Computing a Minimal Representation of the Subsumption Lattice of All Conjunctions of Concepts Defined in a Terminology*

Franz Baader

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

Abstract. For a given TBox of a terminological KR system, the classification algorithm computes (a representation of) the subsumption hierarchy of all concepts introduced in the TBox. In general, this hierarchy does not contain sufficient information to derive all subsumption relationships between conjunctions of these concepts. We show how a method developed in the area of "formal concept analysis" for computing minimal implication bases can be used to determine a minimal representation of the subsumption hierarchy between conjunctions of concepts introduced in a TBox. To this purpose, the subsumption algorithm must be extended such that it yields (sufficient information about) a counterexample in cases where there is no subsumption relationship. For the concept language $\mathcal{ALC}$, this additional requirement does not change the worst-case complexity of the subsumption algorithm. One advantage of the extended hierarchy is that it is a lattice, and not just a partial ordering.

1 Introduction

In knowledge representation systems based on description logics (also called terminological KR systems or KL-ONE-like systems in the literature), one of the main reasoning tasks is *classification*. For a given terminological knowledge base (TBox), the classifier computes the *subsumption* hierarchy, i.e., all subconcept/superconcept relationships between the concepts defined in the TBox. Much of the research in description logics was directed at designing subsumption algorithms, i.e., algorithms that, given a pair of concepts C, D defined in a TBox, decide whether D subsumes C or not. In addition, the complexity of the subsumption problem has been studied for a great variety of concept description languages, with the result that it is now known that the subsumption problem is intractable for all reasonably expressive description languages [18, 21, 5]. Recent empirical investigations concerning the complexity of classification in terminological KR systems [2] have shown, however, that the complexity of a single subsumption test may not be that crucial for the overall process. In the worst case, computing the subsumption hierarchy between n concepts requires n^2 calls of the (expensive) subsumption test. However, by using a clever order in which to make the subsumption calls, and by appropriately propagating the results of each call in the already computed part of the hierarchy, most of the n^2 calls can be avoided.[2] Such techniques are employed in many of the implemented terminological KR systems [15, 16, 19, 28], and similar methods have independently been developed in the conceptual graphs community [12, 8, 13, 9].

The subsumption hierarchy provides only a limited amount of information about the interaction between defined concepts. For example, assume that we have appropriately defined concepts NoDaughter (standing for all persons having no female children), NoSon (standing for all persons having no male children), and NoSmallChild (standing for all persons having no small children). Obviously, there is no subsumption relationship between these three concepts. On the other hand, the conjunction NoDaughter$\sqcap$NoSon of NoDaughter and NoSon would be subsumed by NoSmallChild, i.e., if an individual kruse belongs to NoDaughter and NoSon (i.e., has no child), it also belongs to NoSmallChild. However, this cannot be derived from the information

* This research was supported by the EC Working Group CCL, EP6028.

[2] For the TBoxes considered in [2], between 80% and 90% of the calls could be avoided.

that kruse belongs to NoDaughter and NoSon by just looking at the subsumption hierarchy. This example demonstrates that runtime inferences concerning individuals could be facilitated by precomputing the subsumption hierarchy not only for defined concept but for all conjunctions $C_{i_1} \sqcap \ldots \sqcap C_{i_k}$ of defined concepts. Another advantages of this extended hierarchy is that—unlike the subsumption hierarchy between defined concepts, which can be an arbitrary partial ordering—the subsumption hierarchy between conjunctions of defined concepts is a lattice. In fact, the existence of greatest lower bounds (GLBs) is trivial, and it implies the existence of least upper bounds (LUBs). Having LUBs is, for example, important for applications in concept learning (see Section 5).

Obviously, the problem of computing all subsumption relationships between conjunctions of concepts defined in a TBox is a special case of the classification problem as introduced above, provided that the description language allows for conjunction (which is the case for almost all languages considered in the literature or used in implemented systems). In fact, one simply extends the given TBox $\mathcal{T}$ by introducing names for all conjunctions, and then classifies the extended TBox $\widehat{\mathcal{T}}$. However, this simple approach does not seem to be feasible in practice since the size of the extended TBox would always be exponential in the size of the original TBox, and so would be any graphical or other representation of the resulting extended hierarchy (since any concept of $\widehat{\mathcal{T}}$ would obtain an explicit representation). In the worst case, this exponential blowup can probably not be avoided, but one can still hope to do better for "typical" TBoxes. Thus, we are interested in creating a minimal representation of the subsumption hierarchy between conjunctions of concepts defined in a TBox, and we want to design algorithms that generate this representation with as few as possible applications of the subsumption algorithm to pairs of conjunctions.

To achieve this goal, we employ a method developed in the area of "formal concept analysis" [23, 27]. We shall show that an algorithm used for so-called "attribute exploration" [10, 25, 11] can be adapted to our problem. It creates a minimal representation (sometimes called Duquenne-Guigues base [7]) of the extended hierarchy, from which any subsumption relationship between conjunctions of concepts defined in $\mathcal{T}$ can be deduced in time linear in the size of this representation. The exploration algorithm generates, for a certain pair of conjunctions, a call to the subsumption algorithm. Depending on the answer, it creates a new question, or decides that the construction of the Duquenne-Guigues base is finished. It should be noted, however, that the method requires an extended subsumption algorithm which not just answers "yes" or "no," depending on whether the subsumption relationship holds or not. If the answer is "no" the attribute exploration algorithm needs a (finite) counterexample.

