Computing the hierarchy of conjunctions of concept names and their negations in a Description Logic knowledge base using Formal Concept Analysis [⋆]

Barış Sertkaya

Theoretical Computer Science, TU Dresden, Germany
sertkaya@tcs.inf.tu-dresden.de

Abstract. In a series of previous work, we have presented how attribute exploration can be used in the bottom-up construction of DL knowledge bases to compute a concept lattice that is isomorphic to the subsumption hierarchy of all conjunctions of concept names occurring in a knowledge base, and the negations of these concept names. This work is a continuation in the line of the previous work, that makes a step towards more efficient computation of the mentioned hierarchy. Its specific accomplishment is reducing the number of questions asked to the expert and the number of objects produced during the computation of this hierarchy, thus speeding up the computation. Despite its simple nature, the approach speeds up the computation of this hierarchy drastically.

1 Introduction

Formal Concept Analysis (FCA) [17] is a field of applied mathematics based on a lattice-theoretic formalization of the notions of concept and conceptual hierarchy. It thereby facilitates mathematical reasoning for conceptual data analysis and knowledge processing. On the other hand, Description Logics (DL) [4] are a class of logic-based knowledge representation formalisms that are used to represent the knowledge of an application domain in a structured and formally well-understood way. Although both aim to represent knowledge of an application domain and make reasoning using this knowledge, they follow different methodologies.

In FCA, one starts with a *formal context*, which in its simplest form is a way of specifying which attributes are satisfied by which objects. A *formal concept* is a pair consisting of an object set called the *extent* and an attribute set called the *intent* of the concept. The objects in the extent of a formal context have precisely the attributes in the intent, and the attributes in the intent are satisfied by precisely the objects in the extent. Once all formal concepts of a formal context are obtained, one orders them w.r.t. the inclusion of their extents (equivalently, inverse inclusion of their intents) and uses the resulting complete

[⋆] This work has been supported by Deutsche Forschungsgemeinschaft (DFG) under the grant GRK 334/3

lattice, called the *concept lattice*, for querying the knowledge. In DL, the knowledge of an application domain is captured by first defining relevant concepts of the application domain using unary predicates called *atomic concepts* (corresponding to attributes in FCA language), binary predicates called *atomic roles*, and logical constructors of the particular DL language in use, called *concept constructors*. Then in a second step, these defined concepts are used to describe properties of objects and the roles are used to describe relations between objects. In DLs, there is an obvious distinction between the intensional and the extensional part of knowledge. In FCA, the intensional knowledge is obtained from the extensional part of the knowledge. Furthermore, DLs usually provide a rich language to define concepts, whereas in FCA one can only form conjunctions of attributes. Nevertheless, there have been several efforts [22–24, 27] to use ideas and techniques from one field in the other field.

In this work, we talk about using an FCA tool, *attribute exploration* [14, 17], in the DL domain. Traditionally, DL knowledge bases are built in a top-down manner. In order to support the knowledge engineer in construction and maintenance of knowledge bases, and at the same time to allow him to re-use concepts defined in an existing background knowledge base, a new *bottom-up* approach [5, 6] was introduced. The approach needs to use subsumption relationships between conjunctions of concept names occurring in the background knowledge base, and negations of these concept names. In a series of previous papers [9–12], we have presented how attribute exploration can be used in the bottom-up construction of DL knowledge bases to compute a concept lattice that is isomorphic to the subsumption hierarchy mentioned above. This work is a continuation in the line of the previous work, which makes a step towards more efficient computation of the hierarchy. Its specific accomplishment is reducing the number of questions asked to the expert and the number of objects produced during the computation of this hierarchy, thus speeding up the computation. The main idea is making use of the special context considered here: the attributes are dichotomic. We will take this fact into account while producing counterexamples during attribute exploration. Despite its simple nature, the idea speeds up the computation of this hierarchy drastically. We support our argument in favour of this approach with both theoretical and experimental results.

2 Description Logics

In DL, the knowledge of an application domain is captured by first defining relevant concepts of this domain. For defining concepts, one starts with a set N_C of concept names (unary predicates), a set N_R of role names (binary predicates), and builds complex *concept descriptions* out of them by using the concept constructors provided by the particular DL language being used. As an example, suppose we want to create a small knowledge base on the types of guitars:

$$\mathsf{Guitar} \equiv \mathsf{MusicalInstrument} \sqcap \exists \mathsf{has\text{-}part.String}$$

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

Name of constructor	Syntax	Semantics	$\mathcal{ALC}$	$\mathcal{ALE}$
top-concept	$\top$	$\Delta^{\mathcal{I}}$	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
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
concept definition	$A \equiv C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$	x	x

$$\begin{aligned}\mathsf{AcousticGuitar} &\equiv \mathsf{Guitar} \sqcap \exists \mathsf{has\text{-}part.Resonator}\\ \mathsf{ElectricGuitar} &\equiv \mathsf{Guitar} \sqcap \neg\exists \mathsf{has\text{-}part.Resonator} \sqcap \exists \mathsf{has\text{-}part.Pickup}\end{aligned}$$

In the first line, we define a guitar as a musical instrument, which has a part that is a string. Using this newly defined concept, in the second line we define an acoustic guitar as a guitar, which has a part that is a resonator. Finally in the third line, we define an electric guitar as a guitar which does not have a resonator part, but has a part that is a pickup.

