Building and Structuring Description Logic Knowledge Bases Using Least Common Subsumers and Concept Analysis

Franz Baader and Ralf Molitor

Lehr- und Forschungsgebiet Theoretische Informatik, RWTH Aachen,
Ahornstraße 55, 52074 Aachen, Germany,
{baader,molitor}@informatik.rwth-aachen.de

Abstract. Given a finite set $\mathcal{C} := \{C_1, \ldots, C_n\}$ of description logic concepts, we are interested in computing the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$. This hierarchy can be used to support the bottom-up construction and the structuring of description logic knowledge bases. The point is to compute this hierarchy without having to compute the least common subsumer for all subsets of $\mathcal{C}$. We will show that methods from formal concept analysis developed for computing concept lattices can be employed for this purpose.

1 Introduction

Knowledge representation systems based on description logics (DL) can be used to describe the knowledge of an application domain in a structured and formally well-understood way. Traditionally, the knowledge base of a DL system is built in a *top-down* manner. First, the relevant concepts of the application domain (its terminology) are formalized by *concept descriptions*, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept constructors provided by the DL language. In a second step, these concept descriptions are then used to specify properties of *objects* occurring in the domain. The *standard inference procedures* provided by DL systems (like computing the subsumption hierarchy between concepts, and testing for implied instance relationships between objects and concepts) support this traditional approach to building DL knowledge bases.

The main problem with the top-down approach is that it presupposes a knowledge engineer that is both, an expert in description logics, and in the application domain. Less experienced knowledge engineers encounter (at least one of) the following two problems: (1) it is often not clear which are the relevant concepts in an application; and (2) even if it is clear which (intuitive) concepts should be introduced, it is sometimes difficult to come up with formal definitions of these concepts in the given DL language. It has turned out that providing adequate support for overcoming these problems requires additional (non-standard) inference procedures for DL systems.

In [2,3], we propose to support the construction of DL knowledge bases in a *bottom-up* fashion: instead of directly defining a new concept, the knowledge engineer introduces several typical examples as objects, which are then automatically generalized into a concept description by the system. This description is offered to the knowledge engineer as a possible candidate for a definition of the concept. The task of computing such a concept description can be split into two subtasks: computing the most specific concepts of the given objects, and then computing the least common subsumer of these concepts. The *most specific concept* (msc) of an object o (the *least common subsumer* (lcs) of concept descriptions $C_1, \ldots, C_n$) is the most specific concept description C expressible in the given DL language that has o as an instance (that subsumes $C_1, \ldots, C_n$). The problem of computing the lcs and (to a limited extent) the msc has already been investigated in the literature [5,9,2,3].

Here, we will address an additional problem that occurs in the bottom-up approach: obviously, the *choice of the examples* is crucial for the quality of the result. If the examples are too similar, the resulting concept might be too specific. Conversely, if the examples are too different, the resulting concept is likely to be too general. Our goal in this paper is to support the process of choosing an appropriate set of objects as examples. Assume that $C_1, \ldots, C_n$ are the most specific concepts of a given collection of objects $o_1, \ldots, o_n$, and that we intend to use subsets of this collection for constructing new concepts. In order to avoid obtaining concepts that are too general or too specific, it would be good to know the position of the corresponding lcs in the subsumption hierarchy of all least common subsumers of subsets of $\{C_1, \ldots, C_n\}$. Since there are exponentially many subsets to be considered, and (depending on the DL language) both, computing the lcs and testing for subsumption, can be expensive operations, we want to obtain complete information on how this hierarchy looks like without computing the least common subsumers of all subsets of $\{C_1, \ldots, C_n\}$, and without explicitly making all the subsumption tests between these least common subsumers.

This is where methods from *formal concept analysis* [11] come into play. We shall show that the *dual* of the *attribute exploration* algorithm [10,11] (called *object exploration* in the following) can be adapted to our purposes. To be more precise, given a formal context, the attribute (object) exploration algorithm computes the concept lattice as well as a minimal implication base, the so-called (dual) Duquenne-Guigues base [8], of this context. For a given set of concept descriptions $\{C_1, \ldots, C_n\}$, we will define a formal context that has the property that its concept lattice is isomorphic to the subsumption hierarchy of all least common subsumers of subsets of $\{C_1, \ldots, C_n\}$. Thus, standard tools for drawing concept lattices [17] can be employed to show the hierarchy to the user. In addition, the dual Duquenne-Guigues base provides a small representation of this hierarchy. From this representation, all subsumption relationships can be deduced in time linear in the size of the representation. To compute the concept lattices and the Duquenne-Guigues base, the exploration algorithm employs an algorithm for computing the lcs and a decision procedure for subsumption as

sub-procedures. Although, in the worst case, an exponential number of calls to these sub-procedures cannot be avoided, experiences from applications of formal concept analysis [16] indicate that the exploration algorithm usually does a lot better in practice.

Another application for this method is *structuring of DL knowledge bases.* DL knowledge bases encountered in applications are often rather flat in the sense that a given concept can have a large number of direct successors in the subsumption hierarchy. Deeper hierarchies would be more convenient when browsing the knowledge base along the hierarchy, and they would make searching more efficient. Given a concept C with direct sub-concepts $\{C_1, \ldots, C_n\}$, one could use least common subsumers of selected subsets to provide a better structuring of the knowledge base by inserting additional layers between C and its sub-concepts $C_1, \ldots, C_n$. Again, knowing the hierarchy of all least common subsumers of subsets of $\{C_1, \ldots, C_n\}$ can support the knowledge engineer in choosing the right subsets.

