Dismatching and Local Disunification in $\mathcal{EL}$

Franz Baader, Stefan Borgwardt, and Barbara Morawska*

Theoretical Computer Science, TU Dresden, Germany
{baader,stefborg,morawska}@tcs.inf.tu-dresden.de

Abstract

Unification in Description Logics has been introduced as a means to detect redundancies in ontologies. We try to extend the known decidability results for unification in the Description Logic $\mathcal{EL}$ to disunification since negative constraints on unifiers can be used to avoid unwanted unifiers. While decidability of the solvability of general $\mathcal{EL}$-disunification problems remains an open problem, we obtain NP-completeness results for two interesting special cases: *dismatching problems*, where one side of each negative constraint must be ground, and *local* solvability of disunification problems, where we restrict the attention to solutions that are built from so-called atoms occurring in the input problem. More precisely, we first show that dismatching can be reduced to local disunification, and then provide two complementary NP-algorithms for finding local solutions of (general) disunification problems.

1998 ACM Subject Classification I.2.3 Deduction and Theorem Proving, I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases Unification, Description Logics, SAT

Digital Object Identifier 10.4230/LIPIcs.RTA.2015.40

1 Introduction

Description logics (DLs) [6] are a family of logic-based knowledge representation formalisms, which can be used to represent the conceptual knowledge of an application domain in a structured and formally well-understood way. They are employed in various application areas, but their most notable success so far is the adoption of the DL-based language OWL [21] as standard ontology language for the semantic web. DLs allow their users to define the important notions (classes, relations) of the domain using concepts and roles; to state constraints on the way these notions can be interpreted using terminological axioms; and to deduce consequences such as subsumption (subclass) relationships from the definitions and constraints. The expressivity of a particular DL is determined by the constructors available for building concepts.

The DL $\mathcal{EL}$, which offers the concept constructors conjunction ($\sqcap$), existential restriction ($\exists r.C$), and the top concept ($\top$), has drawn considerable attention in the last decade since, on the one hand, important inference problems such as the subsumption problem are polynomial in $\mathcal{EL}$, even with respect to expressive terminological axioms [16]. On the other hand, though quite inexpressive, $\mathcal{EL}$ is used to define biomedical ontologies, such as the large medical ontology SNOMED CT.[1] For these reasons, the most recent OWL version, OWL 2, contains the profile OWL 2 EL,[2] which is based on a maximally tractable extension of $\mathcal{EL}$ [5].

* Supported by DFG under grant BA 1122/14-2.

[1] http://www.ihtsdo.org/snomed-ct/

[2] http://www.w3.org/TR/owl2-profiles/

Unification in Description Logics was introduced in [12] as a novel inference service that can be used to detect redundancies in ontologies. It is shown there that unification in the DL $\mathcal{FL}_0$, which differs from $\mathcal{EL}$ in that existential restriction is replaced by value restriction ($\forall r.C$), is ExpTime-complete. The applicability of this result was not only hampered by this high complexity, but also by the fact that $\mathcal{FL}_0$ is not used in practice to formulate ontologies.

In contrast, as mentioned above, $\mathcal{EL}$ is employed to build large biomedical ontologies for which detecting redundancies is a useful inference service. For example, assume that one developer of a medical ontology defines the concept of a *patient with severe head injury* as

$$\mathsf{Patient} \sqcap \exists \mathsf{finding}.(\mathsf{Head\_injury} \sqcap \exists \mathsf{severity}.\mathsf{Severe}), \tag{1}$$

whereas another one represents it as

$$\mathsf{Patient} \sqcap \exists \mathsf{finding}.(\mathsf{Severe\_finding} \sqcap \mathsf{Injury} \sqcap \exists \mathsf{finding\_site}.\mathsf{Head}). \tag{2}$$

Formally, these two concepts are not equivalent, but they are nevertheless meant to represent the same concept. They can obviously be made equivalent by treating the concept names Head_injury and Severe_finding as variables, and substituting the first one by $\mathsf{Injury} \sqcap \exists \mathsf{finding\_site}.\mathsf{Head}$ and the second one by $\exists \mathsf{severity}.\mathsf{Severe}$. In this case, we say that the concepts are unifiable, and call the substitution that makes them equivalent a *unifier*. In [10], we were able to show that unification in $\mathcal{EL}$ is of considerably lower complexity than unification in $\mathcal{FL}_0$: the decision problem for $\mathcal{EL}$ is NP-complete. The main idea underlying the proof of this result is to show that any solvable $\mathcal{EL}$-unification problem has a local unifier, i.e., a unifier built from a polynomial number of so-called atoms determined by the unification problem. However, the brute-force "guess and then test" NP-algorithm obtained from this result, which guesses a local substitution and then checks (in polynomial time) whether it is a unifier, is not useful in practice. We thus developed a goal-oriented unification algorithm for $\mathcal{EL}$, which is more efficient since nondeterministic decisions are only made if they are triggered by "unsolved parts" of the unification problem. Another option for obtaining a more efficient unification algorithm is a translation to satisfiability in propositional logic (SAT): in [9] it is shown how a given $\mathcal{EL}$-unification problem Γ can be translated in polynomial time into a propositional formula whose satisfying valuations correspond to the local unifiers of Γ.

Intuitively, a unifier of two $\mathcal{EL}$ concepts proposes definitions for the concept names that are used as variables: in our example, we know that, if we define Head_injury as $\mathsf{Injury} \sqcap \exists \mathsf{finding\_site}.\mathsf{Head}$ and Severe_finding as $\exists \mathsf{severity}.\mathsf{Severe}$, then the two concepts (1) and (2) are equivalent w.r.t. these definitions. Of course, this example was constructed such that the unifier (which is actually local) provides sensible definitions for the concept names used as variables. In general, the existence of a unifier only says that there is a structural similarity between the two concepts. The developer who uses unification as a tool for finding redundancies in an ontology or between two different ontologies needs to inspect the unifier(s) to see whether the definitions it suggests really make sense. For example, the substitution that replaces Head_injury by $\mathsf{Patient} \sqcap \mathsf{Injury} \sqcap \exists \mathsf{finding\_site}.\mathsf{Head}$ and Severe_finding by $\mathsf{Patient} \sqcap \exists \mathsf{severity}.\mathsf{Severe}$ is also a local unifier, which however does not make sense. Unfortunately, even small unification problems like the one in our example can have too many local unifiers for manual inspection. In [2] we propose to restrict the attention to so-called minimal unifiers, which form a subset of all local unifiers. In our example, the nonsensical unifier is indeed not minimal. In general, however, the restriction to minimal unifiers may preclude interesting local unifiers. In addition, as shown in [2], computing minimal unifiers is actually harder than computing local unifiers (unless the polynomial hierarchy collapses). In the present paper, we propose disunification as a more direct approach

for avoiding local unifiers that do not make sense. In addition to positive constraints (requiring equivalence or subsumption between concepts), a disunification problem may also contain negative constraints (preventing equivalence or subsumption between concepts). In our example, the nonsensical unifier can be avoided by adding the dissubsumption constraint

$$\mathsf{Head\_injury} \not\sqsubseteq^? \mathsf{Patient} \tag{3}$$

to the equivalence constraint (1) $\equiv^?$ (2).