DL languages are distinguished based on the concept constructors they allow for. In this paper, we consider the language $\mathcal{ALC}$ and its sub-language $\mathcal{ALE}$, whose constructors are shown in Table 1. In the table, r stands for a role name, A stands for a concept name, and C, D stand for arbitrary concept descriptions. As seen in the last row of the table, assigning a name to a complex concept description is called a *concept definition*. The concept names occurring on the left-hand side of a concept definition are called *defined concepts* and the others are called *primitive concepts*. Again, as we see in the table, $\mathcal{ALE}$ allows for negation of concept names only, whereas $\mathcal{ALC}$ allows for negation of arbitrary concept descriptions as well as concept names. We call a finite set of concept definitions a *TBox* iff it is acyclic, that is, no concept definition refers directly or indirectly to the name it defines, and unambiguous which means that each name has at most one definition.

The semantics of concept expressions is given in terms of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the *domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function* of $\mathcal{I}$. The domain $\Delta^{\mathcal{I}}$ is a non-empty set and the interpretation function $\cdot^{\mathcal{I}}$ is a function which 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 semantics of arbitrary concept descriptions is defined inductively as seen in the third column of the table. An interpretation $\mathcal{I}$ of a TBox $\mathcal{T}$ is a *model* of $\mathcal{T}$, iff it satisfies all concept definitions in $\mathcal{T}$, i.e. $A^{\mathcal{I}} = C^{\mathcal{I}}$ holds for all $A \equiv C$ in $\mathcal{T}$.

Once we get a description of an application domain, we can make inferences using this knowledge. One of the most important traditional inference services provided by DL systems is *subsumption*, which is computing the sub-concept/superconcept relationships between concept descriptions. We say that the concept description C_2 *subsumes* the concept description C_1 w.r.t. the TBox $\mathcal{T}$ (written $C_1 \sqsubseteq_{\mathcal{T}} C_2$) iff $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$. The problem of checking subsumption is extensively investigated in the literature [25, 1, 7, 8]. Besides subsumption, in this work we are interested in a non-standard inference [20] used in bottom-up construction of DL knowledge bases.

In bottom-up approach, instead of directly defining a new concept, the knowledge engineer introduces several typical examples as objects, which are then automatically generalized into a concept description by the system. The task of computing such a concept description can be split into two subtasks: computing the most specific concepts of the given objects, and then computing the least common subsumer of these concepts. The *most specific concept (msc)* of an object o is the most specific concept description C expressible in the given DL language that has o as an instance. 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 subsumes $C_1, \ldots, C_n$. The problem of computing the lcs and the msc has already been investigated in the literature [3, 5, 6].

For a different setting, the non-standard inference *least common subsumer w.r.t. a background terminology*, was recently introduced in [11, 12]. Consider a scenario, where the knowledge engineer wants to construct a knowledge base in the bottom-up fashion, but he wants to re-use concepts from an existing *background terminology* $\mathcal{T}$, defined in an expressive DL $\mathcal{L}_2$ where computing lcs is computationally very costly, or even meaningless due to the disjunction operator in the language. In order to overcome this problem, when defining new concepts he uses only a sub-language $\mathcal{L}_1$ of $\mathcal{L}_2$, for which computing the lcs makes sense, and is feasible. However, his $\mathcal{L}_1$-concept descriptions are allowed to contain concept names defined in the background terminology $\mathcal{T}$, which is written in the more expressive DL $\mathcal{L}_2$. When computing subsumption between such newly defined concepts, this is done w.r.t. $\mathcal{T}$, using a subsumption algorithm for the expressive DL $\mathcal{L}_2$. When computing the lcs of such concepts, we employ an extended version of the lcs algorithm for $\mathcal{L}_1$, which can take into account the subsumption relationships between conjunctions of concepts defined in $\mathcal{T}$. Let $\mathcal{L}_1$, $\mathcal{L}_2$ be DLs such that $\mathcal{L}_1$ is a sub-language of $\mathcal{L}_2$, that is $\mathcal{L}_1$ allows only a subset of the constructors that $\mathcal{L}_2$ allows. 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 1 (lcs w.r.t. a background terminology). *Given an $\mathcal{L}_2$-TBox $\mathcal{T}$ and a collection $C_1, \ldots, C_n$ of $\mathcal{L}_1(\mathcal{T})$-concept descriptions, the* least common subsumer *(lcs) of* $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$ *is the most specific $\mathcal{L}_1(\mathcal{T})$-concept description C 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)

