UEL: Unification Solver for the Description Logic $\mathcal{EL}$ — System Description

Franz Baader, Julian Mendez, and Barbara Morawska

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

Abstract UEL is a system that computes unifiers for unification problems formulated in the description logic $\mathcal{EL}$. $\mathcal{EL}$ is a description logic with restricted expressivity, but which is still expressive enough for the formal representation of biomedical ontologies, such as the large medical ontology SNOMED CT. We propose to use UEL as a tool to detect redundancies in such ontologies by computing unifiers of two formal concepts suspected of expressing the same concept of the application domain. UEL can be used as a plug-in of the popular ontology editor Protégé, or as a standalone unification application.

1 Motivation

The description logic (DL) $\mathcal{EL}$, which offers the concept constructors conjunction ($\sqcap$), existential restriction ($\exists r.C$), and the top concept ($\top$), has recently drawn considerable attention since, on the one hand, important inference problems such as the subsumption problem are polynomial in $\mathcal{EL}$ [1,8,2]. On the other hand, though quite inexpressive, $\mathcal{EL}$ can be used to define biomedical ontologies, such as the large medical ontology SNOMED CT.[1]

Unification in DLs has been proposed in [6] 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

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

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 $\mathsf{Severe\_injury}$ as variables, and substituting the first one by $\mathsf{Injury} \sqcap \exists \mathsf{finding\_site}.\mathsf{Head}$ and the second one by $\mathsf{Injury} \sqcap \exists \mathsf{severity}.\mathsf{Severe}$. In this case, we say that the descriptions are unifiable,

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

Name	Syntax	Semantics
concept name	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role name	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
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	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{x \mid \exists y : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
concept definition	$A \equiv C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$

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

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 Head_injury as Injury $\sqcap$ $\exists$finding_site.Head and Severe_injury as Injury $\sqcap$ $\exists$severity.Severe, then the two concept descriptions (1) and (2) are equivalent w.r.t. these definitions. Of course, this example was constructed such that the unifier actually 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 that 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.

In [3] it was shown that unification in $\mathcal{EL}$ is an NP-complete problem. Basically, this problem is in NP since every solvable unification problem has a "local" unifier, i.e., one built from parts of the unification problem. The NP algorithm introduced in [3] is a brutal "guess and then test" algorithm, which guesses a local substitution and then checks whether it is a unifier. In [5], a more practical $\mathcal{EL}$-unification algorithm was introduced, which tries to transform the given unification problems into a solved form, and makes nondeterministic decisions only if triggered by the problem. While having the potential of becoming quite efficient, this algorithm still requires a high amount of additional optimization work before it can be used in practice. Our system UEL[2] is based on a third kind of algorithm, which encodes the unification problem into a set of propositional clauses [4], and then solves it using an existing highly optimized SAT solver.

2 $\mathcal{EL}$ and unification in $\mathcal{EL}$

In order to explain what UEL actually computes, we need to recall the relevant definitions and results for $\mathcal{EL}$ and unification in $\mathcal{EL}$ (see [7,1,5] for details).

Starting with a finite set N_C of *concept names* and a finite set N_R of *role names*, *$\mathcal{EL}$-concept descriptions* are built from concept names using the constructors *conjunction* ($C \sqcap D$), *existential restriction* ($\exists r.C$ for every $r \in N_R$), and *top* ($\top$). On the semantic side, concept descriptions are interpreted as sets.

[2] Version 1.0.0 of this system, as described in this paper, is available for download at `http://sourceforge.net/projects/uel/files/uel/1.0.0/`.

To be more precise, an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$ 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 descriptions as shown in the semantics column of Table 1.

A *concept definition* is of the form $A \equiv C$ for a concept name A and a concept description C. A *TBox* $\mathcal{T}$ is a finite set of concept definitions such that no concept name occurs more than once on the left-hand side of a definition in $\mathcal{T}$. The TBox $\mathcal{T}$ is called *acyclic* if there are no cyclic dependencies between its concept definitions. Given a TBox $\mathcal{T}$, we call a concept name A a *defined concept* if it occurs as the left-side of a concept definition $A \equiv C$ in $\mathcal{T}$. All other concept names are called *primitive concepts*. An interpretation is a *model* of a TBox $\mathcal{T}$ if $A^{\mathcal{I}} = C^{\mathcal{I}}$ holds for all definitions $A \equiv C$ in $\mathcal{T}$.