Unification and disunification in DLs is actually a special case of unification and disunification modulo equational theories (see [12] and [10] for the equational theories respectively corresponding to $\mathcal{FL}_0$ and $\mathcal{EL}$). Disunification modulo equational theories has, e.g., been investigated in [17, 18]. It is well-known in unification theory that for effectively finitary equational theories, i.e., theories for which finite complete sets of unifiers can effectively be computed, disunification can be reduced to unification: to decide whether a disunification problem has a solution, one computes a finite complete set of unifiers of the equations and then checks whether any of the unifiers in this set also solves the disequations. Unfortunately, for $\mathcal{FL}_0$ and $\mathcal{EL}$, this approach is not feasible since the corresponding equational theories have unification type zero [10, 12], and thus finite complete sets of unifiers need not even exist. Nevertheless, it was shown in [14] that the approach used in [12] to decide unification (reduction to language equations, which are then solved using tree automata) can be adapted such that it can also deal with disunification. This yields the result that disunification in $\mathcal{FL}_0$ has the same complexity (ExpTime-complete) as unification.

For $\mathcal{EL}$, going from unification to disunification appears to be more problematic. In fact, the main reason for unification to be decidable and in NP is locality: if the problem has a unifier then it has a local unifier. We will show that disunification in $\mathcal{EL}$ is not local in this sense by providing an example of a disunification problem that has a solution, but no local solution. Decidability and complexity of disunification in $\mathcal{EL}$ remains an open problem, but we provide partial solutions that are of interest in practice. On the one hand, we investigate *dismatching problems*, i.e., disunification problems where the negative constraints are dissubsumptions $C \not\sqsubseteq^? D$ for which C or D is ground (i.e., does not contain a variable). Note that the dissubsumption (3) from above actually satisfies this restriction since $\mathsf{Patient}$ is not a variable. We prove that (general) solvability of dismatching problems can be reduced to *local disunification*, i.e., the question whether a given $\mathcal{EL}$-disunification problem has a *local* solution, which shows that dismatching in $\mathcal{EL}$ is NP-complete. On the other hand, we develop two specialized algorithms to solve local disunification problems that extend the ones for unification [9, 10]: a goal-oriented algorithm that reduces the amount of nondeterministic guesses necessary to find a local solution, as well as a translation to SAT. The reason we present two kinds of algorithms is that, in the case of unification, they have proved to complement each other well in first evaluations [1]: the goal-oriented algorithm needs less memory and finds minimal solutions faster, while the SAT reduction generates larger data structures (of cubic size), but outperforms the goal-oriented algorithm on unsolvable problems.

Full proofs of the results presented below can be found in [4].

2 Subsumption and dissubsumption in $\mathcal{EL}$

The syntax of $\mathcal{EL}$ is defined based on two sets $\mathsf{N_C}$ and $\mathsf{N_R}$ of *concept names* and *role names*, respectively. *Concept terms* are built from concept names using the constructors *conjunction* ($C \sqcap D$), *existential restriction* ($\exists r.C$ for $r \in \mathsf{N_R}$), and *top* ($\top$). An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$

Table 1 Syntax and semantics of $\mathcal{EL}$.

Name	Syntax	Semantics
top	$\top$	$\top^{\mathcal{I}} := \Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} := \{x \mid \exists y.(x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$

consists of a non-empty domain $\Delta^{\mathcal{I}}$ and an interpretation function that maps concept names to subsets of $\Delta^{\mathcal{I}}$ and role names to binary relations over $\Delta^{\mathcal{I}}$. This function is extended to concept terms as shown in the semantics column of Table 1.

A concept term C is *subsumed* by a concept term D (written $C \sqsubseteq D$) if for every interpretation $\mathcal{I}$ it holds that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. We write a *dissubsumption* $C \not\sqsubseteq D$ to abbreviate the fact that $C \sqsubseteq D$ does not hold. The two concept terms C and D are *equivalent* (written $C \equiv D$) if $C \sqsubseteq D$ and $D \sqsubseteq C$. Note that we use "=" to denote *syntactic* equality between concept terms, whereas "≡" denotes semantic equivalence.

Since conjunction is interpreted as intersection, we can treat $\sqcap$ as a commutative and associative operator, and thus dispense with parentheses in nested conjunctions. An *atom* is a concept name or an existential restriction. Hence, every concept term C is a conjunction of atoms or $\top$. We call the atoms in this conjunction the *top-level atoms* of C. Obviously, C is equivalent to the conjunction of its top-level atoms, where the empty conjunction corresponds to $\top$. An atom is *flat* if it is a concept name or an existential restriction of the form $\exists r.A$ with $A \in \mathsf{N_C}$.

Subsumption in $\mathcal{EL}$ is decidable in polynomial time [8] and can be checked by recursively comparing the top-level atoms of the two concept terms.

▶ **Lemma 1** ([10]). *For two atoms C, D, we have $C \sqsubseteq D$ iff $C = D$ is a concept name or $C = \exists r.C'$, $D = \exists r.D'$, and $C' \sqsubseteq D'$. If C, D are concept terms, then $C \sqsubseteq D$ iff for every top-level atom D' of D there is a top-level atom C' of C such that $C' \sqsubseteq D'$.*

We obtain the following contrapositive formulation characterizing dissubsumption.

▶ **Lemma 2.** *For two concept terms C, D, we have $C \not\sqsubseteq D$ iff there is a top-level atom D' of D such that for all top-level atoms C' of C it holds that $C' \not\sqsubseteq D'$.*

In particular, $C \not\sqsubseteq D$ is characterized by the existence of a top-level atom D' of D for which $C \not\sqsubseteq D'$ holds. By further analyzing the structure of atoms, we obtain the following.

▶ **Lemma 3.** *Let C, D be two atoms. Then we have $C \not\sqsubseteq D$ iff either*
1. *C or D is a concept name and $C \neq D$; or*
2. *$D = \exists r.D'$, $C = \exists s.C'$, and $r \neq s$; or*
3. *$D = \exists r.D'$, $C = \exists r.C'$, and $C' \not\sqsubseteq D'$.*

3 Disunification

As described in the introduction, we now partition the set $\mathsf{N_C}$ into a set of *(concept) variables* ($\mathsf{N_v}$) and a set of *(concept) constants* ($\mathsf{N_c}$). A concept term is *ground* if it does not contain any variables. We define a quite general notion of disunification problems that is similar to the equational formulae used in [18].

▶ **Definition 4.** A *disunification problem* Γ is a formula built from subsumptions of the form $C \sqsubseteq^? D$, where C and D are concept terms, using the logical connectives $\wedge$, $\vee$, and $\neg$. We use equations $C \equiv^? D$ to abbreviate $(C \sqsubseteq^? D) \wedge (D \sqsubseteq^? C)$, disequations $C \not\equiv^? D$ for $\neg(C \sqsubseteq^? D) \vee \neg(D \sqsubseteq^? C)$, and dissubsumptions $C \not\sqsubseteq^? D$ instead of $\neg(C \sqsubseteq^? D)$. A *basic disunification problem* is a conjunction of subsumptions and dissubsumptions. A *dismatching problem* is a basic disunification problem in which all dissubsumptions $C \not\sqsubseteq^? D$ are such that C or D is ground. Finally, a *unification problem* is a conjunction of subsumptions.

The definition of dismatching problems is partially motivated by the definition of *matching* in description logics, where similar restrictions are imposed on unification problems [7, 11, 23]. Another motivation comes from our experience that dismatching problems already suffice to formulate most of the negative constraints one may want to put on unification problems, as described in the introduction.