Let us demonstrate the setting by a trivial example. Assume we have a toy $\mathcal{ALC}$-TBox defining the concept of bass guitar as the union of the concepts electric bass guitar and acoustic bass guitar:

$$\mathcal{T} := \{\mathsf{BassGuitar} \equiv \mathsf{AcousticBassGuitar} \sqcup \mathsf{ElectricBassGuitar}\}$$

If we want to compute the lcs of $\mathcal{ALE}(\mathcal{T})$-concepts $\mathsf{AcousticBassGuitar}$ and $\mathsf{ElectricBassGuitar}$ without taking into account the TBox $\mathcal{T}$, we get the top-concept $\top$ as the result, which does not help much as the lcs, since it subsumes everything. On the other hand, if we were allowed to use the name $\mathsf{BassGuitar}$ defined in $\mathcal{T}$, the lcs w.r.t. $\mathcal{T}$ would obviously be $\mathsf{BassGuitar}$, which is more specific than $\top$.

Depending on the languages $\mathcal{L}_1$ and $\mathcal{L}_2$, the least common subsumer of $\mathcal{L}_1(\mathcal{T})$-concept descriptions w.r.t an $\mathcal{L}_2$-TBox $\mathcal{T}$ may exist or not. In [10], the DLs $\mathcal{EL}$ and $\mathcal{ALC}$ were considered as $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively, and it was shown that the lcs of $\mathcal{EL}(\mathcal{T})$-concept descriptions w.r.t an $\mathcal{ALC}$-TBox $\mathcal{T}$ always exists and can effectively be computed. In [11], the result was extended to the case where $\mathcal{L}_1$ is $\mathcal{ALE}$, and it was shown that the lcs of $\mathcal{ALE}(\mathcal{T})$-concept descriptions w.r.t. an $\mathcal{ALC}$-TBox $\mathcal{T}$ always exists, and can effectively be computed. Unfortunately, these theoretical results are not useful in practice, due to the high computational complexity of the brute-force lcs algorithm following from these results. In fact, in the bottom-up construction of knowledge bases, it is not really necessary to take the *least* common subsumer. It can even result in overfitting. Instead of taking the least one, a common subsumer of the given concept descriptions, which is not too general, can also be used. Such "*good*" common subsumers (gcs) w.r.t. a background terminology were introduced in [10, 11] as a practical alternative to the lcs w.r.t. a background terminology. In order to compute a good common subsumer of $\mathcal{ALE}(\mathcal{T})$-concepts, the gcs algorithm (see [11] for the details of the algorithm) needs to incorporate knowledge from the subsumption hierarchy of the conjunctions of concept names occurring in the $\mathcal{ALC}$-TBox $\mathcal{T}$, and the negations of these concept names. Let us demonstrate the use of this hierarchy by extending our small example on the types of guitars.

$$\begin{aligned}
\mathsf{BassInstrument} &\equiv \mathsf{MusicalInstrument} \sqcap \forall \mathsf{produces\text{-}tone}.\mathsf{Bass} \\
\mathsf{AcousticBassGuitar} &\equiv \mathsf{AcousticGuitar} \sqcap \forall \mathsf{produces\text{-}tone}.\mathsf{Bass} \\
\mathsf{ElectricBassGuitar} &\equiv \mathsf{ElectricGuitar} \sqcap \forall \mathsf{produces\text{-}tone}.\mathsf{Bass} \\
\mathsf{BassGuitar} &\equiv \mathsf{AcousticBassGuitar} \sqcup \mathsf{ElectricBassGuitar}
\end{aligned}$$

Assume that we have the above concept definitions in our $\mathcal{ALC}$-TBox $\mathcal{T}$, in addition to the ones given in the previous example, and we want to compute the lcs of the $\mathcal{ALE}(\mathcal{T})$-concept descriptions C, D defined as follows:

$$\begin{aligned}
\mathsf{C} &\equiv \exists \mathsf{plays}.(\mathsf{ElectricGuitar} \sqcap \mathsf{BassInstrument}) \\
\mathsf{D} &\equiv \exists \mathsf{plays}.(\mathsf{AcousticGuitar} \sqcap \mathsf{BassInstrument})
\end{aligned}$$

If we ignore the TBox $\mathcal{T}$ by treating all concept names as primitive, and just compute the lcs of C,D, we obtain the concept description $\exists \mathsf{plays.BassInstrument}$ which is a common subsumer of C and D, but a too general one. However, if we take into account the TBox $\mathcal{T}$ and consider the fact that both descriptions $\mathsf{ElectricGuitar} \sqcap \mathsf{BassInstrument}$ and $\mathsf{AcousticGuitar} \sqcap \mathsf{BassInstrument}$ are subsumed by the concept $\mathsf{BassGuitar}$, then we obtain the common subsumer

$$\exists \mathsf{plays.BassGuitar}$$

which is more specific than $\exists \mathsf{plays.BassInstrument}$. In this particular example, the result coincides with the least common subsumer, but as stated before it need not be the case in general.

