Optimizations for the role-depth bounded least common subsumer in $\mathcal{EL}^+$

Andreas Ecke and Anni-Yasmin Turhan*

TU Dresden, Institute for Theoretical Computer Science

Abstract. Computing the least common subsumer (lcs) yields a generalization of a collection of concepts, computing such generalizations is a useful reasoning task for many ontology-based applications. Since the lcs need not exist, if computed w.r.t. general TBoxes, an approximative approach, the role-depth bounded lcs, has been proposed. Recently, this approach has been extended to the Description logic $\mathcal{EL}^+$, which covers most of the OWL 2 EL profile.
In this paper we present two kinds of optimizations for the computation of such approximative lcs: one to obtain succinct rewritings of $\mathcal{EL}^+$-concepts and the other to speed-up the k-lcs computation. The evaluation of these optimizations give evidence that they can improve the computation of the role-depth bounded lcs by orders of magnitude.

1 Introduction

Intuitively, the lcs is a new (complex) concept description that captures the commonalities of all of its input concept descriptions. This reasoning service has turned out to be useful for the augmentation of TBoxes and as a subtask when computing the (dis)similarity of concept descriptions [8] or other non-standard inferences.

In particular, bio-medical TBoxes are often written in extensions of $\mathcal{EL}$ that allow to model roles in a more detailed way, such as SNOMED [13] which allows to use role inclusions and is written in $\mathcal{ELH}$ or the Gene Ontology [6] and the FMA ontology [12] which are both written in (sub-languages of) $\mathcal{EL}^+$. For these extensions of $\mathcal{EL}$ standard DL reasoning can still be done in polynomial time [3]. These TBoxes are known to be very large and are mostly built by hand.

If computed w.r.t. general or just cyclic $\mathcal{EL}$-TBoxes, the lcs need not exist [1], since resulting cyclic concept descriptions cannot be expressed in $\mathcal{EL}$. We follow the approach from [10] and compute as an approximative solution the *role-depth bounded lcs*: k-lcs which has a maximal nesting of quantifiers limited to k. The method to compute the k-lcs is to employ the completion method that is used to classify the TBox. This method builds a graph structure, which is saturated by completion rules [2, 3]. In case of $\mathcal{EL}$ the k-lcs is more or less the cross-product of the tree-unravelings of different nodes of the saturated completion graph [10]. It

* Partially supported by the German Research Foundation (DFG) in the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing".

Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
general concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
role inclusion axiom	$r_1 \circ \ldots \circ r_k \sqsubseteq r$	$r_1^{\mathcal{I}} \circ \ldots \circ r_k^{\mathcal{I}} \subseteq r^{\mathcal{I}}$

Table 1. Constructors and axioms for $\mathcal{EL}^+$.

turns out that for $\mathcal{EL}^+$ the computation algorithm is the same as for $\mathcal{EL}$ [7]. Unfortunately, the concept descriptions obtained in this way turn out to be highly redundant and thus take a long time to compute. In order to obtain concise and readable concept descriptions for the k-lcs, we devise a method to obtain smaller, but equivalent rewritings for $\mathcal{EL}^+$-concept descriptions. Furthermore, we devise two methods that avoid the cross-product computation for those parts of the unraveling that would yield a redundant part in the resulting concept description. These optimization methods are highly effective and expedient, when dealing with ontologies with large role hierarchies such as GALEN.

This paper is organized as follows: After preliminaries on DLs, we briefly sketch how completion is employed to compute the k-lcs in $\mathcal{EL}^+$ in Section 3. In Section 4 we introduce the optimization methods and in Section 5 we report on an evaluation of the optimization methods on (variants of) medical ontologies.

2 Preliminaries

We assume that the reader is familiar with the basic notions of DLs, for an introduction see [4]. We introduce the DLs used in this paper formally. *Concept descriptions* are inductively defined from a set of *concepts names* N_C and a set of *role names* N_R by applying the constructors from the upper half of Table 1. In particular, $\mathcal{EL}$-concept descriptions only allow for conjunctions, existential restrictions, and the top concept $\top$. The concept constructors are interpreted in the standard way. Their semantics is displayed in the upper half of Table 1.

In $\mathcal{EL}$ there are *general concept inclusions* (GCIs), which are displayed in the lower half of Table 1. $\mathcal{EL}^+$ additionally allows for complex *role inclusion axioms* (RIAs). These role inclusions can express, in particular, inclusion of roles ($s \sqsubseteq r$) and transitive roles ($r \circ r \sqsubseteq r$). Now, an $\mathcal{EL}$-TBox $\mathcal{T}$ is a set of GCIs and an $\mathcal{EL}^+$-TBox is a set of GCIs and RIAs. We denote by $N_{C,\mathcal{T}}$ and $N_{R,\mathcal{T}}$ the set of concept names and the set of role names that occur in a TBox $\mathcal{T}$. The fundamental reasoning service for TBoxes is *subsumption*. We say that a concept description C subsumes another concept description D w.r.t. the TBox $\mathcal{T}$ (written $\sqsubseteq_{\mathcal{T}}$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models of $\mathcal{T}$.