To define the semantics of disunification problems, we now fix a *finite signature* $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ and assume that all disunification problems contain only concept terms constructed over the symbols in Σ. A *substitution* σ maps every variable in Σ to a ground concept term constructed over the symbols of Σ. This mapping can be extended to all concept terms (over Σ) in the usual way. A substitution σ *solves* a subsumption $C \sqsubseteq^? D$ if $\sigma(C) \sqsubseteq \sigma(D)$; it *solves* $\Gamma_1 \wedge \Gamma_2$ if it solves both Γ_1 and Γ_2; it solves $\Gamma_1 \vee \Gamma_2$ if it solves Γ_1 or Γ_2; and it solves $\neg\Gamma$ if it does not solve Γ. A substitution that solves a given disunification problem is called a *solution* of this problem. A disunification problem is *solvable* if it has a solution.

In contrast to unification, in disunification it does make a difference whether or not solutions may contain variables from $\mathsf{N_v} \cap \Sigma$ or additional symbols from $(\mathsf{N_C} \cup \mathsf{N_R}) \setminus \Sigma$ [17]. In the context of the application sketched in the introduction, restricting solutions to ground terms over Σ is appropriate: the finite signature Σ contains exactly the symbols that occur in the ontology to be checked for redundancy, and since a solution σ is supposed to provide definitions for the variables in Σ, it should not use the variables themselves to define them; moreover, definitions that contain symbols that are not in Σ would be meaningless to the user.

Reduction to basic disunification problems

We will consider only basic disunification problems in the following. The reason is that there is a straightforward NP-reduction from solvability of arbitrary disunification problems to solvability of basic disunification problems. In this reduction, we view all subsumptions occurring in the disunification problem as propositional variables and guess a satisfying valuation of the resulting propositional formula. It then suffices to check solvability of the basic disunification problem obtained as the conjunction of all subsumptions evaluated to true and the negations of all subsumptions evaluated to false. Since the problems considered in the following sections are all NP-complete, the restriction to basic disunification problems does not affect our complexity results. In the following, we thus restrict the attention to basic disunification problems, which we simply call *disunification problems* and consider them to be sets of subsumptions and dissubsumptions.

Reduction to flat disunification problems

We further simplify our analysis by considering *flat* disunification problems, which means that they may only contain *flat* dissubsumptions of the form $C_1 \sqcap \cdots \sqcap C_n \not\sqsubseteq^? D_1 \sqcap \cdots \sqcap D_m$

for flat atoms $C_1, \ldots, C_n, D_1, \ldots, D_m$ with $m, n \geq 0$,[3] and *flat* subsumptions of the form $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D_1$ for flat atoms $C_1, \ldots, C_n, D_1$ with $n \geq 0$.

The restriction to flat disunification problems is without loss of generality: to flatten concept terms, one can simply introduce new variables and equations to abbreviate subterms [10]. Moreover, a subsumption of the form $C \sqsubseteq^? D_1 \sqcap \cdots \sqcap D_m$ is equivalent to $C \sqsubseteq^? D_1$, ..., $C \sqsubseteq^? D_m$. Any solution of a disunification problem Γ can be extended to a solution of the resulting flat disunification problem Γ', and conversely every solution of Γ' also solves Γ.

This flattening procedure also works for unification problems. However, dismatching problems cannot without loss of generality be restricted to being flat since the introduction of new variables to abbreviate subterms may destroy the property that one side of each dissubsumption is ground (see also Section 4).

For solving flat unification problems, it has been shown that it suffices to consider so-called local solutions [10], which are restricted to use only the atoms occurring in the input problem. We extend this notion to disunification as follows. Let Γ be a flat disunification problem. We denote by At the set of all (flat) atoms occurring as subterms in Γ, by Var the set of variables occurring in Γ, and by $\mathsf{At_{nv}} := \mathsf{At} \setminus \mathsf{Var}$ the set of *non-variable atoms* of Γ. Let $S \colon \mathsf{Var} \to 2^{\mathsf{At_{nv}}}$ be an *assignment (for Γ)*, i.e. a function that assigns to each variable $X \in \mathsf{Var}$ a set $S_X \subseteq \mathsf{At_{nv}}$ of non-variable atoms. The relation $>_S$ on Var is defined as the transitive closure of $\{(X, Y) \in \mathsf{Var}^2 \mid Y \text{ occurs in an atom of } S_X\}$. If this defines a strict partial order, i.e. $>_S$ is irreflexive, then S is called *acyclic*. In this case, we can define the substitution σ_S inductively along $>_S$ as follows: if X is minimal, then $\sigma_S(X) := \bigsqcap_{D \in S_X} D$; otherwise, assume that $\sigma_S(Y)$ is defined for all $Y \in \mathsf{Var}$ with $X > Y$, and define

$$\sigma_S(X) := \bigsqcap_{D \in S_X} \sigma_S(D).$$

It is easy to see that the concept terms $\sigma_S(D)$ are ground and constructed from the symbols of Σ, and hence σ_S is a valid candidate for a solution of Γ according to Definition 4.

▶ **Definition 5.** Let Γ be a flat disunification problem. A substitution σ is called *local* if there exists an acyclic assignment S for Γ such that $\sigma = \sigma_S$. The disunification problem Γ is *locally solvable* if it has a local solution, i.e. a solution that is a local substitution. *Local disunification* is the problem of checking flat disunification problems for local solvability.

Note that assignments and local solutions are defined only for *flat* disunification problems.

Obviously, local disunification is decidable in NP: We can guess an assignment S, and check it for acyclicity and whether the induced substitution solves the disunification problem in polynomial time. It has been shown [10] that unification in $\mathcal{EL}$ is *local* in the sense that the equivalent flattened problem has a local solution iff the original problem is solvable. Hence not only local, but also general solvability of unification problems in $\mathcal{EL}$ can be decided in NP. In addition, this shows that NP-hardness already holds for local unification, and thus also for local disunification.

▶ **Fact 6.** *Deciding local solvability of flat disunification problems in $\mathcal{EL}$ is* NP*-complete.*

The next example shows that disunification in $\mathcal{EL}$ is *not local* in this sense.

▶ **Example 7.** Consider the flat disunification problem

$$\Gamma := \{X \sqsubseteq^? B,\ A \sqcap B \sqcap C \sqsubseteq^? X,\ \exists r.X \sqsubseteq^? Y,\ \top \not\sqsubseteq^? Y,\ Y \not\sqsubseteq^? \exists r.B\}$$

[3] Recall that the empty conjunction is $\top$.

with variables X, Y and constants A, B, C. The substitution σ with $\sigma(X) := A \sqcap B \sqcap C$ and $\sigma(Y) := \exists r.(A \sqcap C)$ is a solution of Γ. For σ to be local, the atom $\exists r.(A \sqcap C)$ would have to be of the form $\sigma(D)$ for a non-variable atom D occurring in Γ. But the only candidates for D are $\exists r.X$ and $\exists r.B$, none of which satisfy $\exists r.(A \sqcap C) = \sigma(D)$.

We show that Γ cannot have another solution that is local. Assume to the contrary that Γ has a local solution γ. We know that $\gamma(Y)$ cannot be $\top$ since γ must solve the first dissubsumption. Furthermore, none of the constants A, B, C can be a top-level atom of $\gamma(Y)$ since this would contradict the third subsumption. That leaves only the non-variable atoms $\exists r.\gamma(X)$ and $\exists r.B$, which are ruled out by the last dissubsumption since both $\gamma(X)$ and B are subsumed by B.