In the next section, we introduce the DL languages $\mathcal{ALE}$ and $\mathcal{ALN}$, define the subsumption problem as well as the notion "least common subsumer", and recall results from the literature for deciding subsumption and computing the least common subsumer. In Section 3, we introduce as many of the basic notions of formal concept analysis as are necessary for our purposes. In particular, we sketch the object exploration algorithm. Section 4 applies this technique to our problem of computing the subsumption hierarchy between least common subsumers of subsets of a given collection of concepts. In Section 5, we provide some experimental results that support our thesis that object exploration is an appropriate tool for this purpose. Section 6 concludes with some comments on related and future work.

2 The description logics $\mathcal{ALE}$ and $\mathcal{ALN}$

For the purpose of this paper, it is sufficient to restrict the attention to the formalism for defining concept descriptions (i.e., we need not introduce TBoxes, which allow to abbreviate complex descriptions by names, and ABoxes, which introduce objects and their properties). In order to define concepts in a DL knowledge base, one starts with a set N_C of concept names (unary predicates) and a set N_R of role names (binary predicates), and defines more complex *concept descriptions* using the operations provided by the DL language of the particular system. In this paper, we consider the languages $\mathcal{ALE}$ and $\mathcal{ALN}$,[1] which allow for concept descriptions built from the indicated subsets of the constructors shown in Table 1. In this table, P stands for a concept name, r for a role name, n for a nonnegative integer, and C, D for arbitrary concept descriptions.

The semantics of concept descriptions is defined in terms of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is a non-empty set and the interpretation

[1] It should be noted, however, that the methods developed in this paper apply to arbitrary concept descriptions languages, as long as they are equipped with a subsumption algorithm and an algorithm for computing least common subsumers.

Table 1. Syntax and semantics of concept descriptions.

name of constructor	Syntax	Semantics	$\mathcal{ALE}$	$\mathcal{ALN}$
top-concept	$\top$	$\Delta^{\mathcal{I}}$	x	x
bottom-concept	$\bot$	$\emptyset$	x	x
primitive negation	$\neg P$	$\Delta^{\mathcal{I}} \setminus P^{\mathcal{I}}$	x	x
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	x	x
value restriction	$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$	x	x
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	x	
number restriction	$(\geq n\ r)$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \geq n\}$		x
number restriction	$(\leq n\ r)$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \leq n\}$		x

function $\cdot^{\mathcal{I}}$ maps each concept name $P \in N_C$ to a set $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ to arbitrary concept descriptions is inductively defined, as shown in the third column of Table 1.

One of the most important traditional inference services provided by DL-systems is computing subconcept/superconcept relationships (so-called *subsumption* relationships) between concept descriptions. The concept description C_2 *subsumes* the concept description C_1 ($C_1 \sqsubseteq C_2$) iff $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for all interpretations $\mathcal{I}$; C_2 is *equivalent* to C_1 ($C_1 \equiv C_2$) iff $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. The subsumption relation $\sqsubseteq$ is a quasi order (i.e., reflexive and transitive), but in general not a partial order since it need not be antisymmetric (i.e., there may exist equivalent description that are not syntactically equal). As usual, the quasi order $\sqsubseteq$ induces a partial order $\sqsubseteq_{\equiv}$ on the equivalence classes of concept descriptions:

$$[C_1]_{\equiv} \sqsubseteq_{\equiv} [C_2]_{\equiv} \quad \text{iff} \quad C_1 \sqsubseteq C_2,$$

where $[C_i]_{\equiv} := \{D \mid C_i \equiv D\}$ is the equivalence class of C_i ($i = 1, 2$). When talking about the *subsumption hierarchy* of a set of descriptions, one means this induced partial order.

Deciding subsumption between $\mathcal{ALN}$-concept descriptions is polynomial [4], whereas the subsumption problem for $\mathcal{ALE}$ is NP-complete [6].

In addition to subsumption, we are interested in the non-standard inference problem of computing the least common subsumer of concept descriptions.

Definition 1 *Given $n \geq 2$ concept descriptions $C_1, \ldots, C_n$ in a DL language $\mathcal{L}$, the concept description C of $\mathcal{L}$ is an lcs of $C_1, \ldots, C_n$ ($C = \mathsf{lcs}(C_1, \ldots, C_n)$) iff (i) $C_i \sqsubseteq C$ for all $1 \leq i \leq n$, and (ii) C is the least concept description with this property, i.e., if C' satisfies $C_i \sqsubseteq C'$ for all $1 \leq i \leq n$, then $C \sqsubseteq C'$.*

As an example, consider the $\mathcal{ALE}$-concept descriptions

$$C := \exists \mathsf{has\text{-}child}.\top \sqcap \forall \mathsf{has\text{-}child}.(\mathsf{Male} \sqcap \mathsf{Doctor}) \quad \text{and}$$
$$D := \exists \mathsf{has\text{-}child}.(\mathsf{Male} \sqcap \mathsf{Mechanic}) \sqcap \exists \mathsf{has\text{-}child}.(\mathsf{Female} \sqcap \mathsf{Doctor})$$

respectively describing parents whose children are all male doctors, and parents having a son that is a mechanic and a daughter that is a doctor. The lcs of C and D is given by the $\mathcal{ALE}$-concept description

$$\mathsf{lcs}(C, D) = \exists \mathsf{has\text{-}child}.\mathsf{Male} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Doctor}.$$

It describes all parents having at least one son and at least one child that is a doctor (see [3] for an algorithm for computing such an lcs).

Depending on the DL under consideration, the lcs of two or more descriptions need not always exist, but if it exists, then it is unique up to equivalence. It is also easy to see that one can restrict the attention to the problem of computing the lcs of two concept descriptions, since the lcs of $n > 2$ descriptions can be obtained by iterated application of the binary lcs operation.

