Applying Formal Concept Analysis to Description Logics

Franz Baader* and Baris Sertkaya**

Theoretical Computer Science, TU Dresden, D-01062 Dresden, Germany
{baader,sertkaya}@tcs.inf.tu-dresden.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 $\mathcal{C}$ as well as the hierarchy of all conjunctions of subsets of $\mathcal{C}$. These hierarchies can be used to support the bottom-up construction of description logic knowledge bases. The point is to compute the first hierarchy without having to compute the least common subsumer for all subsets of $\mathcal{C}$, and the second hierarchy without having to check all possible pairs of such conjunctions explicitly for subsumption. We will show that methods from formal concept analysis developed for computing concept lattices can be employed for this purpose.

1 Introduction

The notion of a concept as a collection of objects sharing certain properties is fundamental to both formal concept analysis (FCA) [19] and description logics (DL) [6]. However, the ways concepts are obtained and described differ significantly between these two research areas. In DL, 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 together with the roles are used to specify properties of *objects* occurring in the domain. In FCA, one starts with a *formal context*, which (in its simplest form) specifies which (atomic) properties are satisfied by the objects. A *formal concept* of such a context is a pair consisting of a set of objects (the extent) and a set of properties (the intent) such that the intent consists of exactly those properties that the objects in the extent have in common, and the extent consists of exactly those objects that share all the properties in the intent.

There are several differences between these approaches. First, in FCA one starts with a complete (but simple) extensional description of a domain, and then derives the formal concepts of this specific domain, which provide a useful structuring of the domain. In DL, the (intensional) definition of a concept is

* Partially supported by National ICT Australia Limited, Canberra Research Lab.
** Supported by the German Research Foundation (DFG, GRK 433/3).

given independently of a specific domain (interpretation), and the description of the objects is only partial. Second, in FCA the properties are atomic, and the intensional description of a formal concept (by its intent) is just a conjunction of such properties. DLs usually provide a rich language for the intensional definition of concepts, which can be seen as an expressive, yet decidable sublanguage of first-order predicate logic.

There have been several attempts toward bridging this gap between FCA and DL. For example, researchers from the FCA community have extended FCA to incorporate more complex properties [39, 31, 30, 33]. The present paper is concerned with bridging the gap from the other direction. We will describe how tools from FCA can be used to support the *bottom-up* construction of DL knowledge bases, as introduced in [8, 9]: 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 more limited extent) the msc has already been investigated in the literature [12, 15, 8, 9, 25, 24, 23, 4, 3, 2]. Here, we will address two problems that occur in the context of the bottom-up approach.

First, the methods for computing the least common subsumer are restricted to rather inexpressive descriptions logics not allowing for disjunction (and thus not allowing for full negation). In fact, for languages with disjunction, the lcs of a collection of concepts is just their disjunction, and nothing new can be learned from building it. In contrast, for languages without disjunction, the lcs extracts the "commonalities" of the given collection of concepts. Modern DL systems like FaCT [22] and RACER [21] are based on very expressive DLs, and there exist large knowledge bases that use this expressive power and can be processed by these systems [32, 36, 20]. In order to allow the user to re-use concepts defined in such existing knowledge bases and still support the user during the definition of new concepts with the bottom-up approach sketched above, we propose the following extended bottom-up approach. There is a (background) terminology $\mathcal{T}$ defined in an expressive DL $\mathcal{L}_2$. When defining new concepts, the user employs only a sublanguage $\mathcal{L}_1$ of $\mathcal{L}_2$, for which computing the lcs makes sense. However, in addition to primitive concepts and roles, the concept descriptions written in the DL $\mathcal{L}_1$ may also contain names of concepts defined in $\mathcal{T}$. When computing subsumption between such newly defined concepts, this is done w.r.t. $\mathcal{T}$, using a subsumption algorithm for the expressive DL $\mathcal{L}_2$. When computing the lcs of such concepts, we basically employ the algorithm for $\mathcal{L}_1$, but extend it such that it can take into account the subsumption relationships between conjunctions of

concept defined in $\mathcal{T}$. This is where FCA comes into play: it provides us with an efficient method for computing these relationships. To be more precise, given a TBox $\mathcal{T}$, we define a formal context whose properties are the defined concepts of $\mathcal{T}$, and whose concept lattice is isomorphic to the subsumption hierarchy we are interested in. Then, we employ the so-called "attribute exploration" algorithm [16, 19] to compute this concept lattice. We show that the "expert" for the context required by this algorithm can be realized by the subsumption algorithm for $\mathcal{L}_2$.

The second problem that we will address is that the choice of the examples is crucial for the quality of the result obtained by the bottom-up construction of concepts. 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. Thus, it would be good to have a tool that supports 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. Again, this is where methods of FCA can be utilized. We will define a formal context that has the property that its concept lattice is isomorphic to the *inverse* subsumption hierarchy of all least common subsumers of subsets of $\{C_1, \ldots, C_n\}$. The attribute exploration algorithm can again be used to compute this lattice. In fact, the "expert" required by the algorithm can be realized by the subsumption algorithm and the algorithm for computing the lcs.

In the next section, we introduce the relevant notions from description logics, and 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 attribute exploration algorithm. In Section 4 we show how this algorithm can be used to compute the hierarchy of all conjunctions of concepts defined in a terminology, and in Section 5 we do the same for the hierarchy of all least common subsumers of subsets of a given finite set of concepts. In Section 6, we describe some experimental results. In Section 7 we show that the approaches described in Subsection 4.2 and Section 5 are both instances of a more general approach. Section 8 concludes with some comments on possible future work.

This paper was intended to be an amalgamation of the results originally presented in [1] (see Subsection 4.1) and [10] (see Section 5). However, when reconsidering [1], we saw that in addition to the approach chosen there, one can also use another approach (see Subsection 4.2) to solve the problem addressed in [1]. When comparing this new approach with the approach employed in [10],

Table 1. Syntax and semantics of concept descriptions and definitions.

name of constructor	Syntax	Semantics	$\mathcal{ALC}$	$\mathcal{EL}$
top-concept	$\top$	$\Delta^{\mathcal{I}}$	x	x
bottom-concept	$\bot$	$\emptyset$	x	
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	x	
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	x	x
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	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	
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	x
concept definition	$A \equiv C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$	x	x

we saw that both are instances of a more general approach, which is described in Section 7. The experimental results described in Subsection 6.2 have already been published in [10], whereas the experimental results described in Subsection 6.1 have not been published before.

2 Description logics

For the purpose of this paper, it is sufficient to restrict the attention to the formalism for defining concepts (i.e., we need not introduce ABoxes, which describe 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 concept description language of the particular system. In this paper, we consider the DL $\mathcal{ALC}$ and its sublanguage $\mathcal{EL}$,[1] which allow for concept descriptions built from the indicated subsets of the constructors shown in Table 1. In this table, r stands for a role name, A for a concept name, and C, D for arbitrary concept descriptions. A *concept definition* (as shown in the last row of Table 1) assigns a concept name A to a complex description C. A finite set of such definitions is called a TBox iff it is acyclic (i.e., no definition refers, directly or indirectly, to the name it defines) and unambiguous (i.e., each name has at most one definition).

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 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. The interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$ iff it satisfies all its concept definitions, i.e., $A^{\mathcal{I}} = C^{\mathcal{I}}$ holds for all $A \equiv C$ in $\mathcal{T}$.

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

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 descriptions that are not syntactically equal). As usual, the quasi order $\sqsubseteq$ induces a partial order $\sqsubseteq_{\equiv}$ on the equivalence classes of concept descriptions:

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

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, we mean this induced partial order. In the presence of a TBox, subsumption must be computed w.r.t. this TBox. The concept description C_2 *subsumes* the concept description C_1 *w.r.t. the TBox* $\mathcal{T}$ ($C_1 \sqsubseteq_{\mathcal{T}} C_2$) iff $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$. All the other notions introduced above can be adapted in the obvious way to the case of subsumption w.r.t. a TBox.

Deciding subsumption between $\mathcal{EL}$-concept descriptions (without or with TBox) is polynomial [9, 5], whereas the subsumption problem for $\mathcal{ALC}$ (without or with TBox) is PSPACE-complete [35, 26].