The decidability and complexity of general solvability of disunification problems is still open. In the following, we first consider the special case of solving dismatching problems, for which we show a similar result as for unification: every dismatching problem can be polynomially reduced to a flat problem that has a local solution iff the original problem is solvable. The main difference is that this reduction is nondeterministic. In this way, we reduce dismatching to local disunification. We then provide two different NP-algorithms for the latter problem by extending the rule-based unification algorithm from [10] and adapting the SAT encoding of unification problems from [9]. These algorithms are more efficient than the brute-force "guess and then test" procedure on which our argument for Fact 6 was based.

4 Reducing dismatching to local disunification

As mentioned in Section 3, we cannot restrict our attention to flat dismatching problems without loss of generality. Instead, the nondeterministic algorithm we present in the following reduces any dismatching problem Γ to a flat *disunification* problem Γ' with the property that local solvability of Γ' is equivalent to the solvability of Γ. Since the algorithm takes at most polynomial time in the size of Γ, this shows that dismatching in $\mathcal{EL}$ is NP-complete. For simplicity, we assume that the subsumptions and the non-ground sides of the dissubsumptions have already been flattened using the approach mentioned in the previous section. This retains the property that all dissubsumptions have one ground side and does not affect the solvability of the problem.

Our procedure exhaustively applies a set of rules to the (dis)subsumptions in a dismatching problem (see Figures 1 and 2). In these rules, $C_1, \ldots, C_n$ and $D_1, \ldots, D_m$ are atoms. The rule Left Decomposition includes the special case where the left-hand side of $\mathfrak{s}$ is $\top$, in which case $\mathfrak{s}$ is simply removed from the problem. Note that at most one rule is applicable to any given (dis)subsumption. The choice which (dis)subsumption to consider next is don't care nondeterministic, but the choices in the rules Right Decomposition and Solving Left-Ground Dissubsumptions are don't know nondeterministic.

▶ **Algorithm 8.** Let Γ_0 be a dismatching problem. We initialize $\Gamma := \Gamma_0$. While any of the rules of Figures 1 and 2 is applicable to any element of Γ, choose one such element and apply the corresponding rule. If any rule application fails, then return "failure".

To see that every run of the nondeterministic algorithm terminates in polynomial time, note that each rule application takes only polynomial time in the size of the chosen (dis)subsumption. In particular, subsumptions between ground atoms can be checked in polynomial time [8]. Additionally, we can show that the algorithm needs at most polynomially many rule applications since each rule application decreases the following measure on Γ: we sum up all sizes of (dis)subsumptions in Γ to which a rule is still applicable, where the size

Right Decomposition:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \not\sqsubseteq^? D_1 \sqcap \cdots \sqcap D_m$ if $m = 0$ or $m > 1$, and $C_1, \ldots, C_n$, $D_1, \ldots, D_m$ are atoms.
Action: If $m = 0$, then *fail*. Otherwise, choose an index $i \in \{1, \ldots, m\}$ and replace $\mathfrak{s}$ by $C_1 \sqcap \cdots \sqcap C_n \not\sqsubseteq^? D_i$.

Left Decomposition:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \not\sqsubseteq^? D$ if $n = 0$ or $n > 1$, $C_1, \ldots, C_n$ are atoms, and D is a non-variable atom.
Action: Replace $\mathfrak{s}$ by $C_1 \not\sqsubseteq^? D, \ldots, C_n \not\sqsubseteq^? D$.

Atomic Decomposition:

Condition: This rule applies to $\mathfrak{s} = C \not\sqsubseteq^? D$ if C and D are non-variable atoms.
Action: Apply the first case that matches $\mathfrak{s}$:
a) if C and D are ground and $C \sqsubseteq D$, then *fail*;
b) if C and D are ground and $C \not\sqsubseteq D$, then remove $\mathfrak{s}$ from Γ;
c) if C or D is a constant, then remove $\mathfrak{s}$ from Γ;
d) if $C = \exists r.C'$ and $D = \exists s.D'$ with $r \neq s$, then remove $\mathfrak{s}$ from Γ;
e) if $C = \exists r.C'$ and $D = \exists r.D'$, then replace $\mathfrak{s}$ by $C' \not\sqsubseteq^? D'$.

Figure 1 Decomposition rules.

of $C \sqsubseteq^? D$ or $C \not\sqsubseteq^? D$ is defined as $|C| \cdot |D|$, and $|C|$ is the number of symbols needed to write down C (for details, see [4]).

Note that the Solving rule for left-ground dissubsumptions is not limited to non-flat dissubsumptions, and thus the algorithm completely eliminates all left-ground dissubsumptions from Γ. It is also easy to see that, if the algorithm is successful, then the resulting disunification problem Γ is flat. We now prove that this nondeterministic procedure is correct in the following sense.

▶ **Lemma 9.** *The dismatching problem Γ_0 is solvable iff there is a successful run of Algorithm 8 such that the resulting flat disunification problem Γ has a local solution.*

Proof Sketch. Soundness (i.e., the if direction) is easy to show, using Lemmas 1–3. Showing completeness (i.e., the only-if direction) is more involved. Basically, given a solution γ of Γ_0, we can use γ to guide the rule applications and extend γ to the newly introduced variables such that each rule application is successful and the invariant "γ *solves all (dis)subsumptions of* Γ" is maintained. Once no more rules can be applied, we have a flat disunification problem Γ of which the extended substitution γ is a (possibly non-local) solution. To obtain a local solution, we denote by At, Var, and $\mathsf{At}_{\mathsf{nv}}$ the sets as defined in Section 3 and define the assignment S induced by γ as:

$$S_X := \{D \in \mathsf{At}_{\mathsf{nv}} \mid \gamma(X) \sqsubseteq \gamma(D)\},$$

for all (old and new) variables $X \in \mathsf{Var}$. It can be shown that this assignment is acyclic and that the induced local substitution σ_S solves Γ, and thus also Γ_0 (see [4] for details). ◀

The disunification problem of Example 7 is in fact a dismatching problem. The rule Solving Left-Ground Dissubsumptions can be used to replace $\top \not\sqsubseteq^? Y$ with $Y \sqsubseteq^? \exists r.Z$. The presence of the new atom $\exists r.Z$ makes the solution σ introduced in Example 7 local.

Flattening Right-Ground Dissubsumptions:

Condition: This rule applies to $\mathfrak{s} = X \not\sqsubseteq^? \exists r.D$ if X is a variable and D is ground and is not a concept name.
Action: Introduce a new variable X_D and replace $\mathfrak{s}$ by $X \not\sqsubseteq^? \exists r.X_D$ and $D \sqsubseteq^? X_D$.

Flattening Left-Ground Subsumptions:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqcap \exists r_1.D_1 \sqcap \cdots \sqcap \exists r_m.D_m \sqsubseteq^? X$ if $m > 0$, X is a variable, $C_1, \ldots, C_n$ are flat ground atoms, and $\exists r_1.D_1, \ldots, \exists r_m.D_m$ are non-flat ground atoms.
Action: Introduce new variables $X_{D_1}, \ldots, X_{D_m}$ and replace $\mathfrak{s}$ by $D_1 \sqsubseteq^? X_{D_1}, \ldots, D_m \sqsubseteq^? X_{D_m}$ and $C_1 \sqcap \cdots \sqcap C_n \sqcap \exists r_1.X_{D_1} \sqcap \cdots \sqcap \exists r_m.X_{D_m} \sqsubseteq^? X$.

