On the problem of computing small representations of least common subsumers

Franz Baader and Anni-Yasmin Turhan

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

Abstract. For Description Logics with existential restrictions, the size of the least common subsumer (lcs) of concept descriptions may grow exponentially in the size of the input descriptions. The first (negative) result presented in this paper is that it is in general not possible to express the exponentially large concept description representing the lcs in a more compact way by using an appropriate (acyclic) terminology.
In practice, a second and often more severe cause of complexity was the fact that concept descriptions containing concepts defined in a terminology must first be unfolded (by replacing defined names by their definition) before the known lcs algorithms could be applied. To overcome this problem, we present a modified lcs algorithm that performs lazy unfolding, and show that this algorithm works well in practice.

1 Introduction

In an application in chemical process engineering [5, 11], we support the bottom-up construction of Description Logic (DL) knowledge bases by computing most specific concepts (msc) of individuals and least common subsumers (lcs) of concepts: instead of directly defining a new concept, the knowledge engineer introduces several typical examples as individuals, which are then generalized into a concept description by using the msc and the lcs operation [1, 2, 9]. This description is offered to the knowledge engineer as a possible candidate for a definition of the concept.

Unfortunately, due to the nature of the algorithm for computing the lcs proposed in [2], this algorithm yields concept descriptions that do not contain defined concept names, even if the descriptions of the individuals use concepts defined in a terminology (TBox) $\mathcal{T}$. In addition, due to the inherent complexity of the lcs operation, these descriptions may be quite large (exponentially large in the size of the unfolded input descriptions). For the small DL $\mathcal{EL}$, which allows for conjunctions, existential restrictions, and the top concept, the binary lcs operation is still polynomial, but the n-ary one is already exponential. For the DL $\mathcal{ALE}$, which extends $\mathcal{EL}$ by value restrictions, primitive negation, and the bottom concept, already the binary lcs is exponential in the worst case.

To overcome the problem of large least common subsumers, we have employed rewriting of the computed concept description using the TBox $\mathcal{T}$ in order to obtain a shorter and better readable description [3]. Informally, the problem of rewriting a concept given a terminology can be stated as follows: given a TBox $\mathcal{T}$ and a concept description C that does not contain concept names defined in $\mathcal{T}$, can this description be rewritten into an equivalent smaller description D by using (some of) the names defined in $\mathcal{T}$? First results obtained in our process engineering application were quite encouraging: for a TBox with about 65 defined and 55 primitive names, source descriptions of size about 800 (obtained as results of the lcs computation) were rewritten into descriptions of size about 10 [3].

This paper complements these results in two ways. First, the positive empirical results for the rewriting approach led us to conjecture that maybe TBoxes can always be used to yield a compact representation of the lcs. More formally, our conjecture can be stated as follows. Let $\mathcal{L}$ be a DL for which the lcs operation (binary or n-ary) is exponential (like $\mathcal{EL}$ or $\mathcal{ALE}$). Given input descriptions $C_1, \ldots, C_n$ with lcs D, does there always exist a TBox $\mathcal{T}$ whose size is polynomial in the size of $C_1, \ldots, C_n$ and a defined concept name A in $\mathcal{T}$ such that $A \equiv_{\mathcal{T}} D$, i.e., the TBox defines A such that it is equivalent to the lcs D of $C_1, \ldots, C_n$? A closer look at the worst-case examples for $\mathcal{EL}$ and $\mathcal{ALE}$ from [2] supported this conjecture: the exponentially large least common subsumers constructed there can easily be represented using polynomially large TBoxes. Nevertheless, we will show in Section 4 that the conjecture is false, both for $\mathcal{EL}$ and $\mathcal{ALE}$.

Second, we modify the lcs algorithm presented in [2] such that it works on concept descriptions still containing names defined in a TBox. The idea is that unfolding is not performed a priori, but only if necessary. This technique, called *lazy unfolding*, is a common optimization technique for standard inferences such as subsumption [7, 8], but was until now not employed for non-standard inferences like computing the lcs. Though the lcs computed by this modified algorithm may contain defined concept names, it turned out that rewriting can still reduce the size of the description. However, since it already starts with smaller descriptions, the rewriting step takes less time than with the unmodified algorithm.

2 Preliminaries

First, we introduce the DLs $\mathcal{EL}$ and $\mathcal{ALE}$ in more detail. *Concept descriptions* are inductively defined using a set of *constructors*, starting with a set N_C of *concept names* and a set N_R of *role names*. The constructors determine the expressive power of the DL. In this paper, we consider concept descriptions built from the constructors shown in Table 1. In $\mathcal{EL}$, concept descriptions are formed using the constructors top concept ($\top$), conjunction ($C \sqcap D$) and existential restriction ($\exists r.C$). The DL $\mathcal{ALE}$ provides all the constructors introduced in Table 1.

The semantics of a concept description is defined in terms of an *interpretation* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$. The domain Δ of $\mathcal{I}$ is a non-empty set of individuals and the interpretation function $\cdot^{\mathcal{I}}$ maps each concept name $P \in N_C$ to a set $P^{\mathcal{I}} \subseteq \Delta$ and each role name $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta \times \Delta$. The extension of $\cdot^{\mathcal{I}}$ to arbitrary concept descriptions is inductively defined, as shown in Table 1.