For a concept description C we denote by $rd(C)$ its *role-depth*, i.e., its maximal nesting of quantifiers. We define the central reasoning services of this paper for the Description Logic $\mathcal{EL}^+$.

Definition 1 ((Role-depth bounded) least common subsumer). *Let $\mathcal{T}$ be an $\mathcal{EL}^+$-TBox and $C_1, \ldots, C_n$ be $\mathcal{EL}^+$-concept descriptions. Then the $\mathcal{EL}^+$-*

concept description D is the least common subsumer *of $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$ iff (1) $C_i \sqsubseteq_\mathcal{T} D$ for all $i \in \{1, \ldots, n\}$, and (2) for all $\mathcal{EL}^+$-concept descriptions E: $C_i \sqsubseteq_\mathcal{T} E$ for all $i \in \{1, \ldots, n\}$ implies $D \sqsubseteq_\mathcal{T} E$.*

Let $k \in \mathbb{N}$. Then the $\mathcal{EL}^+$-concept description D is the role-depth bounded least common subsumer *of $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$ and the role-depth k iff (1) $rd(D) \leq k$, (2) $C_i \sqsubseteq_\mathcal{T} D$ for all $i \in \{1, \ldots, n\}$, and (3) for all $\mathcal{EL}^+$-concept descriptions E with $rd(E) \leq k$: $C_i \sqsubseteq_\mathcal{T} E \; \forall i \in \{1, \ldots, n\}$ implies $D \sqsubseteq_\mathcal{T} E$.*

For $\mathcal{EL}^+$ the (k-)lcs is unique up to equivalence, thus we speak of the (k-)lcs.

3 Computing the k-lcs in $\mathcal{EL}^+$ by Completion

The algorithms to compute the role-depth bounded lcs rely on completion graphs produced by completion-based subsumption algorithms. Completion algorithms work on normalized TBoxes for which they build a completion graph and exhaustively apply completion rules to it. After this step, the completion graph contains all subsumption relations from the TBox explicitly.

3.1 Completion algorithm for $\mathcal{EL}^+$

An $\mathcal{EL}^+$-TBox $\mathcal{T}$ is in normal form, if all GCIs in $\mathcal{T}$ are of the form $A \sqsubseteq B$, $A_1 \sqcap A_2 \sqsubseteq B$, $A \sqsubseteq \exists r.B$, or $\exists r.A \sqsubseteq B$ with $A, A_1, A_2, B \in N_C$ and $r \in N_R$; and all role inclusions are of the form $s \sqsubseteq r$ or $s \circ t \sqsubseteq r$ with $\{r, s, t\} \subseteq N_R$. All $\mathcal{EL}^+$-TBoxes can be normalized by applying a set of normalization rules [2].

The *completion graph* for a normalized TBox $\mathcal{T}'$ is of the form (V, E, S), where $V = N_{C,\mathcal{T}'} \cup \{\top\}$ is the *set of nodes*, $E \subseteq V \times N_{R,\mathcal{T}} \times V$ is the *set of role name labeled edges* and $S : V \to 2^{N_{C,\mathcal{T}'} \cup \{\top\}}$ is the *node-labeling*. The completion algorithm starts with an initial graph (V, E, S) with $E = \emptyset$ and $S(A) = \{A, \top\}$ for each $A \in N_{C,\mathcal{T}'} \cup \{\top\}$ and exhaustively applies a set of completion rules from [2] until no more rule applies.

After rule-application all subsumption relations can be directly read-off the completion graph. This completion algorithm is sound and complete as shown in [2]. The intuition is that the completion rules make implicit subsumption relationships explicit in the following sense:

- $B \in S(A)$ implies that $A \sqsubseteq_\mathcal{T} B$,
- $(A, r, B) \in E$ implies that $A \sqsubseteq_\mathcal{T} \exists r.B$.

The resulting completion graph is used to compute the role-depth bounded lcs.

3.2 Computation algorithm of the k-lcs in $\mathcal{EL}^+$

All RIAs from the $\mathcal{EL}^+$-TBox are explicitly captured in the completion graph in the following way: for each edge in the completion graph that is labeled with some role r, the completion algorithm also creates edges for all of r's super-roles. This means that for computing the k-lcs w.r.t. an $\mathcal{EL}^+$-TBox the same algorithm

Algorithm 1 Computation of a role-depth bounded $\mathcal{EL}^+$ k-lcs.