Solving Left-Ground Dissubsumptions:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \not\sqsubseteq^? X$ if X is a variable and $C_1, \ldots, C_n$ are ground atoms.
Action: Choose one of the following options:

- Choose a constant $A \in \Sigma$ and replace $\mathfrak{s}$ by $X \sqsubseteq^? A$. If $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq A$, then *fail*.
- Choose a role $r \in \Sigma$, introduce a new variable Z, replace $\mathfrak{s}$ by $X \sqsubseteq^? \exists r.Z$, $C_1 \not\sqsubseteq^? \exists r.Z$, ..., $C_n \not\sqsubseteq^? \exists r.Z$, and immediately apply Atomic Decomposition to each of these dissubsumptions.

Figure 2 Flattening and solving rules.

Together with Fact 6 and the NP-hardness of unification in $\mathcal{EL}$ [10], Lemma 9 yields the following complexity result.

▶ **Theorem 10.** *Deciding solvability of dismatching problems in $\mathcal{EL}$ is* NP*-complete.*

5 A goal-oriented algorithm for local disunification

In this section, we present an algorithm for local disunification that is based on transformation rules. Basically, to solve the subsumptions, this algorithm uses the rules of the goal-oriented algorithm for unification in $\mathcal{EL}$ [10, 3], which produces only local unifiers. Since any local solution of the disunification problem is a local unifier of the subsumptions in the problem, one might think that it is then sufficient to check whether any of the produced unifiers also solves the dissubsumptions. This would not be complete, however, since the goal-oriented algorithm for unification does *not* produce *all* local unifiers. For this reason, we have additional rules for solving the dissubsumptions. Both rule sets contain (deterministic) *eager* rules that are applied with the highest priority, and *nondeterministic* rules that are only applied if no eager rule is applicable. The goal of the eager rules is to enable the algorithm to detect obvious contradictions as early as possible in order to reduce the number of nondeterministic choices it has to make.

Let now Γ_0 be the flat disunification problem for which we want to decide local solvability, and let the sets At, Var, and $\mathsf{At}_{\mathsf{nv}}$ be defined as in Section 3. We assume without loss of generality that the dissubsumptions in Γ_0 have only a single atom on the right-hand side. If this is not the case, it can easily be achieved by exhaustive application of the nondeterministic rule Right Decomposition (see Figure 1) without affecting the complexity of the overall procedure.

Starting with Γ_0, the algorithm maintains a current disunification problem Γ and a current acyclic assignment S, which initially assigns the empty set to all variables. In addition, for each subsumption or dissubsumption in Γ, it maintains the information on whether it is *solved* or not. Initially, all subsumptions of Γ_0 are unsolved, except those with a variable on the

right-hand side, and all dissubsumptions in Γ_0 are unsolved, except those with a variable on the left-hand side and a non-variable atom on the right-hand side. Subsumptions of the form $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? X$ and dissubsumptions of the form $X \not\sqsubseteq^? D$, for a non-variable atom D, are called *initially solved*. Intuitively, they only specify constraints on the assignment S_X. More formally, this intuition is captured by the process of *expanding* Γ w.r.t. the variable X, which performs the following actions:

- every initially solved subsumption $\mathfrak{s} \in \Gamma$ of the form $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? X$ is expanded by adding the subsumption $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? E$ to Γ for every $E \in S_X$, and
- every initially solved dissubsumption $X \not\sqsubseteq^? D \in \Gamma$ is expanded by adding $E \not\sqsubseteq^? D$ to Γ for every $E \in S_X$.

A (non-failing) application of a rule of our algorithm does the following:

- it solves exactly one unsolved subsumption or dissubsumption,
- it may extend the current assignment S by adding elements of $\mathsf{At}_{\mathsf{nv}}$ to some set S_X,
- it may introduce new flat subsumptions or dissubsumptions built from elements of At,
- it keeps Γ expanded w.r.t. all variables X.

Subsumptions and dissubsumptions are only added by a rule application or by expansion if they are not already present in Γ. If a new subsumption or dissubsumption is added to Γ, it is marked as unsolved, unless it is initially solved (because of its form). Solving subsumptions and dissubsumptions is mostly independent, except for expanding Γ, which can add new unsolved subsumptions and dissubsumptions at the same time, and may be triggered by solving a subsumption or a dissubsumption.

The rules dealing with subsumptions are depicted in Figure 3; these three eager and two nondeterministic rules are essentially the same as the ones in [3], with the only difference that the background ontology $\mathcal{T}$ used there is empty for our purposes. Note that several rules may be applicable to the same subsumption, and there is no preference between them. Using Eager Ground Solving, the algorithm can immediately evaluate ground subsumptions via the polynomial-time algorithm of [8]. If the required subsumption holds, it is marked as solved, and otherwise Γ cannot be solvable and hence the algorithm fails. Eager Solving detects when a subsumption trivially holds because the atom D from the right-hand side is already present on the left-hand side, either directly or via the assignment of a variable. Eager Extension is applicable in case the left-hand side of a subsumption is essentially equivalent to a single variable X due to all its atoms being "subsumed by" S_X. In this case, there is no other option but to add the right-hand side atom to S_X to solve the subsumption, and to expand Γ w.r.t. this new assignment. In case none of the eager rules apply to a subsumption, it can be solved nondeterministically by either extending the assignment of a variable that occurs on the left-hand side (Extension), or decomposing the subsumption by looking for matching existential restrictions on both sides (cf. Lemma 1).

The new rules for solving dissubsumptions are listed in Figure 4. These include variants of the Left Decomposition and Atomic Decomposition rules from the previous section (see Figure 1). In these two rules, which are eager, instead of removing dissubsumptions we mark them as solved. Additionally, Γ may have to be expanded if such a rule adds a new dissubsumption that is initially solved. The new nondeterministic rule Local Extension follows the same idea as the Solving rule for left-ground dissubsumptions (see Figure 2), but does not have to introduce new variables and atoms since we are looking only for local solutions. Note that the left-hand side of $\mathfrak{s}$ may be a variable, and then $\mathfrak{s}$ is of the form $Y \not\sqsubseteq^? X$. This dissubsumption is not initially solved, because X is not a non-variable atom.

▶ **Algorithm 11.** Let Γ_0 be a flat disunification problem. We initialize $\Gamma := \Gamma_0$ and $S_X := \emptyset$ for all variables $X \in \mathsf{Var}$. While Γ contains an unsolved subsumption or dissubsumption, do

Eager Ground Solving:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D \in \Gamma$, if $\mathfrak{s}$ is ground.
Action: The rule application fails if $\mathfrak{s}$ does not hold. Otherwise, $\mathfrak{s}$ is marked as *solved*.

Eager Solving:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D \in \Gamma$, if there is an index $i \in \{1, \ldots, n\}$, such that $C_i = D$ or $C_i = X \in \mathsf{Var}$ and $D \in S_X$.
Action: The application of the rule marks $\mathfrak{s}$ as *solved*.

Eager Extension:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D \in \Gamma$, if there is an index $i \in \{1, \ldots, n\}$, such that $C_i = X \in \mathsf{Var}$ and $\{C_1, \ldots, C_n\} \setminus \{X\} \subseteq S_X$.
Action: The application of the rule adds D to S_X. If this makes S cyclic, the rule application fails. Otherwise, Γ is expanded w.r.t. X and $\mathfrak{s}$ is marked as *solved*.

Decomposition:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? \exists s.D \in \Gamma$, if there is at least one index $i \in \{1, \ldots, n\}$ with $C_i = \exists s.C$.
Action: The application of the rule chooses such an index i, adds $C \sqsubseteq^? D$ to Γ, expands Γ w.r.t. D if D is a variable, and marks $\mathfrak{s}$ as *solved*.