A *TBox* is a finite set of concept definitions of the form $A \doteq C$, where A is a concept name and C a concept description. In addition, we require that TBoxes

Constructor name	Syntax	Semantics
primitive concept, $P \in N_C$	P	$P^{\mathcal{I}} \subseteq \Delta$
top-concept	$\top$	Δ
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction for $r \in N_R$	$\exists r.C$	$\{x \in \Delta \mid \exists y : (x, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction for $r \in N_R$	$\forall r.C$	$\{x \in \Delta \mid \forall y : (x, y) \in r^{\mathcal{I}} \to y \in C^{\mathcal{I}}\}$
primitive negation, $P \in N_C$	$\neg P$	$\Delta \setminus P^{\mathcal{I}}$
bottom-concept	$\bot$	$\emptyset$

Table 1. Syntax and semantics of concept descriptions.

are acyclic and do not contain multiple definitions (see, e.g., [10]). Concept names occurring on the left-hand side of a definition are called *defined concepts*. All other concept names are called *primitive concepts*. In TBoxes of the DL $\mathcal{ALE}$, negation may only be applied to primitive concepts. An interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$ iff it satisfies all its concept definitions, i.e., $A^{\mathcal{I}} = C^{\mathcal{I}}$ for all definitions $A \doteq C$ in $\mathcal{T}$.

One of the most important traditional inference services provided by DL systems is computing the subsumption hierarchy. The concept description C is *subsumed* by the description D w.r.t. the TBox $\mathcal{T}$ ($C \sqsubseteq_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{T}$. The description C is *subsumed* by D ($C \sqsubseteq D$) iff it is subsumed by D w.r.t. the empty TBox (which has all interpretations as models). The concept descriptions C and D are *equivalent* (w.r.t. $\mathcal{T}$) iff they subsume each other (w.r.t. $\mathcal{T}$). We write $C \equiv_{\mathcal{T}} D$ if C and D are *equivalent* w.r.t. $\mathcal{T}$.

In this paper, we are interested in the non-standard inference task of computing the least common subsumer of concept descriptions.

Definition 1 (Least Common Subsumer). *Let $C_1, \ldots, C_n$ be concept descriptions in a DL $\mathcal{L}$. The $\mathcal{L}$-concept description C is a* least common subsumer (lcs) *of $C_1, \ldots, C_n$ in $\mathcal{L}$ (for short $C = \mathsf{LCS}_{\mathcal{L}}(C_1, \ldots, C_n)$) iff*

1. *$C_i \sqsubseteq C$ for all $0 \leq i \leq n$, and*
2. *C is the least concept description with this property, i.e., if D is a concept description satisfying $C_i \sqsubseteq D$ for all $1 \leq i \leq n$, then $C \sqsubseteq D$.*

This definition can naturally be extended to concept definitions containing names defined in a TBox $\mathcal{T}$: simply replace subsumption by subsumption w.r.t. $\mathcal{T}$.

In general (i.e., for an arbitrary DL $\mathcal{L}$), a given collection of n concept descriptions need not have an lcs. However, if an lcs exists, then it is unique up to equivalence. This justifies to talk about *the* lcs of $C_1, \ldots, C_n$ in $\mathcal{L}$.

3 Computing the Least Common Subsumer

In [2] it was shown that, for the DLs $\mathcal{EL}$ and $\mathcal{ALE}$, the lcs always exists. The main idea underlying the algorithms computing the lcs is that concept descriptions are transformed into so-called description trees. Since subsumption can be characterized through the existence of homomorphisms between description trees, the lcs operation corresponds to the product of description trees (see [2]). Because of space limitations, we can only outline the lcs algorithm introduced in [2]. The basic algorithm for computing the lcs of two $\mathcal{EL}$- or $\mathcal{ALE}$-concept descriptions w.r.t. a TBox consists of the following steps:

1. *Unfold the input descriptions:* if the input concept descriptions contain concept names defined in the TBox $\mathcal{T}$, these concept names are recursively replaced by their definitions until no defined names remain in the descriptions. It is well-known that the process of unfolding a description may cause an exponential blow-up [10].
2. *Normalize the input descriptions:* the normal form is computed by removing concept descriptions equivalent to $\top$, replacing inconsistent concept descriptions by $\bot$, joining value restrictions for the same role, and propagating value restrictions into existential restrictions on all role-levels. This last step of the normalization (which is only relevant for $\mathcal{ALE}$) is yet another source of an exponential blow-up [2, 6].
3. *Transform the normalized descriptions into description trees and compute their product:* basically, the description tree of a normalized description is just its syntax tree. The product of the description trees can then be translated back into a concept description, which is the lcs of the input descriptions w.r.t. $\mathcal{T}$. The product construction is explained in the next subsection.

It should be noted that each of the three steps of the lcs algorithm traverses the whole structure of the concept description as obtained in the step before recursively. The basic lcs algorithm is given as a binary operation since the n-ary lcs can be reduced to the binary operation using the fact that $\mathsf{LCS}(C_1, \ldots, C_n) = \mathsf{LCS}(C_1, \mathsf{LCS}(C_2, \ldots, C_n))$. Of course, one can also directly treat the n-ary lcs by using the n-ary product of description trees. We will illustrate the lcs algorithms for $\mathcal{EL}$ and $\mathcal{ALE}$ on two examples, which are the worst-case examples demonstrating that the n-ary lcs in $\mathcal{EL}$ and the binary lcs in $\mathcal{ALE}$ may lead to exponentially large concept descriptions (even without TBox).

3.1 The Least Common Subsumer in $\mathcal{EL}$

For the DL $\mathcal{EL}$, a *description tree* is merely a graphical representation of the syntax of the concept description. Its nodes are labeled with sets of concept names (corresponding to possibly negated concept names occurring in the description) and its edges are labeled with role names (corresponding to the existential restrictions occurring in the description). We call a node w reachable from a node v by an edge labeled with r an r-*successor* of v.