Procedure k-lcs $(C, D, \mathcal{T}, k)$
Input: C, D: $\mathcal{EL}^+$-concept descriptions; $\mathcal{T}$: $\mathcal{EL}^+$-TBox; k: natural number
Output: k-lcs(C, D): role-depth bounded $\mathcal{EL}^+$ k-lcs of C, D w.r.t. $\mathcal{T}$ and k

1: $\mathcal{T}' :=$ normalize$(\mathcal{T} \cup \{A \equiv C, B \equiv D\})$
2: $(V, E, S) :=$ apply-completion-rules$(\mathcal{T}')$
3: $L :=$ k-lcs-r$(A, B, (V, E, S), k)$
4: **return** remove-normalization-names(L)

Procedure k-lcs-r$(A, B, (V, E, S), k)$
Input: A, B: concept names; (V, E, S): completion graph; k: natural number
Output: k-lcs(A, B): role-depth bounded $\mathcal{EL}^+$ k-lcs of A, B w.r.t. $\mathcal{T}$ and k

1: common-names $:= S(A) \cap S(B)$
2: **if** $k = 0$ **then**
3: **return** $\bigsqcap_{P \in \text{common-names}} P$
4: **else**
5: **return** $\bigsqcap_{P \in \text{common-names}} P \sqcap \bigsqcap_{r \in N_R} \Big(\bigsqcap_{(A,r,C) \in E, (B,r,D) \in E} \exists r.\text{k-lcs-r}(C, D, (V, E, S), k-1) \Big)$

can be used as for $\mathcal{EL}$. Such an algorithm was introduced in [10] and, for the convenience of the reader, is shown in Algorithm 1 (for the binary lcs).

The idea is to introduce new concept names for the concept descriptions of interest and to apply the completion algorithm. Then, starting from the newly introduced names A and B, traverse the completion graph simultaneously. More precisely, for the tree unravelings of the completion graph of depth k for A and B, the cross product is computed. In a post-processing step concept names that were introduced by normalization have to be removed from the concept description. Obviously, this method creates heavily redundant concept descriptions.

In cases where the exact lcs exists, the algorithm computes the exact lcs, if started with a big enough k.

4 Optimizations for the k-lcs algorithm

We present two types of optimizations: to obtain succinct rewritings of $\mathcal{EL}^+$-concept descriptions and to speed-up the k-lcs computation.

4.1 Simplifying $\mathcal{EL}^+$-concept descriptions

The highly redundant $\mathcal{EL}^+$-concept descriptions obtained from the k-lcs algorithm need to be simplified, in order to make the resulting concept description readable. The general idea for the simplification is to remove those subtrees

Algorithm 2 Simplification

Procedure simplify$(C, (V, E, S), \mathcal{T})$
Input: C: $\mathcal{EL}^+$-concept description; (V, E, S): completion graph; $\mathcal{T}$: $\mathcal{EL}^+$-TBox
Output: simplify(C): simplified concept description

1: Let $C \equiv A_1 \sqcap \ldots \sqcap A_n \sqcap \exists r_1.D_1 \sqcap \ldots \sqcap \exists r_m.D_m$ with $A_i \in N_C$ for $1 \leq i \leq n$.
2: $Conj := \{A_i \mid 1 \leq i \leq n\} \cup \{\exists r_j.D_j \mid 1 \leq j \leq m\}$
3: **for all** $X \in Conj$ **do**
4: **for all** $Y \in Conj$ **do**
5: **if** $X \neq Y \wedge$ subsumes-H$(X, Y, (V, E, S), \mathcal{T})$ **then**
6: $Conj := Conj \setminus \{X\}$
7: **break**
8: **for all** $X \in Conj$ **do**
9: **if** $X \equiv \exists r_j.D_j$ **then**
10: $Conj := (Conj \setminus \{\exists r_j.D_j\}) \cup \{\exists r_j.\text{simplify}(D_j, (V, E, S), \mathcal{T})\}$
11: **return** $\bigsqcap_{X \in Conj} X$

from the syntax tree which are subsumers of any of their sibling subtrees. For a conjunction of concept names, this results in the least ones (w.r.t. $\sqsubseteq_\mathcal{T}$).

Algorithm 2 computes such simplifications of $\mathcal{EL}^+$-concept descriptions. Note, that it needs to be applied *after* the normalization names are removed. Otherwise it might remove names from the original TBox that subsume a normalization name, which gets removed later during denormalization. For the soundness of the simplification procedure simplify, it is only necessary to ensure that the procedure 'subsumes-H' is sound. However, for our purpose this procedure does not need to be complete. This might result in simplifications that are equivalent to the input concept description, but that are still redundant. Such a heuristic is displayed in Algorithm 3.

