Computing the Least Common Subsumer w.r.t. a Background Terminology [1]

Franz Baader [*], Baris Sertkaya, Anni-Yasmin Turhan

Theoretical Computer Science, TU Dresden, Germany

Abstract

Methods for computing the least common subsumer (lcs) are usually restricted to rather inexpressive Description Logics (DLs) whereas existing knowledge bases are written in very expressive DLs. In order to allow the user to re-use concepts defined in such terminologies and still support the definition of new concepts by computing the lcs, we extend the notion of the lcs of concept descriptions to the notion of the lcs w.r.t. a background terminology. We will show both theoretical results on the existence of the *least* common subsumer in this setting, and describe a practical approach—based on a method from formal concept analysis—for computing *good* common subsumers, which may, however, not be the least ones. We will also describe results obtained in a first evaluation of this practical approach.

Key words: Description Logic, Non-standard Inferences

1 Introduction

Description Logics (DLs) [1] are a class of knowledge representation formalisms in the tradition of semantic networks and frames, which can be used to represent the terminological knowledge of an application domain in a structured and formally well-understood way. The name *description logics* is motivated by the fact that, on the one hand, the important notions of the domain are described by *concept descriptions*, i.e., expressions that are built from atomic concepts

* Corresponding Author
Email addresses: `baader@tcs.inf.tu-dresden.de` (Franz Baader), `sertkaya@tcs.inf.tu-dresden.de` (Baris Sertkaya), `turhan@tcs.inf.tu-dresden.de` (Anni-Yasmin Turhan).

1 This work was supported by the German Science Foundation (DFG) under the grants GRK 334/3 and BA 1122/4-4.

(unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the particular DL. On the other hand, DLs differ from their predecessors, such as semantic networks and frames [2,3], in that they are equipped with a formal, *logic*-based semantics, which can, e.g., be given by a translation into first-order predicate logic.

Knowledge representation systems based on description logics (DL systems) [4,5] provide their users with various inference capabilities that allow them to deduce implicit knowledge from the explicitly represented knowledge. Standard inference services are *subsumption* and *instance checking*. Subsumption allows the user to determine subconcept-superconcept relationships, and hence compute the concept hierarchy: C is subsumed by D iff all instances of C are also instances of D, i.e., the first description is always interpreted as a subset of the second description. Instance checking asks whether a given individual necessarily belongs to a given concept, i.e., whether this instance relationship logically follows from the descriptions of the concept and of the individual.

In order to ensure a reasonable and predictable behaviour of a DL reasoner, these inference problems should at least be decidable for the DL employed by the reasoner, and preferably of low complexity. Consequently, the expressive power of the DL in question must be restricted in an appropriate way. If the imposed restrictions are too severe, however, then the important notions of the application domain can no longer be expressed. Investigating this trade-off between the expressivity of DLs and the complexity of their inference problems has been one of the most important issues of DL research in the 1990ies. As a consequence of this research, the complexity of reasoning in various DLs of different expressive power is now well-investigated (see [6] for an overview of these complexity results). In addition, there are highly optimized implementations of reasoners for very expressive DLs [7–9], which—despite their high worst-case complexity—behave very well in practice [10,11].

DLs have been applied in many domains, such as medical informatics, software engineering, configuration of technical systems, natural language processing, databases, and web-based information systems (see Part III of [1] for details on these and other applications). A recent success story is the use of DLs as ontology languages [12,13] for the Semantic Web [14]. In particular, the W3C recommended ontology web language OWL [15] is based on an expressive description logic [16,17].

Editors—such as OilEd [18] and the OWL plug-in of Protégé [19]—supporting the design of ontologies in various application domains usually allow their users to access a DL reasoner, which realizes the aforementioned *standard inferences* such as subsumption and instance checking. Reasoning is not only useful when working with "finished" ontologies: it can also support the ontology engineer while building an ontology, by pointing out inconsistencies and unwanted consequences. The ontology engineer can thus use reasoning to check whether the definition of a concept or the description of an individual makes sense. However, the standard DL inferences—subsumption and instance checking—provide only little support for actually coming up with a first version of the definition of a concept.

More recently, non-standard inferences [20] were introduced to support building and maintaining large DL knowledge bases. In particular, they overcome the deficit mentioned above, by allowing the user to construct new knowledge from the existing one. For example, such non-standard inferences can be used to support the so-called *bottom-up* construction of DL knowledge bases, as introduced in [21,22]: instead of directly defining a new concept, the knowledge engineer introduces several typical examples as individuals, 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 individuals, and then computing the least common subsumer of these concepts. The *most specific concept* (msc) of an individual i (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 i 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 [23,24,21,22,25–29].

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 [7] and Racer [8] are based on very expressive DLs, and there exist large knowledge bases that use this expressive power and can be processed by these systems [30,31,11]. 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 in this work the following *extended bottom-up approach.* In this approach we assume that there is a fixed *background terminology* defined in an expressive DL; e.g., a large ontology written by experts, which the user has bought from some ontology provider. The user then wants to extend this terminology in order to adapt it to the needs of a particular application domain. However, since the user is not a DL expert, he employs a less expressive DL and needs support through the bottom-up approach when building this user-specific extension of the background terminology. There are several reasons for the user to employing a restricted DL in this setting: first, such a restricted DL may be easier to comprehend and use for a non-expert; second, it may allow for a more intuitive graphical or frame-like user interface; third, to use the bottom-up approach, the lcs must exist and make sense, and it must be possible to compute it with reasonable effort.

To make this more precise, consider a background terminology (TBox) $\mathcal{T}$ 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 $\mathcal{T}$. Let us call such concept descriptions $\mathcal{L}_1(\mathcal{T})$-concept descriptions. Given $\mathcal{L}_1(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, we are now looking for their lcs in $\mathcal{L}_1(\mathcal{T})$, i.e., the least $\mathcal{L}_1(\mathcal{T})$-concept description that subsumes $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$.

In this article, we consider the case where $\mathcal{L}_1$ is the DL $\mathcal{ALE}$ and $\mathcal{L}_2$ is the DL $\mathcal{ALC}$. We first show (in Section 3) the following two results:

- If $\mathcal{T}$ is an acyclic $\mathcal{ALC}$-TBox, then the lcs w.r.t. $\mathcal{T}$ of $\mathcal{ALE}(\mathcal{T})$-concept descriptions always exists.
- If $\mathcal{T}$ is a general $\mathcal{ALC}$-TBox allowing for general concept inclusion axioms (GCIs), then the lcs w.r.t. $\mathcal{T}$ of $\mathcal{ALE}(\mathcal{T})$-concept descriptions need not exist.

The result on the existence and computability of the lcs w.r.t. an acyclic background terminology is theoretical in the sense that it does not yield a practical algorithm.

In Section 4 we follow a more practical approach. Assume that $\mathcal{L}_1$ is a DL for which least common subsumers (without background TBox) always exist. Given $\mathcal{L}_1(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, one can compute a common subsumer w.r.t. $\mathcal{T}$ by just ignoring $\mathcal{T}$, i.e., by treating the defined names in $C_1, \ldots, C_n$ as primitive and computing the lcs of $C_1, \ldots, C_n$ in $\mathcal{L}_1$. However, the common subsumer obtained this way will usually be too general. In Section 4 we sketch a practical method for computing "good" common subsumers w.r.t. background TBoxes, which may not be the *least* common subsumers, but which are better than the common subsumers computed by ignoring the TBox. As a tool, this method uses attribute exploration (possibly with a priori knowledge) [32–34], an algorithm developed in Formal Concept Analysis [35] for computing concept lattices. The application of attribute exploration for this purpose is described in Section 5.

In Section 6 we report on first experimental results. On the one hand, we investigate whether using a priori knowledge in attribute exploration speeds up the exploration process. On the other hand, we compare the approach described above with two other approaches (introduced in Subsection 4.4) for computing common subsumers: one based on approximating $\mathcal{L}_2$-concept descriptions by $\mathcal{L}_1$-concept descriptions, and one using only the information provided by the subsumption relationships between concepts defined in the background TBox $\mathcal{T}$.

2 Basic definitions and results

In this section, we introduce basic notions from description logics and formal concept analysis.

2.1 Description logic

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 constructors provided by the concept description language of the particular system. In this paper, we consider the DL $\mathcal{ALC}$ and its sublanguages $\mathcal{ALE}$ and $\mathcal{EL}$, 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* (see Table 1) assigns a concept name A to a complex concept description C. A finite set of such definitions is called an *acyclic 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). If the TBox is unambiguous, but not acyclic, then it is called a *cyclic TBox.* The concept names occurring on the left-hand side of a concept definition are called *defined* concepts, and the others *primitive.* A *general concept inclusion (GCI)* (see Table 1) states a subconcept/superconcept constraint between two (possibly complex) concept descriptions. A finite set of GCIs is called a *general TBox.* If we say just TBox then this means an acyclic, a cyclic or a general TBox. An acyclic or a cyclic $\mathcal{ALE}$-TBox must satisfy the additional restriction that no defined concept occurs negated in it (i.e., negation can only be applied to primitive concepts).

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 $A \in N_C$ to a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ 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 (a)cyclic 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 $\mathcal{T}$. It is a model of the general TBox $\mathcal{T}$ iff it satisfies all its concept inclusions,

Name of constructor	Syntax	Semantics	$\mathcal{ALC}$	$\mathcal{ALE}$	$\mathcal{EL}$
top-concept	$\top$	$\Delta^{\mathcal{I}}$	x	x	x
bottom-concept	$\bot$	$\emptyset$	x	x	
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	x		
atomic negation	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$	x	x	
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	x	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	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	x
concept definition	$A \equiv C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$	(a)cyclic TBox		
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	general TBox		

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

i.e., $C^{\mathcal{I}} \sqsubseteq D^{\mathcal{I}}$ holds for all $C \sqsubseteq D$ in $\mathcal{T}$.

Given this semantics, we can now define the most important traditional inference service provided by DL systems, i.e., computing subconcept/superconcept relationships, so-called *subsumption* relationships.

Definition 1 *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 $\mathcal{T}$. We write $C_1 \sqsubseteq C_2$ iff C_1 is subsumed by C_2 w.r.t. the empty TBox. Two concept descriptions C_1, C_2 are called* equivalent w.r.t. $\mathcal{T}$ *iff they subsume each other, i.e., $C_1 \equiv_{\mathcal{T}} C_2$ iff $C_1 \sqsubseteq_{\mathcal{T}} C_2$ and $C_2 \sqsubseteq_{\mathcal{T}} C_1$. The concept description C is* unsatisfiable w.r.t. the TBox $\mathcal{T}$ *iff it is subsumed by $\bot$ w.r.t. $\mathcal{T}$; otherwise, it is* satisfiable w.r.t. $\mathcal{T}$.

The subsumption relation $\sqsubseteq_{\mathcal{T}}$ is a preorder (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 preorder $\sqsubseteq_{\mathcal{T}}$ induces a partial order $\sqsubseteq_{\mathcal{T}}^{\equiv}$ on the equivalence classes of concept descriptions:

$$[C_1]_{\equiv} \sqsubseteq_{\mathcal{T}}^{\equiv} [C_2]_{\equiv} \text{ iff } C_1 \sqsubseteq_{\mathcal{T}} C_2,$$

where $[C_i]_{\equiv} := \{D \mid C_i \equiv_{\mathcal{T}} D\}$ is the equivalence class of C_i $(i = 1, 2)$.When talking about the *subsumption hierarchy*, we mean this induced partial order.

The *complexity* of the subsumption problem depends on the DL under consideration, and on what kind of TBox formalism is used. Subsumption w.r.t. the empty TBox (usually called *subsumption of concept descriptions*) is polynomial for $\mathcal{EL}$ [22], NP-complete for $\mathcal{ALE}$ [36], and PSPACE-complete for $\mathcal{ALC}$ [37]. Subsumption in $\mathcal{EL}$ stays polynomial both in the presence of (a)cyclic [38] and general TBoxes [39]. Subsumption in $\mathcal{ALC}$ stays PSPACE-complete w.r.t. acyclic TBoxes [40], but it becomes EXPTIME-complete in the presence of general TBoxes [41]. EXPTIME-completeness already holds for subsumption in $\mathcal{ALE}$ w.r.t. general TBoxes [42].

It should be noted that subsumption w.r.t. acyclic TBoxes can be reduced to subsumption of concept descriptions by *expanding the TBox*, i.e. by replacing the defined concepts by their definitions until no more defined concepts occur in the concept descriptions to be tested for subsumption. To be more precise, let C, D be concept descriptions and $\mathcal{T}$ an acyclic TBox. If C', D' are the concept descriptions obtained by expanding C, D w.r.t. $\mathcal{T}$, then $C \sqsubseteq_{\mathcal{T}} D$ iff $C' \sqsubseteq D'$. However, this reduction cannot be used to obtain the complexity results for subsumption w.r.t. acyclic TBoxes mentioned above since the expansion process may cause an exponential blow-up of the concept descriptions [43].

In addition to standard inferences like computing the subsumption hierarchy, so-called *non-standard inferences* have been introduced and investigated in the DL community (see, e.g., [20]). In this paper, we concentrate on the problem of computing the least common subsumer. Originally, this problem was introduced for concept descriptions (i.e., w.r.t. the empty TBox). In the presence of acyclic TBoxes, one can apply this inference if one first expands the concept descriptions. Let $\mathcal{L}$ be some description logic.

Definition 2 *Given a collection $C_1, \ldots, C_n$ of $\mathcal{L}$-concept descriptions, the* least common subsumer *(lcs) of $C_1, \ldots, C_n$ in $\mathcal{L}$ is the most specific $\mathcal{L}$-concept description that subsumes $C_1, \ldots, C_n$, i.e., it is an $\mathcal{L}$-concept description D such that*

(1) $C_i \sqsubseteq D$ for $i = 1, \ldots, n$ (D is a common subsumer);
(2) if E is an $\mathcal{L}$-concept description satisfying $C_i \sqsubseteq E$ for $i = 1, \ldots, n$, then $D \sqsubseteq E$ (D is least).

As an easy consequence of this definition, the lcs is unique up to equivalence, which justifies talking about *the* lcs. In addition, the n-ary lcs as defined above can be reduced to the binary lcs (the case $n = 2$ above). Indeed, it is easy to see that the lcs of $C_1, \ldots, C_n$ can be obtained by building the lcs of C_1, C_2, then the lcs of this concept description with C_3, etc. Thus, it is enough to devise algorithms for computing the binary lcs.

It should be noted, however, that the lcs need not always exist. This can have different reasons: (a) there may not exist a concept description in $\mathcal{L}$ satisfying (1) of the definition (i.e., subsuming $C_1, \ldots, C_n$); (b) there may be several subsumption incomparable minimal concept descriptions satisfying (1) of the definition; (c) there may be an infinite chain of more and more specific descriptions satisfying (1) of the definition. Obviously, (a) cannot occur for DLs containing the top-concept. It is easy to see that, for DLs allowing for conjunction of descriptions, (b) cannot occur.