For example, the trees depicted in the upper half of Figure 1 were obtained from the concept descriptions

$$
\begin{aligned}
C_1^3 &:= \exists r.(P \sqcap \exists r.(P \sqcap Q \sqcap \exists r.(P \sqcap Q))) \ \sqcap\ \exists r.(Q \sqcap \exists r.(P \sqcap Q \sqcap \exists r.(P \sqcap Q))),\\
C_2^3 &:= \exists r.(P \sqcap Q \sqcap \exists r.(P \sqcap \exists r.(P \sqcap Q)) \sqcap \exists r.(Q \sqcap \exists r.(P \sqcap Q))),\\
C_3^3 &:= \exists r.(P \sqcap Q \sqcap \exists r.(P \sqcap Q \sqcap \exists r.P \sqcap \exists r.Q))).
\end{aligned}
$$

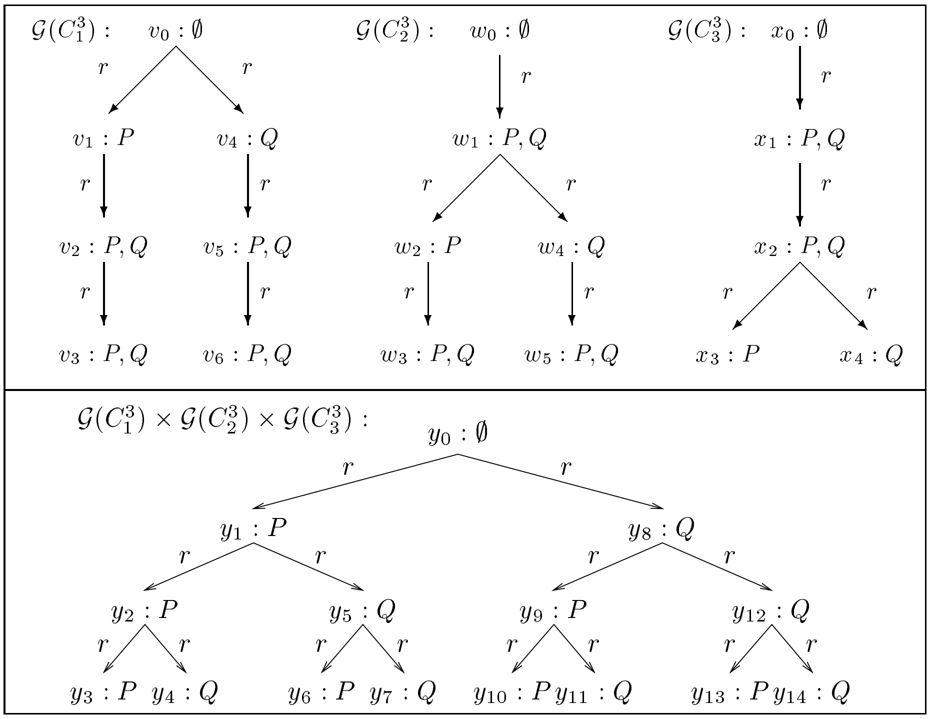

Fig. 1. Description trees of C_1^3, C_2^3, C_3^3 and their product.

The *product* $\mathcal{G}_1 \times \cdots \times \mathcal{G}_n$ of n $\mathcal{EL}$-description trees $\mathcal{G}_1, \ldots, \mathcal{G}_n$ is defined by induction on the depth of the trees. Let $v_{0,1}, \ldots, v_{0,n}$ respectively be the roots of the trees $\mathcal{G}_1, \ldots, \mathcal{G}_n$ with labels $\ell_1(v_{0,1}), \ldots, \ell_n(v_{0,n})$. Then the product $\mathcal{G}_1 \times \cdots \times \mathcal{G}_n$ has the root $(v_{0,1}, \ldots, v_{0,n})$ with label $\ell_1(v_{0,1}) \cap \ldots \cap \ell_n(v_{0,n})$. For each role r and for each n-tuple $v_1, \ldots, v_n$ of r-successors of $v_{0,1}, \ldots, v_{0,n}$, the root $(v_{0,1}, \ldots, v_{0,n})$ has an r-successor $(v_1, \ldots, v_n)$, which is the root of the product of the subtrees of $\mathcal{G}_1, \ldots, \mathcal{G}_n$ with roots $v_1, \ldots, v_n$. The lower half of Figure 1 depicts the tree obtained as the product of the description trees corresponding to the descriptions C_1^3, C_2^3, C_3^3. This tree is a full binary tree of depth 3, where the nodes reached by going to the left are labeled with P and the ones reached by going to the right are labeled with Q.

This example can be generalized to an example demonstrating that the lcs of n $\mathcal{EL}$-concept descriptions of size linear in n may be exponential in n [2].

Example 1. We define for each $n \geq 1$ a sequence $\{C_1^n, \ldots, C_n^n\}$ of $\mathcal{EL}$-concept descriptions. For $n \geq 0$ let

$$D_n := \begin{cases} \top, & n = 0 \\ \exists r.(P \sqcap Q \sqcap D_{n-1}), & n > 0 \end{cases}$$

and for $n \geq 1$ and $1 \leq i \leq n$ we define

$$C_i^n := \begin{cases} \exists r.(P \sqcap D_{n-1}) \sqcap \exists r.(Q \sqcap D_{n-1}), & i = 1 \\ \exists r.(P \sqcap Q \sqcap C_{i-1}^{n-1}), & 1 < i \leq n. \end{cases}$$

It is easy to see that each C_i^n is linear in the size of n. The product of the corresponding description trees is a full binary tree of depth n, where the nodes reached by going to the left are labeled with P and the ones reached by going to the right are labeled with Q. Obviously, the size of this tree is exponential in n. What is less obvious, but can also be shown (see [2], is that there is no smaller description tree representing the same concept (modulo equivalence).