It is easy to see that the procedure subsumes-H is sound by an inspection of the different cases according to the structure of C and D. For instance, if C is a conjunction (line 11), the procedure only returns true if D is a subsumer of F_i, for all $1 \leq i \leq n$. If both C and D are existential restrictions (line 23), then the procedure returns true if $s \sqsubseteq_\mathcal{T} r$ and F is a subsumer of G, or if there is a role t with $s \circ t \sqsubseteq_\mathcal{T} r$ and $G = \ldots \sqcap \exists t.H \sqcap \ldots$ where F is a subsumer of H. In both cases we clearly have $D \sqsubseteq_\mathcal{T} C$. All other cases are proven similarly.

4.2 Speeding-up the k-lcs algorithm

A concept description C is *fully expanded* up to the role-depth k w.r.t. $\mathcal{T}$, if (1) for all $A \in N_{C,\mathcal{T}}$ with $C \sqsubseteq_\mathcal{T} A$ we have that A is a conjunct of C and (2) if $k > 0$ then for all F with $C \sqsubseteq_\mathcal{T} \exists r.F$ we have an F' with $F' \sqsubseteq_\mathcal{T} F$ s.t. $\exists r.F'$ is a conjunct of C and F' is fully expanded up to role-depth $k-1$. Now, the k-lcs-r procedure constructs result descriptions that are the fully expanded and thus heavily redundant. Most of this redundancy is then removed by the simplification procedure. For large ontologies with a deep role hierarchy, the fully expanded result may grow quite large, which causes long run-times and incomprehensible,

Algorithm 3 Subsumes-H

Procedure subsumes-H$(C, D, (V, E, S), \mathcal{T})$
Input: C, D: $\mathcal{EL}^+$-concept descriptions; (V, E, S): completion graph; $\mathcal{T}$: $\mathcal{EL}^+$-TBox
Output: whether C subsumes D w.r.t. $\mathcal{T}$

```
 1: if C ∈ N_C then
 2:    if C = ⊤ then
 3:       return true
 4:    else if D ∈ N_C then
 5:       return C ∈ S(D)
 6:    else if D ≡ F_1 ⊓ ... ⊓ F_n then
 7:       for all 1 ≤ i ≤ n do
 8:          if subsumes-H(C, F_i, (V, E, S), T) then
 9:             return true
10:    return false
11: else if C ≡ F_1 ⊓ ... ⊓ F_n then
12:    for all 1 ≤ i ≤ n do
13:       if not subsumes-H(F_i, D, (V, E, S), T) then
14:          return false
15:    return true
16: else if C ≡ ∃r.F then
17:    if D ∈ N_C then
18:       return (C, r, F) ∈ E
19:    else if D ≡ F_1 ⊓ ... ⊓ F_n then
20:       for all 1 ≤ i ≤ n do
21:          if subsumes-H(C, F_i, (V, E, S), T) then
22:             return true
23:    else if D ≡ ∃s.G then
24:       if s ⊑_T r and subsumes-H(F, G, (V, E, S), T) then
25:          return true
26:       else
27:          for all t ∈ N_R do
28:             if s ∘ t ⊑_T r, G ≡ ... ⊓ ∃t.H ⊓ ...
                and subsumes-H(F, H, (V, E, S), T) then
29:                return true
30:       return false
```

large concept descriptions. Therefore, the idea for optimization is to avoid generating fully expanded descriptions and apply some of the simplifications already during the construction of the result.

Optimization 1: avoid unnecessary role-depth. This simple optimization applies, if one of the input concepts of the k-lcs-r procedure already subsumes the other one, in which case it is the lcs of both. Therefore in k-lcs-r(A, B, S, k), if $A \in S(B)$ we can simply return A and if $B \in S(A)$ we can return B.

This is a well-known optimization for the computation of the lcs for concept descriptions without reference to a TBox [5]. However, if applied in the context of completion, this optimization interacts with the denormalization and the role-

depth bound in non-obvious ways. Consider the TBox

$$\mathcal{T} = \{A \sqsubseteq \exists r.\exists r.K, B \sqsubseteq \exists r.\exists r.K, \exists r.\exists r.K \sqsubseteq \exists s.(L \sqcap M)\}$$

which gets normalized to

$$\mathcal{T}' = \{A \sqsubseteq \exists r.X_1, X_1 \sqsubseteq \exists r.K, B \sqsubseteq \exists r.X_2, X_2 \sqsubseteq \exists r.K, \exists r.K \sqsubseteq X_3, \\ \exists r.X_3 \sqsubseteq X_4, X_4 \sqsubseteq \exists s.X_5, X_5 \sqsubseteq L, X_5 \sqsubseteq M\}.$$

k-lcs-r$(A, B, (\cdots), 1)$ would yield $X_4 \sqcap \exists r.X_3 \sqcap \exists s.(X_5 \sqcap L \sqcap M)$, which denormalizes to $\top \sqcap \exists r.\top \sqcap \exists s.(L \sqcap M)$, while the optimization would yield $X_4 \sqcap \exists r.X_3 \sqcap \exists s.X_5$, which denormalizes to $\top \sqcap \exists r.\top \sqcap \exists s.\top$, which is not the lcs anymore. Therefore, the optimization can only be applied to concept names from the original TBox, never to normalization names. This optimization and the next one are shown in Algorithm 4.

