A Goal-Oriented Algorithm for Unification in $\mathcal{EL}$ w.r.t. Cycle-Restricted TBoxes*

Franz Baader, Stefan Borgwardt, and Barbara Morawska
{baader,stefborg,morawska}@tcs.inf.tu-dresden.de

Theoretical Computer Science, TU Dresden, Germany

1 Introduction

Unification in DLs has been proposed in [7] (for the DL $\mathcal{FL}_0$, which offers the constructors conjunction ($\sqcap$), value restriction ($\forall r.C$), and the top concept ($\top$)) as a novel inference service that can, for instance, be used to detect redundancies in ontologies. 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

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

These two concept descriptions are not equivalent, but they are nevertheless meant to represent the same concept. They can obviously be made equivalent by treating the concept names $\mathsf{Head\_injury}$ and $\mathsf{Severe\_finding}$ 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 descriptions are unifiable, and call the substitution that makes them equivalent a *unifier*. Intuitively, such a unifier proposes definitions for the concept names that are used as variables: in our example, we know that, if we define $\mathsf{Head\_injury}$ as $\mathsf{Injury} \sqcap \exists \mathsf{finding\_site}.\mathsf{Head}$ and $\mathsf{Severe\_finding}$ as $\exists \mathsf{severity}.\mathsf{Severe}$, then the two concept descriptions (1) and (2) are equivalent w.r.t. these definitions. Here equivalence holds without additional GCIs.

To motivate our interest in unification w.r.t. GCIs, assume that the second developer uses the description

$$\mathsf{Patient} \sqcap \exists \mathsf{status}.\mathsf{Emergency} \sqcap \exists \mathsf{finding}.(\mathsf{Severe\_finding} \sqcap \mathsf{Injury} \sqcap \exists \mathsf{finding\_site}.\mathsf{Head}) \tag{3}$$

instead of (2). The descriptions (1) and (3) are not unifiable without additional GCIs, but they are unifiable, with the same unifier as above, if the GCI

$$\exists \mathsf{finding}.\exists \mathsf{severity}.\mathsf{Severe} \sqsubseteq \exists \mathsf{status}.\mathsf{Emergency}$$

* Supported by DFG under grant BA 1122/14-1

is present in a background ontology.

In [4], we were able to show that unification in the DL $\mathcal{EL}$ (which differs from $\mathcal{FL}_0$ by offering existential restrictions ($\exists r.C$) in place of value restrictions) is of considerably lower complexity than in $\mathcal{FL}_0$: the decision problem in $\mathcal{EL}$ is NP-complete rather than ExpTime-complete in $\mathcal{FL}_0$. In addition to a brute-force "guess and then test" NP-algorithm [4], we have developed a goal-oriented unification algorithm for $\mathcal{EL}$, in which nondeterministic decisions are only made if they are triggered by "unsolved parts" of the unification problem [6], and an algorithm that is based on a reduction to satisfiability in propositional logic (SAT) [5], which enables the use of highly-optimized SAT solvers. In [6] it was also shown that the approaches for unification of $\mathcal{EL}$-concept descriptions (without any background ontology) can easily be extended to the case of an acyclic TBox as background ontology without really changing the algorithms or increasing their complexity. Basically, by viewing defined concepts as variables, an acyclic TBox can be turned into a unification problem that has as its unique unifier the substitution that replaces the defined concepts by unfolded versions of their definitions. For GCIs, this simple trick is not possible.

In [2], we extended the brute-force "guess and then test" NP-algorithm from [4] to the case of GCIs, which required the development of a new characterization of subsumption w.r.t. GCIs in $\mathcal{EL}$. Unfortunately, the algorithm is complete only for general TBoxes (i.e., finite sets of GCIs) that satisfy a certain restriction on cycles, which, however, does not prevent all cycles. For example, the cyclic GCI $\exists\mathsf{child}.\mathsf{Human} \sqsubseteq \mathsf{Human}$ satisfies this restriction, whereas the cyclic GCI $\mathsf{Human} \sqsubseteq \exists\mathsf{parent}.\mathsf{Human}$ does not.

In the present paper, we describe a goal-oriented algorithm for unification in $\mathcal{EL}$ w.r.t. cycle-restricted general TBoxes, which extends the one from [6] and reduces the amount of nondeterministic guesses considerably. Full proofs of the presented results can be found in [1].