In [3], we have shown that the lcs of two $\mathcal{ALE}$-concept descriptions always exists and that it can effectively be computed; however, the size of the lcs can be exponential in the size of the input descriptions. For $\mathcal{ALN}$, the lcs of two descriptions also always exists and it can be computed in polynomial time. In addition, the size of the lcs is polynomial in the size of the input descriptions, even if one considers $n > 2$ descriptions (these results for $\mathcal{ALN}$ can easily be obtained by restricting the results in [2] to the acyclic case).

3 Formal concept analysis

We shall introduce only those notions and results from formal concept analysis that are necessary for our application. We will describe how the object exploration algorithm works, but note that explaining why it works is beyond the scope of this paper (see [11] for more information on formal concept analysis).

Definition 2 *A* formal context *is a triple $\mathcal{K} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$, where $\mathcal{O}$ is a set of objects, $\mathcal{P}$ is a set of attributes (or properties), and $\mathcal{S} \subseteq \mathcal{O} \times \mathcal{P}$ is a relation that connects each object o with the attributes satisfied by o.*

Let $\mathcal{K} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ be a formal context. For a set of objects $A \subseteq \mathcal{O}$, the *intent* A' of A is the set of attributes that are satisfied by all objects in A, i.e.,

$$A' := \{p \in \mathcal{P} \mid \forall a \in A\colon (a, p) \in \mathcal{S}\}.$$

Similarly, for a set of attributes $B \subseteq \mathcal{P}$, the *extent* B' of B is the set of objects that satisfy all attributes in B, i.e.,

$$B' := \{o \in \mathcal{O} \mid \forall b \in B\colon (o, b) \in \mathcal{S}\}.$$

It is easy to see that, for $A_1 \subseteq A_2 \subseteq \mathcal{O}$ (resp. $B_1 \subseteq B_2 \subseteq \mathcal{P}$), we have

- $A_2' \subseteq A_1'$ (resp. $B_2' \subseteq B_1'$),
- $A_1 \subseteq A_1''$ and $A_1' = A_1'''$ (resp. $B_1 \subseteq B_1''$ and $B_1' = B_1'''$).

A *formal concept* is a pair (A, B) consisting of an extent $A \subseteq \mathcal{O}$ and an intent $B \subseteq \mathcal{P}$ such that $A' = B$ and $B' = A$. Such formal concepts can be hierarchically ordered by inclusion of their extents, and this order induces a complete lattice, the *concept lattice* of the context. Given a formal context, the first step for analyzing this context is usually to compute the concept lattice.

The following are easy consequences of the definition of formal concepts and the properties of the $\cdot'$ operation mentioned above:

Lemma 3 *All formal concepts are of the form* (A'', A') *for a subset* A *of* $\mathcal{O}$*, and any such pair is a formal concept. In addition,* $(A_1'', A_1') \leq (A_2'', A_2')$ *iff* $A_2' \subseteq A_1'$.

Thus, if the context is finite, the concept lattices can in principle be computed by enumerating the subsets A of $\mathcal{O}$, and applying the operations $\cdot'$ and $\cdot''$. However, this naïve algorithm is usually very inefficient. In many applications [16], one has a large (or even infinite) set of objects, but only a relatively small set of attributes. In such a situation, Ganter's *attribute exploration* algorithm [10, 11] has turned out to be an efficient approach for computing the concept lattice.

In the application considered in this paper, we are faced with the *dual* situation: the set of attributes will be the *infinite* set of all possible concept descriptions of the DL under consideration, and the set of objects will be the *finite* collection of concept descriptions for which we want to compute the subsumption hierarchy of least common subsumers. Consequently, we must dualize Ganter's algorithm and the notions on which it depends. Alternatively, we could have considered the dual context (which is obtained by transposing the matrix corresponding to $\mathcal{S}$) and employed the usual attribute exploration for this context. The correctness of the dual version is an immediate consequence of the fact that its results coincide with the results of the usual algorithm on the dual context.

Object exploration

Before we can describe the dual version of Ganter's algorithm, called object exploration in the following, we must introduce some notation. The most important notion for the algorithm is the one of an implication between sets of objects. Intuitively, such an implication $A_1 \to A_2$ holds if any attribute satisfied by all elements of A_1 is also satisfied by all elements of A_2.

Definition 4 *Let* $\mathcal{K}$ *be a formal context and* A_1*,* A_2 *be subsets of* $\mathcal{O}$*. The* object implication $A_1 \to A_2$ *holds in* $\mathcal{K}$ *(*$\mathcal{K} \models A_1 \to A_2$*) iff* $A_1' \subseteq A_2'$*. An attribute* p violates *the implication* $A_1 \to A_2$ *iff* $p \in A_1' \setminus A_2'$.

It is easy to see that an implication $A_1 \to A_2$ holds in $\mathcal{K}$ iff $A_2 \subseteq A_1''$. In particular, given a set of objects A, the implications $A \to (A'' \setminus A)$ always holds in $\mathcal{K}$. We denote the set of all object implications that hold in $\mathcal{K}$ by $Imp_{\mathcal{O}}(\mathcal{K})$. This set can be very large, and thus one is interested in (small) generating sets.