3.2 The Least Common Subsumer in $\mathcal{ALE}$

$\mathcal{ALE}$*-description trees* are very similar to $\mathcal{EL}$-description trees. The value restrictions just lead to another type of edges, which are labeled by $\forall r$ instead of simply r. However, the unfolded concept descriptions must first be normalized before they can be transformed into description trees. On the one hand, there are normalization rules dealing with negation and the bottom concept. Here we will ignore them since neither negation nor bottom is used in our examples. On the other hand, there are normalization rules dealing with value restrictions and their interaction with existential restrictions:

$$\forall r.E \sqcap \forall r.F \longrightarrow \forall r.(E \sqcap F),$$
$$\forall r.E \sqcap \exists r.F \longrightarrow \forall r.E \sqcap \exists r.(E \sqcap F).$$

The first rule conjoins all value restrictions for the same role into a single value restriction. The second rule is problematic since it duplicates subterms, and thus may lead to an exponential blow-up of the description. The following is a well-known example that demonstrates this effect.

Example 2. We define the following sequence $C_1, C_2, C_3, \ldots$ of $\mathcal{ALE}$-concept descriptions:

$$C_n := \begin{cases} \exists r.P \ \sqcap \ \exists r.Q, & n = 1 \\ \exists r.P \ \sqcap \ \exists r.Q \ \sqcap \ \forall r.C_{n-1}, & n > 1. \end{cases}$$

Obviously, the size of C_n is linear in n. However, applying the second normalization rule to C_n yields a description of size exponential in n. If one ignores the value restrictions (and everything occurring below a value restriction), then the description tree corresponding to the normal form of C_n is again a full binary tree of depth n, where the nodes reached by going to the left are labeled with P and the ones reached by going to the right are labeled with Q. Figure 2 shows the $\mathcal{ALE}$-description tree of the normal form of C_3.

Given the description trees of normalized $\mathcal{ALE}$-concept descriptions, one can again obtain the lcs as the product of these trees. In this product, the bottom concept requires a special treatment, but we ignore this issue since it is irrelevant for our examples.

For each tuple of nodes on the same role-level, existential restrictions and value restrictions are treated symmetrically, i.e., for a role r the r-successors are combined with r-successors in all possible combinations (as before) and the

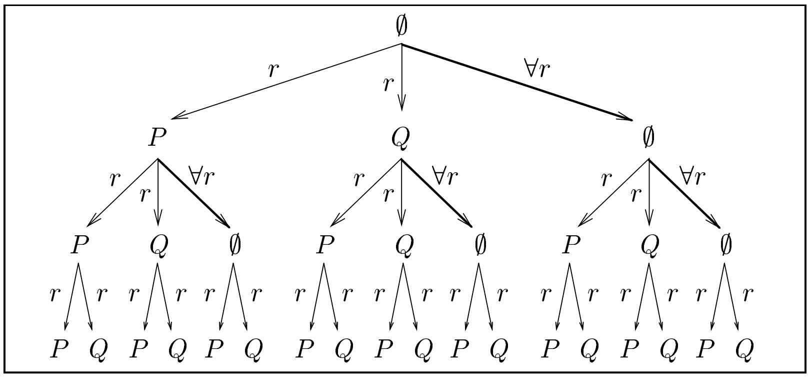

Fig. 2. The $\mathcal{ALE}$-description tree of the normal form of C_3 from Example 2.

(unique) $\forall r$-successors are combined with each other. Note that r-successors are not combined with $\forall r$-successors. The following example is taken from [2].

Example 3. For $n \geq 1$, we consider the concept descriptions C_n introduced in Example 2 and the concept descriptions D_n defined in Example 1. By building the product of the description trees corresponding to the normal forms of C_n and D_n, one basically removes the value restrictions from the normal form of C_n. Thus, one ends up with an lcs E_n that agrees with the binary tree we obtained in Example 1. Again, it can be shown that there is no smaller $\mathcal{ALE}$-concept description equivalent to this lcs.

The lcs computed by the basic algorithm is a concept description not containing names defined in the underlying TBox. If some "parts" of this description have been given names in the TBox, they can be replaced by these names, thus reducing the size of the description. This can be achieved through rewriting [3]. In the next section we show that, though rewriting may be quite effective in some examples (see [3]), it cannot always reduce the size of the lcs.

4 Using TBoxes to Compress the lcs

The exponentially large lcs E_n constructed in Examples 1 and 3 has as its description tree the full binary tree of depth n, where the nodes reached by going to the left were labeled with P and the ones reached by going to the right were labeled with Q. This concept can be defined in a TBox of size linear in n.

Example 4. Consider the following TBox $\mathcal{T}_n$:

$$\begin{array}{l}\{A_1 \doteq \exists r.P \sqcap \exists r.Q\} \;\cup \\ \{A_i \doteq \exists r.(P \sqcap A_{i-1}) \sqcap \exists r.(Q \sqcap A_{i-1}) \mid 1 < i \leq n\}.\end{array}$$

It is easy to see that the size of $\mathcal{T}_n$ is linear in n and that $A_n \equiv_{\mathcal{T}_n} E_n$, i.e., the TBox $\mathcal{T}_n$ provides us with a compact representation of E_n.

In general, however, such a compact representation by structure sharing is not possible. We will first give a counterexample for the n-ary lcs in $\mathcal{EL}$, and then for the binary lcs in $\mathcal{ALE}$. The main idea underlying both counterexamples is to generate description trees having exponentially many leaves that are all labeled by sets of concept names that are incomparable w.r.t. set inclusion. To this purpose, we consider the set of concept names $N_n := \{A_j^0, A_j^1 \mid 1 \leq j \leq n\}$, and define $A^{\mathbf{i}} := A_1^{i_1} \sqcap \ldots \sqcap A_n^{i_n}$ for each n-tuple $\mathbf{i} = (i_1, \ldots, i_n) \in \{0,1\}^n$.