Subsumption asks whether a given concept description C is a subconcept of another concept description D: C is *subsumed* by D w.r.t. $\mathcal{T}$ ($C \sqsubseteq_{\mathcal{T}} D$) if every model of $\mathcal{T}$ satisfies $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. We say that C is *equivalent* to D w.r.t. $\mathcal{T}$ ($C \equiv_{\mathcal{T}} D$) if $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$. For the empty TBox, we write $C \sqsubseteq D$ and $C \equiv D$ instead of $C \sqsubseteq_{\emptyset} D$ and $C \equiv_{\emptyset} D$, and simply talk about subsumption and equivalence (without saying "w.r.t. $\emptyset$").

In order to define unification, we partition the set N_C of concept names 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). Intuitively, N_v are the concept names that have possibly been given another name or been specified in more detail in another concept description describing the same notion. A *substitution* σ maps every variable to a concept description. It can be extended to concept descriptions in the usual way. Unification in $\mathcal{EL}$ was first considered w.r.t. the empty TBox [3]. In this setting, an *$\mathcal{EL}$-unification problem* is a finite set $\Gamma = \{C_1 \equiv^? D_1, \ldots, C_n \equiv^? D_n\}$ of equations. A substitution σ is a *unifier* of Γ if σ *solves* all the equations in Γ, i.e., if $\sigma(C_1) \equiv \sigma(D_1), \ldots, \sigma(C_n) \equiv \sigma(D_n)$. We say that Γ is *solvable* if it has a unifier.

As mentioned before, the main reason for solvability of unification in $\mathcal{EL}$ to be in NP is that any solvable unification problem has a local unifier. Basically, any unification problem Γ determines a polynomial number of so-called *non-variable atoms*, which are concept constants or existential restrictions of the form $\exists r.A$ for a role name r and a concept constant or variable A. An *assignment* S maps every concept variable X to a subset S_X of the set $\mathrm{At}_{\mathsf{nv}}$ of non-variable atoms of Γ. Such an assignment induces the following relation $>_S$ on N_v: $>_S$ is the transitive closure of $\{(X, Y) \in N_v \times N_v \mid Y \text{ occurs in an element of } S_X\}$. 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 is a minimal element of N_v 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 $\sigma = \sigma_S$. In [3] it is shown that any solvable unification

problem has a local unifier. Consequently, one can enumerate (or guess, in a nondeterministic machine) all acyclic assignments and then check whether any of them induces a substitution that is a unifier. Using this brute-force approach, in general many local substitutions will be generated that only in the subsequent check turn out not to be unifiers.

In contrast, the SAT reduction introduced in [4] ensures that only assignments that induce unifiers are generated. The set of propositional clauses $C(\Gamma)$ generated by the reduction contains two kinds of propositional letters: $[A \not\sqsubseteq B]$ for $A, B \in \mathrm{At}_{\mathsf{nv}}$ and $[X > Y]$ for concept variables X, Y. Intuitively, setting $[A \not\sqsubseteq B] = 1$ means that the local substitution σ_S induced by the corresponding assignment S satisfies $\sigma_S(A) \not\sqsubseteq \sigma_S(B)$, and setting $[X > Y] = 1$ means that $X >_S Y$. The clauses in $C(\Gamma)$ are such that Γ has a unifier iff $C(\Gamma)$ is satisfiable. In particular, any propositional valuation τ satisfying $C(\Gamma)$ defines an assignment S^τ with $S^\tau_X := \{A \mid \tau([X \not\sqsubseteq A]) = 0, A \in \mathrm{At}_{\mathsf{nv}}\}$, which induces a local unifier of Γ. Conversely, any local unifier of Γ can be obtained in this way. Thus, by generating all propositional valuations satisfying $C(\Gamma)$ we can generate all local unifiers of Γ.

In [5], *unification w.r.t. an acyclic TBox* $\mathcal{T}$ was introduced. In this setting, the concept variables are a subset of the primitive concepts of $\mathcal{T}$, and substitutions are applied both to the concept descriptions in the unification problem and to the right-hand sides of the definitions in $\mathcal{T}$. To deal with such unification problems, one does not need to develop a new algorithm. In fact, by viewing the defined concepts of $\mathcal{T}$ as variables, one can turn $\mathcal{T}$ into a unification problem, which one simply adds to the given unification problem Γ. As shown in [5], there is a 1–1-correspondence between the unifiers of Γ w.r.t. $\mathcal{T}$ and the unifiers of this extended unification problem.

3 Things not mentioned in the theoretical papers

When implementing UEL, we had to deal with several issues that are abstracted away in the theoretical papers describing unification algorithms for $\mathcal{EL}$.

Primitive definitions In addition to concept definitions, as introduced above, biomedical ontologies often contain so-called *primitive definitions* $A \sqsubseteq C$ where A is a concept name and C is a concept description. Models $\mathcal{I}$ of $A \sqsubseteq C$ need to satisfy $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$. Thus, primitive definitions formulate necessary conditions for concept membership, but these conditions are not sufficient. SNOMED CT contains about 350,000 primitive definitions and only 40,000 concept definitions.