It is also clear that in DLs allowing for disjunction, the lcs of $C_1, \ldots, C_n$ is their disjunction $C_1 \sqcup \ldots \sqcup C_n$. In this case, the lcs is not really of interest. Instead of extracting properties common to $C_1, \ldots, C_n$, it just gives their disjunction, which does not provide us with new information. For the DLs introduced above, this means that it makes sense to look at the lcs in $\mathcal{EL}$ and $\mathcal{ALE}$, but not in $\mathcal{ALC}$. Both for $\mathcal{EL}$ and $\mathcal{ALE}$, the lcs always exists, and can be effectively computed [22]. For $\mathcal{EL}$, the size and computation time for the binary lcs is polynomial, but exponential in the n-ary case. For $\mathcal{ALE}$, already the size of the binary lcs may grow exponentially in the size of the input concept descriptions.

Let us now define the *new non-standard inference* introduced in this paper, which is a generalization of the lcs to (a)cyclic or general background TBoxes. Let $\mathcal{L}_1, \mathcal{L}_2$ be DLs such that $\mathcal{L}_1$ is a sub-DL of $\mathcal{L}_2$, i.e., $\mathcal{L}_1$ allows for less constructors. For a given $\mathcal{L}_2$-TBox $\mathcal{T}$, we call $\mathcal{L}_1(\mathcal{T})$*-concept descriptions* those $\mathcal{L}_1$-concept descriptions that may contain concepts defined in $\mathcal{T}$.

Definition 3 *Given an $\mathcal{L}_2$-TBox $\mathcal{T}$ and $\mathcal{L}_1(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, the* least common subsumer *(lcs) of $C_1, \ldots, C_n$ in* $\mathcal{L}_1(\mathcal{T})$ w.r.t. $\mathcal{T}$ *is the most specific $\mathcal{L}_1(\mathcal{T})$-concept description that subsumes $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$, i.e., it is an $\mathcal{L}_1(\mathcal{T})$-concept description D such that*

(1) $C_i \sqsubseteq_{\mathcal{T}} D$ *for* $i = 1, \ldots, n$ (D is a common subsumer);
(2) if E is an $\mathcal{L}_1(\mathcal{T})$-concept description satisfying
$C_i \sqsubseteq_{\mathcal{T}} E$ *for* $i = 1, \ldots, n$, *then* $D \sqsubseteq_{\mathcal{T}} E$ (D is least).

Depending on the DLs $\mathcal{L}_1$ and $\mathcal{L}_2$, least common subsumers of $\mathcal{L}_1(\mathcal{T})$-concept descriptions w.r.t. an $\mathcal{L}_2$-TBox $\mathcal{T}$ may exist or not. Note that this lcs may use only concept constructors from $\mathcal{L}_1$, but may also contain concept names defined in the $\mathcal{L}_2$-TBox $\mathcal{T}$. This is the main distinguishing feature of this new notion of a least common subsumer w.r.t. a background terminology. Let us illustrate this by a trivial example.

Example 4 Assume that $\mathcal{L}_1$ is the DL $\mathcal{ALE}$ and $\mathcal{L}_2$ is $\mathcal{ALC}$. Consider the $\mathcal{ALC}$-TBox $\mathcal{T} := \{A \equiv P \sqcup Q\}$, and assume that we want to compute the lcs of the $\mathcal{ALE}(\mathcal{T})$-concept descriptions P and Q. Obviously, A is the lcs of P and Q w.r.t. $\mathcal{T}$. If we were not allowed to use the name A defined in $\mathcal{T}$, then the only common subsumer of P and Q in $\mathcal{ALE}$ would be the top-concept $\top$.

At first sight, one might think that, in the case of an acyclic background TBox, the problem of computing the lcs in $\mathcal{ALE}(\mathcal{T})$ w.r.t. an $\mathcal{ALC}$-TBox $\mathcal{T}$ can be reduced to the problem of computing the lcs in $\mathcal{ALE}$ by expanding the TBox and using results on the approximation of $\mathcal{ALC}$ by $\mathcal{ALE}$ [44]. To make this more precise, we must introduce the non-standard inference of approximating concept descriptions of one DL by descriptions of another DL. Let $\mathcal{L}_1, \mathcal{L}_2$ be DLs such that $\mathcal{L}_1$ is a sub-DL of $\mathcal{L}_2$.

Definition 5 *Given an $\mathcal{L}_2$-concept description C, the $\mathcal{L}_1$-concept description D* approximates C from above *iff D is the least $\mathcal{L}_1$-concept description satisfying $C \sqsubseteq D$.*

In [44] it is shown that the approximation from above of an $\mathcal{ALC}$-concept description by an $\mathcal{ALE}$-concept description always exists, and can be computed in double-exponential time.

Thus, given an acyclic $\mathcal{ALC}$-TBox $\mathcal{T}$ and a collection of $\mathcal{ALE}(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, one can first expand $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$ to concept descriptions $C_1', \ldots, C_n'$. These descriptions are $\mathcal{ALC}$-concept descriptions since they may contain constructors of $\mathcal{ALC}$ that are not allowed in $\mathcal{ALE}$. One can then build the $\mathcal{ALC}$-concept description $C := C_1' \sqcup \ldots \sqcup C_n'$, and finally *approximate* C from above by an $\mathcal{ALE}$-concept description D. By construction, D is a common subsumer of $C_1, \ldots, C_n$.

However, D does not contain concept names defined in $\mathcal{T}$, and thus it is not necessarily the *least* $\mathcal{ALE}(\mathcal{T})$-concept description subsuming $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$. Indeed, this is the case in Example 4 above, where the approach based on approximation that we have just sketched yields $\top$ rather than the lcs A. One might now assume that this can be overcome by applying known results on rewriting concept descriptions w.r.t. a terminology [45]. However, in Example 4, the concept description $\top$ cannot be rewritten using the TBox $\mathcal{T} := \{A \equiv P \sqcup Q\}$.

2.2 Formal concept analysis

We will introduce only those notions and results from formal concept analysis (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 [35] for more information on this and FCA in general).

Definition 6 *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. The supremum and infimum in the concept lattice induced by $\mathcal{K}$ can be obtained as follows:

$$\begin{aligned}\textstyle\bigvee_{i\in I}(A_i, B_i) &= \left(\left(\textstyle\bigcup_{i\in I} A_i\right)'', \textstyle\bigcap_{i\in I} B_i\right),\\ \textstyle\bigwedge_{i\in I}(A_i, B_i) &= \left(\textstyle\bigcap_{i\in I} A_i, \left(\textstyle\bigcup_{i\in I} B_i\right)''\right).\end{aligned}$$

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

Lemma 7 *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 $A_2' \subseteq A_1'$.*

The dual of this lemma is also true, i.e., all formal concepts are of the form (B', B'') for a subset B of $\mathcal{P}$, and any such pair is a formal concept. In addition, $(B_1', B_1'') \leq (B_2', B_2'')$ iff $B_1' \subseteq B_2'$.

Given a formal context, the first step for analyzing this context is usually to compute the concept lattice. If the context is finite, then Lemma 7 implies that 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 [46], 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 [32,35] has turned out to be an efficient approach for computing the concept lattice. Before we can describe this algorithm, we must introduce some notation. The most important notion is the one of an implication between sets of attributes. Intuitively, such an implication $B_1 \rightarrow B_2$ holds if any object satisfying all elements of B_1 also satisfies all elements of B_2.

Definition 8 *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 $B_1 \rightarrow B_2$ holds *in $\mathcal{K}$ ($\mathcal{K} \models B_1 \rightarrow B_2$) iff $B_1' \subseteq B_2'$. An object o* violates *the implication $B_1 \rightarrow B_2$ iff $o \in B_1' \setminus B_2'$.*

It is easy to see that an implication $B_1 \rightarrow B_2$ holds in $\mathcal{K}$ iff $B_2 \subseteq B_1''$. In particular, given a set of attributes B, the implications $B \rightarrow B''$ and $B \rightarrow (B'' \setminus B)$ 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 9 *Let $\mathcal{J}$ be a set of implications, i.e., the elements of $\mathcal{J}$ are of the form $B_1 \rightarrow B_2$ 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*
- *$B_1 \rightarrow B_2 \in \mathcal{J}$ and $B_1 \subseteq H$ imply $B_2 \subseteq H$.*

The set of implications generated by $\mathcal{J}$ *consists of all implications $B_1 \rightarrow B_2$ 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.*

From a logician's point of view, computing the implication hull of a set of attributes B is just computing logical consequences. In fact, the notions we have just defined can easily be reformulated in propositional logic. To this purpose, we view the attributes as propositional variables. An implication $B_1 \rightarrow B_2$ can then be expressed by the formula $\phi_{B_1 \rightarrow B_2} := \bigwedge_{p \in B_1} p \rightarrow \bigwedge_{p' \in B_2} p'$. Let $\Gamma_{\mathcal{J}}$ be the set of formulae corresponding to the set of implications $\mathcal{J}$. Then

$$\mathcal{J}(B) = \{b \in \mathcal{P} \mid \Gamma_{\mathcal{J}} \cup \{\bigwedge_{p \in B} p\} \models b\},$$

where $\models$ stands for classical propositional consequence. Obviously, the formulae in $\Gamma_{\mathcal{J}}$ are Horn clauses. For this reason, 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 methods for deciding satisfiability of sets of propositional Horn clauses [47]. Alternatively, these formulae can be viewed as expressing functional dependencies in relational database, and thus the linearity result can also be obtained using methods for deriving new functional dependencies from given ones [48].

If $\mathcal{J}$ is a base for $Imp(\mathcal{K})$, then it can be shown that $B'' = \mathcal{J}(B)$ for all $B \subseteq \mathcal{P}$. 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 $\mathcal{J} \cup \{B_1 \to B_2\}$ since it is equivalent to asking whether $B_2 \subseteq B_1'' = \mathcal{J}(B_1)$.

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 [49]. 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 algorithms in [47,48] appropriately). A subset B of $\mathcal{P}$ is called *pseudo-closed* in a formal context $\mathcal{K}$ iff $Imp(\mathcal{K})^*(B) = B$ and $Imp(\mathcal{K})(B) = B'' \neq B$.

Definition 10 *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 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 11 (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*

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

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:*

[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.

(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.*

One may wonder why, in $(*)$, we compute the hull $\mathcal{J}_{i+1}(\cdot)$ rather than the pseudo-hull $\mathcal{J}^*_{i+1}(\cdot)$. One can show that in this case there actually is no difference between the hull and the pseudo-hull. This is a consequence of the fact that the pseudo-closed sets are enumerated w.r.t. the lectic order.

3 Existence and non-existence of the lcs w.r.t. TBoxes

In this section, we assume that $\mathcal{L}_1$ is $\mathcal{ALE}$ and $\mathcal{L}_2$ is $\mathcal{ALC}$. In addition, we assume that the sets of concept and role names available for building concept descriptions are finite.

Theorem 12 *Let* $\mathcal{T}$ *be an acyclic* $\mathcal{ALC}$*-TBox. The lcs of* $\mathcal{ALE}(\mathcal{T})$*-concept descriptions w.r.t.* $\mathcal{T}$ *always exists and can effectively be computed.*

Since the n-ary lcs can be obtained by iterating the application of the binary lcs, it is sufficient to show the theorem for the case where we want to build the lcs of two $\mathcal{ALE}(\mathcal{T})$-concept descriptions. To show the theorem in this case, we first need to show two propositions.

Given an $\mathcal{ALC}$- or $\mathcal{ALE}(\mathcal{T})$-concept description C, its role depth is the maximal nesting of value restrictions and existential restrictions. For example, the role depth of $\exists r.\forall r.A$ is 2, and the role depth of $\exists r.\forall r.A \sqcup \exists r.\exists r.\exists r.B$ is 3.

Proposition 13 *For a given bound* k *on the role depth, there is only a finite number of inequivalent* $\mathcal{ALE}$*-concept descriptions of role depth at most* k*.*

This is a consequence of the fact that we have assumed that the sets of concept and role names are finite, and can easily be shown by induction on k.[3]

Given this lemma, a first attempt to show Theorem 12 could be the following. Let C_1, C_2 be $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and assume that the role depths of the $\mathcal{ALC}$-concept description C'_1, C'_2 obtained by expanding the descriptions C_i w.r.t. $\mathcal{T}$ are bounded by k. If we could show that this implies that the role depth of any common subsumer of C_1, C_2 w.r.t. $\mathcal{T}$ is also bounded by k, then we could obtain the least common subsumer by simply building the (up to equivalence) finite conjunction of all common subsumers of C_1, C_2 in $\mathcal{ALE}(\mathcal{T})$.

[3] In fact, this is a well-known result, which holds even for full first-order predicate logic formulae of bounded quantifier depth over a finite vocabulary.

However, due to the fact that in $\mathcal{ALC}$ and $\mathcal{ALE}$ we can define unsatisfiable concepts, this simple approach does not work. In fact, $\bot$ has role depth 0, but it is subsumed by any concept description. Given this counterexample, the next conjecture could be that it is enough to prevent this pathological case, i.e., assume that at least one of the concept descriptions C_1, C_2 is satisfiable w.r.t. $\mathcal{T}$, i.e., not subsumed by $\bot$ w.r.t. $\mathcal{T}$. This assumption can be made without loss of generality. In fact, if C_1 is unsatisfiable w.r.t. $\mathcal{T}$ (i.e., equivalent to $\bot$ w.r.t. T), then C_2 is the lcs of C_1, C_2 w.r.t. T. For the DL $\mathcal{EL}$ in place of $\mathcal{ALE}$, this modification of the simple approach sketched above really works (see [50] for details). However, due to the presence of value restrictions, it does not work for $\mathcal{ALE}$. For example, $\forall r.\bot$ is subsumed by $\forall r.F$ for arbitrary $\mathcal{ALE}(\mathcal{T})$-concept descriptions F, and thus the role depth of common subsumers cannot be bounded. However, we can show that common subsumers having a large role depth are too general anyway.

Before giving a more formal statement of this result in Proposition 18, we show some basic model-theoretic facts about $\mathcal{ALE}$ and $\mathcal{ALC}$, which will be employed in the proof of this proposition. An interpretation $\mathcal{I}$ is *tree-shaped* if the role relationships in $\mathcal{I}$ form a tree, i.e., if the directed graph $G_\mathcal{I} = (V_\mathcal{I}, E_\mathcal{I})$ with $V_\mathcal{I} = \Delta^\mathcal{I}$ and

$$E_\mathcal{I} = \{(d, d') \mid (d, d') \in r^\mathcal{I} \text{ for some role } r \in N_R\}$$

is a tree. An interpretation $\mathcal{I}$ is a *tree-shaped counterexample* to the subsumption question $C \sqsubseteq^?_\mathcal{T} D$ iff $\mathcal{I}$ is a tree-shaped model of $\mathcal{T}$ with root $d_0 \in C^\mathcal{I} \setminus D^\mathcal{I}$.