In the next section, we introduce a prototypical description language, called $\mathcal{ALC}$, sketch the well-known subsumption algorithm for this language [21, 1], and show that it can easily be extended to an algorithm producing finite counterexamples. In Section 3, we introduce as many of the basic notions of formal concept analysis as are necessary for our purposes. In particular, we sketch the attribute exploration algorithm. Section 4 applies this technique to our problem of computing a minimal representation of the subsumption hierarchy between conjunctions of concepts defined in a TBox. In addition, it is shown that computing the information about counterexamples that is relevant for the exploration algorithm does not increase the worst-case complexity of the subsumption algorithm. In Section 5, we mention some possible advantages of having a lattice instead of just a partial ordering.

2 The description logic $\mathcal{ALC}$

Terminological knowledge representation systems can be used to represent the taxonomic and conceptual knowledge of an application domain in a structured way. To describe this kind of knowledge, one starts with atomic concepts (unary predicates) and roles (binary predicates), and defines more complex concepts using the operations provided by the concept description language of the particular system. As a prototypical example, we introduce the description language $\mathcal{ALC}$.

Definition 1 (Syntax of $\mathcal{ALC}$). *Concept descriptions* are built from concept and role names using the concept-forming operators *negation* ($\neg C$), *disjunction* ($C \sqcup D$), *conjunction* ($C \sqcap D$), *existential restriction* ($\exists R.C$) and *value restriction* ($\forall R.C$). Here C and D are syntactic variables for concept descriptions, and R

stands for a role name. The set of concept names is assumed to contain the particular names $\top$ and $\bot$ for the *top* and the *bottom* concept.

Let A be a concept name (different from $\top$ and $\bot$), and let D be a concept description. Then $A = D$ is a *terminological axioms*. A *terminology (TBox)* is a finite set $\mathcal{T}$ of terminological axioms with the additional restriction that $\mathcal{T}$ contains no cyclic definitions and no multiple definitions.

For a given TBox $\mathcal{T}$, a concept name occurring on the left-hand side of a definition is called *defined concept*, whereas all other concept names occurring in $\mathcal{T}$ are called *primitive concepts*. $\mathcal{T}$ contains *multiple definitions* iff it contains definitions $A = D_1$ and $A = D_2$ for the same concept name A and distinct descriptions D_1, D_2. Intuitively, a *cyclic definition* is one that refers to itself. More precisely, we say that a concept name A directly uses a concept name B in a TBox $\mathcal{T}$ if B occurs in the description that defines A. Let "uses" be the transitive closure of "directly uses." Then $\mathcal{T}$ contains a cyclic definition iff it contains a defined concept A such that A uses A.

As an example, we consider the TBox of Figure 1, which defines—among others—the concepts NoDaughter, NoSon and NoSmallChild mentioned in the introduction. Here, Female and Small are primitive concepts, and all the other concept names occurring in the TBox are defined.

$$
\begin{array}{ll}
\mathsf{Male} & = \neg\mathsf{Female} \\
\mathsf{Human} & = \mathsf{Male} \sqcup \mathsf{Female} \\
\mathsf{Parent} & = \exists \mathsf{child}.\mathsf{Human} \\
\mathsf{NoDaughter} & = \forall \mathsf{child}.\mathsf{Male} \\
\mathsf{NoSon} & = \forall \mathsf{child}.\mathsf{Female} \\
\mathsf{NoSmallChild} & = \forall \mathsf{child}.\neg\mathsf{Small}
\end{array}
$$

Fig. 1. A family terminology.

Definition 2 (Semantics of $\mathcal{ALC}$). An *interpretation* $\mathcal{I}$ for $\mathcal{ALC}$ consists of a set $dom(\mathcal{I})$ and an extension function that associates with each concept name A a subset $A^{\mathcal{I}}$ of $dom(\mathcal{I})$, and with each role name R a binary relation $R^{\mathcal{I}}$ on $dom(\mathcal{I})$, i.e., a subset of $dom(\mathcal{I}) \times dom(\mathcal{I})$. The special names top and bottom must be interpreted as $\top^{\mathcal{I}} = dom(\mathcal{I})$ and $\bot^{\mathcal{I}} = \emptyset$. For $x \in dom(\mathcal{I})$, we denote the set $\{y \in dom(\mathcal{I}) \mid xR^{\mathcal{I}}y\}$ of R-fillers of x by $R^{\mathcal{I}}(x)$. The extension function can be extended to arbitrary concept descriptions as follows:

- $(\neg C)^{\mathcal{I}} = dom(\mathcal{I}) \setminus C^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, and $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$,
- $(\exists R.C)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid R^{\mathcal{I}}(x) \cap C^{\mathcal{I}} \neq \emptyset\}$,
- $(\forall R.C)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid R^{\mathcal{I}}(x) \subseteq C^{\mathcal{I}}\}$.