2 The Description Logic $\mathcal{EL}$

Syntax and semantics of $\mathcal{EL}$ are defined in the usual way (see, e.g., [9]). Here, we just recall that *$\mathcal{EL}$-concept descriptions* are built from a finite set N_C of *concept names* and a finite set N_R of *role names* using the concept constructors *top-concept* ($\top$), *conjunction* ($C \sqcap D$), and *existential restriction* ($\exists r.C$ for every $r \in N_R$). *Nested existential restrictions* $\exists r_1.\exists r_2.\cdots\exists r_n.C$ will sometimes also be written as $\exists r_1 r_2 \ldots r_n.C$, where $r_1 r_2 \ldots r_n$ is viewed as a word over the alphabet of role names, i.e., an element of N_R^*. As usual, concepts C are interpreted as sets $C^{\mathcal{I}}$ over some domain such that the semantics of the constructors is respected.

A *general concept inclusion (GCI)* is of the form $C \sqsubseteq D$ for concept descriptions C, D, and a general TBox is a finite set of GCIs. An interpretation $\mathcal{I}$ *satisfies* such a GCI if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it is a *model* of the general TBox $\mathcal{T}$ if it satisfies all GCIs in $\mathcal{T}$. Subsumption asks whether a given GCI $C \sqsubseteq D$ follows from a general TBox $\mathcal{T}$, i.e. whether every model of $\mathcal{T}$ satisfies $C \sqsubseteq D$. In this

case we say C is *subsumed* by D w.r.t. $\mathcal{T}$ and write $C \sqsubseteq_{\mathcal{T}} D$. Subsumption in $\mathcal{EL}$ w.r.t. a general TBox is known to be decidable in polynomial time [9].

An $\mathcal{EL}$-concept description is an *atom* if it is an existential restriction or a concept name. The atoms of an $\mathcal{EL}$-concept description C are the subdescriptions of C that are atoms, and the top-level atoms of C are the atoms occurring in the top-level conjunction of C. Obviously, any $\mathcal{EL}$-concept description is the conjunction of its top-level atoms, where the empty conjunction corresponds to $\top$. The atoms of a general TBox $\mathcal{T}$ are the atoms of all the concept descriptions occurring in $\mathcal{T}$.

We say that a subsumption between two atoms is *structural* if their top-level structure is compatible. To be more precise, we define structural subsumption between atoms as follows: the atom C is *structurally subsumed* by the atom D w.r.t. $\mathcal{T}$ ($C \sqsubseteq^{\mathsf{s}}_{\mathcal{T}} D$) iff either (i) $C = D$ is a concept name, or (ii) $C = \exists r.C'$, $D = \exists r.D'$, and $C' \sqsubseteq_{\mathcal{T}} D'$. It is easy to see that subsumption w.r.t. $\emptyset$ between two atoms implies structural subsumption w.r.t. $\mathcal{T}$, which in turn implies subsumption w.r.t. $\mathcal{T}$. The unification algorithm in [2] and the one presented below crucially depend on the following characterization of subsumption:

Lemma 1. *Let $\mathcal{T}$ be a general TBox and $C_1, \ldots, C_n$, $D_1, \ldots, D_m$ atoms. Then $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq_{\mathcal{T}} D_1 \sqcap \cdots \sqcap D_m$ iff for every $j \in \{1, \ldots, m\}$*

1. *there is an index $i \in \{1, \ldots, n\}$ such that $C_i \sqsubseteq^{\mathsf{s}}_{\mathcal{T}} D_j$, or*
2. *there are atoms $A_1, \ldots, A_k, B$ of $\mathcal{T}$ ($k \geq 0$) such that*
   a) *$A_1 \sqcap \cdots \sqcap A_k \sqsubseteq_{\mathcal{T}} B$,*
   b) *for every $\eta \in \{1, \ldots, k\}$ there is $i \in \{1, \ldots, n\}$ with $C_i \sqsubseteq^{\mathsf{s}}_{\mathcal{T}} A_\eta$, and*
   c) *$B \sqsubseteq^{\mathsf{s}}_{\mathcal{T}} D_j$.*

Our proof of this lemma in [1] is based on a new Gentzen-style proof calculus for subsumption w.r.t. a general TBox, which is similar to the one developed in [10] for subsumption w.r.t. cyclic and general TBoxes.

As mentioned in the introduction, our unification algorithm is complete only for general TBoxes that satisfy a certain restriction on cycles.

Definition 2. *The general TBox $\mathcal{T}$ is called* cycle-restricted *iff there is no nonempty word $w \in N_R^+$ and $\mathcal{EL}$-concept description C such that $C \sqsubseteq_{\mathcal{T}} \exists w.C$.*

In [1] we show that a given general TBox can easily be tested for cycle-restrictedness. The main idea is that it is sufficient to consider the cases where C is a concept name or $\top$.