Lemma 14 *Let $\mathcal{T}$ be an acyclic $\mathcal{ALC}$-TBox and C, D $\mathcal{ALC}$-concept descriptions. If $C \not\sqsubseteq_\mathcal{T} D$, then the subsumption question $C \sqsubseteq^?_\mathcal{T} D$ has a tree-shaped counterexample.*

Proof. Assume that $C \not\sqsubseteq_\mathcal{T} D$, and let C', D' be the $\mathcal{ALC}$-concept descriptions obtained by expanding C, D w.r.t. $\mathcal{T}$. Then $C' \sqcap \neg D'$ is satisfiable. It is well-known that the tableau-based satisfiability procedure for $\mathcal{ALC}$ [37] then produces a tree-shaped interpretation $\mathcal{I}$ whose root d_0 satisfies $d_0 \in C'^\mathcal{I} \setminus D'^\mathcal{I}$. Since C', D' do not contain concept names defined in $\mathcal{T}$, and since $\mathcal{T}$ is acyclic, we can assume without loss of generality that $\mathcal{I}$ is a model of $\mathcal{T}$. In fact, otherwise we can modify $\mathcal{I}$ by setting $A^\mathcal{I} := C'^\mathcal{I}_A$ for all defined concepts A, where $A \equiv C_A$ is the definition of A in $\mathcal{T}$, and C'_A is the expansion of C_A w.r.t. $\mathcal{T}$. $\square$

In case $D = \bot$, the statement $C \not\sqsubseteq_\mathcal{T} D$ is equivalent to saying that C is satisfiable w.r.t. $\mathcal{T}$, and thus the lemma also implies that any $\mathcal{ALC}$-concept description that is satisfiable w.r.t. $\mathcal{T}$ has a tree-shaped model, i.e., a tree-shaped model of $\mathcal{T}$ with root $d_0 \in C^\mathcal{I}$. Of course, this and the above lemma also hold when the TBox is empty, i.e., for satisfiability and subsumption of concept descriptions.

Let $\mathcal{I}$ be a tree-shaped model of the acyclic $\mathcal{ALC}$-TBox $\mathcal{T}$, and C_0 be an $\mathcal{ALC}$-concept description. An element d of $\mathcal{I}$ is at *level* k if the unique path from the root d_0 of $\mathcal{I}$ to d has length k. A subdescription F of C_0 is at *level* k if it occurs within k nestings of value and existential restrictions. For example, in the description $A \sqcap \exists r.(B \sqcup \forall r.C)$, the subdescription A occurs at level 0, B occurs at level 1, and C occurs at level 2.

When evaluating C_0 in $\mathcal{I}$, i.e., when checking whether the root d_0 of $\mathcal{I}$ belongs to $C_0^{\mathcal{I}}$, we can directly use the inductive definition of the semantics of $\mathcal{ALC}$-concept descriptions. During this evaluation process, one recursively checks whether certain elements d of $\mathcal{I}$ belong to $F^{\mathcal{I}}$ for subdescriptions F of C_0. It is easy to see that, in such a recursive test, the level of F in C_0 always coincides with the level of d in $\mathcal{I}$. In particular, this means that elements of $\mathcal{I}$ that are at a level higher than the role depth of C_0 are irrelevant when evaluating C_0. The following lemma is an immediate consequence of this observation.

Lemma 15 *Let C_0 be an $\mathcal{ALC}$-concept description of role depth ℓ, and let $\mathcal{I}, \mathcal{I}'$ be tree-shaped interpretations that differ from each other only on elements at levels larger than ℓ. Then $d_0 \in C_0^{\mathcal{I}}$ iff $d_0 \in C_0^{\mathcal{I}'}$, where d_0 is the (common) root of $\mathcal{I}$ and $\mathcal{I}'$.*

In the proof of Proposition 18 we will need a specific result regarding the evaluation of $\mathcal{ALC}$-concept descriptions that are obtained by expanding $\mathcal{ALE}(\mathcal{T})$-concept descriptions, where $\mathcal{T}$ is an acyclic $\mathcal{ALC}$-TBox. Before we can formulate this result in Lemma 17, we must introduce some more notation.

Let C_0 be an $\mathcal{ALC}$-concept description. We define under what conditions a subdescription F of C_0 *occurs conjunctively in* C_0 by induction on the level ℓ of F in C_0:

- if $\ell = 0$, then C_0 must be of the form $F_0 \sqcap F$;[4]
- if $\ell > 0$, then C_0 must be of the form $F_0 \sqcap \exists r.C'$ or $F_0 \sqcap \forall r.C'$, where F occurs conjunctively in C' on level $\ell - 1$.[5]

The following lemma, which can easily be proved by induction on ℓ, links this notion to $\mathcal{ALE}(\mathcal{T})$-concept descriptions. Given an an acyclic $\mathcal{ALC}$-TBox $\mathcal{T}$ and an $\mathcal{ALE}(\mathcal{T})$-concept description C_0, the subdescription F of C_0 is called *positive* if it is not a concept name that occurs within an atomic negation. For example, in the concept description $C_0 = \neg A \sqcap \exists r.\neg B$, the subdescriptions A and B are not positive, but all other subdescriptions (e.g., $\neg A$ or $\exists r.\neg B$) are positive.

[4] The representation of C_0 as $F_0 \sqcap F$ is meant modulo associativity and commutativity of conjunction, and the fact that $\top$ is a unit for conjunction.
[5] Again, this representation of C_0 should be read modulo associativity and commutativity of conjunction, and the fact that $\top$ is a unit for conjunction.

Lemma 16 *Let $\mathcal{T}$ be an acyclic $\mathcal{ALC}$-TBox, and C_0 an $\mathcal{ALE}(\mathcal{T})$-concept description that contains the positive subdescription F at some level ℓ. In addition, let C_0', F' be the $\mathcal{ALC}$-concept descriptions obtained by expanding C_0, F w.r.t. $\mathcal{T}$. Then F' occurs conjunctively in C_0' on level ℓ.*

This lemma will be used to show that the next lemma is applicable in the proof of Proposition 18.

Let C_0 be an $\mathcal{ALC}$-concept description that contains the subdescription F at some level $\ell \geq 0$ conjunctively, and let $\mathcal{I}$ be a tree-shaped interpretation with root d_0 such that $d_0 \in C_0^{\mathcal{I}}$. We modify C_0 into a new $\mathcal{ALC}$-concept description $C_\perp$ by replacing the subdescription F by $\perp$. Now, assume that

- this replacement changes the evaluation of the concept description in $\mathcal{I}$, i.e., $d_0 \notin C_\perp^{\mathcal{I}}$.
- $\neg F$ is satisfiable, and thus there is a tree-shaped interpretation $\mathcal{J}$ with root e_0 such that $e_0 \notin F^{\mathcal{J}}$.

Without loss of generality we may assume that the domains of $\mathcal{I}$ and $\mathcal{J}$ are disjoint.

Lemma 17 *Let C_0 and $\mathcal{I}$ satisfy the properties stated above. Then there is a tree-shaped interpretation $\mathcal{I}'$ with root d_0 that differs from $\mathcal{I}$ only on elements at levels $\geq \ell$ such that $d_0 \notin C_0^{\mathcal{I}'}$.*

Proof. We prove the lemma by induction on ℓ.

Base case: $\ell = 0$. In this case, C_0 is of the form $F_0 \sqcap F$. Let $\mathcal{I}'$ be a renamed copy of $\mathcal{J}$, whose root has the name d_0 instead of e_0. Obviously, $e_0 \notin F^{\mathcal{J}}$ then implies $d_0 \notin F^{\mathcal{I}'}$, and thus $d_0 \notin C_0^{\mathcal{I}'}$.

Induction step: $\ell > 0$. In this case, C_0 is of the form $F_0 \sqcap \exists r.C'$ or $F_0 \sqcap \forall r.C'$, where F is a conjunctive subdescription of C' at level $\ell - 1$. Consequently, $C_\perp$ is of the form $F_0 \sqcap \exists r.C'_\perp$ or $F_0 \sqcap \forall r.C'_\perp$, where $C'_\perp$ is obtained from C' by replacing the subdescription F at level $\ell - 1$ by $\perp$.

First, consider the case where $C_0 = F_0 \sqcap \exists r.C'$ and $C_\perp = F_0 \sqcap \exists r.C'_\perp$. Obviously, $d_0 \in C_0^{\mathcal{I}}$ and $d_0 \notin C_\perp^{\mathcal{I}}$ imply that $d_0 \in (\exists r.C')^{\mathcal{I}}$ and $d_0 \notin (\exists r.C'_\perp)^{\mathcal{I}}$. Let $d_1, \ldots, d_m$ be all the elements of $\mathcal{I}$ that satisfy $(d_0, d_i) \in r^{\mathcal{I}}$ and $d_i \in C'^{\mathcal{I}}$. Now, $d_0 \notin (\exists r.C'_\perp)^{\mathcal{I}}$ implies, for $i = 1, \ldots, m$, that $d_i \notin C'^{\mathcal{I}}_\perp$. Let $\mathcal{I}_1, \ldots, \mathcal{I}_m$ be the tree-shaped interpretations obtained by respectively taking the subtrees of $\mathcal{I}$ with roots $d_1, \ldots, d_m$. For $i = 1, \ldots, m$, we then have $d_i \in C'^{\mathcal{I}_i}$ and $d_i \notin C'^{\mathcal{I}_i}_\perp$. Since F occurs conjunctively at level $\ell - 1$ in C', the induction hypothesis yields a tree-shaped interpretation $\mathcal{I}_i'$ with root d_i that differs from $\mathcal{I}_i$ only on elements at levels $\geq \ell - 1$, and such that $d_i \notin C'^{\mathcal{I}_i'}$.

The interpretation $\mathcal{I}'$ is obtained from $\mathcal{I}$ be replacing, for $i = 1, \dots, m$, the subtree $\mathcal{I}_i$ with root d_i by $\mathcal{I}'_i$. Obviously, $\mathcal{I}$ is tree-shaped and it differs from $\mathcal{I}'$ only on elements at levels $\geq \ell$. We claim that $d_0 \notin (\exists r.C')^{\mathcal{I}'}$, and thus $d_0 \notin C_0^{\mathcal{I}'}$. In fact, let d be such that $(d_0, d) \in r^{\mathcal{I}'}$. By the definition of $\mathcal{I}'$, this implies that $(d_0, d) \in r^{\mathcal{I}}$. If $d = d_i$ for some $i, 1 \leq i \leq m$, then $d = d_i \notin C'^{\mathcal{I}'_i}$, and thus $d = d_i \notin C'^{\mathcal{I}'}$ since the subtree with root d_i of $\mathcal{I}'$ coincides with $\mathcal{I}'_i$. Otherwise, $d \notin C'^{\mathcal{I}}$, and thus $d \notin C'^{\mathcal{I}'}$ since $\mathcal{I}$ coincides with $\mathcal{I}'$ on the respective subtrees with root d.

Second, consider the case where $C_0 = F_0 \sqcap \forall r.C'$ and $C_\bot = F_0 \sqcap \forall r.C'_\bot$. Obviously, $d_0 \in C_0^{\mathcal{I}}$ and $d_0 \notin C_\bot^{\mathcal{I}}$ imply that $d_0 \in (\forall r.C')^{\mathcal{I}}$ and $d_0 \notin (\forall r.C'_\bot)^{\mathcal{I}}$. Let $d_1, \dots, d_m$ be all the elements of $\mathcal{I}$ that satisfy $(d_0, d_i) \in r^{\mathcal{I}}$. Now, $d_0 \in (\forall r.C')^{\mathcal{I}}$ implies $d_i \in C'^{\mathcal{I}}$ for all $i, 1 \leq i \leq m$. In addition, $d_0 \notin (\forall r.C'_\bot)^{\mathcal{I}}$ implies that there exists a $j, 1 \leq j \leq m$, such that $d_j \notin C'^{\mathcal{I}}_\bot$. Let $\mathcal{I}_j$ be the tree-shaped interpretation obtained by taking the subtree of $\mathcal{I}$ with root d_j. Then, we have $d_j \in C'^{\mathcal{I}_j}$ and $d_j \notin C'^{\mathcal{I}_j}_\bot$. Since F occurs conjunctively at level $\ell - 1$ in C', the induction hypothesis yields a tree-shaped interpretation $\mathcal{I}'_j$ with root d_j that differs from $\mathcal{I}_j$ only on elements at levels $\geq \ell - 1$, and such that $d_j \notin C'^{\mathcal{I}'_j}$.

The interpretation $\mathcal{I}'$ is obtained from $\mathcal{I}$ be replacing the subtree $\mathcal{I}_j$ with root d_j by $\mathcal{I}'_j$. Obviously, $\mathcal{I}$ is tree-shaped and it differs from $\mathcal{I}'$ only on elements at levels $\geq \ell$. We claim that $d_0 \notin (\forall r.C')^{\mathcal{I}'}$, and thus $d_0 \notin C_0^{\mathcal{I}'}$. This is an immediate consequence of the following two facts: (i) $(d_0, d_j) \in r^{\mathcal{I}'}$, and (ii) $d_j \notin C'^{\mathcal{I}'_j}$, and thus $d_j \notin C'^{\mathcal{I}'}$ since the subtree with root d_j of $\mathcal{I}'$ coincides with $\mathcal{I}'_j$. $\square$

We are now ready to prove the key proposition.

Proposition 18 *Let C_1, C_2 be $\mathcal{ALE}(\mathcal{T})$-concept descriptions that are both satisfiable w.r.t. $\mathcal{T}$, and assume that the role depths of the $\mathcal{ALC}$-concept descriptions C'_1, C'_2 obtained by expanding the descriptions C_1, C_2 w.r.t. $\mathcal{T}$ are bounded by k. If the $\mathcal{ALE}(\mathcal{T})$-concept description D is a common subsumer of C_1, C_2 w.r.t. $\mathcal{T}$, then there is an $\mathcal{ALE}(\mathcal{T})$-concept description $D_0 \sqsubseteq_{\mathcal{T}} D$ of role depth at most $k+1$ that is also a common subsumer of C_1, C_2 w.r.t. $\mathcal{T}$.*

Proof. Let D be an $\mathcal{ALE}(\mathcal{T})$-concept description that is a common subsumer of C_1, C_2 w.r.t. $\mathcal{T}$. If the role depth of D is bounded by $k+1$, then we are done since we can take $D_0 = D$. Otherwise, D contains at least one subdescription on level $k+1$ that is an existential or a value restriction. Choose such a subdescription F. Obviously, F is positive. We modify D into a concept description $\widehat{D}$ as follows. We replace F by either $\top$ or $\bot$:

- if F is equivalent to $\top$ w.r.t. $\mathcal{T}$, then it is replaced by $\top$;
- otherwise, F is replaced by $\bot$.