Definition 5 *Let $\mathcal{J}$ be a set of object implications, i.e., the elements of $\mathcal{J}$ are of the form $A_1 \to A_2$ for sets of objects $A_1, A_2 \subseteq \mathcal{O}$. For a subset A of $\mathcal{O}$, the* implication hull *of A with respect to $\mathcal{J}$ is denoted by $\mathcal{J}(A)$. It is the smallest subset H of $\mathcal{O}$ such that*

- *$A \subseteq H$, and*
- *$A_1 \to A_2 \in \mathcal{J}$ and $A_1 \subseteq H$ imply $A_2 \subseteq H$.*

The set of object implications generated by $\mathcal{J}$ *consists of all implications $A_1 \to A_2$ such that $A_2 \subseteq \mathcal{J}(A_1)$. It will be denoted by $Cons(\mathcal{J})$. We say that a set of implications $\mathcal{J}$ is a* base *of $Imp_{\mathcal{O}}(\mathcal{K})$ iff $Cons(\mathcal{J}) = Imp_{\mathcal{O}}(\mathcal{K})$ and no proper subset of $\mathcal{J}$ satisfies this property.*

If $\mathcal{J}$ is a base for $Imp_{\mathcal{O}}(\mathcal{K})$, then it can be shown that $A'' = \mathcal{J}(A)$ for all $A \subseteq \mathcal{O}$. The implication hull $\mathcal{J}(A)$ of a set of objects A can be computed in time linear in the size of $\mathcal{J}$ and A using, for example, methods for deciding satisfiability of sets of propositional Horn clauses [7]. Consequently, given a base $\mathcal{J}$ for $Imp_{\mathcal{O}}(\mathcal{K})$, any question of the form "$A_1 \to A_2 \in Imp_{\mathcal{O}}(\mathcal{K})$?" can be answered in time linear in the size of $\mathcal{J} \cup \{A_1 \to A_2\}$.

There may exist different implication bases of $Imp_{\mathcal{O}}(\mathcal{K})$, and not all of them need to be of minimal cardinality. A base $\mathcal{J}$ of $Imp_{\mathcal{O}}(\mathcal{K})$ is called *minimal base* iff no base of $Imp_{\mathcal{O}}(\mathcal{K})$ has a cardinality smaller than the cardinality of $\mathcal{J}$. Duquenne and Guigues have given a description of such a minimal base [8] for the dual case of attribute implications. Ganter's attribute exploration algorithm computes this minimal base as a by-product. In the following, we introduce the *dual Duquenne-Guigues base* and show how it can be computed using the object exploration algorithm.

The definition of the dual Duquenne-Guigues base given below is based on a modification of the closure operator $A \mapsto \mathcal{J}(A)$ defined by a set $\mathcal{J}$ of object implications. For a subset A of $\mathcal{O}$, the *implication pseudo-hull* of A with respect to $\mathcal{J}$ is denoted by $\mathcal{J}^*(A)$. It is the smallest subset H of $\mathcal{O}$ such that

- $A \subseteq H$, and
- $A_1 \to A_2 \in \mathcal{J}$ and $A_1 \subset H$ (strict subset) imply $A_2 \subseteq H$.

Given $\mathcal{J}$, the pseudo-hull of a set $A \subseteq \mathcal{O}$ can again be computed in time linear in the size of $\mathcal{J}$ and A (e.g., by adapting the algorithm in [7] appropriately). A subset A of $\mathcal{O}$ is called *pseudo-closed* in a formal context $\mathcal{K}$ iff $Imp_{\mathcal{O}}(\mathcal{K})^*(A) = A$ and $A'' \neq A$.

Definition 6 *The* dual Duquenne-Guigues base *of a formal context $\mathcal{K}$ consists of all object implications $A_1 \to A_2$ where $A_1 \subseteq \mathcal{O}$ is pseudo-closed in $\mathcal{K}$ and $A_2 = A_1'' \setminus A_1$.*

When trying to use this definition for actually *computing* the dual Duquenne-Guigues base of a formal context, one encounters two problems:

1. The definition of pseudo-closed refers to the set of all valid implications $Imp_{\mathcal{O}}(\mathcal{K})$, and our goal is to avoid explicitly computing all of them.

2. The closure operator $A \mapsto A''$ is used, and computing it via $A \mapsto A' \mapsto A''$ may not be feasible for a context with an infinite set of attributes.

Ganter solves the first problem by enumerating the pseudo-closed sets of $\mathcal{K}$ in a particular order, called lectic order. This order makes sure that it is sufficient to use the already computed part $\mathcal{J}$ of the base when computing the pseudo-hull. To define the lectic order, fix an arbitrary linear order on the set of objects $\mathcal{O} = \{o_1, \ldots, o_n\}$, say $o_1 < \cdots < o_n$. For all $j, 1 \leq j \leq n$, and $A_1, A_2 \subseteq \mathcal{O}$ we define

$$A_1 <_j A_2 \text{ iff } o_j \in A_2 \setminus A_1 \text{ and } A_1 \cap \{o_1, \ldots, o_{j-1}\} = A_2 \cap \{o_1, \ldots, o_{j-1}\}.$$

The lectic order $<$ is the union of all relations $<_j$ for $j = 1, \ldots, n$. It is a linear order on the powerset of $\mathcal{O}$. The lectic smallest subset of $\mathcal{O}$ is the empty set.

The second problem is solved by constructing an increasing chain of finite subcontexts of $\mathcal{K}$. The context $\mathcal{K}_i = (\mathcal{O}_i, \mathcal{P}_i, \mathcal{S}_i)$ is a *subcontext* of $\mathcal{K}$ iff $\mathcal{O}_i = \mathcal{O}$, $\mathcal{P}_i \subseteq \mathcal{P}$, and $\mathcal{S}_i = \mathcal{S} \cap (\mathcal{O} \times \mathcal{P}_i)$. The closure operator $A \mapsto A''$ is always computed with respect to the current finite subcontext $\mathcal{K}_i$. To avoid adding a wrong implication, an "expert" is asked whether the implication $A \to A'' \setminus A$ really holds in the whole $\mathcal{K}$. If it does not hold, the expert must provide a counterexample, i.e., an attribute p from $\mathcal{P} \setminus \mathcal{P}_i$ that violates the implication. This attribute is then added to the current context. Technically, this means that the expert must provide an attribute p, and must say which of the objects of $\mathcal{O}$ satisfy this attribute and which don't.

