th iteration of Step 2. Note that σ_t is undefined if $\textsc{match}_\mathcal{L}$ returns "fail" in the t-th iteration or if the algorithm has stopped before the t-th iteration.

To show that the algorithm is correct, we must show soundness, completeness, and termination, i.e., i) if the algorithm terminates and returns a substitution, then this substitution in fact solves the problem; ii) if the algorithm terminates and returns "fail", then there indeed is no solution; and iii) the algorithm halts on every input. The following lemma proves soundness and completeness of the algorithm. The first two items establish a loop invariant.

Lemma 5 *Let $M := \langle C \equiv^? D, S\rangle$ be an $\mathcal{L}$-matching problem under subsumption conditions.*

1. *If σ_t is defined and τ is a solution of M, then $\sigma_t \sqsubseteq \tau$.*
2. *If σ_t, σ_{t+1} are defined, then $\sigma_t \sqsubseteq \sigma_{t+1}$.*
3. *If $\textsc{match}^\sqsubseteq_\mathcal{L}(M)$ returns the substitution σ, then σ solves M (soundness).*
4. *If $\textsc{match}^\sqsubseteq_\mathcal{L}(M)$ returns "fail", then M has no solution (completeness).*

PROOF. 1. Obviously, the claim is true for σ_0. Assume that $\sigma_t \sqsubseteq \tau$, and that σ_{t+1} is defined. To prove $\sigma_{t+1} \sqsubseteq \tau$, it is sufficient to show that τ solves M_{σ_t} since σ_{t+1} is the least solution of M_{σ_t}. Since τ solves M, we know that it solves $C \equiv^? D$ and that $\tau(X) \sqsubseteq \tau(E)$ for all $X \sqsubseteq^? E \in S$. The induction assumption $\sigma_t \sqsubseteq \tau$ implies $\sigma_t(X) \sqsubseteq \tau(X)$, and thus $\sigma_t(X) \sqsubseteq \tau(E)$, which shows that τ solves M_{σ_t}.

2. Obviously, $\sigma_0 \sqsubseteq \sigma_1$. Now assume that $\sigma_{t-1} \sqsubseteq \sigma_t$. Together with the fact that σ_t solves $M_{\sigma_{t-1}}$, this implies that σ_{t+1} solves the system $M_{\sigma_{t-1}}$. Since σ_t is the least solution of $M_{\sigma_{t-1}}$, we can conclude $\sigma_t \sqsubseteq \sigma_{t+1}$.

3. Assume that $\sigma = \sigma_t$. By definition of $\textsc{match}^\sqsubseteq_\mathcal{L}$, $C \equiv \sigma_t(D)$. It remains to show that σ_t solves the side conditions. We know that $\sigma_t \equiv \sigma_{t+1}$ and σ_{t+1} solves M_{σ_t}. Thus, $\sigma_t(X) \sqsubseteq \sigma_{t+1}(E) \equiv \sigma_t(E)$ for every $X \sqsubseteq^? E \in S$.

4. Assume that $\textsc{match}^\sqsubseteq_\mathcal{L}(M)$ returns "fail," and that σ_t is the last substitution computed by the algorithm. Now assume that τ solves M. As in the proof of 1. we can show that τ solves M_{σ_t}. Consequently, M_{σ_t} is solvable, and thus $\textsc{match}(M_{\sigma_t})$ returns the least matcher of this system, in contradiction to the assumption that $\textsc{match}^\sqsubseteq_\mathcal{L}(M)$ returns "fail" in this step of the iteration. ∎

Proving termination of Algorithm 4 is more involved, and the exact argument depends on the DL $\mathcal{L}$ under consideration. Due to the space limitations, in this paper we sketch the proof for the DL $\mathcal{FL}_\perp$ (see [Baader *et al.*, 2000] for complete proofs). The proof depends on the so-called reduced normal form of $\mathcal{FL}_\perp$-concept descriptions, to be introduced next.