Since F is a positive subdescription of E and all the concept constructors other than atomic negation available in $\mathcal{ALE}$ are monotonic, it is clear that $\widehat{D} \sqsubseteq_{\mathcal{T}} D$. It remains to be shown that $\widehat{D}$ is a common subsumer of C_1, C_2 w.r.t. $\mathcal{T}$. In fact, once we have shown this we can obtain D_0 by applying this construction until all subdescriptions at level $k+1$ that are existential or a value restrictions are replaced by either $\top$ or $\bot$. Obviously, the resulting description D_0 has role depth at most $k+1$ and satisfies $D_0 \sqsubseteq_{\mathcal{T}} D$.

If F was replaced by $\top$, then $F \equiv_{\mathcal{T}} \top$, and thus $\widehat{D} \equiv_{\mathcal{T}} D$ is a common subsumer of C_1, C_2 w.r.t. $\mathcal{T}$. Thus, assume that F was replaced by $\bot$. To show that also in this case $\widehat{D}$ is a common subsumer of C_1, C_2 w.r.t. $\mathcal{T}$, we assume to the contrary that $C_i \not\sqsubseteq_{\mathcal{T}} \widehat{D}$ for $i = 1$ or $i = 2$. We show that this assumption leads to a contradiction.

Let $D', \widehat{D}', F'$ be the $\mathcal{ALC}$-concept descriptions obtained by respectively expanding $D, \widehat{D}, F$. By Lemma 16, F' is a subdescription of D' that occurs conjunctively in D' at level $k+1$. In addition, since F was replaced by $\bot$, F is not equivalent to $\top$ w.r.t. $\mathcal{T}$, and thus $\neg F'$ is satisfiable. Since $C_i \not\sqsubseteq_{\mathcal{T}} \widehat{D}$, we know that $C_i' \not\sqsubseteq \widehat{D}'$, and thus there is a tree-shaped interpretation $\mathcal{I}$ such that the root d_0 of this tree belongs to $C_i'^{\mathcal{I}}$, but not to $\widehat{D}'^{\mathcal{I}}$. Since $C_i \sqsubseteq_{\mathcal{T}} D$, we also know that $C_i' \sqsubseteq D'$, and thus $d_0 \in D'^{\mathcal{I}}$.

Now, $d_0 \notin \widehat{D}'^{\mathcal{I}}$ and $d_0 \in D'^{\mathcal{I}}$ together with the satisfiability of $\neg F'$ and the way $\widehat{D}$ was constructed from D imply that Lemma 17 is applicable. Thus, there is a tree-shaped interpretation $\mathcal{I}'$ with root d_0 that differs from $\mathcal{I}$ only on elements at levels $\geq k+1$, and such that $d_0 \notin D'^{\mathcal{I}'}$.

Since a change of the interpretation at a level larger than k does not influence the evaluation of a concept description of depth at most k (see Lemma 15), $d_0 \in C_i'^{\mathcal{I}}$ implies $d_0 \in C_i'^{\mathcal{I}'}$. However, since $C_i \sqsubseteq_{\mathcal{T}} D$ yields $C_i' \sqsubseteq D'$, this implies $d_0 \in D'^{\mathcal{I}'}$, which yields the desired contradiction. $\square$

Theorem 12 is now an immediate consequence of Proposition 13 and Proposition 18. In fact, to compute the lcs of C_1, C_2 w.r.t. $\mathcal{T}$, it is enough to compute the (up to equivalence) finite set of all $\mathcal{ALE}(\mathcal{T})$-concept descriptions of role depth at most $k+1$, check which of them are common subsumers of C_1, C_2 w.r.t. $\mathcal{T}$, and then build the conjunction E of these common subsumers. Proposition 13 ensures that the conjunction is finite. By definition, E is a common subsumer of C_1, C_2 w.r.t. $\mathcal{T}$, and Proposition 18 ensures that for any common subsumer D of C_1, C_2 w.r.t. $\mathcal{T}$, there is a conjunct D_0 in E such that $D_0 \sqsubseteq_{\mathcal{T}} D$, and thus $E \sqsubseteq_{\mathcal{T}} D$.

If we allow for general TBoxes $\mathcal{T}$, then the lcs w.r.t. $\mathcal{T}$ need not exist.

Theorem 19 *Let* $\mathcal{T} := \{A \sqsubseteq \exists r.A, B \sqsubseteq \exists r.B\}$, *where* A, B *are distinct* *concept names. Then, the lcs of the $\mathcal{ALE}(\mathcal{T})$-concept descriptions A, B w.r.t. $\mathcal{T}$ does not exist.*

Proof. Consider a common subsumer E of A, B w.r.t. $\mathcal{T}$. Without loss of generality we can assume that the $\mathcal{ALE}(\mathcal{T})$-concept description E is a conjunction of (negated) concept names, value restrictions, and existential restrictions. We claim that this conjunction can actually only contain existential restrictions for the role r.

Assume that the concept name P is contained in this conjunction. We restrict our attention to the case where P is different from A (otherwise, P is different from B, and we can proceed analogously). Consider the interpretation $\mathcal{I}$ that consists of one element a, which belongs to A and to no other concept name, and which is related to itself via the role r. Then $\mathcal{I}$ is a model of $\mathcal{T}$, and $a \in A^{\mathcal{I}}$. However, $a \notin P^{\mathcal{I}}$, which is a contradiction since P occurs in the top-level conjunction of E, and we have assumed that $A \sqsubseteq_{\mathcal{T}} E$. Similarly, we can show that no negated concept name can occur in this conjunction.

For similar reasons, the conjunction cannot contain a value restriction $\forall s.F$ where F is not equivalent to $\top$ w.r.t. $\mathcal{T}$.[6] In fact, if F is not equivalent to $\top$ w.r.t. $\mathcal{T}$, then there is a model $\mathcal{I}_{\neg F}$ of $\mathcal{T}$ that contains an element d_0 with $d_0 \notin F^{\mathcal{I}_{\neg F}}$. We extend $\mathcal{I}_{\neg F}$ to an interpretation $\mathcal{I}$ by adding a new element a, which belongs to A and to no other concept name, and which is related to itself via the role r, and to d_0 via the role s. Then $\mathcal{I}$ is a model of $\mathcal{T}$, and $a \in A^{\mathcal{I}}$. However, $a \notin (\forall s.F)^{\mathcal{I}}$, which is a contradiction since $A \sqsubseteq_{\mathcal{T}} E$.

Thus, we may assume without loss of generality that both the conjunction of (negated) concept names and the conjunction of value restrictions is empty. Now, consider an existential restriction $\exists s.F$. By using a construction similar to the ones above, we can show that s must in fact be equal to r, i.e., we have an existential restriction of the form $\exists r.F$. We claim that F is again a common subsumer of A, B w.r.t. $\mathcal{T}$. Otherwise, we assume without loss of generality that $A \not\sqsubseteq_{\mathcal{T}} F$, i.e., there is a model $\mathcal{I}_0$ of $\mathcal{T}$ that contains an element d_0 such that $d_0 \in A^{\mathcal{I}_0} \setminus F^{\mathcal{I}_0}$. This is a contradiction to $A \sqsubseteq_{\mathcal{T}} E \sqsubseteq_{\mathcal{T}} \exists r.F$ since using $\mathcal{I}_0$ we can easily construct a model $\mathcal{I}$ of $\mathcal{T}$ that contains an element a that belongs to A, but not to $\exists r.F$. In fact, $\mathcal{I}$ is obtained from $\mathcal{I}_0$ by adding a new element a, which belongs to A and to no other concept name, and which is related to d_0 via the role r.

We can now apply induction over the role depth of the common subsumer E of A, B to show that E is equivalent w.r.t. $\mathcal{T}$ to an $\mathcal{ALE}$-concept description from the following set of descriptions: $\mathcal{S}$ is the smallest set of $\mathcal{ALE}$-concept descriptions such that

[6] If F is equivalent to $\top$, then $\forall s.F$ is equivalent to $\top$, and thus it can be removed.

- $\top$ belongs to $\mathcal{S}$;
- $\mathcal{S}$ is closed under conjunction;
- if F belongs to $\mathcal{S}$, then $\exists r.F$ also belongs to $\mathcal{S}$.

Conversely, it is easy to show (by induction on the size of elements of $\mathcal{S}$) that any element of $\mathcal{S}$ is a common subsumer of A, B w.r.t. $\mathcal{T}$.

Thus, a least common subsumer of A, B w.r.t. $\mathcal{T}$ must be a least element of $\mathcal{S}$. Since the elements of $\mathcal{S}$ do not contain A, B, least is meant w.r.t. subsumption of concept descriptions (i.e., without a TBox). However, $\mathcal{S}$ does not have a least element w.r.t. subsumption. On the one hand, $\mathcal{S}$ obviously contains elements of arbitrary role depth. On the other hand, an element D of role depth ℓ cannot be subsumed by an element E of role depth $k > \ell$: if $d \in E^{\mathcal{I}}$ for some interpretation $\mathcal{I}$ and element d of $\mathcal{I}$, then there is a path of length k starting from d in the graph $G_{\mathcal{I}}$; in contrast, there is an interpretation $\mathcal{I}_0$ and an element d_0 of $\mathcal{I}_0$ such that $d_0 \in D^{\mathcal{I}_0}$, but all paths in $G_{\mathcal{I}_0}$ starting with d_0 have length $\leq \ell < k$. $\square$

It is easy to see that the same proof also works for the cyclic TBox $\mathcal{T} := \{A \equiv \exists r.A, B \equiv \exists r.B\}$.

Corollary 20 *Let $\mathcal{T} := \{A \equiv \exists r.A, B \equiv \exists r.B\}$, where A, B are distinct concept names. Then, the lcs of the $\mathcal{ALE}(\mathcal{T})$-concept descriptions A, B w.r.t. $\mathcal{T}$ does not exist.*

4 Good common subsumers

The brute-force algorithm for computing the lcs in $\mathcal{ALE}(\mathcal{T})$ w.r.t. an acyclic background $\mathcal{ALC}$-TBox described in the previous section is not useful in practice since the number of concept descriptions that must be considered is very large (super-exponential in the role depth). In addition, w.r.t. cyclic or general TBoxes the lcs need not exist. In the bottom-up construction of DL knowledge bases, it is not really necessary to take the *least* common subsumer,[7] a common subsumer that is not too general can also be used. In this section, we introduce an approach for computing such "good" common subsumers w.r.t. a background TBox. In order to explain this approach, we must first recall how the lcs of $\mathcal{ALE}$-concept descriptions (without background TBox) can be computed.

[7] Using it may even result in over-fitting.

Since the lcs of n concept descriptions can be obtained by iterating the application of the binary lcs, we describe how to compute the least common subsumer $\mathsf{lcs}_{\mathcal{ALE}}(C, D)$ of two $\mathcal{ALE}$-concept descriptions C, D (see [22] for more details and a proof of correctness).

First, the input descriptions C, D are normalized by applying the following equivalence-preserving rules modulo associativity and commutativity of conjunction:

$$\begin{aligned}
\forall r.E \sqcap \forall r.F &\longrightarrow \forall r.(E \sqcap F), & \forall r.E \sqcap \exists r.F &\longrightarrow \forall r.E \sqcap \exists r.(E \sqcap F),\\
\forall r.\top &\longrightarrow \top, & E \sqcap \top &\longrightarrow E,\\
\exists r.\bot &\longrightarrow \bot, & E \sqcap \bot &\longrightarrow \bot,\\
A \sqcap \neg A &\longrightarrow \bot \text{ for each } A \in N_C.
\end{aligned}$$

Note that, due to the second rule in the first line, this normalization may lead to an exponential blow-up of the concept descriptions. In the following, we assume that the input descriptions C, D are normalized.

In order to describe the lcs algorithm, we need to introduce some notation. Let C be a normalized $\mathcal{ALE}$-concept description. Then $\mathsf{names}(C)$ ($\overline{\mathsf{names}}(C)$) denotes the set of (negated) concept names occurring in the top-level conjunction of C, $\mathsf{roles}^{\exists}(C)$ ($\mathsf{roles}^{\forall}(C)$) the set of role names occurring in an existential (value) restriction on the top-level of C, and $\mathsf{restrict}_r^{\exists}(C)$ ($\mathsf{restrict}_r^{\forall}(C)$) denotes the set of all concept descriptions occurring in an existential (value) restriction on the role r in the top-level conjunction of C.

Now, let C, D be normalized $\mathcal{ALE}$-concept descriptions. If C (D) is equivalent to $\bot$, then $\mathsf{lcs}_{\mathcal{ALE}}(C, D) = D$ ($\mathsf{lcs}_{\mathcal{ALE}}(C, D) = C$). Otherwise, we have

$$\begin{aligned}
\mathsf{lcs}_{\mathcal{ALE}}(C, D) = {} & \bigsqcap_{A \in \mathsf{names}(C) \cap \mathsf{names}(D)} A \ \sqcap \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C) \cap \overline{\mathsf{names}}(D)} \neg B \ \sqcap \\
& \bigsqcap_{r \in \mathsf{roles}^{\exists}(C) \cap \mathsf{roles}^{\exists}(D)} \ \bigsqcap_{E \in \mathsf{restrict}_r^{\exists}(C), F \in \mathsf{restrict}_r^{\exists}(D)} \exists r.\mathsf{lcs}_{\mathcal{ALE}}(E, F) \ \sqcap \\
& \bigsqcap_{r \in \mathsf{roles}^{\forall}(C) \cap \mathsf{roles}^{\forall}(D)} \ \bigsqcap_{E \in \mathsf{restrict}_r^{\forall}(C), F \in \mathsf{restrict}_r^{\forall}(D)} \forall r.\mathsf{lcs}_{\mathcal{ALE}}(E, F).
\end{aligned}$$

Here, the empty conjunction stands for the top-concept $\top$. The recursive calls of $\mathsf{lcs}_{\mathcal{ALE}}$ are well-founded since the role depth decreases with each call.