4.1 The Counterexample for $\mathcal{EL}$

For all $n \geq 1$ we define a sequence $C_1, \ldots, C_n$ of n $\mathcal{EL}$-concept descriptions whose size is linear in n:

$$C_j := \exists r. \bigsqcap_{B \in N_n \setminus \{A_j^0\}} B \quad \sqcap \quad \exists r. \bigsqcap_{B \in N_n \setminus \{A_j^1\}} B.$$

Since each of the concepts C_j contains two existential restrictions, the lcs of $C_1, \ldots, C_n$ contains 2^n existential restrictions. The concept descriptions occurring under these restrictions are obtained by intersecting the corresponding concept descriptions under the existential restrictions of the concept descriptions C_j. It is easy to see that these are exactly the 2^n concept descriptions $A^{\mathbf{i}}$ for $\mathbf{i} \in \{0,1\}^n$ introduced above. Since the descriptions $A^{\mathbf{i}}$ are pairwise incomparable w.r.t. subsumption, it is clear that there is no smaller $\mathcal{EL}$-concept description equivalent to this lcs. We show now that a TBox cannot be used to obtain a smaller representation.

Recall that acyclic TBoxes can be unfolded by replacing defined names by their definitions until no more defined names occur on the right-hand sides [10]. If the defined name A represents the lcs of $C_1, \ldots, C_n$ w.r.t. a TBox, then the description defining A in the unfolded TBox is equivalent to this lcs.

Obviously, to get a more compact representation of the lcs using a TBox, one needs duplication of concept names on the right-hand sides of the TBox. During unfolding of the TBox, this would, however, lead to duplication of subconcepts. Since the (description tree of the) lcs we have constructed here has 2^n different leaves, such duplication does not help, since it can only duplicate leaves with the same label, but not generate leaves with different labels. Thus, in general, we cannot represent the lcs in a more compact way by introducing new definitions in an $\mathcal{EL}$ TBox.

4.2 The Counterexample for $\mathcal{ALE}$

For $n \geq 1$ we define concept descriptions C_n of size quadratic in n. For $n \geq 1$, let $F_j^i := \forall r. \cdots \forall r. A_{j+1}^i$ be the concept description consisting of j nested value restrictions followed by the concept name A_{j+1}^i. We define

$$\begin{aligned} C_1 &:= \exists r. A_1^0 \sqcap \exists r. A_1^1, \\ C_n &:= \exists r. F_{n-1}^0 \ \sqcap \ \exists r. F_{n-1}^1 \ \sqcap \ \forall r. C_{n-1} \qquad \text{for } n > 1. \end{aligned}$$

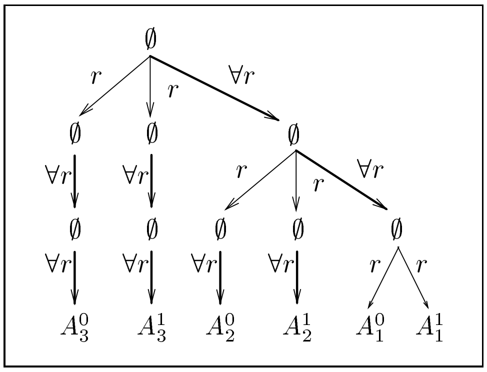

Fig. 3. The $\mathcal{ALE}$-description tree corresponding to C_3.

Figure 3 shows the description tree corresponding to C_3.

Applying the normalization rule $\forall r.E \sqcap \exists r.F \longrightarrow \forall r.E \sqcap \exists r.(E \sqcap F)$ to C_n yields a normalized concept description whose size is exponential in n. If one ignores the value restrictions (and everything occurring below them), then the description tree corresponding to this normal form of C_n is a full binary tree of depth n whose 2^n leaves are labeled by the 2^n concept descriptions $A^{\mathbf{i}}$ for $\mathbf{i} \in \{0,1\}^n$.

Let $D_n := \exists r. \cdots \exists r. \bigsqcap_{B \in N_n} B$ be the concept description consisting of n nested existential restrictions followed by the conjunction of all concept names in N_n. Again, by building the product of the description trees corresponding to the normal forms of C_n and D_n, one basically removes the value restrictions from the normal form of C_n. Thus, the lcs corresponds to the full binary tree of depth n whose leaves are labeled by the concept descriptions $A^{\mathbf{i}}$ for $\mathbf{i} \in \{0,1\}^n$.

By an argument similar to the one for $\mathcal{EL}$ one can show that there is no smaller $\mathcal{ALE}$-concept description equivalent to this lcs, and that a TBox cannot be used to obtain a smaller representation.

The examples given above show that the exponential size of the lcs cannot be avoided by employing structure sharing (i.e., replacing common substructures by a defined name). In practice, however, the complexity of unfolding concept descriptions before applying the lcs algorithm appears to be more problematic than this inherent complexity of the lcs operation. How to avoid this unfolding step is the topic of the next section.

5 Computing the lcs using Lazy Unfolding

Recall from the beginning of Section 3 that the computation of an lcs is realized by three consecutive traversals of the concept descriptions: unfolding, normalization, and construction of the product. The first two steps may cause an exponential blow-up of the descriptions which are in turn the input for the next step, whereas the third step is polynomial for the binary lcs operation.

Before illustrating the shortcomings of the basic lcs algorithm by an example, let us formally define the size of a concept. The *size* $|C|$ of a concept description C is increased by 1 for each occurrence of a role name or a concept name in C (with $|\top| = |\bot| = 0$).