It is easy to see that any $\mathcal{FL}_\perp$-concept description can be transformed into an equivalent description that is either $\top$ or a (nonempty) conjunction of descriptions of the form $\forall r_1.\cdots\forall r_m.A$, where $r_1,\ldots,r_m$ are $m \geq 0$ (not necessarily distinct) roles, and A is the bottom concept $\perp$ or a concept name. We abbreviate $\forall r_1.\cdots\forall r_m.A$ by $\forall r_1\ldots r_m.A$, where $r_1\ldots r_m$ is viewed as a word over the alphabet N_R of all role names. If $m = 0$, then this is the empty word ε, and thus $\forall\varepsilon.A$ is our "abbreviation" for A. In addition, instead of $\forall w_1.A \sqcap \ldots \sqcap \forall w_\ell.A$ we write $\forall L.A$ where $L := \{w_1,\ldots,w_\ell\}$ is a finite set of words over N_R; we define $\forall\emptyset.A \equiv \top$. Using these abbreviations, any $\mathcal{FL}_\perp$-concept description C containing only concept names in the finite set $\mathcal{C} \subseteq N_C$ can be written as

$$C \;\equiv\; \forall U_\perp.\perp \sqcap \bigsqcap_{A\in\mathcal{C}} \forall U_A.A,$$

where U_H for $H \in \mathcal{C} \cup \{\perp\}$ are finite sets of words over N_R (called *role languages*). This representation of C will subsequently be called its *U-labeled normal form.*

As an example, consider the $\mathcal{FL}_\perp$-concept description $C_{ex} := \forall r.(\perp \sqcap \forall r.\perp) \sqcap \forall r.\forall s.A \sqcap \forall s.A$. Its $\mathcal{FL}_0$-normal form C'_{ex} is $\forall\{r, rr\}.\perp \sqcap \forall\{rs, s\}.A$.

The role languages may contain redundant words, i.e., words that, when removed, yield equivalent concept descriptions: i) since $\forall w.\perp \sqsubseteq \forall wv.\perp$ for every $w, v \in N_R^*$, we can require $U_\perp$ to be prefix-free, i.e., $w, wv \in U_\perp$ implies $v = \varepsilon$; and ii) since $\forall w.\perp \sqsubseteq \forall wv.A$, we can require $U_A \cap (U_\perp \cdot N_R^*) = \emptyset$. A normal form satisfying these conditions is called *reduced normal form.*

Obviously, any $\mathcal{FL}_\perp$-concept description can be turned into such a reduced normal form (in polynomial time). In our example, the reduced normal form of C_{ex} is $\forall\{r\}.\perp \sqcap \forall\{s\}.A$, which is obtained from C'_{ex} by removing rr from $U_\perp = \{r, rr\}$ and rs from $U_A = \{rs, s\}$. Reduced normal forms can be used to characterize equivalence of $\mathcal{FL}_\perp$-concept descriptions:

Lemma 6 *Assume that the $\mathcal{FL}_\perp$-concept descriptions C, D are given in their U- and V-labeled reduced normal forms, respectively. Then $C \equiv D$ iff $U_H = V_H$ for all $H \in \mathcal{C} \cup \{\perp\}$.*

Termination of Algorithm 4 for $\mathcal{L} = \mathcal{FL}_\perp$

Let the substitutions σ_t be defined as above. We assume that every $\sigma_t(X)$ is given in $U^{t,X}$-labeled reduced normal form, and that C (as defined in Algorithm 4) is in U-labeled reduced normal form. Then, termination follows from the fact that every solvable matching problem under subsumption conditions has a matcher that only uses concept names already contained in the matching problem M, denoted by the set $\mathcal{C} \subseteq N_C$, and the following three properties of the languages $U_H^{t,X}$ and U_H for $H \in \mathcal{C} \cup \{\perp\}$. In the formulation of these properties we implicitly assume that the substitution σ_t is defined whenever we talk about one of the languages $U_H^{t,X}$.

1. For every variable X and every $H \in \mathcal{C} \cup \{\perp\}$, the set $U_H^{t,X}$ contains only suffixes of U_H.
2. For every word w, if $w \in U_H^{t,X} \setminus U_H^{t+1,X}$, then $w \notin U_H^{t',X}$ for any $t' > t$.
3. If σ_t and σ_{t+1} are defined and $\sigma_t \not\equiv \sigma_{t+1}$, then there exists an $H \in \mathcal{C} \cup \{\perp\}$, a variable X, and a word w such that $w \in U_H^{t,X} \setminus U_H^{t+1,X}$.

A complete proof of these properties can be found in [Baader *et al.*, 2000]. It also depends on details of the algorithm $\text{MATCH}_\mathcal{L}$, which we have here introduced only as a black box. Note that these properties would not hold if we did not use *reduced* normal forms.