Lemma 3. *Let $\mathcal{T}$ be a general TBox. It can be decided in time polynomial in the size of $\mathcal{T}$ whether $\mathcal{T}$ is cycle-restricted or not.*

3 Unification in $\mathcal{EL}$ w.r.t. General TBoxes

We partition the set N_C into a set N_v of concept variables (which may be replaced by substitutions) and a set N_c of concept constants (which must not be replaced by substitutions). A *substitution* σ maps every concept variable to an $\mathcal{EL}$-concept description. It is extended to concept descriptions in the usual way:

– $\sigma(A) := A$ for all $A \in N_c \cup \{\top\}$,
– $\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D)$ and $\sigma(\exists r.C) := \exists r.\sigma(C)$.

An $\mathcal{EL}$-concept description C is *ground* if it does not contain variables. Obviously, a ground concept description is not modified by applying a substitution. A general TBox is *ground* if it does not contain variables.

Definition 4. *Let $\mathcal{T}$ be a general TBox that is ground. An $\mathcal{EL}$-*unification problem *w.r.t. $\mathcal{T}$ is a finite set $\Gamma = \{C_1 \sqsubseteq^? D_1, \ldots, C_n \sqsubseteq^? D_n\}$ of subsumptions between $\mathcal{EL}$-concept descriptions. A substitution σ is a* unifier *of Γ w.r.t. $\mathcal{T}$ if σ* solves *all the subsumptions in Γ, i.e. if $\sigma(C_1) \sqsubseteq_{\mathcal{T}} \sigma(D_1), \ldots, \sigma(C_n) \sqsubseteq_{\mathcal{T}} \sigma(D_n)$. We say that Γ is* unifiable *w.r.t. $\mathcal{T}$ if it has a unifier.*

Note that we have restricted the background general TBox $\mathcal{T}$ to be ground. This is not without loss of generality. If $\mathcal{T}$ contained variables, then we would need to apply the substitution also to its GCIs, and instead of requiring $\sigma(C_i) \sqsubseteq_{\mathcal{T}} \sigma(D_i)$ we would thus need to require $\sigma(C_i) \sqsubseteq_{\sigma(\mathcal{T})} \sigma(D_i)$, which would change the nature of the problem considerably (see [1] for a more detailed discussion).

Preprocessing To simplify the description of the algorithm, it is convenient to first normalize the TBox and the unification problem appropriately. An atom is called *flat* if it is a concept name or an existential restriction of the form $\exists r.A$ for a concept name A. The general TBox $\mathcal{T}$ is called *flat* if it contains only GCIs of the form $A \sqcap B \sqsubseteq C$, where A, B are flat atoms or $\top$ and C is a flat atom. The unification problem Γ is called *flat* if it contains only flat subsumptions of the form $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D$, where $n \geq 0$ and $C_1, \ldots, C_n, D$ are flat atoms.[1]

Let Γ be a unification problem and $\mathcal{T}$ a general TBox. By introducing auxiliary variables and concept names, respectively, Γ and $\mathcal{T}$ can be transformed in polynomial time into a flat unification problem Γ' and a flat general TBox $\mathcal{T}'$ such that the unifiability status remains unchanged, i.e., Γ has a unifier w.r.t. $\mathcal{T}$ iff Γ' has a unifier w.r.t. $\mathcal{T}'$. In addition, if $\mathcal{T}$ was cycle-restricted, then so is $\mathcal{T}'$ (see [1] for details). Thus, we can assume without loss of generality that the input unification problem and general TBox are flat.

Local Unifiers The main idea underlying the "in NP" results in [4,2] is to show that any $\mathcal{EL}$-unification problem that is unifiable has a so-called local unifier. Let $\mathcal{T}$ be a flat cycle-restricted TBox and Γ a flat unification problem. The atoms of Γ are the atoms of all the concept descriptions occurring in Γ. We define

$$\begin{aligned} \mathrm{At} &:= \{C \mid C \text{ is an atom of } \mathcal{T} \text{ or of } \Gamma\} \text{ and} \\ \mathrm{At}_{\mathsf{nv}} &:= \mathrm{At} \setminus N_v \quad \text{(non-variable atoms)}. \end{aligned}$$

Every assignment S of subsets S_X of $\mathrm{At}_{\mathsf{nv}}$ to the variables X in N_v induces the following relation $>_S$ on N_v: $>_S$ is the transitive closure of

$$\{(X, Y) \in N_v \times N_v \mid Y \text{ occurs in an element of } S_X\}.$$

[1] If $n = 0$, then we have an empty conjunction on the left-hand side, which as usual stands for $\top$.

We call the assignment S *acyclic* if $>_S$ is irreflexive (and thus a strict partial order). Any acyclic assignment S induces a unique substitution σ_S, which can be defined by induction along $>_S$:

- If $X \in N_v$ is minimal w.r.t. $>_S$, then we define $\sigma_S(X) := \bigsqcap_{D \in S_X} D$.
- Assume that $\sigma(Y)$ is already defined for all Y such that $X >_S Y$. Then we define $\sigma_S(X) := \bigsqcap_{D \in S_X} \sigma_S(D)$.