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

Definition 1 *Given two concept descriptions C_1, C_2 in a DL $\mathcal{L}$, the concept description C of $\mathcal{L}$ is an lcs of C_1, C_2 in $\mathcal{L}$ ($C = \mathsf{lcs}_{\mathcal{L}}(C_1, C_2)$) iff (i) $C_i \sqsubseteq C$ for $i = 1, 2$, and (ii) C is the least concept description with this property, i.e., if C' satisfies $C_i \sqsubseteq C'$ for $= 1, 2$, then $C \sqsubseteq C'$.*

The lcs of n concept descriptions is obtained by iterating the application of the binary lcs: $\mathsf{lcs}_{\mathcal{L}}(C_1, \ldots, C_n) := lcs_{\mathcal{L}}(C_1, \ldots, lcs_{\mathcal{L}}(C_{n-1}, C_n) \cdots)$. 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. In [9], it is shown that the lcs of *two* $\mathcal{EL}$-concept descriptions always exists and that it can be computed in polynomial time; however, the size of the n-ary lcs can be exponential in the size of the input descriptions, and thus computing it may require exponential time.

As an example for the (binary) lcs in $\mathcal{EL}$, consider the $\mathcal{EL}$-concept descriptions

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

respectively describing parents having a child that is a male doctor, 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 $\mathcal{EL}$-concept description

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

It describes all parents having at least one son and at least one child that is a doctor. Note that $\mathsf{lcs}_{\mathcal{ALC}}(C, D) = C \sqcup D$.

In order to describe the lcs algorithm for $\mathcal{EL}$ in the general case, we need to introduce some notation. Let C be an $\mathcal{EL}$-concept description. Then $\mathsf{names}(C)$ denotes the set of concept names occurring in the top-level conjunction of C, $\mathsf{roles}(C)$ the set of role names occurring in an existential restriction on the top-level of C, and $\mathsf{restrict}_r(C)$ denotes the set of all concept descriptions occurring in an existential restriction on the role r on the top-level of C. In the above example, we have $\mathsf{names}(C) = \mathsf{names}(D) = \emptyset$, $\mathsf{restrict}_{\mathsf{has\text{-}child}}(C) = \{\mathsf{Male} \sqcap \mathsf{Doctor}\}$, and $\mathsf{restrict}_{\mathsf{has\text{-}child}}(D) = \{\mathsf{Male} \sqcap \mathsf{Mechanic}, \mathsf{Female} \sqcap \mathsf{Doctor}\}$.

Now, let C, D be $\mathcal{EL}$-concept descriptions. Then we have

$$\mathsf{lcs}_{\mathcal{EL}}(C, D) = \bigsqcap_{A \in \mathsf{names}(C) \cap \mathsf{names}(D)} A \;\sqcap \bigsqcap_{r \in \mathsf{roles}(C) \cap \mathsf{roles}(D)} \;\; \bigsqcap_{E \in \mathsf{restrict}_r(C), F \in \mathsf{restrict}_r(D)} \exists r.\mathsf{lcs}_{\mathcal{EL}}(E, F)$$

Here, the empty conjunction stands for the top concept $\top$. The recursive call of $\mathsf{lcs}_{\mathcal{EL}}$ is well-founded since the role depth (i.e., the maximal nesting of existential restrictions) of the concept descriptions in $\mathsf{restrict}_r(C)$ ($\mathsf{restrict}_r(D)$) is strictly smaller than the role depth of C (D).

If the $\mathcal{EL}$-concept descriptions C, D contain concept names that are defined in an $\mathcal{ALC}$-TBox $\mathcal{T}$, then it is currently not known how to compute their least common subsumer, i.e., the least $\mathcal{EL}$-concept description containing defined concepts of $\mathcal{T}$ that subsumes C and D w.r.t. $\mathcal{T}$. If we ignore the TBox by treating all concept names as primitive, and just compute the lcs of C, D, then we obtain a common subsumer of C, D also w.r.t. $\mathcal{T}$, but it need not be the least one. As a simple example, consider the TBox $\mathcal{T}$:

$$\begin{aligned}
\mathsf{NoSon} &\equiv \forall \mathsf{has\text{-}child}.\mathsf{Female},\\
\mathsf{NoDaughter} &\equiv \forall \mathsf{has\text{-}child}.\neg\mathsf{Female},\\
\mathsf{SonRichDoctor} &\equiv \forall \mathsf{has\text{-}child}.(\mathsf{Female} \sqcup (\mathsf{Doctor} \sqcap \mathsf{Rich}))\\
\mathsf{DaughterHappyDoctor} &\equiv \forall \mathsf{has\text{-}child}.(\neg\mathsf{Female} \sqcup (\mathsf{Doctor} \sqcap \mathsf{Happy}))\\
\mathsf{ChildrenDoctor} &\equiv \forall \mathsf{has\text{-}child}.\mathsf{Doctor}
\end{aligned}$$

and the $\mathcal{EL}$-concept descriptions

$$\begin{aligned}
C &:= \exists \mathsf{has\text{-}child}.(\mathsf{NoSon} \sqcap \mathsf{DaughterHappyDoctor}),\\
D &:= \exists \mathsf{has\text{-}child}.(\mathsf{NoDaughter} \sqcap \mathsf{SonRichDoctor}).
\end{aligned}$$

If we ignore the TBox, then we obtain the $\mathcal{EL}$-concept description

$$\exists \mathsf{has\text{-}child}.\top$$

as common subsumer of C, D. However, if we take into account that both $\mathsf{NoSon} \sqcap \mathsf{DaughterHappyDoctor}$ and $\mathsf{NoDaughter} \sqcap \mathsf{SonRichDoctor}$ are subsumed by $\mathsf{ChildrenDoctor}$, then we obtain the more specific common subsumer

$$\exists \mathsf{has\text{-}child}.\mathsf{ChildrenDoctor}.$$

This motivates our interest in computing the subsumption hierarchy of all conjunctions of concepts defined in a given TBox. Before we can show how this hierarchy can be computed using tools from FCA, we must introduce the relevant notions from FCA.

3 Formal concept analysis

We will introduce only those notions and results from FCA that are necessary for our purposes. Since it is the main FCA tool that we will employ, we will describe how the attribute exploration algorithm works. Note, however, that explaining why it works is beyond the scope of this paper (see [19] for more information on this and FCA in general).

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 (denoted by $\leq$ in the following) 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 [37], 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 [16, 19] has turned out to be an efficient approach for computing the concept lattice.

In one of the applications considered in this paper, we are faced with the *dual* situation: the set of attributes is the *infinite* set of all possible concept descriptions of the DL under consideration, and the set of objects is the *finite* collection of concept descriptions for which we want to compute the subsumption hierarchy of least common subsumers. To overcome this problem, one can either dualize the attribute exploration algorithm and the notions on which it depends, as done in [10]. In this paper, we follow an alternative approach: we 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 concept lattice of the dual context is obviously the dual of the concept lattice of the original context, i.e., the corresponding orderings on the formal concepts are inverses of each other.

Attribute exploration

Before we can describe the attribute exploration algorithm, we must introduce some notation. The most important notion for the algorithm is the one of an implication between sets of attributes. Intuitively, such an implication $B_1 \to B_2$ holds if any object satisfying all elements of B_1 also satisfies all elements of B_2.

Definition 4 *Let $\mathcal{K} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ be a formal context and B_1, B_2 be subsets of $\mathcal{P}$. The* implication $B_1 \to B_2$ *holds in $\mathcal{K}$ ($\mathcal{K} \models B_1 \to B_2$) iff $B_1' \subseteq B_2'$. An object o* violates *the implication $B_1 \to B_2$ iff $o \in B_1' \setminus B_2'$.*

It is easy to see that an implication $B_1 \to B_2$ holds in $\mathcal{K}$ iff $B_2 \subseteq B_1''$. In particular, given a set of attributes B, the implications $B \to B''$ and $B \to (B'' \setminus B)$ always hold in $\mathcal{K}$. We denote the set of all implications that hold in $\mathcal{K}$ by $Imp(\mathcal{K})$. This set can be very large, and thus one is interested in (small) generating sets.

Definition 5 *Let $\mathcal{J}$ be a set of implications, i.e., the elements of $\mathcal{J}$ are of the form $B_1 \to B_2$ for sets of attributes $B_1, B_2 \subseteq \mathcal{P}$. For a subset B of $\mathcal{P}$, the* implication hull *of B with respect to $\mathcal{J}$ is denoted by $\mathcal{J}(B)$. It is the smallest subset H of $\mathcal{P}$ such that*

- *$B \subseteq H$, and*
- *$B_1 \to B_2 \in \mathcal{J}$ and $B_1 \subseteq H$ imply $B_2 \subseteq H$.*

The set of implications generated by *$\mathcal{J}$ consists of all implications $B_1 \to B_2$ such that $B_2 \subseteq \mathcal{J}(B_1)$. It will be denoted by $Cons(\mathcal{J})$. We say that a set of implications $\mathcal{J}$ is a* base *of $Imp(\mathcal{K})$ iff $Cons(\mathcal{J}) = Imp(\mathcal{K})$ and no proper subset of $\mathcal{J}$ satisfies this property.*

If $\mathcal{J}$ is a base for $Imp(\mathcal{K})$, then it can be shown that $B'' = \mathcal{J}(B)$ for all $B \subseteq \mathcal{P}$. The implication hull $\mathcal{J}(B)$ of a set of attributes B can be computed in time linear in the size of $\mathcal{J}$ and B using, for example, methods for deciding satisfiability of sets of propositional Horn clauses [13]. Consequently, given a base $\mathcal{J}$ for $Imp(\mathcal{K})$, any question of the form "$B_1 \to B_2 \in Imp(\mathcal{K})$?" can be answered in time linear in the size of $\mathcal{J} \cup \{B_1 \to B_2\}$.