Given these properties, it is now easy to show that the algorithm halts after a polynomial number of steps. In fact, Property 1 yields a polynomial upper bound on the size of the role languages $U_H^{t,X}$. Property 3 shows that in every step of the iteration at least one word is removed from one of these languages, and Property 2 ensures that words that have been removed cannot reappear.

To sum up, we have shown the following theorem.

Theorem 7 *Let $\mathcal{L} \in \{\mathcal{FL}_\perp, \mathcal{FL}_\neg, \mathcal{ALN}\}$. The algorithm $\text{MATCH}_\mathcal{L}^\sqsubseteq$ is a polynomial time algorithm that, given an $\mathcal{L}$-matching problem with subsumption conditions, returns a least matcher of this problem if it is solvable, and "fail" otherwise.*

It should be noted that the algorithm $\text{MATCH}_\mathcal{L}^\sqsubseteq$ does not work for $\mathcal{L} = \mathcal{FL}_0$ since this language does not allow for the bottom concept, and thus the initialization step (Step 1) is not possible. However, instead of starting with $\sigma(X) := \perp$, the algorithm can also start from the least matcher of $C \equiv^? D$. In case the side conditions do not introduce new variables (i.e., variables not contained in D), this modification works and yields a polynomial time matching algorithm. In contrast, if new variables are introduced, then we can show [Baader *et al.*, 2000] that the size of the least matcher may grow exponentially in the size of the matching problem, and that there exists an exponential time algorithm for computing such matchers. Nevertheless, the size of the substitutions for variables in D can still be bounded polynomially, and if one is only interested in substitutions for these variables, then these can still be computed in polynomial time.

4 Matching under acyclic side conditions

Matching under acyclic side conditions (i.e., strict and non-strict acyclic subsumption conditions) is more complex than matching under subsumption conditions for two reasons.

First, as already shown in [Baader *et al.*, 1999], deciding the solvability of an $\mathcal{FL}_0$-matching problem under strict (and acyclic) subsumption conditions is NP-hard. It is easy to see that the same reduction works for the DLs $\mathcal{FL}_\perp$, $\mathcal{FL}_\neg$, and $\mathcal{ALN}$. Thus, assuming that P $\neq$ NP, there cannot exist a polynomial time algorithm computing matchers of matching problems under general side conditions.

Second, as shown by the following example, solvable matching problems under strict subsumption conditions no longer need to have a *least* matcher (but rather finitely many *minimal* matchers).

Example 8 Consider the $\mathcal{FL}_\bot$-matching problem $A_1 \sqcap \ldots \sqcap A_n \equiv^? X_1 \sqcap \ldots \sqcap X_n$ under the strict acyclic subsumption conditions $\{X_{i+1} \sqsubset^? X_i \mid 1 \le i \le n-1\} \cup \{X_1 \sqsubset^? \top\}$.

The pure matching problem enforces that each X_i must be replaced by a (possibly empty) conjunction of concept names from $\{A_1, \ldots, A_n\}$. Thus, the strict subsumption conditions can only be satisfied if X_1 is replaced by one of these names, X_2 by a conjunction of this name with an additional one, etc. From this it is easy to derive that the matchers of the problem are of the following form: given a permutation $P := (p_1, \ldots, p_n)$ of $(1, \ldots, n)$, the substitution σ^P is defined by $\sigma^P(X_i) := A_{p_1} \sqcap \ldots \sqcap A_{p_i}$ $(1 \le i \le n)$. Thus, there are $n!$ non-equivalent matchers, and it is easy to see that each of them is minimal.

The new contribution of this section is a (non-deterministic) algorithm, $\textsc{match}_{\mathcal{L}}^{\sqsubset}$, that computes matchers of $\mathcal{L}$-matching problems under acyclic side conditions for $\mathcal{L} \in \{\mathcal{FL}_\bot, \mathcal{FL}_\neg\}$. This non-deterministic algorithm matches the lower complexity bound (NP-hard) for the decision problem in the following sense. The length of every computation path of this algorithm is polynomially bounded in the size of the given matching problem. In case the problem is not solvable, every computation returns "fail". Otherwise, the successful computation paths yield all minimal matchers.