Let $\mathcal{T}$ be a background TBox in some DL $\mathcal{L}_2$ extending $\mathcal{ALE}$ such that subsumption in $\mathcal{L}_2$ w.r.t. this kind of TBoxes is decidable.[8] Let C, D be normalized $\mathcal{ALE}(\mathcal{T})$-concept descriptions. If we ignore the TBox, then we can simply apply the above algorithm for $\mathcal{ALE}$-concept descriptions without background terminology to compute a common subsumer. However, in this context, taking

$$\bigsqcap_{A \in \mathsf{names}(C) \cap \mathsf{names}(D)} A \ \sqcap \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C) \cap \overline{\mathsf{names}}(D)} \neg B$$

is not the best we can do. In fact, some of these concept names may be constrained by the TBox, and thus there may be relationships between them that we ignore by simply using the intersection. Instead, we propose to take the smallest (w.r.t. subsumption w.r.t. $\mathcal{T}$) conjunction of concept names and negated concept names that subsumes (w.r.t. $\mathcal{T}$) both

$$\bigsqcap_{A \in \mathsf{names}(C)} A \sqcap \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C)} \neg B \qquad \text{and} \qquad \bigsqcap_{A' \in \mathsf{names}(D)} A' \sqcap \bigsqcap_{\neg B' \in \overline{\mathsf{names}}(D)} \neg B'.$$

We modify the above lcs algorithm in this way, not only on the top-level of the input concepts, but also in the recursive steps. It is easy to show that the $\mathcal{ALE}(\mathcal{T})$-concept description computed by this modified algorithm still is a common subsumer of A, B w.r.t. $\mathcal{T}$.

Proposition 21 *The $\mathcal{ALE}(\mathcal{T})$-concept description E obtained by applying the modified lcs algorithm to $\mathcal{ALE}(\mathcal{T})$-concept descriptions C, D is a common subsumer of C and D w.r.t. $\mathcal{T}$, i.e., $C \sqsubseteq_{\mathcal{T}} E$ and $D \sqsubseteq_{\mathcal{T}} E$.*

In general, this common subsumer will be more specific than the one obtained by ignoring $\mathcal{T}$, though it need not be the least common subsumer. In the following, we will call the common subsumer computed this way *good common subsumer (gcs)*, and the algorithm that computes it the *gcs algorithm.*

Example 22 As a simple example, consider the $\mathcal{ALC}$-TBox $\mathcal{T}$:

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

[8] Note that the TBox $\mathcal{T}$ used as background terminology may be a general TBox.

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

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

By ignoring the TBox, we obtain the $\mathcal{ALE}(\mathcal{T})$-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 the concept $\mathsf{ChildrenDoctor}$, then we obtain the more specific common subsumer $\exists\mathsf{has\text{-}child}.\mathsf{ChildrenDoctor}$. The gcs of C, D is even more specific. In fact, the least conjunction of (negated) concept names subsuming both $\mathsf{NoSon} \sqcap \mathsf{DaughterHappyDoctor}$ and $\mathsf{NoDaughter} \sqcap \mathsf{SonRichDoctor}$ is

$$\mathsf{ChildrenDoctor} \sqcap \mathsf{DaughterHappyDoctor} \sqcap \mathsf{SonRichDoctor},$$

and thus the gcs of C, D is

$$\exists\mathsf{has\text{-}child}.(\mathsf{ChildrenDoctor} \sqcap \mathsf{DaughterHappyDoctor} \sqcap \mathsf{SonRichDoctor}).$$

The conjunct $\mathsf{ChildrenDoctor}$ is actually redundant since it is implied by the remainder of the conjunction.

In order to implement the gcs algorithm, we must be able to compute the smallest conjunction of (negated) concept names that subsumes two such conjunctions C_1 and C_2 w.r.t. $\mathcal{T}$. In principle, one can compute this smallest conjunction by testing, for every (negated) concept name whether it subsumes both C_1 and C_2 w.r.t. $\mathcal{T}$, and then take the conjunction of those (negated) concept names for which the test was positive. However, this results in a large number of (possibly quite expensive) calls to the subsumption algorithm for $\mathcal{L}_2$ w.r.t. (general or (a)cyclic) TBoxes. Since in our application scenario (bottom-up construction of DL knowledge bases w.r.t. a given background terminology), the TBox $\mathcal{T}$ is assumed to be fixed, it makes sense to precompute this information. In Section 5 we will show that attribute exploration can be used for this purpose.

4.3 Using $\mathcal{ALE}$-expansion when computing the gcs

If the background terminology is an acyclic TBox $\mathcal{T}$, then one can employ an appropriate partial expansion of $\mathcal{T}$ in order to uncover $\mathcal{ALE}$-parts hidden within the defined concepts. The idea is that the gcs algorithm will possibly yield a more specific common subsumer if it can make use of $\mathcal{ALE}$-concepts "hidden" within the defined concepts.

For instance, in Example 22, the concepts defining NoSon and $\mathsf{NoDaughter}$ are actually $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and thus C, D can be expanded to

$$C' := \exists \mathsf{has\text{-}child}.(\forall \mathsf{has\text{-}child}.\mathsf{Female} \sqcap \mathsf{DaughterHappyDoctor}),$$
$$D' := \exists \mathsf{has\text{-}child}.(\forall \mathsf{has\text{-}child}.\neg\mathsf{Female} \sqcap \mathsf{SonRichDoctor}),$$

before computing the gcs. The concepts defining $\mathsf{DaughterHappyDoctor}$ and $\mathsf{SonRichDoctor}$ are not $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and thus these two names cannot be expanded. However, in this example, the common subsumer computed by applying the gcs algorithm to the expanded concepts C', D' is

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

which is actually less specific than the result of applying the gcs algorithm to the unexpanded concepts C, D. To overcome this problem, we do the partial expansion, but also keep the defined concepts that we have expanded. In the example, this yields the expanded concepts

$$C'' := \exists \mathsf{has\text{-}child}.(\forall \mathsf{has\text{-}child}.\mathsf{Female} \sqcap \mathsf{NoSon} \sqcap \mathsf{DaughterHappyDoctor}),$$
$$D'' := \exists \mathsf{has\text{-}child}.(\forall \mathsf{has\text{-}child}.\neg\mathsf{Female} \sqcap \mathsf{NoDaughter} \sqcap \mathsf{SonRichDoctor}).$$

If we apply the gcs algorithm to C'', D'', then we obtain (up to equivalence w.r.t. $\mathcal{T}$) the same common subsumer as obtained from C, D, i.e., in this case the expansion does not yield a more specific result.

However, it is easy to construct examples where this kind of expansion leads to better results. For instance, if we apply the gcs algorithm to $\forall \mathsf{has\text{-}child}.(\mathsf{Female} \sqcap \mathsf{Doctor})$ and $\mathsf{NoSon} \sqcap \forall \mathsf{has\text{-}child}.\mathsf{Happy}$, then the result is $\top$. In contrast, if we apply it to the expanded concept descriptions $\forall \mathsf{has\text{-}child}.(\mathsf{Female} \sqcap \mathsf{Doctor})$ and $\mathsf{NoSon} \sqcap \forall \mathsf{has\text{-}child}.\mathsf{Female} \sqcap \forall \mathsf{has\text{-}child}.\mathsf{Happy}$, then the result is the more specific common subsumer $\forall \mathsf{has\text{-}child}.\mathsf{Female}$.

Before checking whether a defined concept can be expanded, it is useful to transform it into *negation normal form (NNF)* by pushing all negations into the description until they occur only in front of concept names, using de Morgan' rules and the facts that $\neg\neg D \equiv D$ and $\neg\top \equiv \bot$. For example, the concept description $\neg\forall \mathsf{has\text{-}child}.\mathsf{Female}$ is not an $\mathcal{ALE}$-concept description, but its negation normal form $\exists \mathsf{has\text{-}child}.\neg\mathsf{Female}$ is.

More formally, we define the *$\mathcal{ALE}$-expansion* of (negated) concept names defined in $\mathcal{T}$ and of $\mathcal{ALE}(\mathcal{T})$-concept descriptions as follows.

Definition 23 ($\mathcal{ALE}$-expansion) *Let $\mathcal{T}$ be an acyclic TBox, let A be a concept name defined in $\mathcal{T}$, and let $A \equiv C$ be its definition. We first build the negation normal form C' of C. If C' is not an $\mathcal{ALE}(\mathcal{T})$-concept description,* *then the $\mathcal{ALE}$-expansion of A is A. Otherwise, it is $A \sqcap C''$, where C'' is obtained from C' by replacing all (negated) defined concept names in C' by their $\mathcal{ALE}$-expansion. To obtain the $\mathcal{ALE}$-expansion of $\neg A$, we just apply the same approach to $\neg C$. The $\mathcal{ALE}$-expansion of an $\mathcal{ALE}(\mathcal{T})$-concept description is obtained by replacing all (negated) defined concept names by their $\mathcal{ALE}$-expansions.*

Note that this recursive definition of an $\mathcal{ALE}$-expansion is well-founded since the TBox is assumed to be acyclic. As an example, consider the TBox $\mathcal{T}$ consisting of

$$A \equiv \neg\forall r.(B_1 \sqcup B_2), \quad B_1 \equiv P \sqcup Q, \quad B_2 \equiv P \sqcap Q.$$

Then we obtain $A \sqcap \exists r.(\neg B_1 \sqcap \neg P \sqcap \neg Q \sqcap \neg B_2)$ as $\mathcal{ALE}$-expansion of A.

It is easy to see that $\mathcal{ALE}$-expansion may lead to more specific common subsumers, but never to less specific ones.

Proposition 24 *Let $\mathcal{T}$ be an acyclic $\mathcal{L}_2$-TBox, C, D $\mathcal{ALE}(\mathcal{T})$-concept descriptions with $\mathcal{ALE}$-expansion C', D', and let E (E') be the result of applying the gcs algorithm to C, D (C', D'). Then E' is a common subsumer of C, D that is at least as good as E, i.e., $C \sqsubseteq_{\mathcal{T}} E'$, $D \sqsubseteq_{\mathcal{T}} E'$, and $E' \sqsubseteq_{\mathcal{T}} E$.*

$\mathcal{ALE}$-expansion can also be applied to cyclic $\mathcal{L}_2$-TBoxes, provided that the cycles go through non-$\mathcal{ALE}$ parts of the TBox. This is, for example, the case in the TBox $\mathcal{T} := \{A \equiv \exists r.B,\ B \equiv P \sqcup A\}$.

4.4 Alternative approaches for computing common subsumers

In Section 2.1 we have already sketched an approach based on approximation, which works if the TBox $\mathcal{T}$ is acyclic, $\mathcal{L}_2$ allows for disjunction, and one can compute the approximation from above of $\mathcal{L}_2$-concept descriptions by $\mathcal{ALE}$-concept descriptions. For example, if we take $\mathcal{ALC}$ as $\mathcal{L}_2$, then all these conditions are satisfied.

Definition 25 (Common subsumer by approximation) *Assume that $\mathcal{L}_2$ allows for disjunction, and that one can compute the approximation from above of $\mathcal{L}_2$-concept descriptions by $\mathcal{ALE}$-concept descriptions. Let $\mathcal{T}$ be an acyclic $\mathcal{L}_2$-TBox. Given $\mathcal{ALE}(\mathcal{T})$-concept descriptions C, D, one first fully expands them into $\mathcal{L}_2$-concept descriptions C', D'. Then one approximates their disjunction $C' \sqcup D'$ from above by an $\mathcal{ALE}$-concept description. The common subsumer of C, D obtained this way is called the* common subsumer by approximation (acs) *of C, D w.r.t. $\mathcal{T}$.*

In Section 2.1, we have shown by an example that the acs can be less specific than the lcs. In this example (Example 4), the gcs coincides with the lcs, and thus is also more specific than the acs: in fact, w.r.t. the TBox $\mathcal{T} = \{A \equiv P \sqcup Q\}$, the smallest conjunction of concept names above both P and Q is A, and thus the gcs of P and Q is A.

There are, however, also examples where the gcs is less specific than the acs. For instance, consider the TBox

$$\mathcal{T} = \{A \equiv \exists r.A_1 \sqcup \exists r.A_2, \quad B \equiv \exists r.B_1 \sqcup \exists r.B_2\}.$$

With respect to this TBox, the gcs of A, B is $\top$, whereas the acs is the more specific common subsumer $\exists r.\top$.

The gcs algorithm makes use of the subsumption relationships between conjunctions of (negated) concept names. Usually, these relationships are not known for a given TBox, and thus we must either precompute them (see Section 5) or compute them on the fly (as sketched in Section 4.2). Both may be quite expensive. What is usually known for a given TBox $\mathcal{T}$ are all subsumption relationships between the concept names occurring in $\mathcal{T}$.[9] This information can be used as follows. Given two conjunctions

$$\bigsqcap_{A \in \mathsf{names}(C)} A \sqcap \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C)} \neg B \qquad \text{and} \qquad \bigsqcap_{A' \in \mathsf{names}(D)} A' \sqcap \bigsqcap_{\neg B' \in \overline{\mathsf{names}}(D)} \neg B',$$

the gcs algorithm takes the smallest (w.r.t. subsumption w.r.t. $\mathcal{T}$) conjunction of concept names and negated concept names that subsumes (w.r.t. $\mathcal{T}$) both conjunctions. In contrast, the algorithm that just ignores the TBox would take

$$\bigsqcap_{A \in \mathsf{names}(C) \cap \mathsf{names}(D)} A \sqcap \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C) \cap \overline{\mathsf{names}}(D)} \neg B.$$

Using the subsumption relationships between concept names, we can come up with a new approach that lies between these two approaches.

Definition 26 (Subsumption closure) *Let $\mathcal{T}$ be a TBox, and S ($\overline{S}$) a set of (negated) concept names. The* subsumption closure *of S ($\overline{S}$) w.r.t. $\mathcal{T}$ is a set of (negated) concept names, which is defined as follows:*

$$\begin{aligned}\mathsf{SC}(S) &:= \{A \mid \exists B \in S.\ B \sqsubseteq_{\mathcal{T}} A\},\\ \mathsf{SC}(\overline{S}) &:= \{\neg A \mid \exists \neg B \in \overline{S}.\ A \sqsubseteq_{\mathcal{T}} B\}.\end{aligned}$$

Instead of using the intersection $\mathsf{names}(C) \cap \mathsf{names}(D)$ ($\overline{\mathsf{names}}(C) \cap \overline{\mathsf{names}}(D)$), as in the approach that ignores $\mathcal{T}$, one can first build the subsumption closures,

[9] Most DL system compute these automatically when reading in a TBox.

and then intersect the closures, i.e., use

$$\bigsqcap_{A \in \mathsf{SC}(\mathsf{names}(C)) \cap \mathsf{SC}(\mathsf{names}(D))} A \;\sqcap \bigsqcap_{\neg B \in \mathsf{SC}(\overline{\mathsf{names}}(C)) \cap \mathsf{SC}(\overline{\mathsf{names}}(D))} \neg B.$$

We call the algorithm for computing common subsumers obtained this way the *scs algorithm*, and the result of applying it to $\mathcal{ALE}(\mathcal{T})$-concept descriptions C, D the *scs of* C, D *w.r.t.* $\mathcal{T}$.