There may exist different implication bases of $Imp(\mathcal{K})$, and not all of them need to be of minimal cardinality. A base $\mathcal{J}$ of $Imp(\mathcal{K})$ is called *minimal base* iff no base of $Imp(\mathcal{K})$ has a cardinality smaller than the cardinality of $\mathcal{J}$. Duquenne and Guigues have given a description of such a minimal base [14]. Ganter's attribute exploration algorithm computes this minimal base as a by-product. In the following, we define the Duquenne-Guigues base and show how it can be computed using the attribute exploration algorithm.

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

- $B \subseteq H$, and
- $B_1 \to B_2 \in \mathcal{J}$ and $B_1 \subset H$ (strict subset) imply $B_2 \subseteq H$.

Given $\mathcal{J}$, the pseudo-hull of a set $B \subseteq \mathcal{P}$ can again be computed in time linear in the size of $\mathcal{J}$ and B (e.g., by adapting the algorithm in [13] appropriately). A subset B of $\mathcal{P}$ is called *pseudo-closed* in a formal context $\mathcal{K}$ iff $Imp(\mathcal{K})^*(B) = B$ and $B'' \neq B$.

Definition 6 *The* Duquenne-Guigues base *of a formal context $\mathcal{K}$ consists of all implications $B_1 \to B_2$ where $B_1 \subseteq \mathcal{P}$ is pseudo-closed in $\mathcal{K}$ and $B_2 = B_1'' \setminus B_1$.*

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 $Imp(\mathcal{K})$, and our goal is to avoid explicitly computing all of them.
2. The closure operator $B \mapsto B''$ is used, and computing it via $B \mapsto B' \mapsto B''$ may not be feasible for a context with a larger or infinite set of objects.

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 attributes $\mathcal{P} = \{p_1, \ldots, p_n\}$, say $p_1 < \cdots < p_n$. For all $j, 1 \leq j \leq n$, and $B_1, B_2 \subseteq \mathcal{P}$ we define

$$B_1 <_j B_2 \text{ iff } p_j \in B_2 \setminus B_1 \text{ and } B_1 \cap \{p_1, \ldots, p_{j-1}\} = B_2 \cap \{p_1, \ldots, p_{j-1}\}.$$

The lectic order $<$ is the union of all relations $<_j$ for $j = 1, \ldots, n$. It is a linear order on the powerset of $\mathcal{P}$. The lectic smallest subset of $\mathcal{P}$ 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 $\mathcal{O}_i \subseteq \mathcal{O}$, $\mathcal{P}_i = \mathcal{P}$, and $\mathcal{S}_i = \mathcal{S} \cap (\mathcal{O}_i \times \mathcal{P})$. The closure operator $B \mapsto B''$ 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 $B \to B'' \setminus B$ really holds in the whole context $\mathcal{K}$. If it does not hold, the expert must provide a counterexample, i.e., an object o from $\mathcal{O} \setminus \mathcal{O}_i$ that violates the implication. This object is then added to the current context. Technically, this means that the expert must provide an object o, and must say which of the attributes in $\mathcal{P}$ are satisfied for this object.

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

Algorithm 7 (Attribute exploration)
Initialization: *One starts with the empty set of implications, i.e., $\mathcal{J}_0 := \emptyset$, the empty set of concept intents $\mathcal{C}_0 := \emptyset$, and the empty subcontext $\mathcal{K}_0$ of $\mathcal{K}$, i.e., $\mathcal{O}_0 := \emptyset$. The lectic smallest subset of $\mathcal{P}$ is $B_0 := \emptyset$.*

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

If the answer is "no", then let $o_i \in \mathcal{O}$ be the counterexample provided by the expert. Let $B_{i+1} := B_i$, $\mathcal{J}_{i+1} := \mathcal{J}_i$, and let $\mathcal{K}_{i+1}$ be the subcontext of $\mathcal{K}$ with $\mathcal{O}_{i+1} := \mathcal{O}_i \cup \{o_i\}$. The iteration continues with $\mathcal{K}_{i+1}$, $\mathcal{J}_{i+1}$, $\mathcal{C}_{i+1}$, and B_{i+1}.

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

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

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

$$(*) \quad B_i <_j \mathcal{J}_{i+1}^*((B_i \cap \{p_1, \ldots, p_{j-1}\}) \cup \{p_j\})$$

holds. The index j is decreased until one of the following cases occurs:
(1) $j = 0$: In this case, $\mathcal{C}_{i+1}$ is the set of all concept intents and $\mathcal{J}_{i+1}$ the Duquenne-Guigues base of $\mathcal{K}$, and the algorithm stops.
(2) $(*)$ holds for $j > 0$: In this case, $B_{i+1} := \mathcal{J}_{i+1}^*((B_i \cap \{p_1, \ldots, p_{j-1}\}) \cup \{p_j\})$, and the iteration is continued.

4 Subsumption between conjunctions of concepts

Most of the results that we will show are independent of the DL under consideration. The only restriction that we need is that it allows for conjunction.

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

In the following, let $\mathcal{T}$ be a fixed TBox, and let $p_1, \ldots, p_n$ be the concepts defined in $\mathcal{T}$. We are interested in representing all subsumption relationships (w.r.t. $\mathcal{T}$) between finite conjunctions of these defined concepts. In order to employ tools from FCA for this purpose, we want to define a formal context such that the concept lattice of this context is isomorphic to the subsumption hierarchy we are interested in. In this section, we will describe two possible ways for defining such a context. The first one (which was first described in [1]) uses interpretations as objects, whereas the second one uses concepts as objects. The advantage of the second approach is that one can use an arbitrary subsumption algorithm for the DL under consideration as expert. Thus, one can, for example, use highly optimized DL systems like FaCT and RACER for this purpose. In contrast, the first approach requires an extended subsumption algorithm. Though this extended algorithm exists for most of the standard DLs (in particular, for $\mathcal{ALC}$), one cannot use standard implementations.

4.1 The semantic context

The idea underlying the following definition is that a counterexample to a subsumption relationship $C \sqsubseteq_{\mathcal{T}} D$ is a model $\mathcal{I}$ of $\mathcal{T}$ together with an element $d \in \Delta^{\mathcal{I}}$ such that $d \in C^{\mathcal{I}} \setminus D^{\mathcal{I}}$. The objects of the context are all possible such counterexamples.

Definition 8 *The context $\mathcal{K}_{\mathcal{T}} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ is defined as follows:*

$$\begin{aligned}
\mathcal{O} &:= \{(\mathcal{I}, d) \mid \mathcal{I} \text{ is a model of } \mathcal{T} \text{ and } d \in \Delta^{\mathcal{I}}\},\\
\mathcal{P} &:= \{p_1, \ldots, p_n\},\\
\mathcal{S} &:= \{((\mathcal{I}, d), p) \mid d \in p^{\mathcal{I}}\}.
\end{aligned}$$

For a nonempty subset $B = \{p_{i_1}, \ldots, p_{i_r}\}$ of $\mathcal{P}$, we denote the conjunction $p_{i_1} \sqcap \ldots \sqcap p_{i_r}$ by $\sqcap B$. For the empty set, we define $\sqcap\emptyset := \top$.

Lemma 9 *Let B_1, B_2 be subsets of $\mathcal{P}$. The implication $B_1 \to B_2$ holds in $\mathcal{K}_{\mathcal{T}}$ iff $\sqcap B_1 \sqsubseteq_{\mathcal{T}} \sqcap B_2$.*

Proof. Assume that $B_1 \to B_2$ does not hold in $\mathcal{K}_{\mathcal{T}}$. This is the case iff there exists an object $(\mathcal{I}, d) \in B_1' \setminus B_2'$. By definition of $\mathcal{S}$ and of the operator $B \mapsto B'$, this means that (1) $d \in p^{\mathcal{I}}$ for all $p \in B_1$, and (2) there exists $p' \in B_2$ such that $d \notin p'^{\mathcal{I}}$. By the semantics of the conjunction operator, (1) is equivalent to $d \in (\sqcap B_1)^{\mathcal{I}}$, and (2) is equivalent to $d \notin (\sqcap B_2)^{\mathcal{I}}$. Since $\mathcal{I}$ is a model of $\mathcal{T}$, this shows that the subsumption relationship $\sqcap B_1 \sqsubseteq_{\mathcal{T}} \sqcap B_2$ does not hold. Obviously, all of the conclusions we have made are reversible. □

Thus, the Duquenne-Guigues base $\mathcal{L}$ of $\mathcal{K}_{\mathcal{T}}$ also yields a representation of all subsumption relationships of the form $\sqcap B_1 \sqsubseteq_{\mathcal{T}} \sqcap B_2$ for subsets B_1, B_2 of $\mathcal{P}$. As mentioned in Section 3, any question "$\sqcap B_1 \sqsubseteq_{\mathcal{T}} \sqcap B_2$?" can then be answered in time linear in the size of $\mathcal{L} \cup \{B_1 \to B_2\}$.