The algorithm handling acyclic side conditions

In the following, let $M = \langle C \equiv^? D, S\rangle$ be an $\mathcal{L}$-matching problem ($\mathcal{L} \in \{\mathcal{FL}_\bot, \mathcal{FL}_\neg\}$) under acyclic side conditions. We assume that $S = \{X_1 \, \rho_1^? \, E_1, \ldots, X_\ell \, \rho_\ell^? \, E_\ell\}$ for distinct variables $X_1, \ldots, X_\ell$ and patterns $E_1, \ldots, E_\ell$ such that E_i does not contain the variables $X_i, \ldots, X_\ell$. (The case where not all the left-hand side variables are distinct can be treated similarly.) We denote by $S_{\sqsubseteq}$ the set of side conditions obtained from S by replacing every ρ_i by $\sqsubseteq$.

Applied to input M, the algorithm $\textsc{match}_{\mathcal{L}}^{\sqsubset}$ first calls $\textsc{match}_{\mathcal{L}}^{\sqsubseteq}(\langle C \equiv^? D, S_{\sqsubseteq}\rangle)$. If this yields "fail", then M is also unsolvable. Otherwise, the computed substitution σ solves $C \equiv^? D$, but may still violate some of the strict subsumption conditions. Starting with the violated side condition with the largest index, the algorithm tries to modify σ such that this side condition is satisfied.

Assume that $X_k \sqsubset^? E_k$ is this side condition. Since σ solves $X_k \sqsubseteq^? E_k$, we know that $\sigma(X_k) \equiv \sigma(E_k)$. Thus, we must either make $\sigma(X_k)$ more specific or $\sigma(E_k)$ more general. Since $\textsc{match}_{\mathcal{L}}^{\sqsubseteq}$ computes the least solution, the first option cannot lead to a solution of the overall system. Hence, we must try the second one. The idea (which will be explained in more detail later) is that we consider the reduced normal form of $\sigma(E_k)$. We try to make $\sigma(E_k)$ more general by (non-deterministically) choosing one word from one of its role languages and by removing this word by appropriately modifying the role languages of the variables occurring in E_k. Since we want to compute minimal matchers, we make as little changes as possible in order to keep the substitution as specific as possible.

The new substitution σ' obtained this way solves $X_k \sqsubset^? E_k$, and since we only modified variables occurring in E_k, the side conditions with larger index are still satisfied. However, the side conditions with smaller index (even the non-strict ones) as well as the matching problem need no longer be solved by σ'. To overcome this problem, $\textsc{match}_{\mathcal{L}}^{\sqsubseteq}$ is used to compute the least substitution that (i) solves $\langle C \equiv^? D, S_{\sqsubseteq}\rangle$, and (ii) subsumes σ'. It can be shown that the second condition (which can be expressed by a system of matching problems) makes sure that the computed substitution still solves the strict subsumption conditions from index k to ℓ. We can now continue the modification process with this substitution.

Algorithm 9 *Let $M = \langle C \equiv^? D, S\rangle$ be an $\mathcal{L}$-matching problem under acyclic side conditions. Then, $\textsc{match}_{\mathcal{L}}^{\sqsubset}$ works as follows:*

1. *If $\textsc{match}_{\mathcal{L}}^{\sqsubseteq}(\langle C \equiv^? D, S_{\sqsubseteq}\rangle)$ returns "fail", then return "fail";*
2. $k := \ell$; $\sigma := \textsc{match}_{\mathcal{L}}^{\sqsubseteq}\langle\{C \equiv^? D\}, S_{\sqsubseteq}\rangle$;
3. *If $k = 0$, then return σ;*
   If $\sigma(X_k) \; \rho_k \; \sigma(E_k)$, then continue with 5.
4. *Guess modification σ' of σ for $X_k \sqsubset^? E_k$;*
   If $\sigma'(E_k) \equiv \sigma(E_k)$, then return "fail";
   $M' := \langle\{C \equiv^? D\} \cup \{\sigma'(X_j) \sqsubseteq^? X_j | 1 \le j \le \ell\}, S_{\sqsubseteq}\rangle$;
   If $\textsc{match}_{\mathcal{L}}^{\sqsubseteq}(M')$ returns "fail", then return "fail";
   $\sigma := \textsc{match}_{\mathcal{L}}^{\sqsubseteq}(M')$
5. $k := k - 1$*; continue with 3.*

How to guess modifications

Here we sketch the modifications for $\mathcal{FL}_\bot$. (A formal definition for $\mathcal{FL}_\bot$ and $\mathcal{FL}_\neg$ can be found in [Baader *et al.*, 2000].) Recall that the goal is to make $\sigma(E_k)$ more general by (non-deterministically) choosing one word w from one of its role languages and by removing this word by appropriately modifying the role languages of the variables occurring in E_k.