As the example shows, knowing about the subsumption hierarchy of all conjunctions of concept names occurring in a given TBox, and the negations of these concept names helps us to compute more specific common subsumers of some given $\mathcal{ALE}(\mathcal{T})$ concept descriptions. This motivates our interest in computing this hierarchy. Since the background terminology is fixed in our application, it makes sense to precompute it. The point is to compute this hierarchy, whose size is exponential in the number of the concept names, efficiently without having to check subsumption relationships between each pair of conjunctions of concept names and negated concept names. This is the point where FCA comes into play.

3 Hierarchy of conjunctions of DL concept names and their negations

As mentioned in Section 2, the gcs algorithm needs to make use of the hierarchy of all conjunctions of the concept names occurring in the background terminology and the negations of these concept names. Since the background terminology is fixed in our application scenario, it makes sense to precompute this information. Obviously, a naive approach that tests the subsumption relationships between each pair of conjunctions is too expensive for TBoxes of realistic sizes. Instead, we can compute it more efficiently by defining a formal context whose concept lattice is going to be isomorphic to this hierarchy and exploring this context with the help of an expert, which in this case is a subsumption algorithm.

For the hierarchy of conjunctions of concept names only (i.e. without negated concept names), such a formal context, called the *semantic context*, was first defined in [2]. The semantic context needs an expert that is able to come up with counter-models in case an implication question does not hold. The problem with this approach is that the standard subsumption algorithms and their optimized implementations like FaCT [19] and RACER [18] do not produce such counter-models in case a subsumption does not hold. To overcome this problem, a new formal context, called the *syntactic context*, for which a standard subsumption algorithm can act as an expert, was defined in [9] for the same purpose. Let $\{A_1, \ldots, A_n\}$ be the set of concept names occurring in a TBox $\mathcal{T}$ defined in the language $\mathcal{L}$.

Definition 2. *Then the syntactic context is defined as follows.*
$\mathbb{K}_{\mathcal{T}} = (G, M, I)$ *such that:*

$$\begin{aligned} G &:= \{C \mid C \text{ is a concept description in } \mathcal{L}\}, \\ M &:= \{A_1, \ldots, A_n\}, \\ I &:= \{(C, A_i) \mid C \sqsubseteq_{\mathcal{T}} A_i,\ 1 \leq i \leq n\} \end{aligned}$$

The following three results were shown in [9]:

Lemma 1. *Let $B_1, B_2 \subseteq M$ and let $\bigsqcap B_j$ $(j = 1, 2)$ denote the conjunction $A_{r_1} \sqcap \ldots \sqcap A_{r_m}$ for a non-empty $B_j = \{A_{r_1}, \ldots, A_{r_m}\}$. Then the implication $B_1 \to B_2$ holds in $\mathbb{K}_{\mathcal{T}}$ iff $\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} \bigsqcap B_2$.*

Lemma 2. *Any decision procedure for subsumption w.r.t. TBoxes in $\mathcal{L}$ functions as an expert for the context $\mathbb{K}_{\mathcal{T}}$.*

Theorem 1. *The concept lattice of the context $\mathbb{K}_{\mathcal{T}}$ is isomorphic to the subsumption hierarchy of all conjunctions of the concept names occurring in $\mathcal{T}$.*

Lemma 1 states that implication questions in $\mathbb{K}_{\mathcal{T}}$ correspond to subsumption tests w.r.t. $\mathcal{T}$, and Lemma 2 states that a standard subsumption algorithm can be used to answer these questions. Finally, Theorem 1 states that context $\mathbb{K}_{\mathcal{T}}$ serves our purpose of computing the compact representation of a hierarchy that is isomorphic to the one we are interested in. During the exploration of $\mathbb{K}_{\mathcal{T}}$, whenever an implication question $B_1 \to B_2$ does not hold, we insert the concept description $\sqcap B_1$ into $\mathbb{K}_{\mathcal{T}}$ as the counterexample to this question, and for each $A_i \in M$ such that $\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} A_i$, we put a cross in its row in $\mathbb{K}_{\mathcal{T}}$.