We call a substitution σ *local* if it is of this form, i.e., if there is an acyclic assignment S such that $\sigma = \sigma_S$. If the unifier σ of Γ w.r.t. $\mathcal{T}$ is a local substitution, then we call it a *local unifier* of Γ w.r.t. $\mathcal{T}$.

The main technical result shown in [2] is that any unifiable $\mathcal{EL}$-unification problem w.r.t. a cycle-restricted TBox has a local unifier. This yields the following brute-force unification algorithm for $\mathcal{EL}$ w.r.t. cycle-restricted TBoxes: first guess an acyclic assignment S, and then check whether the induced local substitution σ_S solves Γ. As shown in [2], this algorithm runs in nondeterministic polynomial time. NP-hardness follows from the fact that already unification in $\mathcal{EL}$ w.r.t. the empty TBox is NP-hard [4].

4 A Goal-Oriented Unification Algorithm

The brute-force algorithm is not practical since it blindly guesses an acyclic assignment and only afterwards checks whether the guessed assignment induces a unifier. We now introduce a more goal-oriented unification algorithm, in which nondeterministic decisions are only made if they are triggered by "unsolved parts" of the unification problem. In addition, failure due to wrong guesses can be detected early. Any non-failing run of the algorithm produces a unifier, i.e., there is no need for checking whether the assignment computed by this run really produces a unifier. This goal-oriented algorithm generalizes the algorithm for unification in $\mathcal{EL}$ w.r.t. the empty TBox introduced in [6], though the rules look quite different because in the present paper we consider unification problems that consist of subsumptions whereas in [6] we considered equivalences.

We assume without loss of generality that the cycle-restricted TBox $\mathcal{T}$ and the unification problem Γ_0 are flat. Given $\mathcal{T}$ and Γ_0, the sets At and $\mathrm{At}_{\mathsf{nv}}$ are defined as above. Starting with Γ_0, the algorithm maintains a current unification problem Γ and a current acyclic assignment S, which initially assigns the empty set to all variables. In addition, for each subsumption in Γ it maintains the information on whether it is *solved* or not. Initially, all subsumptions are unsolved, except those with a variable on the right-hand side. Rules are applied only to unsolved subsumptions. A (non-failing) rule application does the following:

- it solves exactly one unsolved subsumption,
- it may extend the current assignment S, and
- it may introduce new flat subsumptions built from elements of At.

Each rule application that extends S_X additionally *expands* Γ *w.r.t.* X as follows: every subsumption $\mathfrak{s} \in \Gamma$ of the form $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? X$ is *expanded* by adding the subsumption $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? A$ to Γ for every $A \in S_X$.

Eager Ground Solving:

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

Eager Solving:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D$ if either
- there is $i \in \{1, \ldots, n\}$ such that $C_i = D$ or $C_i = X \in N_v$ and $D \in S_X$, or
- D is ground and $\bigsqcap \mathcal{G} \sqsubseteq_{\mathcal{T}} D$ holds, where $\mathcal{G}$ is the set of all ground atoms in $\{C_1, \ldots, C_n\} \cup \bigcup_{X \in \{C_1, \ldots, C_n\} \cap N_v} S_X$.

Action: Its application marks $\mathfrak{s}$ as *solved*.

Eager Extension:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D$ if there is $i \in \{1, \ldots, n\}$ with $C_i = X \in N_v$ and $\{C_1, \ldots, C_n\} \setminus \{X\} \subseteq S_X$.
Action: Its application 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*.

Fig. 1. The eager rules of the unification algorithm.

Subsumptions are only added if they are not already present in Γ. If a new subsumption is added to Γ, either by a rule application or by expansion of Γ, then it is initially designated unsolved, except if it has a variable on the right-hand side. Once a subsumption is in Γ, it will not be removed. Likewise, if a subsumption in Γ is marked as solved, then it will not become unsolved later.

If a subsumption is marked as solved, this does not mean that it is already solved by the substitution induced by the current assignment. It may be the case that the task of satisfying the subsumption was deferred to solving other subsumptions which are "smaller" than the given subsumption in a well-defined sense. The task of solving a subsumption whose right-hand side is a variable is deferred to solving the subsumptions introduced by expansion.

The rules of the algorithm consist of the three *eager* rules Eager Ground Solving, Eager Solving, and Eager Extension (see Figure 1), and several *nondeterministic* rules (see Figures 2 and 3). Eager rules are applied with higher priority than nondeterministic rules. Among the eager rules, Eager Ground Solving has the highest priority, then comes Eager Solving, and then Eager Extension.

Algorithm 5. Let Γ_0 be a flat $\mathcal{EL}$-unification problem. We set $\Gamma := \Gamma_0$ and $S_X := \emptyset$ for all $X \in N_v$. While Γ contains an unsolved subsumption, apply the steps (1) and (2). Once all subsumptions are solved, return the substitution σ induced by the current assignment.