We call this a $\mathcal{C}$*-modification* if w is picked from a role language corresponding to some atomic concept A. In this case, removing certain words from role languages of the variables in E suffices to obtain a minimal modification.

In case of a $\bot$*-modification*, where w is picked from the role language corresponding to the $\bot$-concept, the removal of some word v in the role language of a variable implicitly removes every continuation vv' of v. To correct this effect, every word in $\{v\}\cdot N_R$ is put back whenever some v is removed. In addition, since v is also implicitly removed from role languages corresponding to atomic concepts, it is also transferred to such role languages. This ensures that the computed substitution is as specific as possible. This is vital both for the proof of correctness and to obtain all minimal solutions.

The following example illustrates in more detail how the modifications work.

Example 10 Consider the $\mathcal{FL}_\bot$-matching problem $A \sqcap \forall\{r, s\}.\bot \equiv^? X_1 \sqcap \forall r.X_2 \sqcap \forall r.X_3$ under the strict subsumption conditions $X_2 \sqsubset^? X_1, \; X_3 \sqsubset^? X_2$.

Executing the above algorithm, we obtain in Step 2 as initial solution σ the following substitution:

$\{X_1 \mapsto \forall\{r,s\}.\bot \sqcap \forall\{\varepsilon\}.A, X_2 \mapsto \forall\{\varepsilon\}.\bot, X_3 \mapsto \forall\{\varepsilon\}.\bot\}$.

The iteration begins in Step 3 by checking the second side condition, which is violated. Choosing a $\bot$-modification in Step 4, we must choose a word from the role language $\{\varepsilon\}$ corresponding to $\bot$ in $\sigma(X_2) = \sigma(X_3)$. In this case, we can only pick ε. To keep the change minimal, we do not simply remove it, but rather replace it by $\{r,s\}$ in the role language corresponding to $\bot$ in $\sigma(X_2)$. In addition, we transfer ε to the role language corresponding to A. This yields $\sigma'(X_2) = \forall\{r,s\}.\bot \sqcap \forall\{\varepsilon\}.A$. The other variables remain unchanged.

In this case, the substitution σ' itself solves the matching problem M' considered in Step 4, and thus $\text{MATCH}^{\sqsubseteq}_{\mathcal{FL}_\bot}(M')$ returns σ'.

In the second iteration, we find in Step 3 that the first side condition $X_2 \sqsubset^? X_1$ no longer holds. In Step 4, we again choose a $\bot$-modification, and choose the word r from the role language $\{r,s\}$ corresponding to $\bot$ in $\sigma(X_1)$. The modification replaces r by rr, rs and adds r to the role language corresponding to A. This yields $\sigma'(X_1) := \forall\{rr,rs,s\}.\bot \sqcap \forall\{\varepsilon,r\}.A$. Again, this substitution solves M', and thus the new value of σ is σ'.

In the next iteration we have $k = 0$, ending the iteration in Step 3. The algorithm finally returns the substitution $\{X_1 \mapsto \forall\{rr,rs,s\}.\bot \sqcap \forall\{\varepsilon,r\}.A, X_2 \mapsto \forall\{r,s\}.\bot \sqcap \forall\{\varepsilon\}.A, X_3 \mapsto \forall\{\varepsilon\}.\bot\}$.

Note that, in the first iteration step, it was not possible to apply a $\mathcal{C}$-modification since the role language corresponding to A was empty. In the second step, we could have applied a $\mathcal{C}$-modification, removing ε from the role language corresponding to A in $\sigma(X_1)$. Then, however, the system M' obtained this way would not have been solvable. In fact, it is easy to see that the two matching problems $A \sqcap \forall\{r,s\}.\bot \equiv^? X_1 \sqcap \forall r.X_2 \sqcap \forall r.X_3$ and $\forall\{r,s\}.\bot \sqsubseteq^? X_1$ occurring in M' cannot be solved simultaneously.