An interpretation $\mathcal{I}$ is a *model* of the TBox $\mathcal{T}$ iff it satisfies $A^{\mathcal{I}} = D^{\mathcal{I}}$ for all terminological axioms $A = D$ in $\mathcal{T}$.

The terminological axioms of a given TBox imply subconcept/superconcept relationships (so-called subsumption relationships) between concept descriptions.

Subsumption: Let $\mathcal{T}$ be a TBox and let C, D be concept descriptions. Then D *subsumes* C with respect to $\mathcal{T}$ (symbolically $C \sqsubseteq_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{T}$.

With respect to the TBox of Figure 1, the description $\mathsf{NoDaughter} \sqcap \mathsf{NoSon}$ is subsumed by NoSmallChild. When a terminological system reads in a terminology, it first computes all the subsumption relationships between the concept names occurring in this TBox. This process, called *classification*, results in an explicit

internal representation of the subsumption ordering, which can directly be accessed during later computations. With respect to the TBox of Figure 1, the names NoDaughter, NoSon and NoSmallChild are not in any subsumption relationship with each other.

The subsumption algorithms described in the literature are usually concerned with subsumption with respect to the empty TBox (which has all interpretations as models). To use these algorithm for answering subsumption questions "$C \sqsubseteq_{\mathcal{T}} D$?" with respect to a nonempty TBox $\mathcal{T}$, one must expand concept definitions, i.e., iteratedly replace defined concepts in C and D by their defining descriptions until all names occurring in the descriptions are primitive. If $\widehat{C}$ and $\widehat{D}$ denote the descriptions obtained this way, then $C \sqsubseteq_{\mathcal{T}} D$ iff $\widehat{C} \sqsubseteq_{\emptyset} \widehat{D}$. The expansion process terminates since TBoxes are assumed to be acyclic. In our example, the expanded description corresponding to NoDaughter $\sqcap$ NoSon is $\forall$child.$\neg$Female $\sqcap$ $\forall$child.Female, and the expanded description corresponding to NoSmallChild is $\forall$child.$\neg$Small.

The first sound and complete subsumption algorithm for $\mathcal{ALC}$ was described in [21] (see also [1] for a more succinct description). In the remainder of this section we explain the ideas underlying this algorithm using descriptions from our example.

First, we show how the algorithm determines that NoDaughter $\sqcap$ NoSon is subsumed by NoSmallChild. As described above, subsumption with respect to the TBox of Figure 1 is reduced to subsumption with respect to the empty TBox by expansion. Thus, the input of the algorithm is actually the pair of expanded descriptions $\forall$child.$\neg$Female $\sqcap$ $\forall$child.Female and $\forall$child.$\neg$Small. The algorithm attempts to show that the subsumption relationship does not hold by trying to construct a counterexample, i.e., an interpretation $\mathcal{I}$ and an individual $d_0 \in dom(\mathcal{I})$ such that $d_0 \in (\forall\mathsf{child}.\neg\mathsf{Female} \sqcap \forall\mathsf{child}.\mathsf{Female})^{\mathcal{I}}$ and $d_0 \in (\exists\mathsf{child}.\mathsf{Small})^{\mathcal{I}}$.[3] Thus, the interpretation $\mathcal{I}$ must satisfy the following set of constraints:

$$\mathcal{C}_0 := \{d_0 \in (\forall\mathsf{child}.\neg\mathsf{Female})^{\mathcal{I}}, d_0 \in (\forall\mathsf{child}.\mathsf{Female})^{\mathcal{I}}, d_0 \in (\exists\mathsf{child}.\mathsf{Small})^{\mathcal{I}}\}.$$

Now, the third constraint is satisfied by introducing a new individual, say d_1, and asserting that it is a small child of d_0. This yields the new set of constraints

$$\mathcal{C}_1 := \mathcal{C}_0 \cup \{(d_0, d_1) \in \mathsf{child}^{\mathcal{I}}, d_1 \in \mathsf{Small}^{\mathcal{I}}\}.$$

Since d_0 has a child, the two value-restrictions in $\mathcal{C}_1$ become active, which means that these restrictions are propagated to the filler d_1 of the child-role. This yields the new set of constraints

$$\mathcal{C}_2 := \mathcal{C}_1 \cup \{d_1 \in \mathsf{Female}^{\mathcal{I}}, d_1 \in (\neg\mathsf{Female})^{\mathcal{I}}\},$$

which is obviously contradictory. Thus, the attempt to show that the subsumption relationship does not hold has failed, and the algorithm concludes that the relationship holds.