The following algorithm computes the set of all extents of formal concepts of $\mathcal{K}$ as well as the dual Duquenne-Guigues base of $\mathcal{K}$. The concept lattice is then given by the usual inclusion ordering between the extents.

Algorithm 7 (Object exploration) Initialization: *One starts with the empty set of object implications, i.e., $\mathcal{J}_0 := \emptyset$, the empty set of concept extents $\mathcal{E}_0 := \emptyset$, and the empty subcontext $\mathcal{K}_0$ of $\mathcal{K}$, i.e., $\mathcal{P}_0 := \emptyset$. The lectic smallest subset of $\mathcal{O}$ is $A_0 := \emptyset$.*

Iteration: *Assume that $\mathcal{K}_i$, $\mathcal{J}_i$, $\mathcal{E}_i$, and A_i ($i \geq 0$) are already computed. Compute A_i'' with respect to the current subcontext $\mathcal{K}_i$. Now the expert is asked whether the implication $A_i \to A_i'' \setminus A_i$ holds in $\mathcal{K}$.*[2]

If the answer is "no", then let $p_i \in \mathcal{P}$ be the counterexample provided by the expert. Let $A_{i+1} := A_i$, $\mathcal{J}_{i+1} := \mathcal{J}_i$, and let $\mathcal{K}_{i+1}$ be the subcontext of $\mathcal{K}$ with $\mathcal{P}_{i+1} := \mathcal{P}_i \cup \{p_i\}$.

If the answer is "yes", then $\mathcal{K}_{i+1} := \mathcal{K}_i$ and

$$(\mathcal{E}_{i+1}, \mathcal{J}_{i+1}) := \begin{cases} (\mathcal{E}_i, \mathcal{J}_i \cup \{A_i \to A_i'' \setminus A_i\}) & \text{if } A_i'' \neq A_i, \\ (\mathcal{E}_i \cup \{A_i\}, \mathcal{J}_i) & \text{if } A_i'' = A_i. \end{cases}$$

To find the new set A_{i+1}, we start with $j = n$, and test whether

$$(*) \quad A_i <_j \mathcal{J}_{i+1}^*((A_i \cap \{o_1, \ldots, o_{j-1}\}) \cup \{o_j\})$$

[2] If $A_i'' \setminus A_i = \emptyset$, then it is not really necessary to ask the expert because implications with empty right-hand side hold in any context.

holds. The index j is decreased until one of the following cases occurs:
(1) $j = 0$: In this case, $\mathcal{E}_{i+1}$ is the set of all concept extents and $\mathcal{J}_{i+1}$ the dual Duquenne-Guigues base of $\mathcal{K}$, and the algorithm stops.
(2) $(*)$ holds for $j > 0$: In this case, $A_{i+1} := \mathcal{J}_{i+1}^*((A_i \cap \{o_1, \ldots, o_{j-1}\}) \cup \{o_j\})$, and the iteration is continued.

4 Computing the hierarchy of least common subsumers

Given a finite set $\mathcal{O} := \{C_1, \ldots, C_n\}$ of concept descriptions, we are interested in the subsumption hierarchy between all least common subsumers of subsets of $\mathcal{C}$. For sets $A \subseteq \mathcal{O}$ of cardinality ≥ 2, we have already defined the notion $\mathsf{lcs}(A)$. We extend this notion to the empty set and singletons in the obvious way: $\mathsf{lcs}(\emptyset) := \bot$ and $\mathsf{lcs}(\{C_i\}) := C_i$.

Our goal is to compute the subsumption hierarchy between all concept descriptions $\mathsf{lcs}(A)$ for subsets A of $\mathcal{O}$ without explicitly computing all these least common subsumers. This is achieved by defining a formal context with objects $\mathcal{O}$ such that the concept lattice of this context is isomorphic to the subsumption hierarchy we are interested in.

Definition 8 *Given a DL language $\mathcal{L}$ and a finite set $\mathcal{O} := \{C_1, \ldots, C_n\}$ of $\mathcal{L}$-concept descriptions, the corresponding formal context $\mathcal{K}_\mathcal{L}(\mathcal{O}) = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ is defined as follows:*

$$\begin{aligned}
&\mathcal{O} := \{C_1, \ldots, C_n\},\\
&\mathcal{P} := \{D \mid D \text{ is an } \mathcal{L}\text{-concept description}\},\\
&\mathcal{S} := \{(C, D) \mid C \sqsubseteq D\}.
\end{aligned}$$

As an easy consequence of the definition of $\mathcal{K}_\mathcal{L}(\mathcal{O})$ and of the lcs, we obtain that the intent of a set $A \subseteq \mathcal{O}$ is closely related to the lcs of this set:

Lemma 9 *Let A_1, A_2 be subsets of $\mathcal{O}$.*

1. $A' = \{D \in \mathcal{P} \mid \mathsf{lcs}(A) \sqsubseteq D\}$;
2. $A_1' \subseteq A_2'$ *iff* $\mathsf{lcs}(A_2) \sqsubseteq \mathsf{lcs}(A_1)$;
3. *the implication* $A_1 \to A_2$ *holds in* $\mathcal{K}_\mathcal{L}(\mathcal{O})$ *iff* $\mathsf{lcs}(A_2) \sqsubseteq \mathsf{lcs}(A_1)$.