For computing gcs, the hierarchy of conjunctions of concept names only does not suffice. The gcs algorithm needs to make use of the hierarchy of conjunctions of concept names and the negations of these concept names. To compute this extended hierarchy, the approach was improved in [11] by simply extending the TBox $\mathcal{T}$ with new names for the negations of the concept names occurring in it. To be more precise, if $\{A_1, \ldots, A_n\}$ is the set of concept names occurring in $\mathcal{T}$, then we introduce the concept names $\overline{A}_1, \ldots, \overline{A}_n$, and extend $\mathcal{T}$ to $\widehat{\mathcal{T}}$ by adding the definitions $\overline{A}_1 \equiv \neg A_1, \ldots, \overline{A}_n \equiv \neg A_n$. As a result of this, the formal context $\mathbb{K}_{\widehat{\mathcal{T}}}$ following from this extended TBox has the attribute set $\{A_1, \ldots, A_n, \overline{A}_1, \ldots, \overline{A}_n\}$. Obviously, the theoretical results given above also hold for $\mathbb{K}_{\widehat{\mathcal{T}}}$.

We can produce counterexamples for $\mathbb{K}_{\widehat{\mathcal{T}}}$, by using the same method used for $\mathbb{K}_{\mathcal{T}}$, which is briefly described above. However, some of the examples produced by this method are reducible. Our experiments showed that this results in huge contexts that are difficult to process. We know that reducing a context does not change the structure of the concept lattice. One idea could be to use the method described for $\mathbb{K}_{\mathcal{T}}$, as counterexample generator for $\mathbb{K}_{\widehat{\mathcal{T}}}$, and during the exploration of $\mathbb{K}_{\widehat{\mathcal{T}}}$ reduce it from time to time. However, since context reduction is a costly operation this approach is not feasible. Instead, by using the dichotomy of the attributes we can improve the method for this particular type of context to produce only irreducible objects. The objects that have either A_i or $\overline{A}_i$ for each $1 \leq i \leq n$ (i.e., objects that have exactly n attributes) are irreducible. Because their intents can not be written as the intersection of other object intents. The following theorem shows that whenever an object with $m < n$ attributes is a counterexample to an implication question, then there is an object with $m + 1$ attributes, which is also a counterexample to this implication.

Theorem 2. *Assume that the implication question $B_1 \rightarrow B_2$ is asked to the expert and he replies that it does not hold in $\mathbb{K}_{\widehat{\mathcal{T}}}$. If $\bigsqcap C$ is a counterexample to this implication with $\bigsqcap C \not\sqsubseteq_{\widehat{\mathcal{T}}} A_i$ and $\bigsqcap C \not\sqsubseteq_{\widehat{\mathcal{T}}} \overline{A}_i$ for some $1 \leq i \leq n$, then one of $\bigsqcap C \sqcap A_i$ or $\bigsqcap C \sqcap \overline{A}_i$ is also a counterexample to this implication.*

Proof. According to the argument, $\bigsqcap C \sqcap A_i \not\sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$ or $\bigsqcap C \sqcap \neg A_i \not\sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$. Assume this were not true. Then, $\bigsqcap C \sqcap A_i \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$ and $\bigsqcap C \sqcap \neg A_i \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$. Then the following also holds $(\bigsqcap C \sqcap A_i) \sqcup (\bigsqcap C \sqcap \neg A_i) \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$, which means $\bigsqcap C \sqcap (A_i \sqcup \neg A_i) \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$. Equivalently, $\bigsqcap C \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$. This is a contradiction. Because, we assumed that $B_1 \rightarrow B_2$ does not hold in $\mathbb{K}_{\widehat{\mathcal{T}}}$. By Lemma 1 this means that, $\bigsqcap B_1 \not\sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$. We further assumed that $\bigsqcap C$ is a counterexample to this implication, that is $\bigsqcap C \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_1$ and $\bigsqcap C \not\sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$. Thus, one of the objects in the argument is a counterexample to this implication. □

By successive applications of the argument, we can produce counterexamples that have exactly n attributes, i.e., irreducible counterexamples. Algorithm 1 describes the idea formally.

Algorithm 1 Counterexample generation for $\mathbb{K}_{\widehat{\mathcal{T}}}$

Assume that during the exploration of $\mathbb{K}_{\widehat{\mathcal{T}}}$, for $B_1, B_2 \subseteq M$, the question $B_1 \rightarrow B_2$? (i.e. $\bigsqcap B_1 \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$? by Lemma 1) is asked to the expert and the expert replies that it does not hold. We create the counterexample as follows:

```
– Initialization: Start with $C_0 := B_1$
– Iteration: Assume $C_j$ is already computed. Let $I$ be the set of indices s.t. for all
  $i \in I$, $A_i \notin C_j$ and $\overline{A}_i \notin C_j$.
    • For each $i \in I$, check if $\bigsqcap C_j \sqcap A_i \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$ holds
        ∗ if yes, then $C_{j+1} = C_j \cup \{\overline{A}_i\}$
        ∗ if no, then $C_{j+1} = C_j \cup \{A_i\}$
– return $\sqcap C_j$
```