(1) **Eager rule application:** If some eager rules apply to an unsolved subsumption $\mathfrak{s}$ in Γ, apply one of highest priority. If the rule application fails, then return "not unifiable".
(2) **Nondeterministic rule application:** If no eager rule is applicable, let $\mathfrak{s}$ be an unsolved subsumption in Γ. If one of the nondeterministic rules applies to $\mathfrak{s}$, nondeterministically choose one of these rules and apply it. If none of these rules apply to $\mathfrak{s}$ or the rule application fails, then return "not unifiable".

Decomposition:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? \exists s.D'$ if there is at least one index $i \in \{1, \ldots, n\}$ with $C_i = \exists s.C'$.
Action: Its application chooses such an index i, adds the subsumption $C' \sqsubseteq^? D'$ to Γ, expands it w.r.t. D' if D' is a variable, and marks $\mathfrak{s}$ as *solved*.

Extension:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D$ if there is at least one $i \in \{1, \ldots, n\}$ with $C_i \in N_v$.
Action: Its application chooses such an 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*.

Fig. 2. The nondeterministic rules *Decomposition* and *Extension*.

In step (2), the choice which unsolved subsumption to consider next is don't care nondeterministic. However, choosing which rule to apply to the chosen subsumption is don't know nondeterministic. Additionally, the application of nondeterministic rules requires don't know nondeterministic guessing.

The *eager rules* are mainly there for optimization purposes, i.e., to avoid nondeterministic choices if a deterministic decision can be made. For example, a ground subsumption, as considered in the *Eager Ground Solving* rule, either follows from the TBox, in which case any substitution solves it, or it does not, in which case it does not have a solution. This condition can be checked in polynomial time using the polynomial time subsumption algorithm for $\mathcal{EL}$ [9]. In the case considered in the *Eager Solving* rule, the substitution induced by the current assignment already solves the subsumption. In fact, if the first (second) condition of the rule is satisfied, then the first (second) condition of Lemma 1 applies. The *Eager Extension* rule solves a subsumption that contains only a variable X and some elements of S_X on the left-hand side. The rule is motivated by the following observation: for any assignment S' extending the current assignment, the induced substitution σ' satisfies $\sigma'(X) \equiv \sigma'(C_1) \sqcap \ldots \sqcap \sigma'(C_n)$. Thus, if S'_X contains D, then $\sigma'(X) \sqsubseteq_{\mathcal{T}} \sigma'(D)$, and σ' solves the subsumption. Conversely, if σ' solves the subsumption, then $\sigma'(X) \sqsubseteq_{\mathcal{T}} \sigma'(D)$, and thus adding D to S'_X yields an equivalent induced substitution.

The *nondeterministic rules* only come into play if no eager rules can be applied. In order to solve an unsolved subsumption $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D$, we consider the two conditions of Lemma 1. Regarding the first condition, which is addressed by the rules *Decomposition* and *Extension*, assume that γ is induced by an acyclic assignment S. To satisfy the first condition of the lemma with γ, the atom $\gamma(D)$ must subsume a top-level atom in $\gamma(C_1) \sqcap \cdots \sqcap \gamma(C_n)$. This atom can either be of the form $\gamma(C_i)$ for an atom C_i, or it can be of the form $\gamma(C)$ for an atom $C \in S_{C_i}$ and a variable C_i. In the second case, the atom C can either already be in S_{C_i} or it can be put into S_{C_i} by an application of the Extension rule. The *Mutation rules* cover the second condition in Lemma 1. For example, let us analyze in detail how *Mutation 1* ensures that all the requirements of the second condition of Lemma 1 are satisfied. Whenever this condition requires a structural

Mutation 1:

Condition: This rule applies to $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? D$ if $n > 1$ and there are atoms $A_1, \ldots, A_k, B$ of $\mathcal{T}$ such that $A_1 \sqcap \cdots \sqcap A_k \sqsubseteq_\mathcal{T} B$ holds.
Action: Its application chooses such atoms, marks $\mathfrak{s}$ as *solved*, and generates the following subsumptions:
- it chooses for each $\eta \in \{1, \ldots, k\}$ an $i \in \{1, \ldots, n\}$ and adds the new subsumption $C_i \sqsubseteq^? A_\eta$ to Γ,
- it adds $B \sqsubseteq^? D$ to Γ.

Mutation 2:

Condition: This rule applies to $\mathfrak{s} = \exists r.X \sqsubseteq^? D$ if X is a variable, D is ground, and there are atoms $\exists r.A_1, \ldots, \exists r.A_k$ of $\mathcal{T}$ such that $\exists r.A_1 \sqcap \cdots \sqcap \exists r.A_k \sqsubseteq_\mathcal{T} D$ holds.
Action: Its application chooses such atoms, adds $A_1, \ldots, A_k$ to S_X, expands Γ w.r.t. X, and marks $\mathfrak{s}$ as *solved*.