As an immediate consequence of 3. of this lemma, the dual Duquenne-Guigues base $\mathcal{J}$ of $\mathcal{K}_\mathcal{L}(\mathcal{O})$ yields a representation of all subsumption relationships of the form $\mathsf{lcs}(A_1) \sqsubseteq \mathsf{lcs}(A_2)$ for subsets A_1, A_2 of $\mathcal{O}$. Given this base $\mathcal{J}$, any question of the form "$\mathsf{lcs}(A_1) \sqsubseteq \mathsf{lcs}(A_2)$?" can then be answered in time linear in the size of $\mathcal{J} \cup \{A_1 \to A_2\}$. Another easy consequence of the lemma is that the concept lattice of $\mathcal{K}_\mathcal{L}(\mathcal{O})$ coincides with the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{O}$.

Theorem 10 *The concept lattice of $\mathcal{K}_\mathcal{L}(\mathcal{O})$ is isomorphic to the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{O}$.*

Proof. We define the mapping π from the formal concepts of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ to the set of (equivalence classes of) least common subsumers of subsets of $\mathcal{O}$ as follows:

$$\pi(A, B) := [\mathsf{lcs}(A)]_{\equiv}.$$

For formal concepts $(A_1, B_1), (A_2, B_2)$ we have $(A_1, B_1) \leq (A_2, B_2)$ iff $A_1 \subseteq A_2$ iff $A_2' \subseteq A_1'$ iff $\mathsf{lcs}(A_1) \sqsubseteq \mathsf{lcs}(A_2)$. As an easy consequence we obtain that π is order preserving (and thus also injective): $(A_1, B_1) \leq (A_2, B_2)$ iff $[\mathsf{lcs}(A_1)]_{\equiv} \sqsubseteq_{\equiv} [\mathsf{lcs}(A_2)]_{\equiv}$.

It remains to be shown that π is surjective as well. Let A be an arbitrary subset of $\mathcal{O}$. We must show that $[\mathsf{lcs}(A)]_{\equiv}$ can be obtained as an image of the mapping π. By Lemma 3, (A'', A') is a formal concept, and thus it is sufficient to show that $\mathsf{lcs}(A) \equiv \mathsf{lcs}(A'')$. Obviously, $A \subseteq A''$ implies $\mathsf{lcs}(A) \sqsubseteq \mathsf{lcs}(A'')$ (by definition of the lcs). To see the converse, note that, for all $C_i \in \mathcal{O}$, we have

$C_i \in A''$	iff	$C_i \sqsubseteq D$ for all $D \in A'$	(def. of $\cdot'$ and $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$)
	iff	$C_i \sqsubseteq D$ for all D such that $\mathsf{lcs}(A) \sqsubseteq D$	(Lemma 9)
	iff	$C_i \sqsubseteq \mathsf{lcs}(A)$.	(def. of the lcs)

Obviously, this implies $\mathsf{lcs}(A'') \sqsubseteq \mathsf{lcs}(A)$. □

If we want to apply Algorithm 7 to compute the concept lattice and the dual Duquenne-Guigues base, we need an "expert" for the context $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$. This expert must be able to answer the questions asked by the object exploration algorithm, i.e., given an object implication $A_1 \to A_2$, it must be able to decide whether this implication holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$. If the implication does not hold, it must be able to compute a counterexample, i.e., an attribute $p \in A_1' \setminus A_2'$.

If the language $\mathcal{L}$ is such that the lcs is computable and subsumption is decidable (which is, e.g., the case for $\mathcal{L} = \mathcal{ALE}$ or $\mathcal{L} = \mathcal{ALN}$), then we can implement such an expert.

Lemma 11 *Given a subsumption algorithm for $\mathcal{L}$ as well as an algorithm for computing the lcs of a finite set of $\mathcal{L}$-concept descriptions, these algorithms can be used to obtain an "expert" for the context $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$.*

Proof. First, we show how to decide whether a given object implication $A_1 \to A_2$ holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ or not. By Lemma 9, we know that $A_1 \to A_2$ holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ iff $\mathsf{lcs}(A_2) \sqsubseteq \mathsf{lcs}(A_1)$. Obviously, $\mathsf{lcs}(A_2) \sqsubseteq \mathsf{lcs}(A_1)$ iff $C_i \sqsubseteq \mathsf{lcs}(A_1)$ for all $C_i \in A_2$. Thus, to answer the question "$A_1 \to A_2$?", we first compute $\mathsf{lcs}(A_1)$ and then use the subsumption algorithm to test whether $C_i \sqsubseteq \mathsf{lcs}(A_1)$ holds for all $C_i \in A_2$.

Second, assume that $A_1 \to A_2$ does not hold in $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$, i.e., $\mathsf{lcs}(A_2) \not\sqsubseteq \mathsf{lcs}(A_1)$. We claim that $\mathsf{lcs}(A_1)$ is a counterexample, i.e., $\mathsf{lcs}(A_1) \in A_1'$ and $\mathsf{lcs}(A_1) \notin A_2'$. This is an immediate consequence of the facts that $A_i' = \{D \in \mathcal{P} \mid \mathsf{lcs}(A_i) \sqsubseteq D\}$ $(i = 1, 2)$ and that $\mathsf{lcs}(A_2) \not\sqsubseteq \mathsf{lcs}(A_1)$.