Proposition 1. *Given an implication $B_1 \rightarrow B_2$ that is rejected by the expert, Algorithm 1 terminates and upon termination it returns a counterexample to this implication, which has either A_i or $\overline{A}_i$ for each $1 \leq i \leq n$.*

Proof. The algorithm iterates over a finite set of indices I, so it terminates. A valid counterexample to $B_1 \rightarrow B_2$ should contain all the attributes in B_1, and should not contain at least one of the attributes in B_2. The object $\bigsqcap B_1$ is a counterexample to this implication. It has all the attributes in B_1, and does not have all the attributes in B_2. So, the algorithm starts generating the counterexample with $C_0 := B_1$. In the iteration, for an $i \in I$ assume that $\bigsqcap C_j \sqcap A_i \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$ does not hold. Then $\bigsqcap C_j \sqcap A_i$ is a counterexample. So, we add A_i to the intent of $\bigsqcap C_j$. If it holds, then by Theorem 2, $\bigsqcap C_j \sqcap \overline{A}_i$ is a counterexample and we add $\overline{A}_i$ to the intent of $\bigsqcap C_j$. Once we iterate over all $i \in I$ in this way, the resultant $\bigsqcap C_j$ has either A_i or $\overline{A}_i$ for each $1 \leq i \leq n$ and it is a counterexample to $B_1 \rightarrow B_2$. $\Box$

The objects generated with the above algorithm are irreducible. Thus the resultant context $\mathbb{K}_{\widehat{\mathcal{T}}}$ obtained from the exploration, which uses this algorithm for counterexample generation, is row-reduced. Using a background terminology which contains n concept names, the upper bound on the size of the resulting formal context obtained by using the new method is 2^n, whereas it was 2^{2n-1} using the previous method. Moreover, generating these objects does not cost more in the number of subsumption tests, compared to the previous method. Assuming that the size of B_1 is k, and the size of M is $2n$, we make exactly $n-k$ subsumption tests to generate a counterexample, which is the same number of subsumption tests we make in the naive version.

4 Experimental results

In order to see the performance gain obtained by the new counterexample generation method, we have performed a series of experiments on several small background TBoxes. Three of these TBoxes are fragments of the medical knowledge base DICE [13], which defines concepts from the intensive care domain. These three TBoxes, called DICE1, DICE2, DICE3 in the tables, are obtained from the original DICE knowledge base by selecting a small number of concept definitions and modifying them such that the fragment belongs to $\mathcal{ALC}$ and such that the number of the concept names occurring in it is small enough to be able to deal with during attribute exploration. The fragment DICE1 contains 10, DICE2 contains 12, and DICE3 contains 13 concept names. One of the other three TBoxes, called PA-6 in the tables, is obtained from a process engineering application [26] which describes reactor models and parts of reactors from a polyamid process. From this knowledge base, we selected a small fragment containing 12 concept names. The other two TBoxes are handcrafted TBoxes. The one called Family is a toy family TBox with 9 concept names, and the other one called HC contains 14 concept names. It should be noted that the formal contexts derived from these examples contain twice as many attributes as the concept names in the examples, due to dichotomizing the attributes.

We computed the subsumption lattice of all conjunctions of concept names occurring in the above example TBoxes, and negations of these concept names using three different variants of Ganter's attribute exploration algorithm. In Tables 2 and 3 below, the field *bg. k. type* (standing for background knowledge type) has values:

– *Type 0*: The usual attribute exploration algorithm that does not use any background knowledge
– *Type 1*: The attribute exploration algorithm that uses the following implicational background knowledge [15]:
  - If $A_i \sqsubseteq_{\widehat{\mathcal{T}}} A_j$ holds, then we know on the FCA side that in the context $\mathbb{K}_{\widehat{\mathcal{T}}}$ all objects satisfying the attribute A_i also satisfy the attribute A_j, i.e., the implication $\{A_i\} \rightarrow \{A_j\}$ holds in $\mathbb{K}_{\widehat{\mathcal{T}}}$.
  - Since $A_i \sqsubseteq_{\widehat{\mathcal{T}}} A_j$ implies $\neg A_j \sqsubseteq_{\widehat{\mathcal{T}}} \neg A_i$, 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 $\mathbb{K}_{\widehat{\mathcal{T}}}$.
  - We know that no object can simultaneously satisfy A_i and $\overline{A}_i$, and thus the implication $\{A_i, \overline{A}_i\} \rightarrow \bot^{\mathbb{K}_{\widehat{\mathcal{T}}}}$ holds, where $\bot^{\mathbb{K}_{\widehat{\mathcal{T}}}}$ stands for the set of all attributes of $\mathbb{K}_{\widehat{\mathcal{T}}}$.