Second, we show how the algorithm determines that NoDaughter is not subsumed by NoSmallChild. As described above, the descriptions are expanded, and the algorithm tries to construct a counterexample. The initial set of constraints is now

$$\mathcal{C}'_0 := \{d_0 \in (\forall\mathsf{child}.\neg\mathsf{Female})^{\mathcal{I}}, d_0 \in (\exists\mathsf{child}.\mathsf{Small})^{\mathcal{I}}\}.$$

To satisfy the second constraint, a new individual d_1 is introduced, and the set of constraints is extended to $\mathcal{C}'_1 := \mathcal{C}'_0 \cup \{(d_0, d_1) \in \mathsf{child}^{\mathcal{I}}, d_1 \in \mathsf{Small}^{\mathcal{I}}\}$. Now, there is only one value-restriction that is propagated onto d_1, and thus the new set of constraints

$$\mathcal{C}'_2 := \mathcal{C}'_1 \cup \{d_1 \in (\neg\mathsf{Female})^{\mathcal{I}}\}$$

is not contradictory. Since there are no more unsatisfied constraints, the algorithm concludes that there exists a counterexample, i.e., the subsumption relationship does not hold. In fact, a finite counterexample can easily be constructed from the final set of constraints: $dom(\mathcal{I}) = \{d_0, d_1\}$ (i.e., the set of all individual contained in the final set of constraints), and the extensions of the primitive concepts and roles are $\mathsf{child}^{\mathcal{I}} = \{(d_0, d_1)\}$

[3] Note that the description $\exists$child.Small is the *negation normal form* of $\neg(\forall$child.$\neg$Small), obtained by pushing negation into descriptions.

(all role-relationships explicitly stated), $\mathsf{Female}^{\mathcal{I}} = \emptyset$, and $\mathsf{Small}^{\mathcal{I}} = \{d_1\}$ (all element-relationships explicitly stated).

To sum up, the algorithm answers with "yes" if the subsumption relationship holds. Otherwise, it answers with "no." In this case, a finite model can easily be constructed from the final non-contradictory set of constraints. As shown in [21], the size of the final set of constraints, and thus also of the model can be exponential in the size of the expanded concept descriptions.

3 Minimal implication bases

We shall introduce only those notions and results from formal concept analysis that are necessary for our application (see, e.g., [27] for more information).

Definition 3 (Context). A *context* is a triple $\mathcal{K} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ where $\mathcal{O}$ is a (possibly infinite) set of objects, $\mathcal{P}$ is a finite set of properties,[4] and $\mathcal{S} \subseteq \mathcal{O} \times \mathcal{P}$ is a relation that connects each object o with the properties satisfied by o.

Let $\mathcal{K} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ be a context. For a set of objects $A \subseteq \mathcal{O}$, the *intent* A' of A is the set of properties 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 properties $B \subseteq \mathcal{P}$, the *extent* B' of B is the set of objects that satisfy all properties 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 $B_1' = B_1'''$).

We are now interested in implications between sets of properties that are satisfied in a given context $\mathcal{K}$.

Implication: Let $\mathcal{K}$ be a context and let B_1, B_2 be subsets of $\mathcal{P}$. The implication $B_1 \to B_2$ holds in $\mathcal{K}$ (symbolically $\mathcal{K} \models B_1 \to B_2$) iff $B_1' \subseteq B_2'$.

Because of the properties mentioned above, $B_1' \subseteq B_2'$ holds iff $B_2 \subseteq B_1''$. The set of all implications that hold in $\mathcal{K}$ is denoted by $\mathit{Imp}(\mathcal{K})$. This set can be very large, and thus one is interested in (small) generating sets for $\mathit{Imp}(\mathcal{K})$.

Definition 4. Let $\mathcal{L}$ be a set of implications, i.e., the elements of $\mathcal{L}$ are of the form $B_1 \to B_2$ for sets of properties $B_1, B_2 \subseteq \mathcal{P}$. For a subset B of $\mathcal{P}$, the *implication hull* of B with respect to $\mathcal{L}$ is denoted by $\mathcal{L}(B)$. It is the smallest subset H of $\mathcal{P}$ such that

- $B \subseteq H$, and
- $B_1 \to B_2 \in \mathcal{L}$ and $B_1 \subseteq H$ implies $B_2 \subseteq H$.

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

[4] In formal concept analysis, the notion "attribute" is used instead of "property." We prefer the second notion to avoid confusion since, in the description logics community, the name "attribute" is sometimes used for functional roles. In our application, the set of properties will be the set of concept names occurring in a TBox.

If $\mathcal{L}$ is a base for $Imp(\mathcal{K})$ then $B'' = \mathcal{L}(B)$ for all $B \subseteq \mathcal{P}$, i.e., the closure operator $B \mapsto B''$ can be computed without computing the extent B' of B.

The notions we have just defined can easily be reformulated in propositional logic. In fact, we can consider the properties as propositional variables. An implication $B_1 \to B_2$ can then be expressed by the formula $\phi_{B_1 \to B_2} := \bigwedge_{p \in B_1} p \to \bigwedge_{p' \in B_2} p'$. Let Γ be the set of formulae corresponding to the set of implications $\mathcal{L}$. Then $B_1 \to B_2 \in Cons(\mathcal{L})$ iff $\phi_{B_1 \to B_2}$ is a logical consequence of Γ. It is now easy to see that the linear decision method for satisfiability of sets of propositional Horn clauses [6] can easily be adapted to the problem of deciding whether $B_1 \to B_2 \in Cons(\mathcal{L})$. Thus, a base $\mathcal{L}$ for $Imp(\mathcal{K})$ yields a representation of $Imp(\mathcal{K})$ with the property that any question "$B_1 \to B_2 \in Imp(\mathcal{K})$?" can be answered in time linear in the size of $\mathcal{L} \cup \{B_1 \to B_2\}$.[5]