Example 5 (naïve lcs algorithm). Given the following TBox

$$\begin{array}{llll} \mathcal{T} := \{ & C_1 \doteq A_1 \sqcap \exists r.D_1, & & C_2 \doteq A_2 \sqcap \exists r.D_1, \\ & D_1 \doteq (\forall s.B_1) \sqcap (\exists s.D_2) \sqcap (\exists s.D_3), & & \\ & D_2 \doteq B_2 \sqcap B_3, & & D_3 \doteq B_3 \sqcap B_4 \ \}, \end{array}$$

we compute the lcs of C_1 and C_2 as sketched in Section 3. After the first and second step we obtain the unfolded and normalized descriptions

$$C_i' := A_i \sqcap \exists r.\big(\forall s.B_1 \sqcap \exists s.(B_1 \sqcap B_2 \sqcap B_3) \sqcap \exists s.(B_1 \sqcap B_3 \sqcap B_4)\big)$$

for $i \in \{1,2\}$. In the third step, the algorithm first determines the concept names appearing on the top-level of the lcs—in this case none since $\{A_1\} \cap \{A_2\} = \emptyset$. Then, the algorithm makes a recursive call to compute the lcs of the descriptions occurring in the existential restriction of C_1' and C_2', respectively. This in turn leads to recursive calls for the pair of value restrictions and for all four pairs of existential restrictions for the s-successors. As output, the algorithm yields

$$\begin{array}{ll} \mathsf{LCS}(C_1, C_2) = \exists r.(\forall s.B_1 \sqcap & \\ \qquad \exists s.(B_1 \sqcap B_2 \sqcap B_3) \sqcap & \\ \qquad \exists s.(B_1 \sqcap B_3) \sqcap & \\ \qquad \exists s.(B_1 \sqcap B_3) \sqcap & \\ \qquad \exists s.(B_1 \sqcap B_3 \sqcap B_4)) & \text{which is of size } |\mathsf{LCS}(C_1, C_2)| = 17. \end{array}$$

Thus, the result is computed by three recursive traversals of the input descriptions. The size of the output description results from the actually unnecessary unfolding of D_1. Even if the redundant existential restrictions are eliminated from the result, the size of the returned lcs concept description is still quite big in comparison to the equivalent description $\exists r.D_1$. Of course, rewriting the computed lcs w.r.t. $\mathcal{T}$ would also yield this result, but it would be better to avoid obtaining such an unnecessarily large intermediate result.

The idea of lazy unfolding is to replace a defined concept in a concept description only if examination of that part of the description is necessary. Lazy unfolding unfolds all defined concepts appearing on the top-level of the concept description under consideration, while defined concepts on deeper role-levels remain unchanged. Note, however, that unfolding may still be applied iteratedly if unfolding has produced another defined name on the top-level. Once this is finished, all (negated) primitive concepts, value restrictions, and existential restrictions that would occur on the top-level of the completely unfolded concept description are visible, and one can in principle continue as in the case of the naïve lcs algorithm.

The algorithm for computing the lcs with lazy unfolding, as shown in Figure 4, is based on the following sets:

Input: Two $\mathcal{ALE}$-concept descriptions C, D and an $\mathcal{ALE}$-TBox $\mathcal{T}$

Algorithm: $\mathsf{LCS_{lu}}(C, D)$

if $C \sqsubseteq_{\mathcal{T}} D$ **then** $\mathsf{LCS_{lu}}(C, D) = D$
if $D \sqsubseteq_{\mathcal{T}} C$ **then** $\mathsf{LCS_{lu}}(C, D) = C$
else

$C' := \mathsf{lazy\text{-}unfold}(C, \mathcal{T})$,
$\{P_1, \ldots, P_n\} := \mathsf{prim}(C')$,
for all $r \in N_R$:
$C_0 := \mathsf{val}_r(C')$,
$\{C_1, \ldots, C_n\} := \mathsf{ex}_r(C')$,
end for

$D' := \mathsf{lazy\text{-}unfold}(D, \mathcal{T})$,
$\{P_1, \ldots, P_n\} := \mathsf{prim}(D')$,
for all $r \in N_R$:
$D_0 := \mathsf{val}_r(D')$,
$\{D_1, \ldots, D_n\} := \mathsf{ex}_r(D')$
end for

$$\begin{aligned}\mathsf{LCS_{lu}}(C, D) = & \bigsqcap_{P \in \mathsf{prim}(C') \cap \mathsf{prim}(D')} P \quad \sqcap \\ & \bigsqcap_{r \in N_R} (\ \forall r.\mathsf{LCS_{lu}}\ (\mathsf{val}_r(C'), \mathsf{val}_r(D'))\ ) \quad \sqcap \\ & \bigsqcap_{r \in N_R} (\bigsqcap_{\substack{C_i \in \mathsf{ex}_r(C') \\ D_j \in \mathsf{ex}_r(D')}} \exists r.\ \mathsf{LCS_{lu}}\ (C_i \sqcap \mathsf{val}_r(C'),\ D_j \sqcap \mathsf{val}_r(D')))\end{aligned}$$

Fig. 4. The lcs algorithm $\mathsf{LCS_{lu}}$ for $\mathcal{ALE}$ using lazy unfolding.

$\mathsf{prim}(C)$: denotes the set of all (negated) primitive names occurring on the top-level of C.

$\mathsf{val}_r(C)$: denotes the conjunction of the concept descriptions occurring in value restrictions on the top-level of C, where $\mathsf{val}_r(C) := \top$ if there is no value restriction.

$\mathsf{ex}_r(C)$: denotes the set $\{C_1, \ldots, C_n\}$ of concept descriptions occurring in existential restrictions of the form $\exists r.C_i$ on the top-level of C.