Theorem 10 *The concept lattice of the context $\mathcal{K}_\mathcal{T}$ is isomorphic to the subsumption hierarchy of all conjunctions of subsets of $\mathcal{P}$ w.r.t. $\mathcal{T}$.*

Proof. In order to obtain an appropriate isomorphism, we define a mapping π from the formal concepts of the context $\mathcal{K}_\mathcal{T}$ to the set of all (equivalence classes of) conjunctions of subsets of $\mathcal{P}$ as follows:

$$\pi(A, B) = [\sqcap B]_\equiv.$$

For formal concepts $(A_1, B_1), (A_2, B_2)$ of $\mathcal{K}_\mathcal{T}$ we have $(A_1, B_1) \leq (A_2, B_2)$ iff $B_2 \subseteq B_1$. Since B_1 is the intent of the formal concept (A_1, B_1), we have $B_1 = A_1' = A_1''' = B_1''$, and thus $B_2 \subseteq B_1$ iff $B_2 \subseteq B_1''$ iff the implication $B_1 \rightarrow B_2$ holds in $\mathcal{K}_\mathcal{T}$ iff $\sqcap B_1 \sqsubseteq_\mathcal{T} \sqcap B_2$. Overall, we have thus shown that π is an order embedding (and thus injective): $(A_1, B_1) \leq (A_2, B_2)$ iff $[\sqcap B_1]_\equiv \sqsubseteq_\equiv [\sqcap B_2]_\equiv$.

It remains to be shown that π is surjective as well. Let B be an arbitrary subset of $\mathcal{P}$. We must show that $[\sqcap B]_\equiv$ can be obtained as an image under the mapping π. We know that (B', B'') is a formal concept of $\mathcal{K}_\mathcal{T}$, and thus it is sufficient to show that $\pi(B', B'') = [\sqcap B]_\equiv$, i.e., $\sqcap(B'') \equiv_\mathcal{T} \sqcap B$. Obviously, $B \subseteq B''$ implies $\sqcap(B'') \sqsubseteq_\mathcal{T} \sqcap B$. Conversely, the implication $B \rightarrow B''$ holds in $\mathcal{K}_\mathcal{T}$, and thus Lemma 9 yields $\sqcap B \sqsubseteq_\mathcal{T} \sqcap B''$. □

If we want to apply Algorithm 7 to compute the concept lattice and the Duquenne-Guigues base, we need an "expert" for the context $\mathcal{K}_\mathcal{T}$. This expert must be able to answer the questions asked by the attribute exploration algorithm, i.e., given an implication $B_1 \rightarrow B_2$, it must be able to decide whether this implication holds in $\mathcal{K}_\mathcal{T}$. If the implication does not hold, it must be able to compute a counterexample, i.e., an object $o \in B_1' \setminus B_2'$.

By Lemma 9, $B_1 \rightarrow B_2$ holds in $\mathcal{K}_\mathcal{T}$ iff $\sqcap B_1 \sqsubseteq_\mathcal{T} \sqcap B_2$. Thus, validity of implications in $\mathcal{K}_\mathcal{T}$ can be decided using a standard subsumption algorithm. A counterexample to the implication $B_1 \rightarrow B_2$ is a pair $(d, \mathcal{I}) \in \mathcal{O}$ such that $d \in (\sqcap B_1)^\mathcal{I} \setminus (\sqcap B_2)^\mathcal{I}$. Since the usual tableau-based subsumption algorithms [11] in principle try to generate finite countermodels to subsumption relationships, they can usually be extended such that they yield such an object in case the subsumption relationship does not hold. In [1], this is explicitly shown for the DL $\mathcal{ALC}$.

Proposition 11 *Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox. The tableau-based subsumption algorithm for $\mathcal{ALC}$ can be extended such that it functions as an "expert" for the context $\mathcal{K}_\mathcal{T}$ without increasing its worst-case complexity of PSPACE.*

However, the highly optimized algorithms in systems like FaCT and RACER do not produce such countermodels as output. For this reason, we are interested in a context that has the same attributes and the same concept lattice (up to isomorphism), but for which a standard subsumption algorithm can function as an expert.

4.2 The syntactic context

The intuition underlying this context is that the subsumption relationship $C \sqsubseteq_\mathcal{T} D$ does not hold iff there is a concept E such that $E \sqsubseteq_\mathcal{T} C$ and $E \not\sqsubseteq_\mathcal{T} D$.

Definition 12 *The context $\mathcal{K}'_\mathcal{T} = (\mathcal{O}', \mathcal{P}, \mathcal{S}')$ is defined as follows:*

$$\mathcal{O}' := \{E \mid E \text{ is a concept description of the DL under consideration}\};$$
$$\mathcal{P} := \{p_1, \ldots, p_n\},$$
$$\mathcal{S}' := \{(E, p) \mid E \sqsubseteq_\mathcal{T} p\}.$$

The context $\mathcal{K}'_\mathcal{T}$ satisfies the analogs of Lemma 9 and Theorem 10.

Lemma 13 *Let B_1, B_2 be subsets of $\mathcal{P}$. The implication $B_1 \to B_2$ holds in $\mathcal{K}'_\mathcal{T}$ iff $\sqcap B_1 \sqsubseteq_\mathcal{T} \sqcap B_2$.*

Proof. First, we prove the *only if* direction. If the implication $B_1 \to B_2$ holds in $\mathcal{K}'_\mathcal{T}$, then this means that the following holds for all objects $E \in \mathcal{O}'$: if $E \sqsubseteq_\mathcal{T} p$ holds for all $p \in B_1$, then $E \sqsubseteq_\mathcal{T} p$ also holds for all $p \in B_2$. Thus, if we take $\sqcap B_1$ as object E, we obviously have $\sqcap B_1 \sqsubseteq_\mathcal{T} p$ for all $p \in B_1$, and thus $\sqcap B_1 \sqsubseteq_\mathcal{T} p$ for all $p \in B_2$, which shows $\sqcap B_1 \sqsubseteq_\mathcal{T} \sqcap B_2$.

Second, we prove the *if* direction. If $\sqcap B_1 \sqsubseteq_\mathcal{T} \sqcap B_2$, then any object E satisfying $E \sqsubseteq_\mathcal{T} \sqcap B_1$ also satisfies $E \sqsubseteq_\mathcal{T} \sqcap B_2$ by the transitivity of the subsumption relation. Consequently, if E is a subconcept of all concepts in B_1, then it is also a subconcept of all concepts in B_2, i.e., if E satisfies all attributes in B_1, it also satisfies all attributes in B_2. This shows that the implication $B_1 \to B_2$ holds in $\mathcal{K}'_\mathcal{T}$. □

The proof of the following theorem is identical to the proof of Theorem 10.

Theorem 14 *The concept lattice of the context $\mathcal{K}'_\mathcal{T}$ is isomorphic to the subsumption hierarchy of all conjunctions of subsets of $\mathcal{P}$ w.r.t. $\mathcal{T}$.*

Again, attribute exploration can be used to compute the concept lattice since any standard subsumption algorithm for the DL under consideration can be used as an expert for $\mathcal{K}'_\mathcal{T}$.

Proposition 15 *Any decision procedure for subsumption functions as an expert for the context $\mathcal{K}'_\mathcal{T}$.*

Proof. The attribute exploration algorithm asks questions of the form "$B_1 \to B_2$?" By Lemma 13, we can translate these questions into subsumption questions of the form "$\sqcap B_1 \sqsubseteq_\mathcal{T} \sqcap B_2$?" Obviously, any decision procedure for subsumption can answer these questions correctly.