Optimization 2: avoid unnecessary branching. For this optimization consider the TBox

$$\mathcal{T}_n = \{A \sqsubseteq D \sqcap \exists r.C_1, B \sqsubseteq E \sqcap \exists r.C_1\} \cup \{C_1 \sqsubseteq C_i \mid i \in \{2 \dots n\}\}.$$

In this case, the lcs of A and B is just $\exists r.C_1$. However, the algorithm would generate the complete product set $\{C_i \mid i \in \{1, \dots, n\}\} \times \{C_i \mid i \in \{1, \dots, n\}\}$ and recursively call k-lcs-r for each pair, just to eliminate all $\exists r.C_i$ for $i > 1$ and all $\exists r.\top$ afterwards in the simplification step. Even for this simple example, the algorithm would require time quadratic in the size of the input TBox. Clearly, evaluating the C_is for $i > 1$ is not necessary, as they all subsume C_1. The same is true for role hierarchies, where for example

$$\mathcal{T}'_n = \{A \sqsubseteq D \sqcap \exists r.C_1, B \sqsubseteq E \sqcap \exists r.C_1\} \cup \{r \sqsubseteq r_i \mid i \in \{2 \dots n\}\}$$

would lead to the same unnecessary quadratic runtime.

The idea to avoid this kind of branching is to explicitly create the sets SA and SB of all role-successors $(A, r, C) \in E$ and $(B, r, C) \in E$ for input concepts A and B and to remove all role-successors which are subsumers of other role-successors in the same set. Recursive calls to k-lcs-r-o can then be made with one successor from each set SA and SB. However, we have to be careful with role-successors with different role-names. For example, for a role-successor $(r, C) \in SA$ and a role-successor $(s, D) \in SB$, a recursive call has to be made for all minimal (w.r.t. $\sqsubseteq_\mathcal{T}$) role names t with $r \sqsubseteq_\mathcal{T} t$ and $s \sqsubseteq_\mathcal{T} t$. The following lemma shows that this optimization is correct.

Lemma 1. *The results of the k-lcs-r and k-lcs-r-o procedures are equivalent.*

Proof. Let $\mathcal{T}$ be a normalized TBox, (V, E, S) be its completion graph, $A, B \in N_{C,\mathcal{T}}$, $k \in \mathbb{N}$ and $L = k$-lcs-r$(A, B, (V, E, S), k)$ and $L_o = k$-lcs-r-o$(A, B, (V, E, S), k)$. We have to show that $L \equiv_\mathcal{T} L_o$. Note, that if optimization 1 is applicable, then $A \in S(B)$ or $B \in S(A)$ and hence A (resp. B) is equivalent to the lcs. Otherwise, we show that $L \equiv_\mathcal{T} L_o$ by induction on the role-depth bound k.

Algorithm 4 Computation of a role-depth bounded $\mathcal{EL}^+$-lcs.

Procedure k-lcs-o$(C, D, \mathcal{T}, k)$
Input: C, D: $\mathcal{EL}^+$ concept descriptions; $\mathcal{T}$: $\mathcal{EL}^+$ TBox; k: natural number
Output: k-lcs(C, D): role-depth bounded $\mathcal{EL}^+$-lcs of C, D w.r.t. $\mathcal{T}$ and k

1: $\mathcal{T}' :=$ normalize$(\mathcal{T} \cup \{A \equiv C, B \equiv D\})$
2: $(V, E, S) :=$ apply-completion-rules$(\mathcal{T}')$
3: $L :=$ k-lcs-r-o$(A, B, (V, E, S), k)$
4: **if** $L \equiv_{\mathcal{T}'} A$ **then**
5: **return** C
6: **else if** $L \equiv_{\mathcal{T}'} B$ **then**
7: **return** D
8: **else**
9: **return** remove-normalization-names(L)

Procedure k-lcs-r-o$(A, B, (V, E, S), k)$
Input: A, B: concept names; (V, E, S): completion graph; k: natural number
Output: k-lcs(A, B): role-depth bounded $\mathcal{EL}^+$-lcs of A, B w.r.t. $\mathcal{T}$ and k