Of this counterexample, Algorithm 7 really needs the column corresponding to this attribute in the matrix corresponding to $\mathcal{S}$. This column can easily be computed using the subsumption algorithm: for each $C_i \in \mathcal{O}$, we use the subsumption algorithm to test whether $C_i \sqsubseteq \mathsf{lcs}(A)$ holds or not. □

Using this expert, an application of Algorithm 7 yields

- all extents of formal concepts of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$, and thus the concept lattice of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$, which coincides with the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{O}$ (by Theorem 10);
- the dual Duquenne-Guigues base of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$, which yields a compact representation of this hierarchy (by 3. of Lemma 9); and
- a finite subcontext of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ that has the same concept extents as $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ and the same $\cdot''$ operation on sets of objects.

Using the output of Algorithm 7, one can then employ the usual tools for drawing concept lattices [17] in order to present the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{O}$ to the knowledge engineer.

5 First experimental results

In the previous section, we have shown that the object exploration algorithm can be used to compute the hierarchy of least common subsumers of a given set of concept descriptions. What remains is to analyze whether object exploration really is a good approach for solving this task. Our reason for trying it in the first place was that computing this hierarchy is the same as computing a certain concept lattice (as shown above), and that Ganter's algorithm is known to be a very good method for doing this. The problem with this generic argument in favour of object exploration is, of course, that we consider a very specific context, and that it might well be that, for this context, object exploration is not the best thing to do. The first experimental results that will be described below are, however, rather encouraging.

We intend to use the bottom-up construction of knowledge bases in a chemical process engineering application [13, 14], where the knowledge base describes standard building blocks of process models (such as certain types of reactors). Currently, this knowledge base consists of about 600 definitions of building blocks, which we translate into $\mathcal{ALE}$-concept descriptions.

In order to test the object exploration algorithm, we have taken 7 descriptions of reactors of a similar type, which the process engineers considered to be good examples for generating a new concept. These descriptions were translated into $\mathcal{ALE}$-concept descriptions $R_1, \ldots, R_7$, and we applied the object exploration algorithm to this set of objects. The resulting concept lattice, which coincides with the hierarchy of all least common subsumers of subsets of $\mathcal{O} := \{R_1, \ldots, R_7\}$, is depicted in Figure 1. The top concept corresponds to the lcs obtained from the whole set of examples, and the bottom concept is the lcs obtained from the empty set, i.e., the description $\bot$. The node labelled $\mathsf{lcs}(i_1 \ldots i_m)$ corresponds to the formal concept with extent $R_{i_1}, \ldots, R_{i_m}$, and thus to $\mathsf{lcs}(R_{i_1}, \ldots, R_{i_m})$. Note that in many cases $\mathsf{lcs}(R_{i_1}, \ldots, R_{i_m})$ can also be obtained as the lcs of a strict subset of $\{R_{i_1}, \ldots, R_{i_m}\}$. This can be easily seen by using the least-upper-bound operation of the concept lattice. For example, $\mathsf{lcs}(R_i, R_7) = \mathsf{lcs}(R_1, \ldots, R_7)$ for all $i, 1 \leq i \leq 6$.

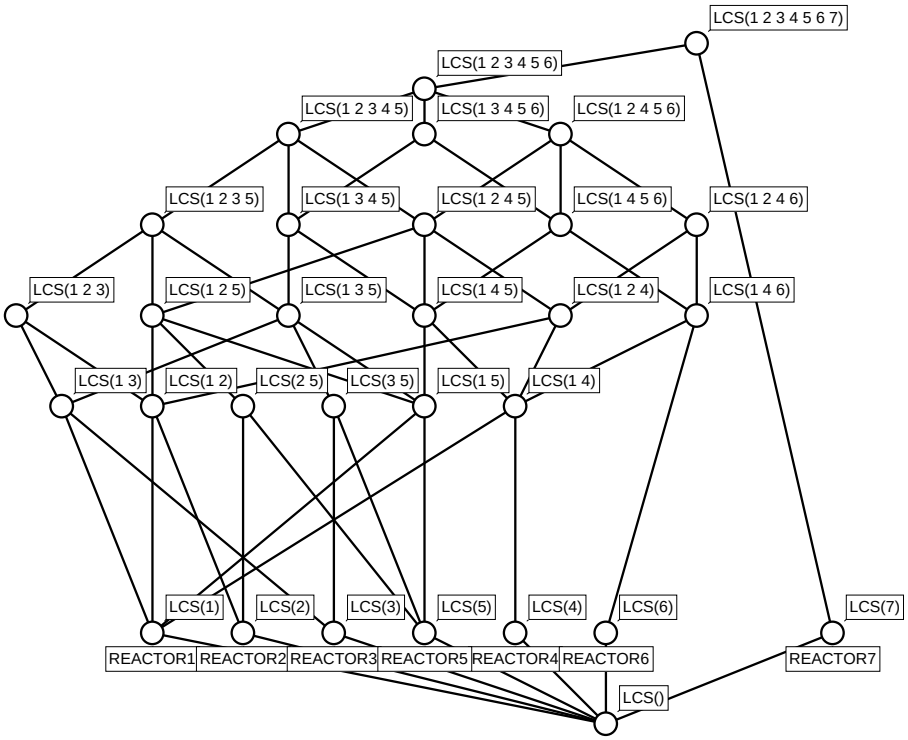

Fig. 1. The hierarchy of least common subsumers of seven reactor descriptions.

Statistical information: The Duquennes-Guigues base of the context consists of 15 implications, and the concept lattice of 30 formal concepts. If we subtract the trivial least common subsumers $\bot, R_1, \ldots, R_7$ as well as $\mathsf{lcs}(R_1, \ldots, R_7)$, which turned out to be equivalent to an already existing description, we end up with 21 candidates for new concepts. Of these 21 interesting least common subsumers, only 10 have explicitly been computed during the exploration.