Proposition 27 *Let $\mathcal{T}$ be an $\mathcal{L}_2$-TBox, C, D $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and let E (E') be the result of applying the gcs (scs) algorithm to C, D. Then E' is a common subsumer of C, D that is at most as good as the gcs E, i.e., $C \sqsubseteq_{\mathcal{T}} E'$, $D \sqsubseteq_{\mathcal{T}} E'$, and $E \sqsubseteq_{\mathcal{T}} E'$.*

5 Computing the subsumption lattice of conjunctions of (negated) concept names w.r.t. a TBox

To obtain a practical gcs algorithm, we must be able to compute in an efficient way the smallest conjunction of (negated) concept names that subsumes two such conjunctions w.r.t. $\mathcal{T}$. Since in our application scenario (bottom-up construction of DL knowledge bases w.r.t. a given background terminology), the TBox $\mathcal{T}$ is assumed to be fixed, it makes sense to precompute this information. Obviously, a naïve approach that calls the subsumption algorithm for each pair of conjunctions of (negated) concept names is too expensive for TBoxes of a realistic size. Instead, we propose to use attribute exploration for this purpose.

In order to apply attribute exploration to the task of computing the subsumption lattice[10] of conjunctions of (negated) concept names (some of which may be defined concepts in an $\mathcal{L}_2$-TBox $\mathcal{T}$), we define a formal context whose concept lattice is isomorphic to the subsumption lattice we are interested in.

For the case of conjunctions of concept names (without negated names), this problem was first addressed in [51], where the objects of the context were basically all possible counterexamples to subsumption relationships, i.e., interpretations together with an element of the interpretation domain. The resulting "semantic context" has the disadvantage that an "expert" for this context must be able to deliver such counterexamples, i.e., it is not sufficient to have a simple subsumption algorithm for the DL in question. One needs one that, given a subsumption problem "$C \sqsubseteq D$?", is able to compute a counterexample

[10] In general, the subsumption relation induces a partial order, and not a lattice structure on concepts. However, in the case of conjunctions of (negated) concept names, all infima exist, and thus also all suprema.

if the subsumption relationship does not hold, i.e., an interpretation $\mathcal{I}$ and an element d of its domain such that $d \in C^{\mathcal{I}} \setminus D^{\mathcal{I}}$. Since the usual tableau-based subsumption algorithms [52] 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. For instance, in [51], this is explicitly shown for the DL $\mathcal{ALC}$. 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. Such a context was first introduced in [53]:

Definition 28 *Let $\mathcal{T}$ be an $\mathcal{L}_2$-TBox. The context $\mathcal{K}_{\mathcal{T}} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ is defined as follows:*

$\mathcal{O} := \{E \mid E$ *is an $\mathcal{L}_2$-concept description*$\}$,
$\mathcal{P} := \{A_1, \ldots, A_n\}$ *is the set of concept names occurring in* $\mathcal{T}$,
$\mathcal{S} := \{(E, A) \mid E \sqsubseteq_{\mathcal{T}} A\}$.

Before we can prove that this context has the desired properties, it is useful to show the following lemma, which relates subsumption in $\mathcal{T}$ with implications holding in $\mathcal{K}_{\mathcal{T}}$. If B is a set of attributes of $\mathcal{K}_{\mathcal{T}}$ (i.e., a set of concept names occurring in $\mathcal{T}$), then $\sqcap B$ denotes their conjunction, i.e., $\sqcap B := \bigsqcap_{A \in B} A$.

Lemma 29 *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 subsumed by all concepts in B_1, then it is also subsumed by 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 $\mathcal{K}_{\mathcal{T}}$. □

Given this lemma, we can now prove the main theorem of this section:

Theorem 30 *The concept lattice of the context $\mathcal{K}_{\mathcal{T}}$ is isomorphic to the sub-*

sumption 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 $B_1 \to B_2$ 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 $[\sqcap B_1]_\equiv \sqsubseteq_\mathcal{T}^\equiv [\sqcap B_2]_\equiv$.

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 $B \to B''$ holds in $\mathcal{K}_\mathcal{T}$, and thus Lemma 29 yields $\sqcap B \sqsubseteq_\mathcal{T} \sqcap B''$. □

Attribute exploration can be used to compute the concept lattice of $\mathcal{K}_\mathcal{T}$ since any standard subsumption algorithm for the DL under consideration is an expert for $\mathcal{K}_\mathcal{T}$.

Proposition 31 *Any decision procedure for subsumption w.r.t. TBoxes in $\mathcal{L}_2$ 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 29, 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 $\sqcap B_1 \not\sqsubseteq_\mathcal{T} \sqcap B_2$.

The expert must provide for each counterexample the information which attributes it satisfies and which it does not. This can again be realized by the subsumption algorithm: $\sqcap B_1$ satisfies the attribute A_i iff $\sqcap B_1 \sqsubseteq_\mathcal{T} A_i$. □

In order to compute the gcs, we consider not only conjunctions of concept names, but rather conjunctions of concept names *and negated* concept names. The above results can easily be adapted to this case. In fact, one can simply extend the TBox $\mathcal{T}$ by a definition for each negated concept name, and then apply the approach to this extended TBox. To be more precise, if $\{A_1, \ldots, A_n\}$ is the set of concept names occurring in $\mathcal{T}$, then we introduce new concept names $\overline{A}_1, \ldots, \overline{A}_n$, and extend $\mathcal{T}$ to a TBox $\widehat{\mathcal{T}}$ by adding the definitions $\overline{A}_1 \equiv \neg A_1, \ldots, \overline{A}_n \equiv \neg A_n$.[11]

Corollary 32 *The concept lattice of the context $\mathcal{K}_{\widehat{\mathcal{T}}}$ is isomorphic to the subsumption hierarchy of all conjunctions of concept names and negated concept names occurring in $\mathcal{T}$.*

How can this result be used to support computing the gcs? The above corollary together with Proposition 31 shows that attribute exploration applied to $\mathcal{K}_{\widehat{\mathcal{T}}}$ can be used to compute the Duquenne-Guigues base of $\mathcal{K}_{\widehat{\mathcal{T}}}$. Given this base, we can compute the supremum in the concept lattice of $\mathcal{K}_{\widehat{\mathcal{T}}}$ as follows:

Lemma 33 *Let $\mathcal{J}$ be the Duquenne-Guigues base of $\mathcal{K}_{\widehat{\mathcal{T}}}$, and let B_1, B_2 be sets of attributes of $\mathcal{K}_{\widehat{\mathcal{T}}}$. Then $\mathcal{J}(B_1) \cap \mathcal{J}(B_2)$ is the intent of the supremum of the formal concepts (B_1', B_1'') and (B_2', B_2'').*

Proof. We know that the intent of the supremum of the formal concepts (B_1', B_1'') and (B_2', B_2'') is just the intersection $B_1'' \cap B_2''$ of their intents. In addition, since $Cons(\mathcal{J}) = Imp(\mathcal{K}_{\widehat{\mathcal{T}}})$, we know that $B_i'' = \mathcal{J}(B_i)$ for $i = 1, 2$. $\square$

As an immediate consequence of this lemma together with Theorem 30 and its proof, the supremum in the hierarchy of all conjunctions of concept names and negated concept names occurring in $\mathcal{T}$ can be computed as follows:

Proposition 34 *Let $\mathcal{J}$ be the Duquenne-Guigues base of $\mathcal{K}_{\widehat{\mathcal{T}}}$, and let B_1, B_2 be sets of (negated) concept names occurring in $\mathcal{T}$. Then*

$$\bigsqcap_{L \in \mathcal{J}(B_1) \cap \mathcal{J}(B_2)} L$$

is the least conjunction of (negated) concept names occurring in $\mathcal{T}$ that lies above both $\bigsqcap_{L \in B_1} L$ and $\bigsqcap_{L \in B_2} L$.

As mentioned in Section 2.2, computing the implication hull $\mathcal{J}(B)$ for a set of attributes B can be done in time linear in the size of $\mathcal{J}$ and B. This means that the supremum can be computed efficiently as long as the Duquenne-Guigues base of $\mathcal{K}_{\widehat{\mathcal{T}}}$ is relatively small.

The experimental results reported in [53] show that using attribute exploration for computing the subsumption lattice of all conjunctions of (negated)

[11] For $\widehat{\mathcal{T}}$ to be an $\mathcal{L}_2$-TBox, we must assume that $\mathcal{L}_2$ allows for full negation.

concept names gives a huge increase of efficiency compared to the semi-naïve approach, which introduces a new definition for each of the exponentially many such conjunctions, and then applies the usual algorithm for computing the subsumption hierarchy. Nevertheless, these results also show that the approach can only be applied if the number of concept names is relatively small. Although these experiments were done almost 10 years ago on a rather slow computer, using randomly generated TBoxes and the semantic context, our more recent experiments (see Section 6) come to a similar result. For this reason, we have also experimented with an improved algorithm for computing concept lattices [33,34], which can employ additional knowledge that is readily available in our case, but not used by the basic attribute exploration algorithm.

Using attribute exploration with a priori knowledge

When starting the exploration process, all the basic attribute exploration algorithm knows about the context is the set of its attributes. It acquires all the necessary knowledge about the context by asking the expert, which in our setting means: by calling the subsumption algorithm for $\mathcal{L}_2$. Since $\mathcal{L}_2$ is usually an expressive DL, the complexity of the subsumption problem is usually quite high, and thus asking the expert may be expensive. For this reason it makes sense to employ approaches that can avoid some of these expensive calls of the subsumption algorithm.

In our application, we already have some a priori knowledge about relationships between attributes. In fact, we know that the attributes A_i and $\overline{A}_i$ stand for a concept and its negation. In addition, since the background TBox $\mathcal{T}$ is assumed to be an existing terminology, we can usually assume that the subsumption hierarchy between the concept names occurring in $\mathcal{T}$ has already been computed. This provides us with the following *a priori knowledge* about the relationships between attributes:

(1) If $A_i \sqsubseteq_{\mathcal{T}} A_j$ holds, then we know on the FCA side that in the context $\mathcal{K}_{\widehat{\mathcal{T}}}$ all objects satisfying the attribute A_i also satisfy the attribute A_j, i.e., the implication $\{A_i\} \rightarrow \{A_j\}$ holds in $\mathcal{K}_{\widehat{\mathcal{T}}}$.
(2) Since $A_i \sqsubseteq_{\mathcal{T}} A_j$ implies $\neg A_j \sqsubseteq_{\mathcal{T}} \neg A_i$, we also know that all objects satisfying the attribute $\overline{A}_j$ also satisfy the attribute $\overline{A}_i$, i.e., the implication $\{\overline{A}_j\} \rightarrow \{\overline{A}_i\}$ holds in $\mathcal{K}_{\widehat{\mathcal{T}}}$.
(3) We know that no object can simultaneously satisfy A_i and $\overline{A}_i$, and thus the implication $\{A_i, \overline{A}_i\} \rightarrow \bot^{\mathcal{K}_{\widehat{\mathcal{T}}}}$ holds, where $\bot^{\mathcal{K}_{\widehat{\mathcal{T}}}}$ stands for the set of all attributes of $\mathcal{K}_{\widehat{\mathcal{T}}}$.
(4) Every object satisfies either A_i or $\overline{A}_i$.

The a priori knowledge mentioned in (4) differs from the one mentioned in the other points in that it cannot be represented by Horn clauses. This is the reason why it does not correspond to an implication of the context. In addition, whereas reasoning with respect to Horn clauses (implications) is polynomial, the presence of knowledge of the form mentioned in (4) means that reasoning about the a priori knowledge becomes NP-complete (since it is general propositional reasoning).

Depending on the TBox, there may exist other relationships between attributes that can be deduced, but it should be noted that deducing them makes sense only if this can be done without too much effort: otherwise, the advantage obtained by using the information might be outweighed by the effort of obtaining the a priori knowledge.

Attribute exploration with a priori knowledge [54,33,34] is able to use such additional information (like the one mentioned in (1)–(4)) to create less calls to the expert and generate less additional implications.[12]

If the *a priori knowledge is purely implicational* (in our case, if we use only the implications mentioned in (1)–(3)), then it is easy to modify the attribute exploration algorithm such that it takes this knowledge into account. In fact, in the initialization phase we simply replace the assignment $\mathcal{J}_0 := \emptyset$ by

$$\begin{aligned}\mathcal{J}_0 := \{&\{A_i\} \to \{A_j\} \mid A_i \sqsubseteq_{\mathcal{T}} A_j\} \cup \\ \{&\{\overline{A}_j\} \to \{\overline{A}_i\} \mid A_i \sqsubseteq_{\mathcal{T}} A_j\} \cup \\ \{&\{A_i, \overline{A}_i\} \to \bot^{\mathcal{K}_{\widehat{\mathcal{T}}}} \mid i = 1, \ldots, n\},\end{aligned}$$

where $\bot^{\mathcal{K}_{\widehat{\mathcal{T}}}} = \{A_1, \ldots, A_n, \overline{A}_1, \ldots, \overline{A}_n\}$. The effect of this modification is that, when computing the implication hull $\mathcal{J}_{i+1}(\cdot)$ during attribute exploration, these a priori known implications are also taken into account. The other effect is, of course, that the overall set of implications obtained by the exploration process need not be of minimal cardinality or even free of redundancies.

In principle, *non-implicational a priori knowledge* (in our case, the one mentioned in (4)) can be utilized in the same way. In this case, the implication hull $\mathcal{J}_{i+1}(\cdot)$ is replaced by the deductive closure, i.e., given a set of computed implications $\mathcal{J}_{i+1}$, the a priori knowledge Γ (which is a finite set of propositional formulae), and a set of attributes B (which we view as propositional variables), we ask which other propositional variables follow from $\Gamma \cup \Gamma_{\mathcal{J}_{i+1}} \cup \{\bigwedge_{b \in B} b\}$, where $\Gamma_{\mathcal{J}_{i+1}}$ is the set of propositional formulae obtained by translating the implications in $\mathcal{J}_{i+1}$ to the corresponding propositional formulae.

[12] In [54,33,34], the a priori knowledge is actually called background knowledge. Here we prefer the name "a priori knowledge" to avoid confusion with our notion of a background TBox.

The problem with using the non-implicational knowledge in this way is that computing the deductive closure of propositional formulae is an NP-complete problem, whereas computing the implication hull is polynomial. During the attribute exploration process, it might make sense to use such a more complex closure operator if this saves us calls to the (in general even more complex) subsumption algorithm realizing the expert. However, we also need to use this closure operation when computing the supremum of two conjunctions of (negated) concept names during the gcs computation. Computing the concept lattice is done only once for the given background TBox, whereas the supremum operation is executed several times whenever the user wants the system to compute a gcs. For this reason, the algorithm for computing the supremum operation should be very efficient, and thus we do not want to use full propositional reasoning when computing the supremum.