Now, assume that $B_1 \to B_2$ does *not* hold in $\mathcal{K}'_\mathcal{T}$, i.e., $\sqcap B_1 \not\sqsubseteq_\mathcal{T} \sqcap B_2$. We claim that $\sqcap B_1$ is a counterexample, i.e., $\sqcap B_1 \in B'_1$, but $\sqcap B_1 \notin B'_2$. This is an immediate consequence of the facts that $B'_i = \{E \mid E \sqsubseteq_\mathcal{T} \sqcap B_i\}$ $(i = 1, 2)$ and that $\sqcap B_1 \sqsubseteq_\mathcal{T} \sqcap B_1$ and $\sqcap B_1 \not\sqsubseteq_\mathcal{T} \sqcap B_2$. □

5 Computing the hierarchy of least common subsumers

In the following, we assume that in the DL $\mathcal{L}$ under consideration the lcs always exists and can effectively be computed.

Given a finite set $\mathcal{C} := \{C_1, \ldots, C_n\}$ of concept descriptions, we are interested in the subsumption hierarchy between all least common subsumers of subsets of $\mathcal{C}$. For sets $B \subseteq \mathcal{C}$ of cardinality ≥ 2, we have already defined the notion $\mathsf{lcs}(B)$. 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 $\mathsf{lcs}(B)$ for subsets B of $\mathcal{C}$ without explicitly computing all these least common subsumers. This is again achieved by defining a formal context (with attribute set $\mathcal{C}$) such that the concept lattice of this context is isomorphic to the subsumption hierarchy we are interested in. The following context is similar to the syntactic context defined in the previous section. The main difference is the definition of the incidence relation, where subsumption is used in the opposite direction.

Definition 16 *Given a DL language $\mathcal{L}$ and a finite set $\mathcal{C} := \{C_1, \ldots, C_n\}$ of $\mathcal{L}$-concept descriptions, the corresponding formal context $\mathcal{K}_{\mathcal{L}}(\mathcal{C}) = (\mathcal{O}'', \mathcal{P}'', \mathcal{S}'')$ is defined as follows:*

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

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

Lemma 17 *Let B, B_1, B_2 be subsets of $\mathcal{P}''$.*

1. $B' = \{D \in \mathcal{O}'' \mid \mathsf{lcs}(B) \sqsubseteq D\}$.
2. $B_1' \subseteq B_2'$ *iff* $\mathsf{lcs}(B_2) \sqsubseteq \mathsf{lcs}(B_1)$.

Proof. First, let B be a subset of $\mathcal{P}''$. We have $D \in B'$ iff $C \sqsubseteq D$ for all $C \in B$ iff $\mathsf{lcs}(B) \sqsubseteq D$.

Second, by 1. we have $B_1' \subseteq B_2'$ iff $\mathsf{lcs}(B_1) \sqsubseteq D$ implies $\mathsf{lcs}(B_2) \sqsubseteq D$ for all $D \in \mathcal{O}''$. Thus, if $B_1' \subseteq B_2'$, then $\mathsf{lcs}(B_1) \sqsubseteq \mathsf{lcs}(B_1)$ yields $\mathsf{lcs}(B_2) \sqsubseteq \mathsf{lcs}(B_1)$. Conversely, if $\mathsf{lcs}(B_2) \sqsubseteq \mathsf{lcs}(B_1)$, then $\mathsf{lcs}(B_1) \sqsubseteq D$ obviously implies $\mathsf{lcs}(B_2) \sqsubseteq D$ for arbitrary concepts D. □

Now, we can again show that implications correspond to subsumption relationships between the corresponding least common subsumers.

Lemma 18 *Let B_1, B_2 be subsets of $\mathcal{P}''$. The implication $B_1 \rightarrow B_2$ holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$ iff $\mathsf{lcs}(B_2) \sqsubseteq \mathsf{lcs}(B_1)$.*

Proof. $B_1 \to B_2$ holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$
iff $B_1' \subseteq B_2'$ (by definition)
iff $\mathsf{lcs}(B_2) \sqsubseteq \mathsf{lcs}(B_1)$ (by 2. of the above lemma).

□

As an immediate consequence of this lemma, the Duquenne-Guigues base $\mathcal{J}$ of $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$ yields a representation of all subsumption relationships of the form $\mathsf{lcs}(B_1) \sqsubseteq \mathsf{lcs}(B_2)$ for subsets B_1, B_2 of $\mathcal{O}$. Given this base $\mathcal{J}$, any question of the form "$\mathsf{lcs}(B_1) \sqsubseteq \mathsf{lcs}(B_2)$?" can then be answered in time linear in the size of $\mathcal{J} \cup \{B_1 \to B_2\}$. Another easy consequence of the lemma is that the concept lattice of $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$ coincides with the inverse subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$. The proof of this fact (which we include for the sake of completeness) is very similar to the proof of Theorem 10.

Theorem 19 *The concept lattice of $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$ is isomorphic to the inverse subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$.*

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

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

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

$$(A_1, B_1) \leq (A_2, B_2) \text{ iff } [\mathsf{lcs}(B_1)]_{\equiv} \sqsupseteq_{\equiv} [\mathsf{lcs}(B_2)]_{\equiv}.$$

It remains to be shown that π is surjective as well. Let B be an arbitrary subset of $\mathcal{C} = \mathcal{P}''$. We must show that $[\mathsf{lcs}(B)]_{\equiv}$ can be obtained as an image under the mapping π. By the dual of Lemma 3, (B', B'') is a formal concept, and thus it is sufficient to show that $\mathsf{lcs}(B) \equiv \mathsf{lcs}(B'')$. Obviously, $B \subseteq B''$ implies $\mathsf{lcs}(B) \sqsubseteq \mathsf{lcs}(B'')$ (by definition of the lcs). Conversely, the implication $B \to B''$ holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$, and thus $\mathsf{lcs}(B'') \sqsubseteq \mathsf{lcs}(B)$ (by Lemma 18). □

If we want to apply Algorithm 7 to compute the concept lattice and the Duquenne-Guigues base, we need an "expert" for the context $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$. This expert must be able to answer the questions asked by the attribute exploration algorithm, i.e., given an implication $B_1 \to B_2$, it must be able to decide whether this implication holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$. If the implication does not hold, it must be able to compute a counterexample, i.e., an object $o \in B_1' \setminus B_2'$.

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

Proposition 20 *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 $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$.*

Proof. First, we show how to decide whether a given implication $B_1 \to B_2$ holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$ or not. By Lemma 18, we know that $B_1 \to B_2$ holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$ iff $\mathsf{lcs}(B_2) \sqsubseteq \mathsf{lcs}(B_1)$. Obviously, $\mathsf{lcs}(B_2) \sqsubseteq \mathsf{lcs}(B_1)$ iff $C_i \sqsubseteq \mathsf{lcs}(B_1)$ for all $C_i \in B_2$. Thus, to answer the question "$B_1 \to B_2$?", we first compute $\mathsf{lcs}(B_1)$ and then use the subsumption algorithm to test whether $C_i \sqsubseteq \mathsf{lcs}(B_1)$ holds for all $C_i \in B_2$.

Second, assume that $B_1 \to B_2$ does not hold in $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$, i.e., $\mathsf{lcs}(B_2) \not\sqsubseteq \mathsf{lcs}(B_1)$. We claim that $\mathsf{lcs}(B_1)$ is a counterexample, i.e., $\mathsf{lcs}(B_1) \in B_1'$ and $\mathsf{lcs}(B_1) \notin B_2'$. This is an immediate consequence of the facts that $B_i' = \{D \in \mathcal{O}'' \mid \mathsf{lcs}(B_i) \sqsubseteq D\}$ $(i = 1, 2)$ and that $\mathsf{lcs}(B_1) \sqsubseteq \mathsf{lcs}(B_1)$ and $\mathsf{lcs}(B_2) \not\sqsubseteq \mathsf{lcs}(B_1)$.

Of this counterexample, Algorithm 7 really needs the row corresponding to this object in the matrix corresponding to $\mathcal{S}''$. This row can easily be computed using the subsumption algorithm: for each $C_i \in \mathcal{C} = \mathcal{P}''$, we use the subsumption algorithm to test whether $C_i \sqsubseteq \mathsf{lcs}(B_1)$ holds or not. □

Using this expert, an application of Algorithm 7 yields

- all intents of formal concepts of $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$, and thus the concept lattice of $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$, which coincides with the inverse subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$ (by Theorem 19);
- the Duquenne-Guigues base of $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$, which yields a compact representation of this hierarchy (by Lemma 18); and
- a finite subcontext of $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$ that has the same concept intents as $\mathcal{K}_{\mathcal{L}}(\mathcal{C})$ and the same $\cdot''$ operation on sets of attributes.

Using the output of Algorithm 7, one can then employ the usual tools for drawing concept lattices [38] in order to present the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$ to the knowledge engineer.