This shows that it is important to find bases of small size. A base $\mathcal{L}$ of $Imp(\mathcal{K})$ is called *minimal base* iff no base of $Imp(\mathcal{K})$ has a cardinality smaller than the cardinality of $\mathcal{L}$. Duquenne and Guigues have given a description of such a minimal base [7], and Ganter [10, 11] has shown how this base can be computed.[6]

One way of defining the Duquenne-Guigues base of a context is to modify the closure operator $B \mapsto \mathcal{L}(B)$ defined by a set $\mathcal{L}$ of implications. For a subset B of $\mathcal{P}$, the *implication pseudo-hull* of B with respect to $\mathcal{L}$ is denoted by $\mathcal{L}^*(B)$. It is the smallest subset H of $\mathcal{P}$ such that

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

Given $\mathcal{L}$, the pseudo-hull of a set $B \subseteq \mathcal{P}$ can be computed in time linear in the size of $\mathcal{L}$ and B. A subset B of $\mathcal{P}$ is called *pseudo-closed* in a context $\mathcal{K}$ iff $Imp(\mathcal{K})^*(B) = B$ and $B'' \neq B$.

Definition 5. The *Duquenne-Guigues base* of a context $\mathcal{K}$ consist of all implications $B_1 \to B_2$ where B_1 is pseudo-closed in $\mathcal{K}$ and $B_2 = B_1'' \setminus B_1$.

When trying to use this definition for computing the Duquenne-Guigues base of a 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 an 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 $\mathcal{L}$ of the base when computing the pseudo-hull. To define the lectic order, fix an an arbitrary linear order on the set of properties, say $\mathcal{P} = \{p_1, \ldots, p_n\}$. 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"[7] is asked whether the implication $B \to B'' \setminus B$ really holds in $\mathcal{K}$. If it does not hold, the expert must provide a counterexample, i.e., an object 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 name o, and must say which of the properties of $\mathcal{P}$ hold for o and which don't.

[5] Another possibility to obtain this linearity result is to use algorithms developed for deriving functional dependencies in relational databases (see [17], Section 4.6).

[6] Computation of minimal bases for sets of functional dependencies is considered in [17].

[7] In the applications mentioned in [27, 10], this "expert" is a human expert who knows the context $\mathcal{K}$; in our application, the expert will be a subsumption algorithm.

Algorithm 6 (Computation of the Duquenne-Guigues base of $\mathcal{K}$).
Initialization: One starts with the empty set of implications, i.e., $\mathcal{L}_0 := \emptyset$, 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 $\mathcal{K}_i$, $\mathcal{L}_i$, and B_i $(i \geq 0)$ are already defined. 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 $\mathcal{K}$.[8]

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

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

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

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

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

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

4 Subsumption between conjunctions of concepts

In the following, let $\mathcal{T}$ be a fixed TBox, and let $p_1, \ldots, p_n$ be the concept names occurring in $\mathcal{T}$. We are interested in representing all subsumption relationships (w.r.t. $\mathcal{T}$) between finite conjunctions of these names. To this purpose we define a context such that the implications that hold in the context are in a 1–1-relationship with these subsumption relationships.

Definition 7. The context $\mathcal{K}_\mathcal{T} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ is defined as follows:

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

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

Lemma 8. *Let B_1, B_2 be subsets of $\mathcal{P}$. The implication $B_1 \to B_2$ holds in $\mathcal{K}_\mathcal{T}$ iff $\sqcap B_1 \sqsubseteq_\mathcal{T} \sqcap B_2$.*

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

Thus, the Duquenne-Guigues base $\mathcal{L}$ of $\mathcal{K}_\mathcal{T}$ also yields a representation of all subsumption relationships of the form $\sqcap B_1 \sqsubseteq_\mathcal{T} \sqcap B_2$ for subsets B_1, B_2 of $\mathcal{P}$. As mentioned in Section 3, any question "$\sqcap B_1 \sqsubseteq_\mathcal{T} \sqcap B_2$?" can then be answered in time linear in the size of $\mathcal{L} \cup \{B_1 \to B_2\}$. If we want to apply Algorithm 6 to compute this base, we must show:

Lemma 9. *The subsumption algorithm sketched in Section 2 can function as an "expert" for the context $\mathcal{K}_\mathcal{T}$.*

[8] 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.