– *Type 2*: In addition to the implicational background knowledge in Type 1, it uses the following non-implicational background knowledge as pre-expert to optimize the expert. Propositional consequences are computed using the algorithm in [16], which is linear in the size of the implicational part of the background knowledge, but exponential in the size of the non-implicational part.
  - Every object satisfies either A_i or $\overline{A}_i$, i.e. $\top \rightarrow A_i \vee \overline{A}_i$ In contrast to the other forms of background knowledge mentioned until now, this knowledge cannot be encoded in an implication.

The experiments were performed on a computer with one Pentium 4 processor at 2.40 GHz and 2GB of memory, under the Linux operating system. The implementation was made in the LISP programming language using the version 19a of the CMU Common Lisp interpreter. We used the version 1.7.23 of the Description Logic System Racer as the expert for attribute exploration. For implicational closure calculation, we implemented the linear time implicational closure algorithm *linclosure* in Section 4.6 of [21].

Tables 2 and 3 below show the number of calls to the expert, implicational closure calculation, the pre-expert, and the respective CPU times. The numbers show that when the improved version of the counterexample generation method is used, number of calls to the expert and the respective CPU times decrease by drastic amounts up to 90%. This drastic decrease is due to the sizes of the formal contexts obtained by using the improved version, which are much smaller compared to the ones obtained by using the naive method. Our experiments using the naive method showed that, when the context sizes get large, most of the time is spent computing the double-prime ($''$) operator. Using implicational background knowledge speeds up the computation of the hierarchy by reducing the number of expert calls, although the speed up is rather moderate. However, using non-implicational background knowledge in some cases even slows down the process due to high complexity of reasoning with non-implicational knowledge. The results also show that the gain obtained by the improved method is independent of using background knowledge. The percentage of the speed up obtained is the same whether we use background knowledge or not. Table 4 gives

TBox	bg. k. type	number of calls			cpu time (secs)			
		expert	imp. clos.	pre expert	expert	imp. clos.	pre expert	total
DICE1 10 names	0	96	3,124	-	0.15	0.37	-	1.14
	1	75	2,999	-	0.18	0.90	-	1.89
	2	73	2999	75	0.05	0.93	0.15	2.14
DICE2 12 names	0	853	163,375	-	1.52	26.62	-	59.030
	1	835	163,245	-	1.27	36.3	-	68.17
	2	833	163,245	835	1.18	37.49	2.56	72.28
DICE3 13 names	0	1,115	327,526	-	1.78	81.61	-	147.86
	1	1,094	327,363	-	1.73	108.4	-	172.77
	2	1090	327,363	1,094	2.06	107.99	4.16	178.94
PA-6 12 names	0	559	77,767	-	0.97	16.04	-	31.87
	1	535	77,580	-	0.73	26.88	-	41.91
	2	530	77,580	535	0.75	26.88	1.68	44.28
HC 14 names	0	184	21,356	-	0.30	19.25	-	22.43
	1	160	21,164	-	0.23	35.59	-	38.50
	2	140	21,164	160	0.22	35.67	3.40	42.68
Family 9 names	0	240	16,925	-	0.30	1.49	-	4.84
	1	227	16,868	-	0.29	1.62	-	4.80
	2	225	16,868	227	0.4	1.54	0.26	5.28

Table 2. Experimental results using the improved version

TBox	bg. k. type	number of calls			cpu time (secs)			
		expert	imp. clos.	pre expert	expert	imp. clos.	pre expert	total
DICE1 10 names	0	1,309	4,537	-	2.13	1.22	-	23.81
	1	1,290	3,905	-	2.18	0.72	-	23.78
	2	1,288	3,905	1,290	1.83	0.70	1.15	21.33
DICE2 12 names	0	54,696	132,731	-	91.62	32.44	-	2072.21
	1	54,678	132,589	-	93.55	22.01	-	2058.08
	2	54,676	132,589	54,678	92.60	22.54	66.65	2123.30
DICE3 13 names	0	91,880	246,616	-	157.66	90.83	-	4862.17
	1	91,860	246,437	-	154.33	57.78	-	4795.51
	2	91,856	246,437	91,860	154.96	56.68	183.62	5021.54
PA-6 12 names	0	30,484	110,671	-	93.52	55.06	-	943.25
	1	30,462	95,572	-	52.42	24.69	-	907.22
	2	30,457	95,572	30,462	50.47	24.84	53.77	927.13
HC 14 names	0	4,794	17,816	-	8.19	33.86	-	131.34
	1	4,776	17,629	-	7.89	19.18	-	112.99
	2	4,755	17,629	4,776	7.79	19.21	77.35	129.81
Family 9 names	0	6,334	16,962	-	9.31	2.22	-	102.89
	1	6,321	16,905	-	9.74	1.48	-	103.83
	2	6,319	16,905	6,321	9.22	0.67	2.87	97.81

Table 3. Experimental results using the naive version

number of objects	DICE1 10 names	DICE2 12 names	DICE3 13 names	PA-6 12 names	HC 14 names	Family 9 names
improved version	72	832	1088	528	128	224
naive version	1285	54675	91853	30453	4738	6318

Table 4. Comparison of context sizes

the sizes of the formal contexts obtained at the end of attribute exploration using the improved counterexample generation method and the naive method. As seen in the table, the improved method enables up to 90% decrease in the size of the formal contexts obtained.