1: **if** $A \in S(B)$ and A is not a normalization name **then**
2: **return** A
3: **else if** $B \in S(A)$ and B is not a normalization name **then**
4: **return** B
5: common-names $:= S(A) \cap S(B)$
6: SA := remove-redundant$(\{(r, C) \mid (A, r, C) \in E\})$
7: SB := remove-redundant$(\{(s, D) \mid (B, s, D) \in E\})$
8: **if** $k = 0$ **then**
9: **return** $\prod_{P \in \text{common-names}} P$
10: **else**
11: **return** $\prod_{P \in \text{common-names}} P \sqcap \prod_{\substack{(r,C) \in SA,\ (s,D) \in SB,\\ t \in N_R \text{ minimal with } r \sqsubseteq_{\mathcal{T}} t \wedge s \sqsubseteq_{\mathcal{T}} t}} \exists\, t.k\text{-lcs-r-o}(C, D, (V, E, S), \text{k-1})$

Procedure remove-redundant(S)
Input: S: set of role-successors
Output: remove-redundant(S): simplified set

1: **for all** $(r, C) \in S$ **do**
2: **for all** $(s, D) \in S$ **do**
3: **if** $(r, C) \neq (s, D)$ and $r \sqsubseteq_{\mathcal{T}} s$ and $D \in S(C)$ **then**
4: $S := S \setminus \{(s, D)\}$
5: **return** S

For $k = 0$ both procedures return the same concept description, i.e., $L = L_o$.
For $k > 0$, we first show that for each $r \in N_{R,\mathcal{T}}$ and all $(A, r, C) \in E$, $(B, r, D) \in E$, we have $L_o \sqsubseteq_{\mathcal{T}} \exists r.k$-lcs-r$(C, D, (V, E, S), k-1)$. If $(A, r, C) \in E$, then there exists $(s_1, C') \in SA$ with $s_1 \sqsubseteq_{\mathcal{T}} r$ and $C \in S(C')$. Similarly, there exists $(s_2, D') \in SB$ with $s_2 \sqsubseteq_{\mathcal{T}} r$ and $D \in S(D')$. Then $\exists t.k$-lcs-r-o$(C', D', (V, E, S), k-1)$ is a conjunct of L_o for all minimal $t \in N_{R,\mathcal{T}}$ with $s_1 \sqsubseteq_{\mathcal{T}} t$ and $s_2 \sqsubseteq_{\mathcal{T}} t$. Since $s_1 \sqsubseteq_{\mathcal{T}} r$ and $s_2 \sqsubseteq_{\mathcal{T}} r$, there is at least one minimal $t_0 \in N_{R,\mathcal{T}}$ with $t_0 \sqsubseteq r$

for which $\exists t_0.k$-lcs-r-o$(C', D', (V, E, S), k-1)$ is conjunct of L_o. Together with k-lcs-r-o$(C', D', (V, E, S), k-1) \equiv_{\mathcal{T}} k$-lcs-r$(C', D', (V, E, S), k-1)$ by the induction hypothesis and k-lcs-r$(C', D', (V, E, S), k-1) \sqsubseteq_{\mathcal{T}} k$-lcs-r$(C, D, (V, E, S), k-1)$ for $C' \sqsubseteq_{\mathcal{T}} C$ and $D' \sqsubseteq_{\mathcal{T}} D$ we have $\exists t_0.k$-lcs-r-o$(C', D', (V, E, S), k-1) \sqsubseteq_{\mathcal{T}} \exists r.k$-lcs-r$(C, D, (V, E, S), k-1)$. Since $L_o \sqsubseteq_{\mathcal{T}} \exists r.k$-lcs-r$(C, D, (V, E, S), k-1)$ for each $r \in N_{R,\mathcal{T}}$ and all edges $(A, r, C) \in E$, $(B, r, D) \in E$ and also $L_o \sqsubseteq_{\mathcal{T}} C$ for $C \in$ common-names, we have $L_o \sqsubseteq_{\mathcal{T}} L$. Obviously, k-lcs-r computes all of the recursive concept descriptions (and possibly more) that k-lcs-r-o computes and hence $L \sqsubseteq_{\mathcal{T}} L_o$. Thus $L \equiv_{\mathcal{T}} L_o$.

Obviously, both optimizations also yield shorter, but equivalent concept descriptions than the naive algorithm.

5 Implementation and Evaluation

We implemented the role-depth bounded lcs for $\mathcal{EL}^+$ in the new version of our Protégé plug-in GEL[1]. This program uses the reasoner jCEL[2], which implements the completion-based classification algorithms for $\mathcal{EL}^+$ (and $\mathcal{ELHI}f_{\mathcal{R}^+}$). The earlier version of GEL implemented this reasoning service for $\mathcal{EL}$ [10, 9].