Proof. Algorithm 6 asks questions of the form "$B_1 \rightarrow B_2$?". By Lemma 8, these questions can be translated into subsumption questions "$\sqcap B_1 \sqsubseteq_{\mathcal{T}} \sqcap B_2$?". Let $\widehat{\sqcap B_1}, \widehat{\sqcap B_2}$ be the corresponding expanded concept descriptions. Obviously, the subsumption algorithm sketched in Section 2 can answer the question whether "$\widehat{\sqcap B_1} \sqsubseteq_\emptyset \widehat{\sqcap B_2}$?". In addition, we have seen in Section 2 that this algorithm can be modified such that it yields a counterexample if the answer is "no." This counterexample consists of a finite interpretation $\widehat{\mathcal{I}}$ and an element d of $dom(\widehat{\mathcal{I}})$ such that $d \in (\widehat{\sqcap B_1})^{\widehat{\mathcal{I}}} \setminus (\widehat{\sqcap B_2})^{\widehat{\mathcal{I}}}$. Since the extended concept descriptions contain only primitive concepts and roles of $\mathcal{T}$, we may assume that $\widehat{\mathcal{I}}$ interprets only these concepts and roles, whereas the interpretation of the defined concepts is still open. Since $\mathcal{T}$ is a TBox without cyclic definitions, there is a unique way of extending $\widehat{\mathcal{I}}$ to a model $\mathcal{I}$ of $\mathcal{T}$. In addition, if $\widehat{D}$ is the expanded version of a description D, then we have $\widehat{D}^{\widehat{\mathcal{I}}} = D^{\mathcal{I}}$. This shows that $d \in dom(\mathcal{I})$ satisfies $d \in (\sqcap B_1)^{\mathcal{I}} \setminus (\sqcap B_2)^{\mathcal{I}}$, and thus we know that $(\mathcal{I}, d) \in \mathcal{O}$ is a counterexample for the implication $B_1 \rightarrow B_2$ in $\mathcal{K}_{\mathcal{T}}$.

In order to extend the current finite subcontext of $\mathcal{K}_{\mathcal{T}}$ by this counterexample, Algorithm 6 must know which of the properties $p_1, \ldots, p_n$ are satisfied for $(\mathcal{I}, d)$, i.e., it must know for which of the concept names p_i one has $d \in p_i^{\mathcal{I}}$. Since $\mathcal{I}$ is a finite model, this is an instance of finite model checking, which for the description logic $\mathcal{ALC}$ is decidable in polynomial time.[9] □

The method of realizing the expert for $\mathcal{K}_{\mathcal{T}}$ described in the above proof contains two exponential steps that can partially be avoided. First, as shown in [18], the expanded concept descriptions can have a size that is exponential in the size of the TBox. Second, as already mentioned in Section 2, the size of the final set of constraints, and thus also of the model, can be exponential in the size of the expanded concept descriptions. It is, however, possible to design a subsumption algorithm that uses only polynomial space (but not polynomial time). The first exponential step is avoided by not completely expanding concept descriptions in the beginning. Instead, one makes one-step expansions during the algorithm by need (see [2], Section 6). To avoid obtaining (and thus also storing) an exponential set of constraints, [21] uses the fact that constraints on a given role-successor of an individual d do not influence any of the other role successors of d. Thus, different role-successors can be check separately, which means that there is no need to keep the constraints generated by checking one successor when checking the next one (see [1] for a succinct description of a (functional) algorithm realizing this idea).

Since we are interested in generating a counterexample, one might think that it is not possible to avoid storing the (exponentially large) model. However, Algorithm 6 does not really need a complete description of the counterexample $(\mathcal{I}, d)$. It is sufficient to know how to extend the current context, i.e., which properties p_j are satisfied by the counterexample. Then one can simply generate a generic name for this counterexample, and extend the relation $\mathcal{S}_i$ appropriately.

Lemma 10. *It is possible to implement a "PSPACE-expert" for the context $\mathcal{K}_{\mathcal{T}}$, i.e., the relevant information about the counterexample can be computed by an algorithm that uses space that is polynomially bounded by the size of $\mathcal{T}$.*

Proof. A detailed proof would require a more detailed description of the functional subsumption algorithm introduced in [1]. Because of the space limitations, we give only a sketch of the extended algorithm. In addition to the pair C, D of concept descriptions for which subsumption is to be checked, the extended algorithm takes a list $E_1, \ldots, E_n$ of concept descriptions.[10] If the subsumption relationship does not hold, and $(\mathcal{I}, d)$ is the counterexample that would be generated by the simple algorithm, then the extended algorithm returns a list $b_1, \ldots, b_n$ of Boolean values, where $b_i = \mathit{true}$ iff $d \in E_i^{\mathcal{I}}$.

[9] In fact, as shown in [20], $\mathcal{ALC}$ is a syntactic variant of the multi-modal logic **K**. In addition, [3] shows that model checking for an even larger logic can be done in time $O(nm)$ where n is the size of the formula and m is the size of the model.

[10] Without mentioning it explicitly, we always assume that concept descriptions are expanded by need according to the definitions in $\mathcal{T}$.