6 Some experimental results

In the previous two sections, we have shown that the attribute exploration algorithm can be used to compute the hierarchy of least common subsumers of a given set of concept descriptions and the hierarchy of all conjunctions of concepts defined in a terminology. What remains is to analyze whether attribute 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 attribute exploration is known to be a very good method for doing this. The problem with this generic argument in favor of attribute exploration is, of course, that we consider a very specific context, and that it might well be that, for this context, attribute exploration is not the best thing to do.

6.1 Results for computing the hierarchy of conjunctions of defined concepts

In [29], the approach for computing this hierarchy based on the semantic context (see Subsection 4.1) was implemented and then evaluated on randomly generated

$\mathcal{ALC}$ TBoxes. For each TBox size (where size is the number of defined concepts), at least 50 different TBoxes were generated and attribute exploration was applied to the semantic context induced by these TBoxes. The main observations made during these experiments are the following:

1. For TBoxes of size ≥ 30, the computation of the full hierarchy in all cases took longer than the time out of 5 minutes imposed on each experiment.
2. The size of the Duquenne-Guigues base was very small compared with the number of computed counterexamples and the number of formal concepts of the context. For example, for TBoxes of size 20, there was an average of 13 implications in the base, 850 counterexamples, and 13500 formal concepts.
3. The overhead of the "concept analysis part" of the algorithm was considerable. Although calls to the expert are quite expensive (since the expert is realized by a PSPACE algorithm, the extended subsumption algorithm) less than 60% of the time was spent by the expert.
4. The time required by the extended subsumption algorithm (computing a counterexample) was compared with the time required by a normal subsumption algorithm (answering only "yes" or "no" to subsumption queries). It turned out that for all TBox sizes the runtime of the extended subsumption algorithm was 2.5 times the runtime of the normal subsumption algorithm.
5. The subsumption hierarchy of all conjunctions of concepts defined in a TBox can, of course, also be computed by extending the TBox by a new definition for each such conjunction. However, even if one employs the sophisticated optimization techniques described in [7] to compute the subsumption hierarchy of this extended TBox, the number of calls to the subsumption algorithm is much larger than the number of expert calls during attribute exploration of the semantic context of the unextended TBox. For example, for TBoxes of size 10, attribute exploration required an average of 75 calls of the expert, whereas computing the subsumption hierarchy of the extended TBoxes required 19700 calls of the normal subsumption algorithm.

A few comments regarding these results are in order. First, there are two reasons why TBoxes with more than 30 defined concepts could not be handled in reasonable time. One the one hand, the extended subsumption algorithm was implemented in a naive and almost unoptimized way, and thus could not handle the large concept descriptions (obtained by unfolding the TBox definitions) in an efficient way. On the other hand, for large TBoxes, the size of the concept lattice was huge (and thus a lot of time was spent in the "concept analysis part" of the algorithm). The reason for the huge number of formal concepts compared to the number of implications is probably that the dependencies between the different attributes was quite low in the semantic contexts induced by the randomly generated TBoxes. By varying the probability of using an already defined concept in the definition of a new concept, we were able to generate TBoxes with more or less dependencies between the attributes. However, even with high dependency between attributes, the concept lattice was quite large. As long as this is the case, the "concept analysis part" of the algorithm will require a considerable amount of time.

The constant factor of 2.5 between the runtime of the expert and the runtime of the normal subsumption algorithm is consistent with the theoretical result that both algorithms belong to the same complexity class (PSPACE). For practical purposes, the situation is, however, worse than indicated by this rather small factor. The algorithms we tested were both naive, almost unoptimized implementations. For the normal subsumption algorithm there are now highly optimized implementations that are several orders of magnitude better than the naive implementation, whereas there is no such optimized implementation available for the extended algorithm. For this reason, using the syntactic context (which just requires a normal subsumption algorithm as expert) appears to be the better option (though this must still be verified in experiments).

In spite of the overhead caused by the expert and the "concept analysis part" of the algorithm, attribute exploration is much better than the naive approach of extending the TBox by a new definition for each conjunction of defined concepts. Thus, the generic argument in favor of attribute exploration is indeed supported by our experiments. As we will see in the next subsection, the same is true for the problem of computing the hierarchy of least common subsumers.

6.2 Results for computing the hierarchy of least common subsumers

We have used the bottom-up construction of knowledge bases in a chemical process engineering application [27, 34, 28], where the knowledge base describes standard building blocks of process models (such as certain types of reactors). When we performed the experiments, this knowledge base consisted of about 600 definitions of building blocks.

In order to test the attribute 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 concept descriptions $R_1, \ldots, R_7$, and we applied the attribute exploration algorithm to this set of attributes. The resulting hierarchy of all least common subsumers of subsets of $\mathcal{C} := \{R_1, \ldots, R_7\}$ is depicted in Figure 1. The concept on the top corresponds to the lcs obtained from the whole set of examples, and the concept at the bottom is the lcs obtained from the empty set, i.e., the description $\bot$. The node labeled $\mathsf{lcs}(i_1 \ldots i_m)$ corresponds to the formal concept with intent $R_{i_1}, \ldots, R_{i_m}$, 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 (inverse) concept lattice. For example, $\mathsf{lcs}(R_i, R_7) = \mathsf{lcs}(R_1, \ldots, R_7)$ for all $i, 1 \leq i \leq 6$.

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.

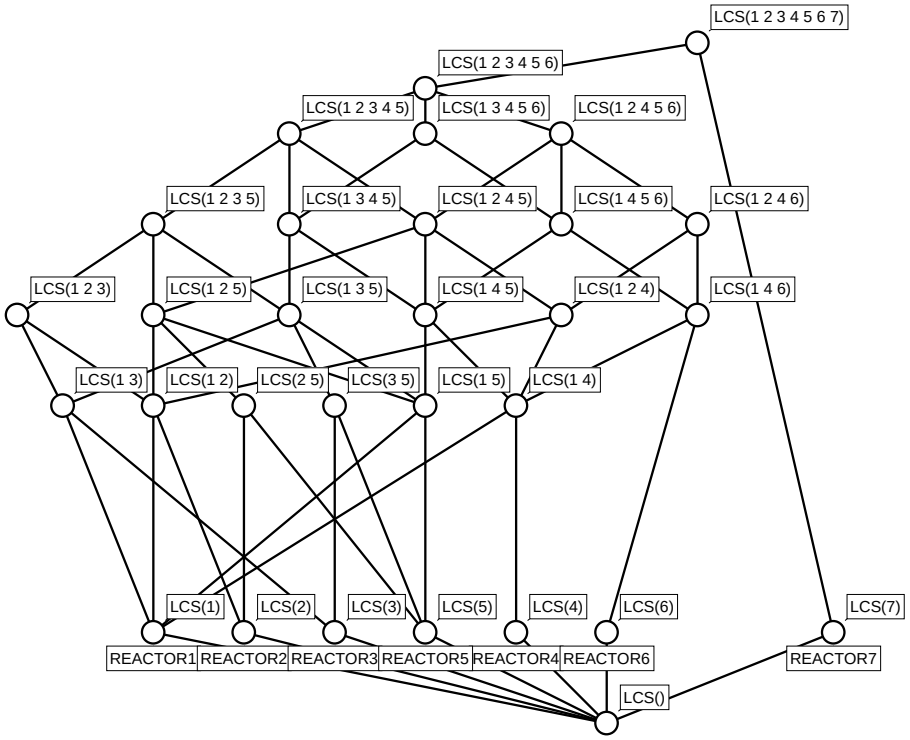

Fig. 1. The hierarchy of least common subsumers of seven reactor descriptions.

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 attribute 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.[3] The fact that computing the lcs needed a lot more time than testing subsumption is probably due to

[3] This is in strong contrast to the experimental results described in Subsection 6.1. The main difference appears to be that, for the present context, the concept lattice and the number of counterexamples is not much larger than the number of implications in the base.

the fact that we used a highly optimized subsumption algorithm [22], 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 intents. 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 intents 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 intents of cardinality 4) were consider by the process engineers to be the most interesting new concepts among the 9 candidates.

7 A more abstract point of view

The results and proofs presented in Subsection 4.2 and in Section 5 are very similar, and the proofs are based on rather generic arguments (i.e., they use almost no specific properties of the lcs or the conjunctions of defined concepts). Thus, one may ask whether the constructions and arguments used there can be generalized. The purpose of this section is to show that this is indeed the case.