GEL processes concept descriptions and ontologies in OWL API format. According to Algorithm 4, it first adds new concept definitions for the input $\mathcal{EL}^+$-concept descriptions to the ontology and then uses jCEL to apply the normalization and the completion rules. After all completion sets are computed, the GEL-processor applies the recursive k-lcs-r-o procedure. The resulting k-lcs is then (possibly simplified,) translated back to OWL API format and returned. Although Algorithm 4 is formulated for the binary k-lcs, the implementation can handle input tuples of concept descriptions by applying the binary k-lcs to the first two concepts and then successively computing the k-lcs of the result with the next concept in the tuple. In previous tests we found this to be faster then computing the n-ary k-lcs directly.

Our implementation includes a Protégé plug-in with a Protégé ontology view, i.e., a component where the user can input concept descriptions and select processing options such as the role-depth bound and whether to apply the simplification or optimizations. Figure 1 shows a screen-shot of the GEL plug-in.

5.1 Evaluation

Test data. The real-world ontologies used in our tests were the Gene Ontology [6] and Not-Galen, a version of the Galen ontology [11] pruned to $\mathcal{EL}^+$. As most concepts in these ontologies are unrelated, we selected around 50 different tuples of named concepts tuples by hand, taking concepts which were classified as sibling concepts and had similar exisential restrictions. These turned out to be the most interesting cases for testing. We computed the k-lcs for these ergeben nicht top!

[1] http://sourceforge.net/p/gen-el
[2] http://jcel.sourceforge.net

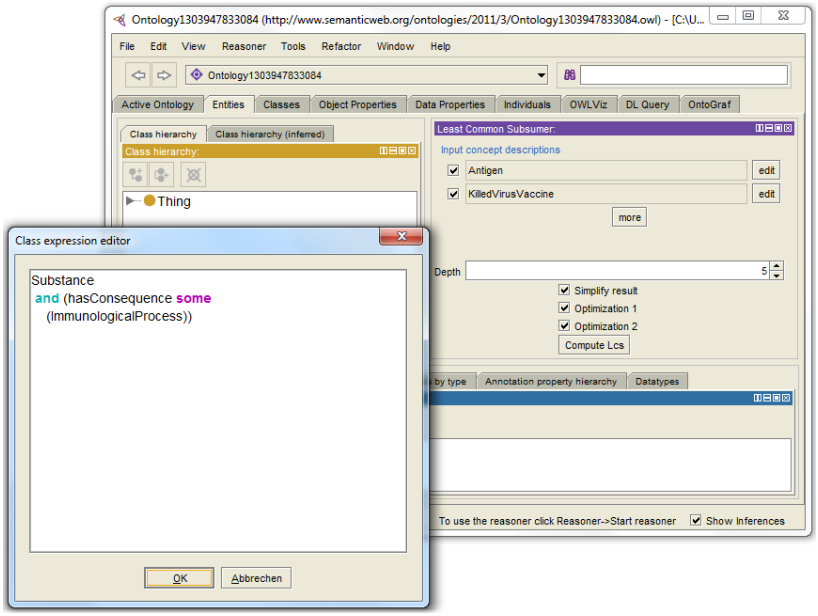

Fig. 1. Screen-shot of the PROTÉGÉ plug-in GEL

input concept descriptions with various role-depth bounds both with and without optimizations.

Evaluation. In the worst case the role-depth bounded lcs can have a size that is exponential in the role-depth bound k. However, it largely depends on the ontology whether such worst-case behavior occurs. For the Gene Ontology, the role-depth bounded lcs was always constructed and simplified almost instantly. The runtime was totally dominated by the classification time by JCEL.

For NOT-GALEN some input concept pairs resulted in long runtimes, which were mostly dominated by the runtime of the k-lcs-r-procedure and not by classification. Figure 2 shows the average k-lcs-r-construction runtime of GEL on various input pairs for different values of k. Figure 2 also shows the effect of the optimizations on the construction time. Optimization 1, which returns one of the input concepts if it subsumes the other, is able to cut-off the k-lcs-r-o recursion before the maximum role-depth is reached. This improves the runtime by a factor of around 2 compared to the basic k-lcs-r procedure with no optimizations. Optimization 2, which removes redundant successor nodes before product construction, reduces the branching factor of the recursion. In our tests, it yielded even better runtime improvements than Optimization 1—on average by factor 30. Each optimization yields a larger speed-up for increasing role-depth bounds. Combining the optimizations yielded the best runtime for most cases, which indicates independance of the optimizations.