The interesting case occurs when the current set of constraints[11] is of the form

$$\{d \in L_1^{\mathcal{I}}, \ldots, d \in L_k^{\mathcal{I}},$$
$$d \in (\exists R_1.C_{1,1})^{\mathcal{I}}, \ldots, d \in (\exists R_1.C_{1,m_1})^{\mathcal{I}}, d \in (\forall R_1.C_1)^{\mathcal{I}}, \ldots,$$
$$d \in (\exists R_r.C_{r,1})^{\mathcal{I}}, \ldots, d \in (\exists R_r.C_{r,m_r})^{\mathcal{I}}, d \in (\forall R_r.C_r)^{\mathcal{I}}\},$$

where $L_1, \ldots, L_k$ are literals, i.e., primitive concepts or negations of primitive concepts, and $R_1, \ldots, R_r$ are different role names. If $L_1, \ldots, L_k$ is inconsistent, i.e., there is i, j such that $L_i = A$ and $L_j = \neg A$, then this constraint system is obviously contradictory. Otherwise, $m_1 + \cdots + m_r$ new constraint systems $\mathcal{C}_{i,j}$ are generated, which are checked separately for consistency:

$$\mathcal{C}_{i,j} := \{d_{i,j} \in C_{i,j}^{\mathcal{I}}, d_{i,j} \in C_i^{\mathcal{I}}\}.$$

The concepts descriptions $E_1, \ldots, E_n$ are (without loss of generality) Boolean combinations of literals L and descriptions of the form $\exists S.E$ and $\forall S.E$. We call these concept descriptions simple components of $E_1, \ldots, E_n$. Obviously, it is sufficient to know for each simple component F whether $d \in F^{\mathcal{I}}$ holds in the model generated by the algorithm. For a literal L this is easy to decide. For a primitive concept A we have $d \in A^{\mathcal{I}}$ iff there is i such that $L_i = A$ (by construction of the model; see Section 2). For a negated primitive concept $\neg A$ we have $d \in (\neg A)^{\mathcal{I}}$ iff there is no i such that $L_i = A$.

Now, consider a simple component $\exists S.E$. If there is no i such that $S = R_i$ then $d \notin (\exists S.E)^{\mathcal{I}}$. If $S = R_i$, then we have $d \in (\exists S.E)^{\mathcal{I}}$ iff there is $j, 1 \leq j \leq m_i$, such that $d_{i,j} \in E^{\mathcal{I}}$. In order to determine whether this is the case, the recursive calls of the algorithm with $\mathcal{C}_{i,j}$ $(j = 1, \ldots, m_i)$ must simply be equipped with an additional list of descriptions that includes E. The recursive call with $\mathcal{C}_{i,j}$ returns *true* for an element E of this additional list iff $d_{i,j} \in E^{\mathcal{I}}$.

Simple components of the form $\forall S.E$ can be treated similarly: If there is no i such that $S = R_i$ then $d \in (\forall S.E)^{\mathcal{I}}$. If $S = R_i$, then we have $d \in (\forall S.E)^{\mathcal{I}}$ iff for all $j, 1 \leq j \leq m_i$, we have $d_{i,j} \in E^{\mathcal{I}}$, i.e., E must be included in the additional list for all calls $\mathcal{C}_{i,j}$ $(j = 1, \ldots, m_i)$.

By induction on the structure of descriptions we can assume that the recursive calls yield the correct lists of Boolean values. Hence the overall algorithm can easily determine the correct Boolean values for the simple components, and thus also for the descriptions $E_1, \ldots, E_n$. □

Recall that our goal was to find a method that generates a minimal representation of the subsumption hierarchy between conjunctions of concepts defined in a TBox. In addition, the algorithm computing this representation should create as few as possible calls of the subsumption algorithm for pairs of conjunctions. Since the Duquenne-Guigues base is a minimal implication base, our method meets the first requirement. It yields a subset of all subsumption relationships between conjunctions from which all others can be generated, and there is no smaller subset with this property. In the worst-case, the size of the Duquenne-Guigues can be exponential in the number of properties (i.e., in our case the number of concept names occurring in $\mathcal{T}$). Empirical results from other applications [22, 14] of the methods described in Section 3 seem to indicate, however, that one can expect a polynomial behaviour for non-random contexts.

In general, Algorithm 6 does not create a minimal number of calls of the subsumption algorithm. In fact, it is a well-known phenomenon that sometimes counterexamples are generated that become redundant because of counterexamples introduced later on. Thus, too many calls with negative results of the subsumption algorithm may be generated. In this respect, the behaviour of the algorithm strongly depends on how the set of properties is ordered. Thus it would be interesting to develop good heuristics for choosing this order.

5 Possible applications

In addition to the fact that the extended concept hierarchy provides more information about the interaction between concepts than the usual subsumption hierarchy does, an advantage of this hierarchy is that it is a

[11] As mentioned above, the PSPACE-algorithm considers constraints for different individuals separately, i.e., each set of constraints is only concerned with one individual.

lattice and not just a partial ordering. On the one hand, this means that algorithms developed for drawing and structuring line diagrams of lattices can be employed (see, e.g., [26, 24]) to obtain good graphical representations of this hierarchy. On the other hand, applications that depend on the existence of least common subsumers (i.e., least upper bounds in the subsumption hierarchy) are facilitated. For example, as pointed out in [4], finding a least general concept that is consistent with a set of positive examples is an operation that is frequently used in learning. For this reason, the extended hierarchy computed by our method could be of interest in the following setting: Assume that we have a fixed TBox $\mathcal{T}$, and that both the class of target concepts to be learned and the examples are conjunctions of concepts introduced in $\mathcal{T}$. An example conjunction is positive for a target conjunction iff the example is subsumed by the target with respect to $\mathcal{T}$. Thus, the least general concept (from the class of target concepts) that is consistent with a set of positive examples is the least upper bound of the set of positive examples in the extended hierarchy computed by our method.