This does not mean that the non-implicational a priori knowledge cannot be used at all during attribute exploration. For example, one can use it to *optimize the expert*. In fact, assume that we use only the implicational knowledge when computing the hull during attribute exploration. If the exploration process generates an implication question "$B_i \to B_i'' \setminus B_i$," then one can first check (by propositional reasoning) whether this implication follows from the implications together with the non-implicational a priori knowledge. If the answer is "yes," then one knows that this is a valid implication. Only if the answer is "no" does one need to call the expert. These propositional pre-test make sense if the expert is realized by an algorithm that is more complex than the algorithm used for propositional reasoning. Since this approach only optimizes the expert, the set of implications computed by it is identical to the one computed when using only the implicational a priori knowledge. Thus, the supremum can be obtained by computing only the implication hull.

6 Experimental results

In order to obtain a first impression of the applicability of the gcs algorithm and to identify parts where further optimization is necessary, we have done experiments using several rather small background TBoxes. The reason for using small knowledge bases is that computing the subsumption hierarchy of all conjunctions of (negated) concept names is rather time consuming. In fact, if the TBox contains n concept names (this includes the primitive ones), then the corresponding context has $2n$ attributes, and in the worst case attribute exploration must run through 2^{2n} iterations. Since this is done only once for a given background TBox, long run-times are not prohibitive as long as the computed implication base is rather small, and thus computing the supremum in the gcs algorithm is fast. However, these long run-times would have prevented us from experimenting with different kinds of TBoxes.

TBox name	DL	number concept names	number role names	size expanded TBox	role depth expanded TBox	layers subs. hier.
DICE1	$\mathcal{ALC}$	10	5	88	1	4
DICE2	$\mathcal{ALC}$	12	4	cyclic	cyclic	3
DICE3	$\mathcal{ALC}$	13	4	cyclic	cyclic	3
PA-6	$\mathcal{ALE}$	12	3	118	3	3
HC	Boolean	14	0	33	0	4
Family	$\mathcal{ALC}$	9	1	18	1	2

Table 2
The background TBoxes used in our experiments

The experiments were performed on a machine with a Pentium 4 processor at 2.40 GHz and 2GB of memory, under Linux. The implementation was made in LISP using version 19a of CMU Common Lisp. We used version 1.7.23 of the DL System RACER [8] as the expert answering subsumption questions during attribute exploration. For the computation of the implication hull, we implemented the linear time implicational closure algorithm *linclosure* described in Section 4.6 of [48].

6.1 The background TBoxes

In order to obtain experimental results that are relevant in practice, we used TBoxes that closely resemble fragments of knowledge bases from applications.

Three such fragments, called DICE1, DICE2, and DICE3 in the following, were obtained from the DICE knowledge base [55], which comes from a medical application and defines concepts from the intensive care domain. The original DICE knowledge base contains more than 2000 concept definitions, is acyclic, and is written in the DL $\mathcal{ALCQ}$, which extends $\mathcal{ALC}$ by so-called qualified number restrictions [56]. The TBoxes DICE1, DICE2, and DICE3 were obtained from DICE by selecting a relatively small number of concept definitions and modifying these definitions such that the obtained fragment belongs to $\mathcal{ALC}$ and the number of concept names used in the TBox is small. In addition, two of these TBoxes (DICE2 and DICE3) were modified such that they contain cyclic concept definitions.

A fourth fragment, called PA-6 in the following, was obtained from a process engineering application [57]. The original knowledge base describes reactor models and parts of reactors from a polyamide process, consists of about 60 acyclic concept definitions, and is an $\mathcal{ALE}$ knowledge base. Again, we selected a small number of concept definitions from this knowledge base to construct the TBox used in our experiments. Note that the fact that PA-6 is an $\mathcal{ALE}$-TBox does not mean that we can simply expand $\mathcal{ALE}$(PA-6)-concept descriptions into $\mathcal{ALE}$-concept descriptions. The reason is that concepts defined in PA-6 may occur negated in $\mathcal{ALE}$(PA-6)-concept descriptions.

The other two knowledge bases used in our experiments were handcrafted. One is the family TBox from Example 22, called Family in the following. The other one is a small acyclic TBox, called HC, which uses only Boolean operations. It was built such that there are relationships between conjunctions of (negated) concept names that do not follow from the subsumption relationships between the names. This was achieved by introducing concept names for disjunctions of other concept names.

Table 2 contains information on the structure of the six TBoxes used in our experiments. Most of the column names should be self-explaining. The most important number is probably the number of concept names since this number determines the number of attributes in the context: if it is n then we have $2n$ attributes. The size of the expanded TBox gives an upper-bound on the size of the concepts for which the expert must decide subsumption. The number of layers of the subsumption hierarchy counts the maximal chain in the hierarchy, not counting $\top$ or $\bot$.

In order to obtain $\mathcal{ALE}(\mathcal{T})$-concept descriptions for which to compute common subsumers, we randomly generated $\mathcal{ALE}$-concept descriptions of size at least 6 using the concept and role names of the respective background KB. The size bound was meant to ensure that the description has a significant $\mathcal{ALE}$ part.

6.2 Computing the subsumption lattice

We computed the subsumption lattices of all conjunctions of (negated) concept names for the six TBoxes introduced in the previous subsection, using three different variants of Ganter's attribute exploration algorithm:

Type 0: The usual attribute exploration algorithm that does not use any a priori knowledge.

Type 1: The attribute exploration algorithm that uses the implicational a priori knowledge (1)–(3) introduced in Section 5.

Type 2: Like Type 1, but now the non-implicational a priori knowledge (4) is used to optimize the expert, as sketched at the end of Section 5. Propositional consequences are computed using an algorithm by Ganter [34], which is linear in the size of the implicational part of the a priori knowledge, but exponential in the size of the non-implicational part.

TBox	a.k. type	number of calls: expert	number of calls: imp. hull	number of calls: pre expert	cpu time (secs): expert	cpu time (secs): imp. hull	cpu time (secs): pre expert	cpu time (secs): total
DICE1	0	1,309	4,537	-	2.13	1.22	-	23.81
10	1	1,290	3,905	-	2.18	0.72	-	23.78
names	2	1,288	3,905	1,290	1.83	0.70	1.15	21.33
DICE2	0	54,696	132,731	-	91.62	32.44	-	2,072.21
12	1	54,678	132,589	-	93.55	22.01	-	2,058.08
names	2	54,676	132,589	54,678	92.60	22.54	66.65	2,123.30
DICE3	0	91,880	246,616	-	157.66	90.83	-	4,862.17
13	1	91,860	246,437	-	154.33	57.78	-	4,795.51
names	2	91,856	246,437	91,860	154.96	56.68	183.62	5,021.54
PA-6	0	30,484	110,671	-	93.52	55.06	-	943.25
12	1	30,462	95,572	-	52.42	24.69	-	907.22
names	2	30,457	95,572	30,462	50.47	24.84	53.77	927.13
HC	0	4,794	17,816	-	8.19	33.86	-	131.34
14	1	4,776	17,629	-	7.89	19.18	-	112.99
names	2	4,755	17,629	4,776	7.79	19.21	77.35	129.81
Family	0	6,334	16,962	-	9.31	2.22	-	102.89
9	1	6,321	16,905	-	9.74	1.48	-	103.83
names	2	6,319	16,905	6,321	9.22	0.67	2.87	97.81

Table 3
Attribute exploration (part 1).

Table 3 shows the number of calls to the expert, the number of computations of the implication hull during attribute exploration, and the number of calls to the pre-expert realized using non-implicational a priori knowledge. It also shows the time spent in the respective calls, where for the expert we only measured the time spent in answering the implication question, but not the time spent to produce the counterexample (since only this part can be addressed by the pre-expert).

The numbers show that using implicational a priori knowledge leads to an improvement of the attribute exploration algorithm since it reduces the number of calls to the expert and the hull computation, and it also reduces the overall run-time. However, these gains are rather marginal. Using non-implicational a priori knowledge in the way we currently do is not advisable: there is only a very moderate reduction of the number of calls to the expert, and the additional calls to the pre-expert often take more time than is is gained this way. However, things may change if one uses a highly optimized propositional reasoner to realize the pre-expert, and when calls to the expert become more expensive for larger background TBoxes.

What Table 3 also shows is that most of the time spent in attribute exploration is *not* spent answering implication questions or computing the implication hull. The major culprit actually turns out to be the computation of the operation $B_i \mapsto B_i''$. The reason is that the number of objects in the contexts computed during attribute exploration becomes very large. For example, consider the DICE3 TBox. There, we have 91,880 calls to the expert, but only 27 implications (see Table 4 below). This means that in all but 27 cases, the expert produces a counterexample. Thus, the final context consists of 91,853 objects. In addition to the problem caused by the large number of objects when computing the operation $B_i \mapsto B_i''$, the expert also spends quite some time actually producing the counterexample, i.e., checking by subsumption tests which attributes the counterexample satisfies and which it does not. As mentioned above, this time was not measured in the column for the run-time of the expert. The high number of calls to the expert and the long run-times for the TBoxes DICE-2, DICE-3, and PA-6 are due to the fact that for these TBoxes (i) the number of concept names is relatively high, while (ii) there are not many subsumption relationships between conjunctions of (negated) concept names. For the TBox HC, the measured values are much lower, although this TBox contains more concept names. As mentioned before, HC was built to contain new subsumption relationships between conjunctions of (negated) concept names, i.e., relationships that do not directly follow from the subsumption relationships between the names. Thus, there are less counterexamples and more implications. This is the reason why computing the extended subsumption hierarchy for this TBox takes much less time compared to the TBoxes DICE-2, DICE-3, and PA-6.

Table 4 shows the sizes of the final set of implications computed by the attribute exploration algorithm. Since the third variant of the attribute exploration algorithm (type 2) only “optimizes” the expert, but does not change the the attribute exploration process, the results for it coincide with the second variant (type 1). Thus, it is not explicitly included in the table. For type 1 we distinguish between the set of all implications and the ones computed during the exploration process. For example, for PA-6 we had 24 implications as a priori knowledge and computed 7 additional implications.

The most interesting result that can be drawn from this table is that the number of implications stays rather small (in particular compared to the huge number of objects in the final context). Consequently, computing implication

	a.k.	size of base	
TBox	type	computed	total
DICE1	0	24	24
10 names	1	3	25
DICE2	0	21	21
12 names	1	3	22
DICE3	0	27	27
13 names	1	6	28
PA-6	0	31	31
12 names	1	7	31
HC	0	56	56
14 names	1	32	62
Family	0	16	16
9 names	1	3	16

Table 4
Attribute exploration (part 2).

hulls for these sets of implications will be fast (see Subsection 6.4 below).

6.3 Computing the gcs

The first set of experiments was used to find out whether using $\mathcal{ALE}$-expansion leads to a more specific common subsumer. To this purpose we ran experiments w.r.t. all six background TBoxes, computing the gcs of randomly generated $\mathcal{ALE}(\mathcal{T})$-concept descriptions once with $\mathcal{ALE}$-expansion and once without, and comparing the computed common subsumers w.r.t. subsumption. Table 5 summarizes the obtained results: the entries describe how often

- the gcs computed from the $\mathcal{ALE}$-expanded concept descriptions was equivalent to the gcs computed from the unexpanded descriptions ($\mathcal{ALE}$-expansion same);
- the gcs computed from the $\mathcal{ALE}$-expanded concept descriptions was strictly more specific than the gcs computed from the unexpanded descriptions ($\mathcal{ALE}$-expansion better);
- the gcs computed from the $\mathcal{ALE}$-expanded concept descriptions was more specific than $\top$ whereas the gcs computed from the unexpanded descriptions was equivalent to $\top$ ($\mathcal{ALE}$-expansion much better).

The results show that in the majority of the tests $\mathcal{ALE}$-expansion does not yield a more specific common subsumer. Nevertheless, the fact that in 13% of all tests $\mathcal{ALE}$-expansion results in a more specific common subsumer justifies using it. The comparison with $\top$ was done since the common subsumer $\top$ is totally useless, and thus the last column of Table 5 shows how often $\mathcal{ALE}$-expansion generated a useful result whereas the common subsumer obtained from the unexpanded descriptions was useless.

TBox name	nr. tests	$\mathcal{ALE}$-expansion same	$\mathcal{ALE}$-expansion better	$\mathcal{ALE}$-expansion much better
DICE1	52	48 (92.3 %)	4 (7.7 %)	0
DICE2	61	52 (85.2 %)	9 (14.8 %)	5 (8.2 %)
DICE3	66	61 (92.4 %)	5 (7.6 %)	2 (3.0 %)
PA-6	60	45 (75.0 %)	15 (25.0 %)	3 (5.0 %)
HC	20	20 (100.0 %)	0	0
Family	25	22 (88.0 %)	3 (12.0 %)	0

Table 5
Gcs with $\mathcal{ALE}$-expansion versus gcs without $\mathcal{ALE}$-expansion.

TBox name	Nr. tests	gcs $\sqsubset$ scs	gcs $\sqsubset$ acs	acs $\sqsubset$ gcs	gcs $\sqsubset$ scs $= \top$	gcs $\sqsubset$ acs $= \top$
DICE1	52	3.8%	30.8%	15.4 %	3.8%	1.9%
DICE2	61	8.2%	—	—	3.3%	—
DICE3	66	6.1%	—	—	1.5%	—
PA-6	60	3.3%	25.0%	0.0%	0.0%	0.0%
HC	20	20.0%	40.0%	0.0%	0.0%	10.0%
Family	25	0.0%	28.0%	0.0%	0.0%	0.0%

Table 6
Gcs versus acs and scs.

In the second set of experiments, summarized in Table 6, we used the same randomly generated $\mathcal{ALE}(\mathcal{T})$-concept descriptions as in the first set. But this time we checked how often the gcs was better than the scs and the acs (see Subsection 4.4 for the definition of these to alternative ways of obtaining common subsumers). Since the acs can also be more specific than the gcs, we also checked for strict subsumption in the other direction in this case. In addition, we again checked how often the gcs was much better than the other common subsumer in the sense that it was not $\top$ in cases where the others were. Note that the acs requires the TBox to be acyclic, and thus we could note compute it for DICE2 and DICE3.

The results show that using the gcs is clearly better than using the acs. The comparison between the gcs and the scs yields less conclusive results. Although for all but one of the test TBoxes there were cases where the gcs turned out to be more specific, there were relatively few such cases. Thus, it is not yet clear whether this difference really justifies the additional effort required for computing the gcs. The difference between the gcs and the scs might, however, become more important for larger knowledge bases with more complex relationships between the concept names occurring in them. This is also supported by the results for the handcrafted TBox HC, which was designed to contain such relationships.