Consider a partially ordered set $(M, \preceq)$ for which all finite infima exist, i.e., if B is a finite subset of M, then there exists an element $\mathit{inf}(B) \in M$ that is the greatest element of M smaller than all elements of B. From the algorithmic point of view we assume that there are algorithms for deciding the relation $\preceq$ and for computing $\mathit{inf}(B)$ for all finite subsets B of M. In Subsection 4.2, M is the set of all concept descriptions of the DL under consideration, $\preceq$ is subsumption w.r.t. the TBox ($\sqsubseteq_{\mathcal{T}}$), and the infimum of a finite set of such descriptions is their conjunction.[4] In Section 5, M is again the set of all concept descriptions of the

[4] To be more precise, M is the set of all equivalence classes $[C]_{\equiv}$ of concept descriptions C and the partial order is the partial order $\sqsubseteq_{\equiv}$ induced by subsumption w.r.t. $\mathcal{T}$ on these equivalence classes.

DL under consideration, $\preceq$ is inverse subsumption ($\sqsupseteq$) and the infimum is given by the lcs.[5]

Definition 21 *Given a* finite *subset N of M, its* infimum closure *is the set*

$$\mathit{InfC}(N) := \{\mathit{inf}(B) \mid B \subseteq N\}.$$

In Subsection 4.2, N is the set of all concept names defined in the TBox, and $\mathit{InfC}(N)$ is the set of all conjunctions of such names. In Section 5, N is a finite set $\mathcal{C}$ of concept descriptions, and $\mathit{InfC}(N)$ is the set of all least common subsumers of subsets of $\mathcal{C}$.

Since N is finite, $(\mathit{InfC}(N), \preceq)$ is a complete semilattice, and thus a complete lattice. We are interested in computing this lattice. In Subsection 4.2, $(\mathit{InfC}(N), \preceq)$ is the hierarchy of all conjunctions of defined concepts, and in Section 5, $(\mathit{InfC}(N), \preceq)$ is the hierarchy of all least common subsumers of subsets of $\mathcal{C}$. In order to compute $(\mathit{InfC}(N), \preceq)$, we define a formal context with attribute set N such that the concept lattice of this context is isomorphic to $(\mathit{InfC}(N), \preceq)$.

Definition 22 *Let $(M, \preceq)$ be a partially ordered set for which all finite infima exist, and let N be a finite subset of M. The formal context $\mathcal{K}_{\preceq}(N) = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ is defined as follows:*

$$\begin{aligned} \mathcal{O} &:= M, \\ \mathcal{P} &:= N, \\ \mathcal{S} &:= \{(m, n) \mid m \preceq n\}. \end{aligned}$$

Valid implications in $\mathcal{K}_{\preceq}(N)$ correspond to $\preceq$-relationships between infima of subsets of N. The proof of this result is an easy generalization of the proof of Lemma 13.

Lemma 23 *Let B_1, B_2 be subsets of $\mathcal{P} = N$. The implication $B_1 \to B_2$ holds in $\mathcal{K}_{\preceq}(N)$ iff $\mathit{inf}(B_1) \preceq \mathit{inf}(B_2)$.*

Proof. First, we prove the *only if* direction. If the implication $B_1 \to B_2$ holds in $\mathcal{K}_{\preceq}(N)$, then this means that the following holds for all objects $m \in \mathcal{O}$: if $m \preceq n$ holds for all $n \in B_1$, then $m \preceq n$ also holds for all $n \in B_2$. Thus, if we take $\mathit{inf}(B_1)$ as object m, we obviously have $\mathit{inf}(B_1) \preceq n$ for all $n \in B_1$, and thus $\mathit{inf}(B_1) \preceq n$ for all $n \in B_2$, which shows $\mathit{inf}(B_1) \preceq \mathit{inf}(B_2)$.

Second, we prove the *if* direction. If $\mathit{inf}(B_1) \preceq \mathit{inf}(B_2)$, then any object m satisfying $m \preceq \mathit{inf}(B_1)$ also satisfies $m \preceq \mathit{inf}(B_2)$ by the transitivity of $\preceq$. Consequently, if $m \preceq n$ for all $n \in B_1$, then $m \preceq \mathit{inf}(B_1) \preceq \mathit{inf}(B_2) \preceq n$ for all $n \in B_2$, i.e., if m satisfies all attributes in B_1, it also satisfies all attributes in B_2. This shows that the implication $B_1 \to B_2$ holds in $\mathcal{K}_{\preceq}(N)$. □

The proof of the following theorem is an easy generalization of the proof of Theorem 10 and Theorem 14.

[5] Since we take inverse subsumption, the lcs, which is the supremum w.r.t. subsumption, is indeed the infimum.

Theorem 24 *The concept lattice of the context $\mathcal{K}_{\preceq}(N)$ is isomorphic to the lattice $(\mathit{InfC}(N), \preceq)$.*

Proof. In order to obtain an appropriate isomorphism, we define a mapping π from the formal concepts of the context $\mathcal{K}_{\preceq}(N)$ to $\mathit{InfC}(N)$ as follows:

$$\pi(A, B) = \mathit{inf}(B).$$

Since B is a finite subset of $\mathcal{P} = N$, the definition of $\mathit{InfC}(N)$ implies that $\mathit{inf}(B) \in \mathit{InfC}(N)$.

For formal concepts $(A_1, B_1), (A_2, B_2)$ of $\mathcal{K}_{\preceq}(N)$ we have $(A_1, B_1) \leq (A_2, B_2)$ iff $B_2 \subseteq B_1$. Since B_1 is the intent of the formal concept (A_1, B_1), we have $B_1 = A_1' = A_1''' = B_1''$, and thus $B_2 \subseteq B_1$ iff $B_2 \subseteq B_1''$ iff the implication $B_1 \to B_2$ holds in $\mathcal{K}_{\preceq}(N)$ iff $\mathit{inf}(B_1) \preceq \mathit{inf}(B_2)$. Overall, we have thus shown that π is an order embedding (and thus injective): $(A_1, B_1) \leq (A_2, B_2)$ iff $\mathit{inf}(B_1) \preceq \mathit{inf}(B_2)$.

It remains to be shown that π is surjective as well. Let B be an arbitrary subset of $\mathcal{P} = N$. We must show that $\mathit{inf}(B)$ can be obtained as an image under the mapping π. We know that (B', B'') is a formal concept of $\mathcal{K}_{\preceq}(N)$, and thus it is sufficient to show that $\pi(B', B'') = \mathit{inf}(B)$, i.e., $\mathit{inf}(B'') = \mathit{inf}(B)$. Obviously, $B \subseteq B''$ implies $\mathit{inf}(B'') \preceq \mathit{inf}(B)$. Conversely, the implication $B \to B''$ holds in $\mathcal{K}_{\preceq}(N)$, and thus Lemma 23 yields $\mathit{inf}(B) \preceq \mathit{inf}(B'')$. Since $\preceq$ is antisymmetric, this shows $\mathit{inf}(B) = \mathit{inf}(B'')$. □

If we want to apply Algorithm 7 to compute the concept lattice and the Duquenne-Guigues base of $\mathcal{K}_{\preceq}(N)$, we need an "expert" for this context. The proof of the next proposition is a generalization of the proofs of Proposition 20 and of Proposition 15.

Proposition 25 *Given a decision procedure for $\preceq$ as well as an algorithm for computing the infima of all finite subsets of M, these algorithms can be used to obtain an expert for the context $\mathcal{K}_{\preceq}(N)$.*

Proof. First, we show how to decide whether a given implication $B_1 \to B_2$ holds in $\mathcal{K}_{\preceq}(N)$ or not. By Lemma 23, we know that $B_1 \to B_2$ holds in $\mathcal{K}_{\preceq}(N)$ iff $\mathit{inf}(B_1) \preceq \mathit{inf}(B_2)$. Obviously, $\mathit{inf}(B_1) \preceq \mathit{inf}(B_2)$ iff $\mathit{inf}(B_1) \preceq n$ for all $n \in B_2$. Thus, to answer the question "$B_1 \to B_2$?", we first compute $\mathit{inf}(B_1)$ and then use the decision procedure for $\preceq$ to test whether $\mathit{inf}(B_1) \preceq n$ holds for all $n \in B_2$.

Second, assume that $B_1 \to B_2$ does not hold in $\mathcal{K}_{\preceq}(N)$, i.e., $\mathit{inf}(B_1) \not\preceq \mathit{inf}(B_2)$. We claim that $\mathit{inf}(B_1)$ is a counterexample, i.e., $\mathit{inf}(B_1) \in B_1'$ and $\mathit{inf}(B_1) \notin B_2'$. This is an immediate consequence of the facts that $B_i' = \{m \in \mathcal{O} \mid m \preceq n \text{ for all } n \in B_i\} = \{m \in \mathcal{O} \mid m \preceq \mathit{inf}(B_i)\}$ $(i = 1, 2)$ and that $\mathit{inf}(B_1) \preceq \mathit{inf}(B_1)$ and $\mathit{inf}(B_1) \not\preceq \mathit{inf}(B_2)$.