In practice, the runtime with only optimization 1 (or no optimization) was sometimes too long to be useful. For some input concepts, like *PepticUlcer* and

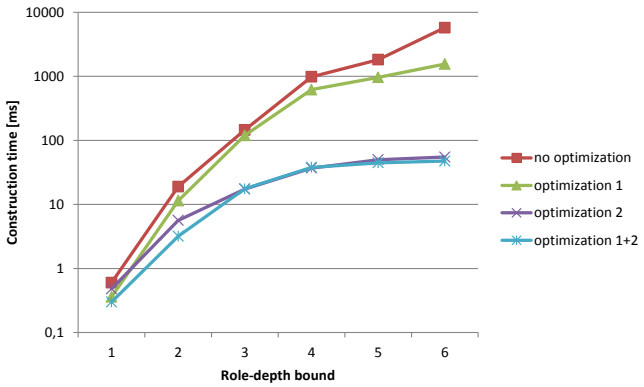

Fig. 2. Average k-lcs-r-construction time for concept descriptions from NOT-GALEN

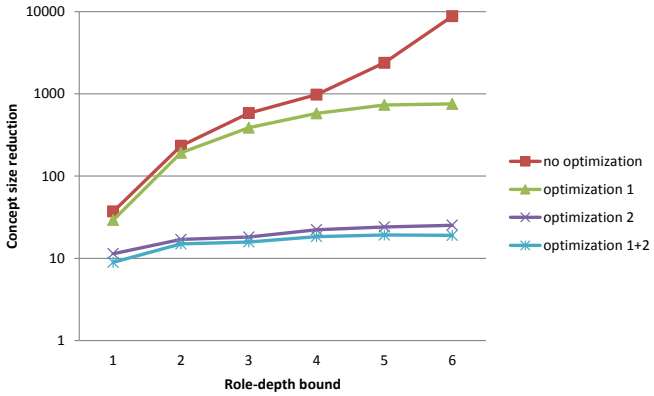

Fig. 3. Average size-reduction by simplification for k-lcses computed w.r.t. NOT-GALEN

AreaOfAtrophicGastritis, it did not return a result within an hour for a role-depth bound for as low as 4. However, with both optimizations enabled, the result for a role-depth bound of 4 was computed in 1.9 seconds (with a classification time of 330ms). The reason that optimization 2 is so effective on NOT-GALEN is its a deep role hierarchy. Without the optimization for each role also all subsuming roles would generate a recursive k-lcs-r call, which yields large branching factors.

The runtime for the simplification is proportional to the size of the concept description before simplification (which is roughly proportional to the runtime of the k-lcs-r-construction, with a little bit of overhead for the second optimization). However, Figure 3 shows that simplification reduces the concept sizes dramatically – even with both optimizations enabled the size of the result is still reduced by factor 16 on average. Of course, since both optimizations apply simplification steps during the construction of the k-lcs, the size-reduction for results computed without optimizations would be even larger.

6 Conclusions

In this paper we have presented highly effective optimization methods for the computation of approximative solutions for least common subsumers w.r.t. general TBoxes written in $\mathcal{EL}^+$. Our optimization methods address two aspects of the role-depth bounded lcs. On the one hand we devised optimization methods for speeding-up the computation and on the other, we devised a simplification procedure that yields smaller, but equivalent rewritings of $\mathcal{EL}^+$-concept descriptions. This procedure turned out to be extremely helpful, when reducing the concept size. For the NOT-GALEN ontology the result concepts were reduced by several orders of magnitude. With the here devised methods a large part of the constructors available in the OWL 2 EL profile can be covered when computing generalizations.

References

1. F. Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI-03)*. Morgan Kaufm., 2003.
2. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, 2005.
3. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope further. In *In Proc. of the OWLED Workshop*, 2008.
4. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
5. F. Baader and A.-Y. Turhan. On the problem of computing small representations of least common subsumers. In *Proc. of the 25th German Annual Conf. on Artificial Intelligence (KI'02)*, vol. 2479 of *LNAI*. Springer, 2002.
6. The Gene Ontology Consortium. Gene Ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.
7. A. Ecke and A.-Y. Turhan. Role-depth bounded least common subsumers for $\mathcal{EL}^+$ and $\mathcal{ELI}$. Submitted to DL'12.
8. K. Janowicz. *Computing Semantic Similarity Among Geographic Feature Types Represented in Expressive Description Logics*. PhD thesis, Instit. f. Geoinformatics, Univ. of Münster, Germany, 2008.
9. J. Mendez, A. Ecke, and A.-Y. Turhan. Implementing completion-based inferences for the $\mathcal{EL}$-family. In *Proc. of the 2011 Description Logic Workshop (DL'11)*, 2011.
10. R. Peñaloza and A.-Y. Turhan. A practical approach for computing generalization inferences in $\mathcal{EL}$. In *Proc. of the 8th European Semantic Web Conf. (ESWC'11)*, LNCS. Springer, 2011.
11. A. Rector and I. Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proc. of the WS on Ontological Engineering, AAAI Spring Symp. (AAAI'97)*, 1997.
12. C. Rosse and J. L. V. Mejino Jr. A reference ontology for biomedical informatics: the foundational model of anatomy. *Journal of Biomedical Informatics*, 36(6).
13. K. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with snomed-rt. *Journal of the American Medical Informatics Assoc.*, 2000. Fall Symposium Special Issue.