Mutation 3:

Condition: This rule applies to $\mathfrak{s} = \exists r.X \sqsubseteq^? \exists s.Y$ if X and Y are variables, and there are atoms $\exists r.A_1, \ldots, \exists r.A_k, \exists s.B$ of $\mathcal{T}$ with $\exists r.A_1 \sqcap \cdots \sqcap \exists r.A_k \sqsubseteq_\mathcal{T} \exists s.B$.
Action: Its application chooses such atoms, marks $\mathfrak{s}$ as *solved*, and generates the following subsumptions:
- it adds $A_1, \ldots, A_k$ to S_X and expands Γ w.r.t. X,
- it adds the subsumption $B \sqsubseteq^? Y$ to Γ and expands it w.r.t. Y.

Mutation 4:

Condition: This rule applies to $\mathfrak{s} = C \sqsubseteq^? \exists s.Y$ if C is a ground atom or $\top$, Y is a variable, and there is an atom $\exists s.B$ of $\mathcal{T}$ such that $C \sqsubseteq_\mathcal{T} \exists s.B$ holds.
Action: Its application chooses such an atom, adds the new subsumption $B \sqsubseteq^? Y$ to Γ, expands this subsumption w.r.t. Y, and marks $\mathfrak{s}$ as *solved*.

Fig. 3. The nondeterministic *Mutation* rules of the unification algorithm.

subsumption $\gamma(E) \sqsubseteq^\mathfrak{s}_\mathcal{T} \gamma(F)$ to hold for a (hypothetical) unifier γ of Γ, the rule creates the new subsumption $E \sqsubseteq^? F$, which has to be solved later on. This way, the rule ensures that the substitution built by the algorithm actually satisfies the conditions of the lemma. To check the subsumption $A_1 \sqcap \cdots \sqcap A_k \sqsubseteq_\mathcal{T} B$, the rule again employs a polynomial-time subsumption algorithm.

The *other mutation rules* follow the same idea, but they implicitly apply one or more Decomposition or Eager Extension rules after mutation. This ensures that the generated subsumptions are "smaller" than the subsumption that triggers their introduction.

Soundness We will show that, if Algorithm 5 returns a substitution σ on input Γ_0, then σ is a unifier of Γ_0 w.r.t. $\mathcal{T}$. In the following, let S be the final acyclic assignment computed by a non-failing run of Algorithm 5 on input Γ_0, and σ the substitution induced by S. By $\widehat{\Gamma}$ we denote the final set of subsumptions computed by this run, i.e., the original subsumptions of Γ_0 together with the

new ones generated by rule applications. To show that σ solves all subsumptions in $\widehat{\Gamma}$, we use well-founded induction [8] on the well-founded order $\succ$ on $\widehat{\Gamma}$:

Definition 6. *Let* $\mathfrak{s} = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? C_{n+1} \in \widehat{\Gamma}$.

- $\mathfrak{s}$ *is* small *if* $n = 1$ *and* C_1 *is ground or* C_{n+1} *is ground.*
- *We define* $m(\mathfrak{s}) := (m_1(\mathfrak{s}), m_2(\mathfrak{s}), m_3(\mathfrak{s}))$, *where*
  - $m_1(\mathfrak{s}) := 0$ *if* $\mathfrak{s}$ *is small, and* $m_1(\mathfrak{s}) := 1$ *otherwise;*
  - $m_2(\mathfrak{s}) := X$ *if* $C_{n+1} = X$ *or* $C_{n+1} = \exists r.X$ *for a variable* X *and some* $r \in N_R$, *and* $m_2(\mathfrak{s}) := \bot$ *otherwise;*
  - $m_3(\mathfrak{s}) := \max\{rd(\sigma(C_i)) \mid i \in \{1, \ldots, n+1\}\}$ *where* rd *yields the* role depth *of a concept description, i.e., the maximal nesting of existential restrictions.*
- *The strict partial order* $\succ$ *on such triples is the lexicographic order, where the first and the third component are compared w.r.t. the normal order* $>$ *on natural numbers. The variables in the second component are compared w.r.t. the relation* $>_S$ *induced by* S, *and* $\bot$ *is smaller than any variable.*
- *We extend* $\succ$ *to* $\widehat{\Gamma}$ *by setting* $\mathfrak{s}_1 \succ \mathfrak{s}_2$ *iff* $m(\mathfrak{s}_1) \succ m(\mathfrak{s}_2)$.

As the lexicographic product of well-founded strict partial orders is again well-founded [8], $\succ$ is a well-founded strict partial order on $\widehat{\Gamma}$.