The $\mathsf{LCS_{lu}}$ algorithm as given in Figure 4 tests in each recursion, if the input concepts subsume each other, in this case the lcs is trivial. The algorithm uses the function $\mathsf{lazy\text{-}unfold}()$ for unfolding the top-level of the input concept descriptions w.r.t. the input $\mathcal{ALE}$-TBox $\mathcal{T}$. Next, the auxiliary sets and concept descriptions defined above are computed. The returned lcs concept description is a conjunction of three components:

1. the conjunction of all (negative) primitive concepts appearing on both the top-level of C' and D',
2. the conjunction of all value restrictions derived from recursive calls of $\mathsf{LCS_{lu}}$ for each role that has a value restriction on top-level of C' and D',
3. a conjunction of the existential restrictions derived from recursive calls of $\mathsf{LCS_{lu}}$ for each combination of existential restrictions where the value restrictions are propagated "on the fly".

In contrast to the three independent recursions in the naïve algorithm, the $\mathsf{LCS}_{\mathsf{lu}}$ algorithm traverses the structure of the concept descriptions recursively only once. The three steps of the basic algorithm are now interwoven on each role-level. In particular, the normalization of the concept descriptions is realized role-level-wise (i) by the definition of the description $\mathsf{val}_r(C')$ and $\mathsf{val}_r(D')$; (ii) by including the conjuncts $\mathsf{val}_r(C')$ and $\mathsf{val}_r(D')$, respectively, in the recursive calls for the existential restrictions. The normalization rules dealing with negation and the bottom concept (which we have not described above) are taken care of by the subsumption test at the beginning of the algorithm.

There are two reasons why the new lcs algorithm may avoid computations done by the naïve algorithm. First, if one of the subsumption tests at the beginning are successful, then one of the input descriptions is returned without unfolding or normalizing any of the two descriptions. This can also happen in recursive calls of the algorithm. In particular, this also means that the returned description may still contain defined names. Second, if an existential restriction for a role r has no matching restriction in the other description, then this restriction need not be processed (i.e., its concept description is not unfolded and normalized). Consider once more Example 5 to illustrate the first effect.

Example 6 (lcs using lazy unfolding). Assume we apply the new lcs algorithm to the descriptions C_1, C_2 of Example 5. In the first step, none of the two subsumption conditions hold and the algorithm calls $\mathsf{lazy\text{-}unfold}(C_1, \mathcal{T})$ and $\mathsf{lazy\text{-}unfold}(C_2, \mathcal{T})$. There is no defined concept name to replace on top-level. Then the algorithm calls $\mathsf{LCS}_{\mathsf{lu}}$ recursively for the pair of existential restrictions. Because the subsumption test is successful, this call directly yields D_1, without considering the definition of D_1. Thus the returned concept description is $\mathsf{LCS}_{\mathsf{lu}}(C_1, C_2) = \exists r.D_1$, which has size $|\mathsf{LCS}_{\mathsf{lu}}(C_1, C_2)| = 2$.

Comparing the lcs from Example 5 to the result obtained here, we see that $\mathsf{LCS}_{\mathsf{lu}}$ needs less recursive calls with less recursion depth, and furthermore returns a smaller concept description.

Using lazy unfolding is advantageous in most cases when computing a lcs w.r.t. a TBox, but there are, of course, combinations of input concept descriptions where an exponential growth of the lcs concept description is still unavoidable.

6 Implementations of the lcs

We have implemented both, the naïve and the lazy unfolding based lcs algorithm [12] in Lisp. The FaCT system [8] is used to compute subsumption. The core of both implementations is a binary lcs function wrapped by a function that successively calls the binary lcs function.

The "old lcs" is a straightforward implementation of the fundamental algorithm outlined in Section 3 and discussed in [2]. It also uses an implementation of the heuristic rewriting algorithm for computing small (but not always minimal) rewritings of $\mathcal{ALE}$-concept descriptions mentioned earlier (see [3]), which we

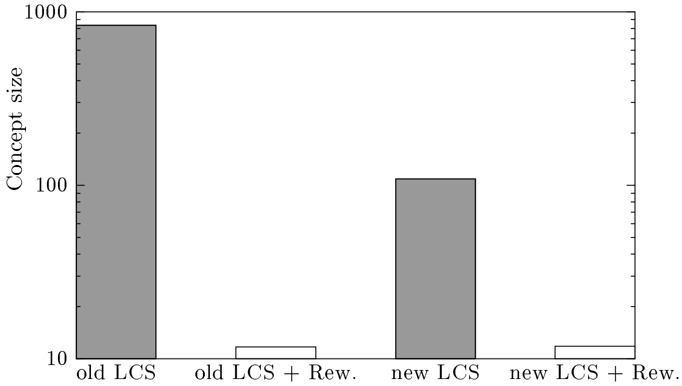

Fig. 5. Average concept sizes obtained from the four settings.

use in our evaluation. The "new lcs" implements the algorithm $\mathsf{LCS}_{\mathsf{lu}}$ introduced in Section 5. It is also a straightforward implementation of this algorithm and does not employ special low-level encoding tricks to improve its performance. In contrast to the old lcs implementation, which is strongly linked to FaCT, the new lcs may be coupled with arbitrary DL reasoners.

6.1 A first Evaluation of the Implementations

To compare the performance of our implementations of both algorithms we use a TBox derived from our application in chemical process engineering. It contains 52 primitive concepts, 67 defined concepts and 43 roles. It has a rather deep concept hierarchy, which makes it likely that least common subsumers computed for concept descriptions defined in this TBox will not collapse to $\top$.