6.4 Computing the supremum

Table 7 shows the time spent in supremum calls that were generated during the computation of the gcs in the experiments described in the previous subsection. We measured the run-time for three different methods of computing the supremum:

Type 0: The supremum is computed by building the implication hull w.r.t. the implication base computed without using a priori knowledge during attribute exploration.

Type 1: The supremum is computed by building the implication hull w.r.t. the implication base computed using a priori knowledge during attribute exploration.

Type 2: The supremum is computed naïvely using iterated calls of the subsumption algorithm, as sketched at the end of Subsection 4.2.

Since the implication base computed with a priori knowledge is usually a bit larger than the one without, using this base takes a bit longer, but in both cases the supremum is computed very fast. In contrast, the naïve approach is two orders of magnitude slower. Thus, it really pays to have the implication base available.

7 Related and future work

In a preliminary version of this paper [50], we have considered computing the lcs in $\mathcal{EL}$ w.r.t. a background $\mathcal{ALC}$-terminology. We have shown that the lcs w.r.t. acyclic TBoxes always exists in this setting, and that the lcs w.r.t. general TBoxes need not exist. We have also sketched a practical approach for computing good common subsumers similar to the one described above. Since $\mathcal{EL}$ does not allow for atomic negation, we only had to consider conjunctions of concepts names (no negated names). The present version of the paper (which was first published in a shorter version as [58]) improves on this by using the considerably more expressive DL $\mathcal{ALE}$ in place of $\mathcal{EL}$ (which makes the proof of Theorem 12 much harder), by extending the approach for computing the subsumption lattice of all conjunctions of concept names to conjunctions of concept names and negated concept names, by using attribute exploration

	exp.	cpu time (secs)		base
TBox	type	average	total	size
DICE1	0	0.00015	0.01	24
67 supremum	1	0.0003	0.02	25
calls	2	0.03045	2.04	–
DICE2	0	0.00016	0.02	21
121 supremum	1	0.00016	0.02	22
calls	2	0.04330	5.24	–
DICE3	0	0.00039	0.04	27
101 supremum	1	0.00049	0.05	28
calls	2	0.0403	4.07	–
PA-6	0	0.00046	0.11	31
237 supremum	1	0.00046	0.11	31
calls	2	0.04810	11.40	–
HC	0	0.00166	0.07	56
42 supremum	1	0.00238	0.10	62
calls	2	0.06	2.52	–
Family	0	0.0001	0.01	16
98 supremum	1	0.0002	0.02	16
calls	2	0.03775	3.70	–

Table 7
Computing the supremum of conjunctions of (negated) concept names.

with a priori knowledge and $\mathcal{ALE}$-expansion, and by providing first experimental results on the run-time for computing the subsumption lattice, and on the quality of the computed gcs (compared to other approaches for computing common subsumers).

It should be noted that formal concept analysis and attribute exploration have already been applied in a different context to the problem of computing the least common subsumer. In [59], the following problem is addressed: given a finite collection $\mathcal{C}$ of concept descriptions, compute the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$. Again, this extended subsumption hierarchy can be computed by defining a formal context whose concept lattice is isomorphic to the subsumption lattice we are interested in, and then applying attribute exploration (see [59] for details). In [53], it is shown that this approach and the one used in the present paper can be seen as two instances of a more abstract approach.

On the theoretical side, the main topic for future research is to try to find exact algorithms for computing the *least* common subsumer that are better than the brute-force algorithm sketched in the proof of Theorem 12. On the practical side, we will, on the one hand, try to improve the formal concept analysis part of our approach by (i) improving the implementation of the attribute exploration algorithm such that it can handle larger knowledge bases, and (ii) checking whether efficient SAT solvers (see, e.g., [60]) can be used to handle non-implicational a priori knowledge. On the other hand, we want to integrate the implementation of the scs computation into our non-standard inference system SONIC [61]. So far, this system offers the lcs and approximation inference as a plug-in of the ontology editor OilEd. The integration of the scs in SONIC will enable us to see whether this fairly inexpensive way of computing a common subsumer is already useful for practical applications, or whether the more expensive gcs or lcs is needed.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.

[2] M. R. Quillian, Word concepts: A theory and simulation of some basic capabilities, Behavioral Science 12 (1967) 410–430, republished in [62].

[3] M. Minsky, A framework for representing knowledge, in: J. Haugeland (Ed.), Mind Design, The MIT Press, 1981, a longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [62].

[4] W. A. Woods, J. G. Schmolze, The KL-ONE family, in: F. W. Lehmann (Ed.), Semantic Networks in Artificial Intelligence, Pergamon Press, 1992, pp. 133–178, published as a special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2–9.

[5] R. Möller, V. Haarslev, Description logic systems, in: [1], 2003, pp. 282–305.

[6] F. Donini, Complexity of reasoning, in: [1], 2003, pp. 96–136.

[7] I. 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), 1998, pp. 636–647.

[8] V. Haarslev, R. Möller, RACER system description, in: Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001), 2001.

[9] I. Horrocks, Implementation and optimization techniques, in: [1], 2003, pp. 306–346.

[10] I. Horrocks, The FaCT system, in: H. de Swart (Ed.), Proc. of the 2nd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'98), Vol. 1397 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 1998, pp. 307–312.

[11] V. Haarslev, R. 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.

[12] F. Baader, I. Horrocks, U. Sattler, Description logics for the semantic web, KI – Künstliche Intelligenz 2002 (4).

[13] F. Baader, I. Horrocks, U. Sattler, Description logics, in: S. Staab, R. Studer (Eds.), Handbook on Ontologies, International Handbooks in Information Systems, Springer–Verlag, Berlin, Germany, 2003, pp. 3–28.

[14] T. Berners-Lee, J. Hendler, O. Lassila, The semantic Web, Scientific American 284 (5) (2001) 34–43.

[15] I. Horrocks, P. F. Patel-Schneider, F. van Harmelen, From SHIQ and RDF to OWL: The making of a web ontology language, Journal of Web Semantics 1 (1) (2003) 7–26.

[16] I. Horrocks, U. Sattler, S. Tobies, Practical reasoning for very expressive description logics, J. of the Interest Group in Pure and Applied Logic 8 (3) (2000) 239–264.

[17] I. Horrocks, U. Sattler, Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic, in: Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001), Morgan Kaufmann, Los Altos, 2001.

[18] S. Bechhofer, I. Horrocks, C. Goble, R. Stevens, OilEd: A reason-able ontology editor for the semantic web, in: F. Baader, G. Brewka, T. Eiter (Eds.), KI-2001: Advances in Artificial Intelligence, Vol. 2174 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2001, pp. 396–408.

[19] H. Knublauch, R. W. Fergerson, N. F. Noy, M. A. Musen, The Protégé OWL plugin: An open development environment for semantic web applications, in: Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004.

[20] R. Küsters, Non-standard Inferences in Description Logics, Vol. 2100 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2001.

[21] F. Baader, R. 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), Vol. 1504 of Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 129–140.

[22] F. Baader, R. Küsters, R. Molitor, Computing least common subsumers in description logics with existential restrictions, in: Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99), 1999, pp. 96–101.

[23] W. W. Cohen, H. Hirsh, Learning the CLASSIC description logics: Theoretical and experimental results, in: J. Doyle, E. Sandewall, P. Torasso (Eds.), Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94), 1994, pp. 121–133.

[24] M. Frazier, L. Pitt, CLASSIC learning, Machine Learning 25 (1996) 151–193.

[25] R. Küsters, R. Molitor, Computing least common subsumers in $\mathcal{ALEN}$, in: Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001), 2001, pp. 219–224.

[26] R. Küsters, R. Molitor, Approximating most specific concepts in description logics with existential restrictions, in: F. Baader, G. Brewka, T. Eiter (Eds.), Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI 2001), Vol. 2174 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Vienna, Austria, 2001, pp. 33–47.

[27] R. Küsters, A. Borgida, What's in an attribute? Consequences for the least common subsumer, J. of Artificial Intelligence Research 14 (2001) 167–203.

[28] F. Baader, Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles, in: G. Gottlob, T. Walsh (Eds.), Proceedings of the 18th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 2003, pp. 319–324.

[29] S. Brandt, A.-Y. Turhan, R. Küsters, Extensions of non-standard inferences for description logics with transitive roles, in: M. Vardi, A. Voronkov (Eds.), Proceedings of the 10th International Conference on Logic for Programming and Automated Reasoning (LPAR'03), Vol. 2850 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2003, pp. 122–136.

[30] A. Rector, I. 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), AAAI Press, Stanford, CA, 1997.

[31] S. Schultz, U. Hahn, Knowledge engineering by large-scale knowledge reuse—experience from the medical domain, in: A. G. Cohn, F. Giunchiglia, B. Selman (Eds.), Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'2000), Morgan Kaufmann, 2000, pp. 601–610.

[32] B. Ganter, Finding all closed sets: A general approach, Order 8 (1991) 283–290.

[33] B. Ganter, Attribute exploration with background knowledge, Theoretical Computer Science 217 (2) (1999) 215–233.

[34] B. Ganter, R. Krauße, Pseudo models and propositional Horn inference, Tech. Rep. MATH-AL-15-1999, Institut für Algebra, Technische Universität Dresden, Dresden, Germany (1999).

[35] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer-Verlag, Berlin, 1999.

[36] F. M. Donini, B. Hollunder, M. Lenzerini, A. M. Spaccamela, D. Nardi, W. Nutt, The complexity of existential quantification in concept languages, Artificial Intelligence 2–3 (1992) 309–327.

[37] M. Schmidt-Schauß, G. Smolka, Attributive concept descriptions with complements, Artificial Intelligence 48 (1) (1991) 1–26.

[38] F. Baader, Terminological cycles in a description logic with existential restrictions, in: G. Gottlob, T. Walsh (Eds.), Proceedings of the 18th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 2003, pp. 325–330.

[39] S. Brandt, Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else?, in: R. L. de Mántaras, L. Saitta (Eds.), Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI 2004), 2004, pp. 298–302.

[40] C. Lutz, Complexity of terminological reasoning revisited, in: Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99), Vol. 1705 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 1999, pp. 181–200.

[41] K. Schild, A correspondence theory for terminological logics: Preliminary report, in: Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91), 1991, pp. 466–471.

[42] F. Baader, S. Brandt, C. Lutz, Pushing the $\mathcal{EL}$ envelope, in: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05, Morgan-Kaufmann Publishers, Edinburgh, UK, 2005.

[43] B. Nebel, Terminological reasoning is inherently intractable, Artificial Intelligence 43 (1990) 235–249.

[44] S. Brandt, R. Küsters, A.-Y. Turhan, Approximation and difference in description logics, in: D. Fensel, F. Giunchiglia, D. McGuiness, M.-A. Williams (Eds.), Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), Morgan Kaufman, San Francisco, CA, 2002, pp. 203–214.

[45] F. Baader, R. Küsters, R. Molitor, Rewriting concepts using terminologies, in: Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'2000), 2000, pp. 297–308.

[46] G. Stumme, R. Wille, Begriffliche Wissensverarbeitung – Methoden und Anwendungen, Springer-Verlag, 2000.

[47] W. F. Dowling, J. H. Gallier, Linear-time algorithms for testing the satisfiability of propositional horn formulae, Journal of Logic Programming 1 (3) (1984) 267–284.

[48] D. Maier, The Theory of Relational Databases, Computer Science Press, Potomac, Maryland, 1983.

[49] V. Duquenne, Contextual implications between attributes and some representational properties for finite lattices, in: B. Ganter, R. Wille, K. E. Wolf (Eds.), Beiträge zur Begriffsanalyse, B.I. Wissenschaftsverlag, Mannheim, 1987, pp. 213–239.

[50] F. Baader, B. Sertkaya, A.-Y. Turhan, Computing the least common subsumer w.r.t. a background terminology, in: Proceedings of the 2004 International Workshop on Description Logics (DL2004), 2004.

[51] F. 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, V. Dahl (Eds.), Knowledge Retrieval, Use and Storage for Efficiency: Proc. of the 1st Int. KRUSE Symposium, 1995, pp. 168–178.

[52] F. Baader, U. Sattler, An overview of tableau algorithms for description logics, Studia Logica 69 (2001) 5–40.

[53] F. Baader, B. Sertkaya, Applying formal concept analysis to description logics, in: P. Eklund (Ed.), Proceedings of the 2nd International Conference on Formal Concept Analysis (ICFCA 2004), Vol. 2961 of Lecture Notes in Computer Science, Springer-Verlag, Sydney, Australia, 2004, pp. 261–286.

[54] G. Stumme, Attribute exploration with background implications and exceptions, in: H.-H. Bock, W. Polasek (Eds.), Data Analysis and Information Systems: Statistical and Conceptual Approaches (GfKl'95), Vol. 7 of Studies in Classification, Data Analysis, and Knowledge Organization, 1996, pp. 457–469.

[55] R. Cornet, A. Abu-Hanna, Using description logics for managing medical terminologies, in: M. Dojat, E. Keravnou, P. Barahona (Eds.), 9th Conference on Artificial Intelligence in Medicine in Europe (AIME 2003), Lecture Notes in Computer Science, Springer-Verlag, 2003, pp. 61–70.

[56] B. Hollunder, F. Baader, Qualifying number restrictions in concept languages, Tech. Rep. RR-91-03, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern (Germany), an abridged version appeared in *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*(1991).

[57] G. Schopfer, A. Yang, L. von Wedel, W. Marquardt, CHEOPS: A tool-integration platform for chemical process modelling and simulation, International Journal on Software Tools for Technology Transfer 6 (3) (2004) 186–202.

[58] F. Baader, B. Sertkaya, A.-Y. Turhan, Computing the least common subsumer w.r.t. a background terminology, in: J. J. Alferes, J. A. Leite (Eds.), Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004), Vol. 3229 of Lecture Notes in Computer Science, Springer-Verlag, Lisbon, Portugal, 2004, pp. 400–412.

[59] F. Baader, R. Molitor, Building and structuring description logic knowledge bases using least common subsumers and concept analysis,

in: B. Ganter, G. Mineau (Eds.), Conceptual Structures: Logical, Linguistic, and Computational Issues – Proceedings of the 8th International Conference on Conceptual Structures (ICCS2000), Vol. 1867 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2000, pp. 290–303.

[60] L. Zhang, S. Malik, The quest for efficient Boolean satisfiability solvers, in: A. Voronkov (Ed.), 18th International Conference on Automated Deduction (CADE-18), Vol. 2392 of Lecture Notes in Computer Science, Springer-Verlag, Copenhagen, Denmark, 2002, pp. 295–313.

[61] A.-Y. Turhan, C. Kissig, Sonic — Non-standard inferences go OilEd, in: D. Basin, M. Rusinowitch (Eds.), Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2004), Vol. 3097 of Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 321–325.

[62] R. J. Brachman, H. J. Levesque (Eds.), Readings in Knowledge Representation, Morgan Kaufmann, Los Altos, 1985.