By using a trick first introduced by Nebel [9], primitive definitions $A \sqsubseteq C$ can be turned into concept definitions $A \equiv C \sqcap A\_UNDEF$, where A_UNDEF is a new concept name that stands for the undefined part of the definition of A. In the resulting acyclic TBox, these new concept names are primitive concepts, and thus can be declared to be variables. In this case, a unifier σ suggests how to complete the definition of A by providing the concept description $\sigma(A\_UNDEF)$.

Unifiers as acyclic TBoxes Given an acyclic assignment S computed by the SAT reduction, our system UEL actually does not produce the corresponding local unifier σ_S as output, but rather the acyclic TBox $\mathcal{T}_S := \{X \equiv \bigsqcap_{D \in S_X} D \mid X \in N_v\}$. This TBox solves the input unification problem Γ w.r.t. $\mathcal{T}$ in the sense that $C \equiv_{\mathcal{T} \cup \mathcal{T}_S} D$ holds for all equations $C \equiv^? D$ in Γ. This is actually what the developer that employs unification wants to know: how must the concept variables be defined such that the concept descriptions in the equations become equivalent? Another advantage of this representation of the output is that the size of S and thus of $\mathcal{T}_S$ is polynomial in the size of the input Γ and $\mathcal{T}$, while the size of the concept descriptions $\sigma_S(X)$ may be exponential in this size. In the following, we will also call the TBoxes $\mathcal{T}_S$ unifiers.

Internal variables The unification algorithms for $\mathcal{EL}$ actually assume that the unification problem is first transformed into a so-called flat form. This form can easily be generated by introducing auxiliary variables. These new variables have system-generated names, which do not make sense to the user. Thus, they should not show up in the output acyclic TBox $\mathcal{T}_S$. By replacing such auxiliary defined concepts in $\mathcal{T}_S$ by their definitions as long as auxiliary names occur, we can transform $\mathcal{T}_S$ into an acyclic TBox that satisfies this requirement, actually without causing an exponential blow-up of the size of the TBox.

Reachable sub-TBox As mentioned above, acyclic TBoxes are treated by viewing them as part of the unification problem. For very large TBoxes like SNOMED CT, adding the whole TBox to the unification problem is neither viable nor necessary. In fact, it is sufficient to add the reachable part of the TBox, i.e., the definitions onto which the concept descriptions in the unification problem depend. This reachable part is usually rather small, even for very large TBoxes.

Enumeration of all local unifiers Depending on how many concept names are turned into variables, a unification problem can have many local unifiers. If the SAT solver has provided a satisfying propositional valuation, we can add a clause to the SAT problem that prevents the re-computation of this unifier, and call the SAT solver with this new SAT instance. While computing a single unifier is usually quite fast, computing all of them can take much longer. Thus, we enable the user to compute and then inspect one unifier at a time. If this unifier makes sense, i.e., suggests reasonable definitions for the variables, then the user can stop. Otherwise, by pressing a button, the computation of the next local unifier can be initiated. For this to work well, it is important that "good" unifiers are computed first. For the moment, we have interpreted "good" as meaning small, i.e., we want to compute those unifiers first that are generated by acyclic assignments for which the sets S_X are small. It has turned out that the SAT reduction sketched above actually leads to computing unifiers in the opposite order, at least if we use a SAT solver that tries to minimize the number of propositional letters that are set to 1. In fact, setting a letter of the form $[X \not\sqsubseteq A]$ for $X \in N_v$ and $A \in \mathsf{At_{nv}}$ to 0 rather than 1 adds A to S_X. This problem can be overcome by using propositional letters $[A \sqsubseteq B]$ with the obvious meaning, and basically replacing $[A \not\sqsubseteq B]$ in the SAT reduction by $\neg[A \sqsubseteq B]$.

4 The system UEL and how to use it

UEL was implemented in Java 1.6 and is compatible with Java 1.7. It uses the OWL API 3.2.4[3] to read ontologies. It has a visual interface that can be used as a Protégé 4.1 plug-in, or as a standalone application. The unification problem generated by the user through this interface is translated into a propositional formula in conjunctive normal form using the DIMACS CNF format,[4] which is the most popular format used by SAT solvers. As SAT solver, we currently use SAT4J,[5] which is implemented in Java. This configuration is, however, parametrized and can be easy changed to any SAT solver that accepts DIMACS CNF input and returns the computed satisfying propositional valuation.