Extension:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D \in \Gamma$, if there is at least one index $i \in \{1, \ldots, n\}$ with $C_i \in \mathsf{Var}$.
Action: The application of the rule chooses such an index i and adds D to S_{C_i}. If this makes S cyclic, the rule application fails. Otherwise, Γ is expanded w.r.t. C_i and $\mathfrak{s}$ is marked as *solved*.

Figure 3 Rules for subsumptions.

the following:

1. **Eager rule application:** If eager rules are applicable to some unsolved subsumption or dissubsumption $\mathfrak{s}$ in Γ, apply an arbitrarily chosen one to $\mathfrak{s}$. If the rule application fails, return "failure".
2. **Nondeterministic rule application:** If no eager rule is applicable, let $\mathfrak{s}$ be an unsolved subsumption or dissubsumption in Γ. If one of the nondeterministic rules applies to $\mathfrak{s}$, choose one and apply it. If none of these rules apply to $\mathfrak{s}$ or the rule application fails, then return "failure".

Once all (dis)subsumptions in Γ are solved, return the substitution σ_S that is induced by the current assignment.

As with Algorithm 8, the choice which (dis)subsumption to consider next and which eager rule to apply is don't care nondeterministic, while the choice of which nondeterministic rule to apply and the choices inside the rules are don't know nondeterministic. Each of these latter choices may result in a different solution σ_S. All proof details for the following results can be found in [4].

▶ **Lemma 12.** *Every run of Algorithm 11 terminates in time polynomial in the size of* Γ_0.

Proof Sketch. We can show that each (dis)subsumption that is added by a rule or by expansion is either of the form $C \sqsubseteq^? D$ or $C \not\sqsubseteq^? D$, where $C, D \in \mathsf{At}$, or of the form $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? E$, where $C_1 \sqcap \cdots \sqcap C_n$ is the left-hand side of a subsumption from the original problem Γ_0 and $E \in \mathsf{At}$. Obviously, there are only polynomially many such

Eager Top Solving:

Condition: This rule applies to $\mathfrak{s} = C \not\sqsubseteq^? \top \in \Gamma$.
Action: The rule application fails.

Eager Left Decomposition:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \not\sqsubseteq^? D \in \Gamma$ if $n = 0$ or $n > 1$, and $D \in \mathsf{At}_{\mathsf{nv}}$.
Action: The application of the rule marks $\mathfrak{s}$ as *solved* and, for each $i \in \{1, \ldots, n\}$, adds $C_i \not\sqsubseteq^? D$ to Γ and expands Γ w.r.t. C_i if C_i is a variable.

Eager Atomic Decomposition:

Condition: This rule applies to $\mathfrak{s} = C \not\sqsubseteq^? D \in \Gamma$ if $C, D \in \mathsf{At}_{\mathsf{nv}}$.
Action: The application of the rule applies the first case that matches $\mathfrak{s}$:

a) if C and D are ground and $C \sqsubseteq D$, then the rule application fails;

b) if C and D are ground and $C \not\sqsubseteq D$, then $\mathfrak{s}$ is marked as *solved*;

c) if C or D is a concept name, then $\mathfrak{s}$ is marked as *solved*;

d) if $C = \exists r.C'$ and $D = \exists s.D'$ with $r \neq s$, then $\mathfrak{s}$ is marked as *solved*;

e) if $C = \exists r.C'$ and $D = \exists r.D'$, then $C' \not\sqsubseteq^? D'$ is added to Γ, Γ is expanded w.r.t. C' if C' is a variable and D' is not a variable, and $\mathfrak{s}$ is marked as *solved*.

Local Extension:

Condition: This rule applies to $\mathfrak{s} = C \not\sqsubseteq^? X \in \Gamma$ if $X \in \mathsf{Var}$.
Action: The application of the rule chooses $D \in \mathsf{At}_{\mathsf{nv}}$ and adds D to S_X. If this makes S cyclic, the rule application fails. Otherwise, the new dissubsumption $C \not\sqsubseteq^? D$ is added to Γ, Γ is expanded w.r.t. X, Γ is expanded w.r.t. C if C is a variable, and $\mathfrak{s}$ is marked as *solved*.

Figure 4 New rules for dissubsumptions.

(dis)subsumptions. Additionally, each rule application solves at least one (dis)subsumption and takes at most polynomial time. ◀

To show *soundness* of the procedure, assume that a run of the algorithm terminates with success, i.e. all subsumptions and dissubsumptions are solved. Let $\hat{\Gamma}$ be the set of all subsumptions and dissubsumptions produced by this run, S be the final assignment, and σ_S the induced substitution (see Section 3). To show that σ_S solves $\hat{\Gamma}$, and hence also Γ_0, we use induction on the following order on (dis)subsumptions.

▶ **Definition 13.** Consider any (dis)subsumption $\mathfrak{s}$ of the form $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? C_{n+1}$ or $C_1 \sqcap \cdots \sqcap C_n \not\sqsubseteq^? C_{n+1}$ in $\hat{\Gamma}$.

- We define $m(\mathfrak{s}) := (m_1(\mathfrak{s}), m_2(\mathfrak{s}))$, where
  - $m_1(\mathfrak{s}) := \emptyset$ if $\mathfrak{s}$ is ground; otherwise, $m_1(\mathfrak{s}) := \{X_1, \ldots, X_m\}$, where $\{X_1, \ldots, X_m\}$ is the multiset of all variables occurring in $C_1, \ldots, C_n, C_{n+1}$.
  - $m_2(\mathfrak{s}) := |\mathfrak{s}|$, where $|\mathfrak{s}|$ is the size of $\mathfrak{s}$, i.e. the number of symbols in $\mathfrak{s}$.
- The strict partial order $\succ$ on such pairs is the lexicographic order, where the second components are compared w.r.t. the usual order on natural numbers, and the first components are compared w.r.t. the multiset extension of $>_S$ [13].
- We extend $\succ$ to $\hat{\Gamma}$ by setting $\mathfrak{s}_1 \succ \mathfrak{s}_2$ iff $m(\mathfrak{s}_1) \succ m(\mathfrak{s}_2)$.

Since multiset extensions and lexicographic products of well-founded strict partial orders are again well-founded [13], $\succ$ is a well-founded strict partial order on $\hat{\Gamma}$. We can then use the fact that the (dis)subsumptions produced by Algorithm 11 are always smaller w.r.t. this

order than the (dis)subsumptions they were created from to prove the following lemma by well-founded induction over $\succ$.

▶ **Lemma 14.** *σ_S is a local solution of $\hat{\Gamma}$, and thus also of its subset Γ_0.*

To prove *completeness*, assume that σ is a local solution of Γ_0. We can show that σ can guide the choices of Algorithm 11 to obtain a local solution σ' of Γ_0 such that, for every variable X, we have $\sigma(X) \sqsubseteq \sigma'(X)$. The following invariants will be maintained throughout the run of the algorithm for the current set of (dis)subsumptions Γ and the current assignment S:

I. σ is a solution of Γ. II. For each $D \in S_X$, we have that $\sigma(X) \sqsubseteq \sigma(D)$.

By Lemma 1, chains of the form $\sigma(X_1) \sqsubseteq \sigma(\exists r_1.X_2)$, $\dots \sigma(X_{n-1}) \sqsubseteq \sigma(\exists r_{n-1}.X_n)$ with $X_1 = X_n$ are impossible, and thus invariant II implies that S is acyclic. Hence, if extending S during a rule application preserves this invariant, this extension will not cause the algorithm to fail. In [4] it is shown that

- the invariants are maintained by the operation of expanding Γ;
- the application of an eager rule never fails and maintains the invariants; and
- if $\mathfrak{s}$ is an unsolved (dis)subsumption of Γ to which no eager rule applies, then there is a nondeterministic rule that can be successfully applied to $\mathfrak{s}$ while maintaining the invariants.