During the calls of the "expert", 255 subsumption tests and 25 n-ary lcs operations have been executed. Because we re-used already computed least common subsumers, the 25 n-ary lcs operations only required 25 binary lcs operations. The number of counterexamples computed by the expert was also 25.

Finally, we measured the time needed for executing the interesting subtasks, namely computing the lcs, testing subsumption, and realizing the overhead introduced by the object exploration algorithm (e.g., computing the $\cdot''$ operation, the pseudo-hull, etc). It turned out that more than 84% of the time was used for computing least common subsumers, 15% for subsumption tests, and less than 1% for the rest. This shows that, at least for this small example, the exploration algorithm does not introduce any measurable overhead. The fact that computing the lcs needed a lot more time than testing subsumption is probably due to the

fact that we used a highly optimized subsumption algorithm [12], but only a first prototypical implementation of the lcs algorithm.

What can be learned from the concept lattice? Two important facts about the reactor descriptions can be read off immediately. First, there is no subsumption relationship between any of the 7 concepts since all singleton sets occur as extents. Second, Reactor 7 is quite different from the other reactors since its lcs with any of the others yields a very general concept description. Thus, it should not be used for generating new concepts together with the other ones. In fact, a closer look at R_7 revealed that, though it describes a reactor of a type similar to that of the other ones, this description was given on a completely different level of abstraction.

Next, let us consider the question of which of the least common subsumers occurring in the lattice appear to be good candidates for providing an interesting new concept. First, the lcs of the whole set is ruled out since it involves Reactor 7, which does not fit well with the other examples (see above). Second, in order to avoid concepts that are too specific, least common subsumers that do not cover more than half of the reactors should also be avoided. If we use these two criteria, then we are left with 9 candidates (the formal concepts with extents of cardinality 4, 5, and 6), which is a number of concepts that can well be inspected by the process engineer. In our example, the 5 least common subsumers on the first layer of these interesting candidates (the formal concepts with extents of cardinality 4) turned out to be the most promising, though this must still be checked in more detail with the process engineers.

6 Related and future work

The idea of using tools from formal concept analysis in description logics is not new. In [1], the attribute exploration algorithm was used to compute a small representation of the subsumption hierarchy of all conjunctions of concepts defined in a terminology, and in [15] this approach was extended such that it could handle both, conjunction and disjunction. There are, however, significant differences to the approach considered in the present paper. First, the formal context defined in [1, 15] is quite different from the one introduced above: its objects are pairs consisting of an interpretation $\mathcal{I}$ and an element of $\Delta^{\mathcal{I}}$. To be able to compute the counterexamples required by the attribute exploration algorithm, the subsumption algorithm had to be extended such that it computes appropriate finite countermodels [1]. Second, [1] was only interested in the Duquennes-Guigues base computed by the algorithm, and not in the concept lattice. In fact, in the experiments made with the approach introduced in [1], the base turned out to by usually rather small, whereas the lattice was very large (and thus it did not make sense to visualize it) [**?**].

In the future we will test the approach introduced in this paper with more examples from our chemical process engineering application. In particular, we will more closely analyze which of the least common subsumers yield concepts

that make sense in the application domain, with the goal to develop appropriate heuristics for suggesting good candidates based on the computed concept lattice. In addition, we will try to optimize the method, with the goal to avoid even more explicit lcs computations by using information from the partial implication base and the subcontext already computed.

References

1. F. Baader. Computing a Minimal Representation of the Subsumption Lattice of all Conjunctions of Concepts Defined in a Terminology. In *Proceedings of KRUSE'95*, 1995.
2. 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 *Proceedings of KI'98*, Springer LNCS 1504, 1998.
3. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proceedings of IJCAI'99*, Morgan Kaufmann, 1999.
4. A. Borgida and P. Patel-Schneider. A semantics and complete algorithm for subsumption in the Classic description logic. *J. Artificial Intelligence Research*, 1, 1994.
5. W.W. Cohen and H. Hirsh. Learning the Classic description logic: Theoretical and experimental results. In *Proceedings of KR'94*, Morgan Kaufmann, 1994.
6. F.M. Donini, M. Lenzerini, D. Nardi, B. Hollunder, W. Nutt, and A.M. Spaccamela. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 53, 1992.
7. W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 3, 1984.
8. V. Duquenne. Contextual implications between attributes and some representational properties for finite lattices. In *Beiträge zur Begriffsanalyse*, B.I. Wissenschaftsverlag, 1987.
9. M. Frazier and L. Pitt. Classic learning. *Machine Learning*, 25, 1996.
10. B. Ganter. Finding all closed sets: A general approach. *Order*, 8, 1991.
11. B. Ganter and R. Wille. *Formal Concept Analysis – Mathematical Foundations*, Springer–Verlag, 1999.
12. I. Horrocks. The FaCT system. In *Proceedings of Tableaux'98*, LNAI 1397, 1998.
13. W. Marquardt. Trends in computer-aided process modeling. *Computers and Chemical Engineering*, 20(6/7), 1996.
14. U. Sattler. *Terminological Knowledge Representation Systems in a Process Engineering Application*. PhD thesis, RWTH Aachen, 1998.
15. G. Stumme. The concept classification of a terminology extended by conjunction and disjunction. In *Proceedings of PRICAI'96*, Springer LNCS 1114, 1996.
16. G. Stumme and R. Wille. *Begriffliche Wissensverarbeitung — Methoden und Anwendungen*. Springer–Verlag, 2000.
17. F. Vogt. *Formale Begriffsanalyse mit C^{++}*. Springer–Verlag, 1996.