Lemma 7. σ *is an* $\mathcal{EL}$*-unifier of* $\widehat{\Gamma}$ *w.r.t.* $\mathcal{T}$, *and thus also of its subset* Γ_0.

Proof. Let $\mathfrak{s} \in \widehat{\Gamma}$ and assume that σ solves all subsumptions $\mathfrak{s}' \in \widehat{\Gamma}$ with $\mathfrak{s}' \prec \mathfrak{s}$.

- If $\mathfrak{s}$ has a *non-variable atom as its right-hand side*, then it was initially marked as unsolved and must have been marked solved by a successful rule application. As an example, we consider the application of the Decomposition rule (the other rules can be treated similarly [1]). Then $\mathfrak{s}$ is of the form $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? \exists s.D'$ with $C_i = \exists s.C'$ for some $i \in \{1, \ldots, n\}$ and we have $\mathfrak{s}' = C' \sqsubseteq^? D' \in \widehat{\Gamma}$. We will show that $\mathfrak{s} \succ \mathfrak{s}'$ holds. By induction, this implies that σ solves $\mathfrak{s}'$, and by Lemma 1 thus also $\mathfrak{s}$.
  Observe first that $m_2(\mathfrak{s}) = m_2(\mathfrak{s}')$ since either $\exists s.D'$ and D' contain the same variable or are both ground. We now make a case distinction based on $m_1(\mathfrak{s}')$. If $\mathfrak{s}'$ is small, then $\mathfrak{s}$ is either non-small, i.e. $m_1(\mathfrak{s}) > m_1(\mathfrak{s}')$, or small and of the form $\exists s.C' \sqsubseteq^? \exists s.D'$. In the second case, we have $m_1(\mathfrak{s}) = m_1(\mathfrak{s}')$ and $m_3(\mathfrak{s}) > m_3(\mathfrak{s}')$. If $\mathfrak{s}'$ is non-small, then both C' and D' are variables, and thus $\mathfrak{s}$ is also non-small, which yields $m_1(\mathfrak{s}) = m_1(\mathfrak{s}')$. Furthermore, the maximal role depth obviously decreases when going from $\mathfrak{s}$ to $\mathfrak{s}'$, and thus $m_3(\mathfrak{s}) > m_3(\mathfrak{s}')$. In all cases we have shown $m(\mathfrak{s}) \succ m(\mathfrak{s}')$, i.e., $\mathfrak{s} \succ \mathfrak{s}'$.
- If $\mathfrak{s}$ has a *variable as its right-hand side*, it is of the form $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? X$ and for every $A \in S_X$ there is a subsumption $\mathfrak{s}_A = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? A$ in $\widehat{\Gamma}$. If $\mathfrak{s}$ is small, then $n = 1$ and C_1 is ground, and thus the subsumptions $\mathfrak{s}_A$ are also small. Thus, we have $m_1(\mathfrak{s}) \geq m_1(\mathfrak{s}_A)$ for every $A \in S_X$. Furthermore, we have $m_2(\mathfrak{s}) > m_2(\mathfrak{s}_A)$ since A is ground or contains a variable on which X depends. This yields $\mathfrak{s} \succ \mathfrak{s}_A$, and thus by induction $\sigma(C_1) \sqcap \cdots \sqcap \sigma(C_n) \sqsubseteq_{\mathcal{T}} \sigma(A)$ for every $A \in S_X$, which implies that $\sigma(C_1) \sqcap \cdots \sqcap \sigma(C_n) \sqsubseteq_{\mathcal{T}} \sigma(X)$ by the definition of σ. ◻

Completeness Assume that Γ_0 is unifiable w.r.t. $\mathcal{T}$ and let γ be a ground unifier of Γ_0 w.r.t. $\mathcal{T}$. We use this unifier to guide the application of the nondeterministic rules such that Algorithm 5 does not fail. The following invariants for Γ and S will be maintained:

(I) γ is a unifier of Γ.
(II) For all $B \in S_X$ we have $\gamma(X) \sqsubseteq_{\mathcal{T}} \gamma(B)$.

Since S_X is initialized to $\emptyset$ for all variables $X \in N_v$ and Γ is initialized to Γ_0, these invariants are satisfied after the initialization of the algorithm.

The invariants immediately rule out one cause of failure for the algorithm, namely that the current assignment becomes cyclic. This is the only place in the whole proof where our assumption on cycle-restrictedness of $\mathcal{T}$ is needed.

Lemma 8. *If invariant (II) is satisfied, then the current assignment S is acyclic.*

The proofs of this and of the next lemma can be found in [1].