This concludes the proof of correctness of Algorithm 11, which provides a more goal-directed way to solve local disunification problems than blindly guessing an assignment as described in Section 4.

▶ **Theorem 15.** *The flat disunification problem Γ_0 has a local solution iff there is a successful run of Algorithm 11 on Γ_0.*

6 Encoding local disunification into SAT

The following reduction to SAT is a generalization of the one for unification problems in [9]. We again consider a flat disunification problem Γ and the sets At, Var, and $\mathsf{At}_{\mathsf{nv}}$ as in Section 3. Since we are restricting our considerations to *local* solutions, we can without loss of generality assume that the sets N_{v}, N_{c}, and N_{R} contain exactly the variables, constants, and role names occurring in Γ. To further simplify the reduction, we assume in the following that all flat dissubsumptions in Γ are of the form $X \not\sqsubseteq^? Y$ for variables X, Y. This is without loss of generality, which can be shown using a transformation similar to the flattening rules from Section 4.

The translation into SAT uses the propositional variables $[C \sqsubseteq D]$ for all $C, D \in \mathsf{At}$. The SAT problem consists of a set of clauses $\mathsf{Cl}(\Gamma)$ over these variables that express properties of (dis)subsumption in $\mathcal{EL}$ and encode the elements of Γ. The intuition is that a satisfying valuation of $\mathsf{Cl}(\Gamma)$ induces a local solution σ of Γ such that $\sigma(C) \sqsubseteq \sigma(D)$ holds whenever $[C \sqsubseteq D]$ is true under the valuation. The solution σ is constructed by first extracting an acyclic assignment S out of the satisfying valuation and then computing $\sigma := \sigma_S$. We additionally introduce the variables $[X > Y]$ for all $X, Y \in \mathsf{N}_{\mathsf{v}}$ to ensure that the generated assignment S is indeed acyclic. This is achieved by adding clauses to $\mathsf{Cl}(\Gamma)$ that express that $>_S$ is a strict partial order, i.e. irreflexive and transitive.

Finally, we use the auxiliary variables $p_{C,X,D}$ for all $X \in \mathsf{N}_{\mathsf{v}}$, $C \in \mathsf{At}$, and $D \in \mathsf{At}_{\mathsf{nv}}$ to express the restrictions imposed by dissubsumptions of the form $C \not\sqsubseteq^? X$ in clausal form. More precisely, whenever $[C \sqsubseteq X]$ is false for some $X \in \mathsf{N}_{\mathsf{v}}$ and $C \in \mathsf{At}$, then the

dissubsumption $\sigma(C) \not\sqsubseteq \sigma(X)$ should hold. By Lemma 2, this means that we need to find an atom $D \in \mathsf{At_{nv}}$ that is a top-level atom of $\sigma(X)$ and satisfies $\sigma(C) \not\sqsubseteq \sigma(D)$. This is enforced by making the auxiliary variable $p_{C,X,D}$ true, which makes $[X \sqsubseteq D]$ true and $[C \sqsubseteq D]$ false (see Definition 167).

▶ **Definition 16.** The set $\mathsf{Cl}(\Gamma)$ contains the following propositional clauses:

(I) *Translation of* Γ.

a. For every subsumption $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D$ in Γ with $D \in \mathsf{At_{nv}}$:
$\rightarrow [C_1 \sqsubseteq D] \vee \cdots \vee [C_n \sqsubseteq D]$

b. For every subsumption $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? X$ in Γ with $X \in \mathsf{N_v}$, and every $E \in \mathsf{At_{nv}}$:
$[X \sqsubseteq E] \rightarrow [C_1 \sqsubseteq E] \vee \cdots \vee [C_n \sqsubseteq E]$

c. For every dissubsumption $X \not\sqsubseteq^? Y$ in Γ: $[X \sqsubseteq Y] \rightarrow$

(IV) *Properties of subsumptions between non-variable atoms.*

a. For every $A \in \mathsf{N_c}$: $\rightarrow [A \sqsubseteq A]$

b. For every $A, B \in \mathsf{N_c}$ with $A \neq B$: $[A \sqsubseteq B] \rightarrow$

c. For every $\exists r.A, \exists s.B \in \mathsf{At_{nv}}$ with $r \neq s$: $[\exists r.A \sqsubseteq \exists s.B] \rightarrow$

d. For every $A \in \mathsf{N_c}$ and $\exists r.B \in \mathsf{At_{nv}}$:
$[A \sqsubseteq \exists r.B] \rightarrow$ and $[\exists r.B \sqsubseteq A] \rightarrow$

e. For every $\exists r.A, \exists r.B \in \mathsf{At_{nv}}$:
$[\exists r.A \sqsubseteq \exists r.B] \rightarrow [A \sqsubseteq B]$ and $[A \sqsubseteq B] \rightarrow [\exists r.A \sqsubseteq \exists r.B]$

(VI) *Transitivity of subsumption.*
For every $C_1, C_2, C_3 \in \mathsf{At}$: $[C_1 \sqsubseteq C_2] \wedge [C_2 \sqsubseteq C_3] \rightarrow [C_1 \sqsubseteq C_3]$

(VII) *Dissubsumptions of the form* $C \not\sqsubseteq^? X$ *with a variable* X.
For every $C \in \mathsf{At}$, $X \in \mathsf{N_v}$:
$\rightarrow [C \sqsubseteq X] \vee \bigvee_{D \in \mathsf{At_{nv}}} p_{C,X,D}$,
and additionally for every $D \in \mathsf{At_{nv}}$:
$p_{C,X,D} \rightarrow [X \sqsubseteq D]$ and $p_{C,X,D} \wedge [C \sqsubseteq D] \rightarrow$

(VIII) *Properties of* $>$.

a. For every $X \in \mathsf{N_v}$: $[X > X] \rightarrow$

b. For every $X, Y, Z \in \mathsf{N_v}$: $[X > Y] \wedge [Y > Z] \rightarrow [X > Z]$

c. For every $X, Y \in \mathsf{N_v}$ and $\exists r.Y \in \mathsf{At}$: $[X \sqsubseteq \exists r.Y] \rightarrow [X > Y]$

The main difference to the encoding in [9] (apart from the fact that we consider (dis)subsumptions here instead of equivalences) lies in the clauses 7 that ensure the presence of a non-variable atom D that solves the dissubsumption $C \not\sqsubseteq^? X$ (cf. Lemma 2). We also need some additional clauses in 4 to deal with dissubsumptions. It is easy to see that $\mathsf{Cl}(\Gamma)$ can be constructed in time cubic in the size of Γ (due to the clauses in 6 and 2).

To show *soundness* of the reduction, let τ be a valuation of the propositional variables that satisfies $\mathsf{Cl}(\Gamma)$. We define the assignment S^τ as follows:

$$S^\tau_X := \{D \in \mathsf{At_{nv}} \mid \tau([X \sqsubseteq D]) = 1\}.$$

In [4] it is shown that $X >_{S^\tau} Y$ implies $\tau([X > Y]) = 1$ and that this implies irreflexivity of $>_{S^\tau}$. This in particular shows that S^τ is acyclic. In the following, let σ_τ denote the substitution σ_{S^τ} induced by S^τ. In [4] it is shown that σ_τ is a solution of Γ by proving that for all atoms $C, D \in \mathsf{At}$ it holds that $\tau([C \sqsubseteq D]) = 1$ iff $\sigma_\tau(C) \sqsubseteq \sigma_\tau(D)$.