Of this counterexample, Algorithm 7 really needs the row corresponding to this object in the matrix corresponding to $\mathcal{S}$. This row can easily be computed using the decision procedure for $\preceq$: for each $n \in \mathcal{P} = N$, we use this decision procedure to test whether $\mathit{inf}(B_1) \preceq n$ holds or not. □

To sum up, we have shown that attribute exploration can be used to compute (a representation of) the lattice $(\mathit{InfC}(N), \preceq)$ provided that $\preceq$ is decidable and

all finite infima are computable. The results presented in Subsection 4.2 and in Section 5 are instances of this general result. The fact that the approach described in Section 5 (and first presented in [10]) can be generalized in this direction has already been mentioned in [18] (Section 3), but not worked out in detail.

8 Conclusion

We have described two cases where a tool from FCA (attribute exploration) can be used to compute an extended subsumption hierarchy in DL (the hierarchy of conjunctions of concepts defined in a terminology, and the hierarchy of all least common subsumers of a finite set of concept descriptions). The experimental results show that this approach is much better than the naive approach for computing these hierarchies. Nevertheless, there is still room for improvements. For example, in the first case the overhead of the FCA part of the algorithm was quite high due to the large number of formal concepts in the concept lattice. For most applications, one does not really need to compute all formal concepts since the implication base already contains all relevant information. Attribute exploration (as described in Section 3) generates all intents of formal concepts, since it enumerates all pseudo-closed sets, which are either concept intents, left-hand sides of implications in the base, or left-hand sides of implications that are not valid (i.e., implications that yield a counterexamples during the exploration process). In contexts whose concept lattice is quite large compared to the size of the Duquenne-Guigues base and the number of counterexamples, it would be better to have a modified attribute exploration algorithm that enumerates only those pseudo-closed sets that are not concept intents (only for those, the expert is called).

We have also seen that the approaches described in Subsection 4.2 and Section 5 are both instances of a more general approach. Thus, one can try to find other interesting instances of this general approach. One example could be to compute the hierarchy of all conjunctions of defined concepts and their negations. In fact, when considering the lcs in DLs that are more expressive than $\mathcal{EL}$ (e.g., in $\mathcal{ALE}$), one must deal with such conjunctions. Since in this case one has background knowledge about the relationship between different attributes (the concept A is disjoint from its negation $\neg A$), one can probably employ methods developed for attribute exploration with background knowledge [17].

Acknowledgment We should like to thank Madjid Nassiri for the implementation of and the experiments with the approach described in Subsection 4.2 and Ralf Molitor for the implementation of and the experiments with the approach described in Section 5.

References

1. Franz Baader. Computing a minimal representation of the subsumption lattice of all conjunctions of concepts defined in a terminology. In G. Ellis, R. A. Levinson,

A. Fall, and V. Dahl, editors, *Knowledge Retrieval, Use and Storage for Efficiency: Proc. of the 1st Int. KRUSE Symposium*, pages 168–178, 1995.

2. Franz Baader. Computing the least common subsumer in the description logic $\mathcal{EL}$ w.r.t. terminological cycles with descriptive semantics. In *Proceedings of the 11th International Conference on Conceptual Structures, ICCS 2003*, volume 2746 of *Lecture Notes in Artificial Intelligence*, pages 117–130. Springer-Verlag, 2003.
3. Franz Baader. The instance problem and the most specific concept in the description logic $\mathcal{EL}$ w.r.t. terminological cycles with descriptive semantics. In *Proceedings of the 26th Annual German Conference on Artificial Intelligence, KI 2003*, volume 2821 of *Lecture Notes in Artificial Intelligence*, pages 64–78, Hamburg, Germany, 2003. Springer-Verlag.
4. Franz Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 319–324. Morgan Kaufmann, 2003.
5. Franz Baader. Terminological cycles in a description logic with existential restrictions. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 325–330. Morgan Kaufmann, 2003.
6. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
7. Franz Baader, Enrico Franconi, Bernhard Hollunder, Bernhard Nebel, and Hans-Jürgen 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. Franz Baader and Ralf Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In *Proc. of the 22nd German Annual Conf. on Artificial Intelligence (KI'98)*, volume 1504 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1998.
9. Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 96–101, 1999.
10. Franz Baader and Ralf Molitor. Building and structuring description logic knowledge bases using least common subsumers and concept analysis. In B. Ganter and G. Mineau, editors, *Conceptual Structures: Logical, Linguistic, and Computational Issues – Proceedings of the 8th International Conference on Conceptual Structures (ICCS2000)*, volume 1867 of *Lecture Notes in Artificial Intelligence*, pages 290–303. Springer-Verlag, 2000.
11. Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
12. William W. Cohen and Haym Hirsh. Learning the CLASSIC description logics: Theorethical and experimental results. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 121–133, 1994.
13. William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.
14. Vincent Duquenne. Contextual implications between attributes and some representational properties for finite lattices. In Bernhard Ganter, Rudolf Wille, and

Karl Erich Wolf, editors, *Beiträge zur Begriffsanalyse*, pages 213–239. B.I. Wissenschaftsverlag, Mannheim, 1987.
15. Michael Frazier and Leonard Pitt. CLASSIC learning. *Machine Learning*, 25:151–193, 1996.
16. Bernhard Ganter. Finding all closed sets: A general approach. *Order*, 8:283–290, 1991.
17. Bernhard Ganter. Attribute exploration with background knowledge. *Theoretical Computer Science*, 217(2):215–233, 1999.
18. Bernhard Ganter and Sergei O. Kuznetsov. Pattern structures and their projections. In Harry S. Delugach and Gerd Stumme, editors, *Conceptual Structures: Broadening the Base, Proceedings of the 9th International Conference on Conceptual Structures, ICCS 2001*, volume 2120 of *Lecture Notes in Computer Science*, pages 129–142. Springer-Verlag, 2001.
19. Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin, 1999.
20. Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, 2001.
21. Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, 2001.
22. Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.
23. Ralf Küsters and Alex Borgida. What's in an attribute? Consequences for the least common subsumer. *J. of Artificial Intelligence Research*, 14:167–203, 2001.
24. Ralf Küsters and Ralf Molitor. Approximating most specific concepts in description logics with existential restrictions. In Franz Baader, Gerd Brewka, and Thomas Eiter, editors, *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI 2001)*, volume 2174 of *Lecture Notes in Artificial Intelligence*, pages 33–47, Vienna, Austria, 2001. Springer-Verlag.
25. Ralf Küsters and Ralf Molitor. Computing least common subsumers in $\mathcal{ALEN}$. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 219–224, 2001.
26. Carsten Lutz. Complexity of terminological reasoning revisited. In *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 181–200. Springer-Verlag, 1999.
27. Wolfgang Marquardt. Trends in computer-aided process modeling. *Computers and Chemical Engineering*, 20(6/7):591–609, 1996.
28. Ralf Molitor. *Unterstützung der Modellierung verfahrenstechnischer Prozesse durch Nicht-Standardinferenzen in Beschreibungslogiken (Supporting the Modelling of of Chemical Processes by Using Non-standard Inferences in Description Logics)*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2000. In German.
29. Madjid Nassiri. Berechnung einer erweiterten Subsumtionshierarchie (Computation of an extended subsumption hierarchy). Diploma thesis, RWTH Aachen, Germany, 1997. In German.
30. Susanne Prediger. Terminologische Merkmalslogik in der Formalen Begriffsanalyse. In *[37]*. 2000.

31. Susanne Prediger and Gerd Stumme. Theory-driven logical scaling: Conceptual information systems meet description logics. In Enrico Franconi and Michael Kifer, editors, *Proc. of the 6th Int. Workshop on Knowledge Representation meets Databases (KRDB'99)*, 1999.
32. Alan Rector and Ian Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*, Stanford, CA, 1997. AAAI Press.
33. Sebastian Rudolf. An FCA method for the extensional exploration of relational data. In Bernhard Ganter and Aldo de Moor, editors, *Using Conceptual Structures – Contributions to ICCS 2003*. Shaker Verlag, 2003.
34. Ulrike Sattler. *Terminological Knowledge Representation Systems in a Process Engineering Application*. PhD thesis, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1998.
35. Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
36. Stefan Schultz and Udo Hahn. Knowledge engineering by large-scale knowledge reuse—experience from the medical domain. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'2000)*, pages 601–610. Morgan Kaufmann, 2000.
37. Gerd Stumme and Rudolf Wille. *Begriffliche Wissensverarbeitung – Methoden und Anwendungen*. Springer-Verlag, 2000.
38. Frank Vogt. *Formale Begriffsanalyse mit C++*. Springer-Verlag, 1996.
39. Monika Zickwolff. *Rule Exploration: First Order Logic in Formal Concept Analysis*. PhD thesis, TH Darmstadt, Germany, 1991.