Lemma 9. *Assume that the current set of subsumptions Γ and the current assignment S satisfy the invariants (I) and (II), and let $\mathfrak{s} \in \Gamma$ be unsolved.*

1. *If an eager rule applies to $\mathfrak{s}$, then its application does not fail and the resulting set Γ' and assignment S' also satisfy the invariants (I) and (II).*
2. *If no eager rule applies to $\mathfrak{s}$, then there is a nondeterministic rule that can successfully be applied to $\mathfrak{s}$ such that the resulting set Γ' and assignment S' also satisfy the invariants (I) and (II).*

An immediate consequence of this lemma is that, if Γ_0 is unifiable, then there is a non-failing run of Algorithm 5 on Γ_0 during which the invariants (I) and (II) are satisfied. Together with the fact that every run of the algorithm terminates (see below), this shows completeness, i.e., whenever Γ_0 has a unifier w.r.t. $\mathcal{T}$, the algorithm computes one.

Termination Consider a run of Algorithm 5. It is easy to show that any subsumption encountered during this run falls into one of the following categories:

1. subsumptions from Γ_0;
2. subsumptions created by expansion from Γ_0: these are of the form $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? A$ for a subsumption $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq^? X \in \Gamma_0$ and $A \in \mathrm{At}_{\mathsf{nv}}$;
3. subsumptions of the form $C \sqsubseteq^? D$ for $C, D \in \mathrm{At}$.

Since the cardinality of At is polynomially bounded by the size of Γ_0 and $\mathcal{T}$, there are only polynomially many subsumptions of this form. Rules are only applicable to subsumptions that are marked unsolved, and the application of a rule marks at least one subsumption as solved. Thus, only polynomially many rules can be applied during the run. In addition, each rule application takes only polynomial time. This shows that every run of the algorithm terminates in polynomial time.

Theorem 10. *Algorithm 5 is an* NP*-decision procedure for unifiability in $\mathcal{EL}$ w.r.t. cycle-restricted TBoxes.*

5 Conclusions

We have presented a goal-oriented NP-algorithm for unification in $\mathcal{EL}$ w.r.t. cycle-restricted TBoxes. In [3], we developed a reduction of this problem to a propositional satisfiability problem (SAT), which is based on a characterization of subsumption different from the one in Lemma 1. Though clearly better than the brute-force algorithm introduced in [2], both algorithms suffer from a high degree of nondeterminism introduced by having to guess atoms and GCIs from the underlying cycle-restricted TBox. We have to find optimizations to tackle this problem before an implementation becomes feasible.

On the theoretical side, the main topic for future research is to consider unification w.r.t. unrestricted general TBoxes. In order to generalize the brute-force algorithm in this direction, we need to find a more general notion of locality. Starting with the goal-oriented algorithm, the idea would be not to fail when a cyclic assignment is generated, but rather to add rules that can break such cycles, similar to what is done in procedures for general E-unification [11].

References

1. Baader, F., Borgwardt, S., Morawska, B.: Unification in the description logic $\mathcal{EL}$ w.r.t. cycle-restricted TBoxes. LTCS-Report 11-05, Chair of Automata Theory, TU Dresden, Germany (2011), see http://lat.inf.tu-dresden.de/research/reports.html.
2. Baader, F., Borgwardt, S., Morawska, B.: Extending unification in $\mathcal{EL}$ towards general TBoxes. In: Proc. KR'12. AAAI Press (2012), short paper. To appear.
3. Baader, F., Borgwardt, S., Morawska, B.: SAT encoding of unification in $\mathcal{ELH}_{R^+}$ w.r.t. cycle-restricted ontologies. LTCS-Report 12-02, Chair for Automata Theory, TU Dresden (2012), see http://lat.inf.tu-dresden.de/research/reports.html. A short version of this report has been submitted to a conference.
4. Baader, F., Morawska, B.: Unification in the description logic $\mathcal{EL}$. In: Proc. RTA'09. LNCS, vol. 5595, pp. 350–364. Springer (2009)
5. Baader, F., Morawska, B.: SAT encoding of unification in $\mathcal{EL}$. In: Proc. LPAR'10. LNCS, vol. 6397, pp. 97–111. Springer (2010)
6. Baader, F., Morawska, B.: Unification in the description logic $\mathcal{EL}$. Log. Meth. Comput. Sci. 6(3) (2010)
7. Baader, F., Narendran, P.: Unification of concept terms in description logics. J. Symb. Comput. 31(3), 277–305 (2001)
8. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, United Kingdom (1998)
9. Brandt, S.: Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In: Proc. ECAI'04. pp. 298–302. IOS Press (2004)
10. Hofmann, M.: Proof-theoretic approach to description-logic. In: Proc. LICS'05. pp. 229–237. IEEE Press (2005)
11. Morawska, B.: General E-unification with eager variable elimination and a nice cycle rule. J. Autom. Reasoning 39(1), 77–106 (2007)