Applicability of the method described in the present paper is, of course, not restricted to the language $\mathcal{ALC}$. In fact, it can be employed for any description formalism that (1) allows for conjunction of descriptions, and (2) has a subsumption algorithm that can function as an "expert" for Algorithm 6 (i.e., yields sufficient information about a counterexample if the answer to the subsumption question is "no.")

References

1. F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithms. In M. Richter and H. Boley, editors, *International Workshop on Processing Declarative Knowledge*, volume 567 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1991.
2. F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological systems. *J. Applied Intelligence*, 4:109–132, 1994.
3. R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In *Proceedings of the 3rd International Workshop on Computer Aided Verification, CAV'91*, pages 48–58, volume 575 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.
4. W.W. Cohen and H. Hirsch. Learning the CLASSIC description logic: Theoretical and experimental results. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 4th International Conference*, pages 121–133, Bonn, 1994. Morgan Kaufmann.
5. F.M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J.A. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 2nd International Conference*, pages 151–162, Cambridge, MA, 1991. Morgan Kaufmann.
6. W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming* 3:267–284, 1984.
7. V. Duquenne. Contextual implications between attributes and some representational properties for finite lattices. In B. Ganter, R. Wille, K.E. Wolf, editors, *Beiträge zur Begriffsanalyse*, pages 213–239, B.I. Wissenschaftsverlag, Mannheim, 1987.
8. G. Ellis. Compiled hierarchical retrieval. In *6th Annual Conceptual Graphs Workshop*, 1991.
9. G. Ellis and R.A. Levinson. The birth of PEIRCE: A conceptual graphs workbench. In *Proceedings of the Seventh Annual Conceptual Graphs Workshop*, Las Cruces, New Mexico, July 1992.
10. B. Ganter. Algorithmen zur Formalen Begriffsanalyse. In B. Ganter, R. Wille, K.E. Wolf, editors, *Beiträge zur Begriffsanalyse*, pages 241–254, B.I. Wissenschaftsverlag, Mannheim, 1987.
11. B. Ganter. Finding all closed sets: A general approach. *Order*, 8:283–290, 1991.
12. R.A. Levinson. A self-organizing pattern retrieval system for graphs. In *Proceedings of the 4th National Conference of the American Association for Artificial Intelligence*, pages 203–206, Austin, TX, 1984.
13. R.A. Levinson. Pattern associativity and the retrieval of semantic networks. *Journal of Computers & Mathematics with Applications*, 23(6–9):573–600, 1992.
14. C. Lindig. Inkrementelle, rückgekoppelte Suche in Software-Bibliotheken. Informatik-Bericht 94-07 (Technical Report), Technical University of Braunschweig, 1994.
15. T. Lipkis. A KL-ONE classifier. In J.G. Schmolze and R.J. Brachman, editors, *Proceedings of the 1981 KL-ONE Workshop*, pages 128–145, Cambridge, MA, 1982. The proceedings have been published as BBN Report No. 4842 and Fairchild Technical Report No. 618.

16. R. MacGregor. A deductive pattern matcher. In *Proceedings of the 7th National Conference of the American Association for Artificial Intelligence*, pages 403–408, Saint Paul, MI, Aug. 1988.
17. D. Maier. *The Theory of Relational Databases*, Computer Science Press, Rockville, 1983.
18. B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.
19. C. Peltason, A. Schmiedel, C. Kindermann, and J. Quantz. The BACK system revisited. KIT Report 75, Department of Computer Science, Technische Universität Berlin, Berlin, Germany, Sept. 1989.
20. K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sydney, Australia, 1991.
21. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.
22. G. Snelting. Reengineering of configurations based on mathematical concept analysis. Informatik-Bericht 95-02 (Technical Report), Technical University of Braunschweig, 1995.
23. R. Wille. Restructuring lattice theory. In I. Rival, editor, *Ordered Sets*, pages 445–470, Reidel, Dodrecht, Boston, 1982.
24. R. Wille. Tensorial decomposition of concept lattices. *Order*, 2:81–95, 1985.
25. R. Wille. Knowledge acquisition by methods of formal concept analysis. In E. Diday, editor, *Data Analysis, Learning Symbolic and Numeric Knowledge*, pages 365–380, Nova Science Publ., New York, Budapest, 1989.
26. R. Wille. Lattices in data analysis: How to draw them with a computer. In I. Rival, editor, *Algorithms and Order*, pages 33–58, Kluwer, Dodrecht, Boston, 1989.
27. R. Wille. Concept lattices and conceptual knowledge systems. In F. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 493–516, Pergamon Press, Oxford, New York, 1992.
28. W.A. Woods. Understanding subsumption and taxonomy: A framework for progress. In J.F. Sowa, editor, *Principles of Semantic Networks*, pages 45–94. Morgan Kaufmann, San Mateo, CA, 1991.

This article was processed using the LaTeX macro package with LLNCS style