Since σ_τ is obviously local, this suffices to show soundness of the reduction.

▶ **Lemma 17.** *If* $\mathsf{Cl}(\Gamma)$ *is solvable, then* Γ *has a local solution.*

To show *completeness*, let σ be a local solution of Γ and $>_\sigma$ the resulting partial order on $\mathsf{N_v}$, defined as follows for all $X, Y \in \mathsf{N_v}$:

$$X >_\sigma Y \text{ iff } \sigma(X) \sqsubseteq \exists r_1. \ldots \exists r_n. \sigma(Y) \text{ for some } r_1, \ldots, r_n \in \mathsf{N_R} \text{ with } n \geq 1.$$

Note that $>_\sigma$ is irreflexive since $X >_\sigma X$ is impossible by Lemma 1, and it is transitive since $\sqsubseteq$ is transitive and closed under applying existential restrictions on both sides. Thus, $>_\sigma$ is a strict partial order. We define a valuation τ_σ as follows for all $C, D \in \mathsf{At}$, $E \in \mathsf{At_{nv}}$, and $X, Y \in \mathsf{N_v}$:

$$\tau_\sigma([C \sqsubseteq D]) := \begin{cases} 1 & \text{if } \sigma(C) \sqsubseteq \sigma(D) \\ 0 & \text{otherwise} \end{cases} \qquad \tau_\sigma([X > Y]) := \begin{cases} 1 & \text{if } X >_\sigma Y \\ 0 & \text{otherwise} \end{cases}$$

$$\tau_\sigma(p_{C,X,E}) := \begin{cases} 1 & \text{if } \sigma(X) \sqsubseteq \sigma(E) \text{ and } \sigma(C) \not\sqsubseteq \sigma(E) \\ 0 & \text{otherwise} \end{cases}$$

In [4] it is proved that τ_σ satisfies $\mathsf{CI}(\Gamma)$, which shows completeness of the reduction.

▶ **Lemma 18.** *If Γ has a local solution, then $\mathsf{CI}(\Gamma)$ is solvable.*

This completes the proof of the correctness of the translation presented in Definition 16, which provides us with a reduction of local disunification (and thus also of dismatching) to SAT. This SAT reduction has been implemented in our prototype system UEL,[4] which uses SAT4J[5] as external SAT solver. First experiments show that dismatching is indeed helpful for reducing the number and the size of unifiers. The runtime performance of the solver for dismatching problems is comparable to the one for pure unification problems.

7 Related and future work

Since Description Logics and Modal Logics are closely related [26], results on unification in one of these two areas carry over to the other one. In Modal Logics, unification has mostly been considered for expressive logics with all Boolean operators [19, 20, 25]. An important open problem in the area is the question whether unification in the basic modal logic K, which corresponds to the DL $\mathcal{ALC}$, is decidable. It is only known that relatively minor extensions of K have an undecidable unification problem [27]. Disunification also plays an important role in Modal Logics since it is basically the same as the admissibility problem for inference rules [15, 22, 24] (see [4] for details).

Regarding future work, we want to investigate the decidability and complexity of general disunification in $\mathcal{EL}$, and consider also the case where non-ground solutions are allowed. From a more practical point of view, we plan to implement also the goal-oriented algorithm for local disunification, and to evaluate the performance of both presented algorithms on real-world problems.

References

1 Franz Baader, Stefan Borgwardt, Julian Alfredo Mendez, and Barbara Morawska. UEL: Unification solver for $\mathcal{EL}$. In *Proc. DL'12*, volume 846 of *CEUR-WS*, pages 26–36, 2012.

[4] version 1.3.0, available at `http://uel.sourceforge.net/`
[5] `http://www.sat4j.org/`

2 Franz Baader, Stefan Borgwardt, and Barbara Morawska. Computing minimal $\mathcal{EL}$-unifiers is hard. In *Proc. AiML'12*, 2012.

3 Franz Baader, Stefan Borgwardt, and Barbara Morawska. A goal-oriented algorithm for unification in $\mathcal{EL}$ w.r.t. cycle-restricted TBoxes. In *Proc. DL'12*, volume 846 of *CEUR-WS*, pages 37–47, 2012.

4 Franz Baader, Stefan Borgwardt, and Barbara Morawska. Dismatching and local disunification in $\mathcal{EL}$. LTCS-Report 15-03, Chair for Automata Theory, TU Dresden, Germany, 2015. See `http://lat.inf.tu-dresden.de/research/reports.html`.

5 Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope further. In *Proc. OWLED'08*, 2008.

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

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

8 Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proc. IJCAI'99*, pages 96–101. Morgan Kaufmann, 1999.

9 Franz Baader and Barbara Morawska. SAT encoding of unification in $\mathcal{EL}$. In *Proc. LPAR'10*, volume 6397 of *LNCS*, pages 97–111. Springer, 2010.

10 Franz Baader and Barbara Morawska. Unification in the description logic $\mathcal{EL}$. *Log. Meth. Comput. Sci.*, 6(3), 2010.

11 Franz Baader and Barbara Morawska. Matching with respect to general concept inclusions in the description logic $\mathcal{EL}$. In *Proc. KI'14*, volume 8736 of *LNCS*, pages 135–146. Springer, 2014.

12 Franz Baader and Paliath Narendran. Unification of concept terms in description logics. *J. Symb. Comput.*, 31(3):277–305, 2001.

13 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1999.

14 Franz Baader and Alexander Okhotin. Solving language equations and disequations with applications to disunification in description logics and monadic set constraints. In *Proc. LPAR'12*, volume 7180 of *LNCS*, pages 107–121. Springer, 2012.

15 Sergey Babenyshev, Vladimir V. Rybakov, Renate Schmidt, and Dmitry Tishkovsky. A tableau method for checking rule admissibility in $S4$. In *Proc. M4M-6*, 2009.

16 Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In *Proc. ECAI'04*, pages 298–302, 2004.

17 Wray L. Buntine and Hans-Jürgen Bürckert. On solving equations and disequations. *J. of the ACM*, 41(4):591–629, 1994.

18 Hubert Comon. Disunification: A survey. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 322–359. MIT Press, 1991.

19 Silvio Ghilardi. Unification through projectivity. *J. Logic and Computation*, 7(6):733–752, 1997.

20 Silvio Ghilardi. Unification in intuitionistic logic. *J. Logic and Computation*, 64(2):859–880, 1999.

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

22 Rosalie Iemhoff and George Metcalfe. Proof theory for admissible rules. *Ann. Pure Appl. Logic*, 159(1-2):171–186, 2009.

23 Ralf Küsters. Chapter 6: Matching. In *Non-Standard Inferences in Description Logics*, volume 2100 of *LNCS*, pages 153–227. Springer, 2001.

24 Vladimir V. Rybakov. *Admissibility of logical inference rules*, volume 136 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, 1997.

25 Vladimir V. Rybakov. Multi-modal and temporal logics with universal formula - reduction of admissibility to validity and unification. *J. Logic and Computation*, 18(4):509–519, 2008.

26 Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. IJCAI'91*, pages 466–471, 1991.

27 Frank Wolter and Michael Zakharyaschev. Undecidability of the unification and admissibility problems for modal and description logics. *ACM Trans. Comput. Log.*, 9(4), 2008.