The input descriptions we used for the evaluation are combinations of seven $\mathsf{REACTOR}$ concepts defined in the application TBox. To compute the lcs of all possible combinations of these $\mathsf{REACTOR}$ concepts, it suffices to compute some combinations determined by the attribute exploration algorithm as described in [4]. Our test suite included 22 different lcs calls, ranging from binary lcs calls to lcs calls with seven input concepts. For each computation of these least common subsumers, we measured run-times and sizes of the output concept descriptions of four settings: both of the lcs implementations and both of the lcs implementations followed by a rewriting step. The latter two use the same rewriting implementation of the heuristic algorithm. To obtain representative run-times we ran each LCS in each setting 100 times.

The results for the average concept size are shown in Figure 5, where one should note the logarithmic scale. The measured values indicate that an lcs computed by the $\mathsf{LCS}_{\mathsf{lu}}$ implementation returns concept descriptions that are about an order of magnitude smaller than the concept descriptions returned by the naïve algorithm. The rewritten lcs concept descriptions are again one order of magnitude smaller than the lcs concept description returned by the $\mathsf{LCS}_{\mathsf{lu}}$

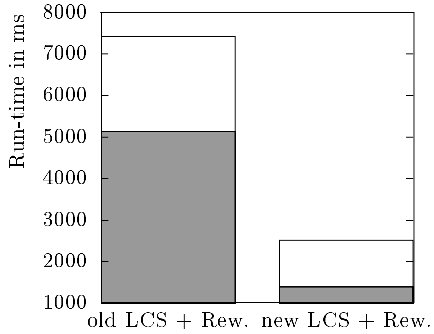

Fig. 6. Average run-times needed by the different settings.

implementation. Comparing the concept sizes obtained for the settings including the heuristic rewriting shows that starting from a smaller concept description does not yield a smaller rewritten concept. This is probably due to the fact that for our examples the heuristic algorithm produced the optimal result.

The average concept description obtained with the $\mathsf{LCS}_{\mathsf{lu}}$ implementation has a concept size of about 100. These concept descriptions are still too big to be comprehensible to a human reader. In our application scenario, the knowledge engineer is supposed to choose an appropriate description from a set of computed least common subsumers, and possibly also modifying the chosen description by hand. Therefore, rewriting remains necessary as an additional step in this application.

Figure 6 shows the sum of run-times for computing the lcs (grey) and rewriting the obtained lcs concept description (white). The comparison of run-times for the lcs implementations indicates a speed-up of factor 3.5. The run-time for rewriting an lcs concept description returned by the $\mathsf{LCS}_{\mathsf{lu}}$ implementation is also lower than for rewriting the description produced by the naïve algorithm (by a factor of about 2). Taking lcs computation and rewriting together, the overall run-time differs by a factor of three.

7 Conclusion and Future Work

The worst-case examples presented in Section 4 are quite contrived and not likely to occur in practice. Nevertheless, they show that, in principle, the exponential blow-up inherent to the lcs operation cannot be avoided, even if one can introduce "abbreviations" for subdescriptions. An interesting question for future research is to characterize situations in which this exponential blow-up cannot occur, and to check whether these situations are likely to occur in practice.

The performance of the lcs algorithm using lazy unfolding greatly depends on the structure of the TBox and the input descriptions. There are, of course, examples where there is no improvement over the naïve algorithm. However, since no overhead is generated by using lazy unfolding, it is advantageous to use the new algorithm in any case.

The first evaluation of the lcs implementations in our application framework indicates that using lazy unfolding can substantially decrease the size of the resulting concept descriptions. However, our results also indicate that it is still necessary to perform rewriting after computing the lcs in order to obtain concept descriptions that are small enough to be inspected by human users. As indicated by our tests, lazy unfolding will also decrease the run-time of the subsequent rewriting step.

References

1. F. Baader and R. Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In O. Herzog and A. Günter, eds., *Proc. of KI-98*, volume 1504 of *Lecture Notes in Computer Science*, p. 129–140, Bremen, Germany, 1998. Springer-Verlag.
2. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumer in description logics with existential restrictions. In T. Dean, ed., *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, p. 96–101, Stockholm, Sweden, 1999. Morgan Kaufmann, Los Altos. An extended version appeared as LTCS-Report LTCS-98-09, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1998. See http://www-lti.informatik.rwth-aachen.de/Forschung/Papers.html.
3. F. Baader, R. Küsters, and R. Molitor. Rewriting concepts using terminologies. In A.G. Cohn, F. Giunchiglia, and B. Selman, eds., *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-00)*, p. 297–308, San Francisco, CA, 2000. Morgan Kaufmann Publishers.
4. F. Baader and R. Molitor. Building and structuring description logic knowledge bases using least common subsumers and concept analysis. In B. Ganter and G. Mineau, eds., *Proc. of ICCS-00*, volume 1867 of *Lecture Notes in Artificial Intelligence*, p. 290–303. Springer-Verlag, 2000.
5. F. Baader and U. Sattler. Knowledge representation in process engineering. In *Proc. of DL-96*, 1996.
6. F. Baader and A.-Y. Turhan. TBoxes do not yield a compact representation of the least common subsumer. In *Proc. of DL-2001*, 2001.
7. F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.
8. I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.
9. R. Küsters and R. Molitor. Approximating most specific concepts in description logics with existential restrictions. In T. Eiter F. Baader, G. Brewka, eds., *Proc. of the 24th German Annual Conf. on Artificial Intelligence (KI'01)*, number 2174 in Lecture Notes In Artificial Intelligence, p. 33–47. Springer-Verlag, 2001.
10. B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence Journal*, 43:235–249, 1990.
11. U. Sattler. *Terminological knowledge representation systems in a process engineering application.* PhD thesis, RWTH Aachen, 1998.
12. A.-Y. Turhan and R. Molitor. *Using lazy unfolding for the computation of least common subsumers.* In *Proc. of DL-2001*, 2001.