Soundness and completeness for $\mathcal{FL}_\bot$ and $\mathcal{FL}_\neg$ is proved in [Baader *et al.*, 2000]. It is easy to see that the length of each computation branch of the nondeterminstic algorithm $\text{MATCH}^{\sqsubseteq}_{\mathcal{L}}$ is polynomially bounded. Because matching under strict acyclic subsumption conditions in $\mathcal{FL}_\bot$ and $\mathcal{FL}_\neg$ is known to be NP-hard, we obtain the following theorem.

Theorem 11 *Let $\mathcal{L} \in \{\mathcal{FL}_\bot, \mathcal{FL}_\neg\}$. Deciding the solvability of $\mathcal{L}$-matching problems under acyclic side conditions is an NP-complete problem.*

The algorithm described above not only decides solvability of matching problems under acyclic side conditions. It also computes matchers in case the problem is solvable. To be more precise, all minimal matchers (w.r.t. subsumption of substitutions) are computed.

5 Future work

In contrast to the algorithm for matching under subsumption conditions, the algorithm $\text{MATCH}^{\sqsubseteq}_{\mathcal{L}}$ presently is restricted to (i) $\mathcal{L} \in \{\mathcal{FL}_\bot, \mathcal{FL}_\neg\}$ and (ii) acyclic side conditions. Naturally, the next step is to overcome the first restriction by extending the result for acyclic side conditions to $\mathcal{ALN}$. We strongly conjecture that an algorithm similar to $\text{MATCH}^{\sqsubseteq}_{\mathcal{L}}$ can be devised for this purpose.

In [Baader *et al.*, 1999], $\mathcal{FL}_0$-matching problems with cyclic subsumption conditions are transformed into equivalent ones with acyclic subsumption conditions. Unfortunately, this approach cannot be extended to $\mathcal{FL}_\bot$ in a straightforward way. Instead, in order to overcome the second restriction, one might try to adapt the general idea of the algorithm $\text{MATCH}^{\sqsubseteq}_{\mathcal{L}}$ to cyclic side conditions. Hence, one would first compute the solution to the analogous non-strict matching problem and then proceed by iteratively modifying the solution to satisfy the strict side conditions. Proving termination of this approach appears to be the main problem.

Apart from $\mathcal{ALN}$ and its sublanguages, it also seems promising to consider matching problems under side conditions in $\mathcal{ALE}$, which extends $\mathcal{FL}_\neg$ by existential restrictions. Since the algorithm $\text{MATCH}^{\sqsubseteq}_{\mathcal{L}}$ proposed here is defined on top of an existing algorithm for pure matching problems (with certain properties), one might similarly take advantage of the matching algorithm proposed in [Baader and Küsters, 2000] for $\mathcal{ALE}$-matching problems.

References

[Baader *et al.*, 2000] F. Baader, S. Brandt, and R. Küsters. Matching under Side Conditions in Description Logics. LTCS-Report 01-02, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 2000. See http://www-lti.informatik.rwth-aachen.de/Forschung/Reports.html.

[Baader and Küsters, 2000] F. Baader and R. Küsters. Matching in description logics with existential restrictions. In *Proc. of KR2000*, pp. 261–272, Morgan Kaufmann Publishers, 2000.

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

[Baader and Narendran, 1998] F. Baader and P. Narendran. Unification of concept terms in description logics. In *Proc. of ECAI-98*, pp. 331–335, John Wiley & Sons Ltd., 1998.

[Borgida and Küsters, 2000] A. Borgida and R. Küsters. What's not in a name: some properties of a purely structural approach to integrating large DL knowledge bases. In *Proc. of DL2000*, number 33 in *Proc.* of CEUR-WS, 2000.

[Borgida and McGuinness, 1996] A. Borgida and D. L. McGuinness. Asking queries about frames. In *Proc. of KR'96*, pp. 340–349, Morgan Kaufmann Publishers, 1996.

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

[Brachmann *et al.*, 1991] 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. *Principles of Semantic Networks*, pp. 401–456. Morgan Kaufmann Publishers, 1991.

[Donini *et al.*, 1996] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. *Principles of Knowledge Representation*, Studies in Logic, Language and Information, pp. 193–238. CSLI Publications, 1996.

[Horrocks, 1998] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR'98*, pp. 636–647, Morgan Kaufmann Publishers, 1998.

[McGuinness, 1996] D. L. McGuinness. *Explaining Reasoning in Description Logics*. PhD thesis, Department of Computer Science, Rutgers University, October, 1996.