After opening UEL's visual interface, the first step is to open one or two ontologies. The second option enables unification of concepts defined in different ontologies. The user can then choose two concepts to be unified.[6] This is done by choosing two concept names that occur on the left-hand sides of concept definitions or primitive definitions. UEL then computes the subontologies reachable from these concept names, and turns the primitive definitions in these subontologies into concept definitions.

After choosing the concepts to be unified, pressing the button opens a dialog window in which the user is presented with the primitive concepts contained in these subontologies (including the ones with ending *_UNDEF*). The user can then decide which of these primitive concepts should be viewed as variables in the unification problem

Once the user has chosen the variables, UEL computes the unification problem defined this way, and transforms it into a clause set in DIMACS CNF format. It also opens a dialog window with control buttons. By pressing the button , the user triggers the computation of the first unifier (or later, of the next one). Each computed unifier is shown (as an acyclic TBox) in the dialog window. The button can be used to go back to the previously computed unifier. The button can be used to trigger the computation of all (remaining) unifiers, and the button allows to jump back to the first unifier. Unifiers already computed are stored, and thus need not be recomputed during navigation. Each unifier (i.e., the acyclic TBox representing it) can be saved using the RDF/OWL or the KRSS format by pressing the button . The format for saving is determined by the file ending typed by the user (.krss or .owl).

The user can use the button to retrieve internal details about the computation process. These details include the unification problem created internally by UEL, the number of all concept variables (user chosen and internal variables), the number of propositional letters, and the number of propositional clauses that are checked for satisfiability by the SAT solver.

---

[3] http://owlapi.sourceforge.net

[4] http://www.satcompetition.org/2004/format-solvers2004.html

[5] http://www.sat4j.org

[6] Note that a finite set of equations $\{C_1 \equiv^? D_1, \ldots, C_n \equiv^? D_n\}$ can always be encoded into the single equation $\{\exists r_1.C_1 \sqcap \ldots \sqcap \exists r_n.C_n \equiv^? \exists r_1.D_1 \sqcap \cdots \sqcap \exists r_n.D_n\}$, where $r_1, \ldots, r_n$ are pairwise distinct role names.

5 An example

We consider a modified version of our example in the first section, where the TBox gives (1) as definition for the concept name Patient_with_severe_head_injury and (2) as definition for the concept name Patient_with_severe_injury_at_head. In addition, the TBox contains two primitive definitions, saying that Head_injury and Severe_injury are subconcepts of Injury. We load this TBox into UEL and choose Patient_with_severe_head_injury and Patient_with_severe_injury_at_head as the concepts to be unified. The system then offers us the primitive concepts Patient, Severe, Head as well as Head_injury_UNDEF, Severe_injury_UNDEF as possible variables, of which we choose only the latter two.

The SAT translation generates a SAT problem consisting of 3976 clauses and containing 320 different propositional letters. The first unifier computed by UEL is the substitution

$$\{\mathsf{Head\_injury\_UNDEF} \mapsto \exists \mathsf{finding\_site.Head},$$
$$\mathsf{Severe\_injury\_UNDEF} \mapsto \exists \mathsf{severity.Severe}\}.$$

This unifier thus completes the primitive definitions of the concepts Head_injury and Severe_injury to concept definitions $\mathsf{Head\_injury} \equiv \mathsf{Injury} \sqcap \exists \mathsf{finding\_site.Head}$ and $\mathsf{Severe\_injury} \equiv \mathsf{Injury} \sqcap \exists \mathsf{severity.Severe}$.

However, the unification problem has 127 additional local unifiers. Some of them are similar to the first one, but contain "redundant" conjuncts. Others do not make much sense in the application (e.g., ones where Patient occurs in the images of the variables). Computing all 128 local unifiers at once (after pressing the button ⏩) takes less than 1 second.

References

1. Franz Baader. Terminological cycles in a description logic with existential restrictions. In *Proc. IJCAI'03*, 2003.
2. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. IJCAI'05*, 2005.
3. Franz Baader and Barbara Morawska. Unification in the description logic $\mathcal{EL}$. In *Proc. RTA'09*, Springer LNCS 5595, 2009.
4. Franz Baader and Barbara Morawska. SAT encoding of unification in $\mathcal{EL}$. In *Proc. (LPAR-17)*, Springer LNCS 6397, 2010.
5. Franz Baader and Barbara Morawska. Unification in the description logic $\mathcal{EL}$. *Logical Methods in Computer Science*, 6(3), 2010.
6. Franz Baader and Paliath Narendran. Unification of concept terms in description logics. *J. of Symbolic Computation*, 31(3):277–305, 2001.
7. Franz Baader and Werner Nutt. Basic Description Logics. In *The Description Logic Handbook*, Cambridge University Press, 2003.
8. Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In *Proc. ECAI'04*, 2004.
9. Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*, Springer LNCS 422, 1990.