5 Conclusion

We presented how counterexample generation method could be improved in our application where we compute the subsumption hierarchy of all conjunctions of concept names occurring in a Description Logic TBox, and the negations of these concept names, by using attribute exploration method of Formal Concept Analysis. The hierarchy obtained at the end of the exploration is used to support knowledge engineer during the bottom-up construction of knowledge bases. The improved method is obtained by a minor modification of the algorithm we have used in our previous works for the same purpose. The main idea of the improvement is taking into account the fact that the attributes of formal context used, are dichotomic. We have given both theoretical and experimental results to support our argument. The experimental results show that the improvement speeds up the calculation of this hierarchy quite considerably.

As future work, we are going to investigate if the pseudo-intents of the formal context used have regularities which makes it possible to compute them without having to go over all intents, and thus without suffering from the exponential nature of the attribute exploration algorithm.

References

1. F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*. Morgan Kaufmann, 1991.
2. 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, and V. Dahl, editors, *Knowledge Retrieval, Use and Storage for Efficiency: Proceedings of the 1st International KRUSE Symposium*, pages 168–178, 1995.
3. F. Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In *Proceedings of the*

18th International Joint Conference on Artificial Intelligence (IJCAI 2003), pages 319–324, 2003.

4. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
5. F. Baader and R. Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In *Proceedings of the 22nd German Annual Conference on Artificial Intelligence (KI 1998)*, volume 1504 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1998.
6. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pages 96–101, 1999.
7. F. Baader and U. Sattler. Expressive number restrictions in description logics. *Journal of Logic and Computation*, 9(3):319–350, 1999.
8. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
9. F. Baader and B. Sertkaya. Applying formal concept analysis to description logics. In P. Eklund, editor, *Proceedings of the 2nd International Conference on Formal Concept Analysis (ICFCA 2004)*, volume 2961 of *Lecture Notes in Computer Science*, pages 261–286, Sydney, Australia, 2004. Springer-Verlag.
10. F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In V. Haarslev and R. Möller, editors, *Proceedings of the 2004 International Workshop on Description Logics (DL2004)*, volume 104 of *CEUR Workshop Proceedings*, Whistler, Canada, 2004. CEUR-WS.org.
11. F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In J. J. Alferes and J. A. Leite, editors, *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *Lecture Notes in Computer Science*, pages 400–412, Lisbon, Portugal, 2004. Springer-Verlag.
12. F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *Journal of Applied Logic*, 2006. To appear.
13. R. Cornet and A. Abu-Hanna. Using description logics for managing medical terminologies. In M. Dojat, E. T. Keravnou, and P. Barahona, editors, *Proceedings of the 9th Conference on Artificial Intelligence in Medicine in Europe (AIME 2003)*, volume 2780 of *Lecture Notes in Computer Science*, pages 61–70. Springer-Verlag, 2003.
14. B. Ganter. Two basic algorithms in concept analysis. Technical Report Preprint-Nr. 831, Technische Hochschule Darmstadt, Darmstadt, Germany, 1984.
15. B. Ganter. Attribute exploration with background knowledge. *Theoretical Computer Science*, 217(2):215–233, 1999.
16. B. Ganter and R. Krauße. Pseudo models and propositional Horn inference. Technical Report MATH-AL-15-1999, Institut für Algebra, Technische Universität Dresden, Dresden, Germany, 1999.
17. B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations.* Springer-Verlag, Berlin, Germany, 1999.
18. V. Haarslev and R. Möller. RACER system description. In *Proceedings International Joint Conference on Automated Reasoning (IJCAR 2001)*, 2001.

19. I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proceedings of the 6th International Conference on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.
20. R. Küsters. *Non-Standard Inferences in Description Logics*, volume 2100 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001. Ph.D. thesis.
21. D. Maier. *The Theory of Relational Databases.* Computer Science Press, Maryland, 1983.
22. S. Prediger. Terminologische Merkmalslogik in der Formalen Begriffsanalyse. In *Begriffliche Wissensverarbeitung – Methoden und Anwendungen*, Heidelberg, Germany, 2000. Springer-Verlag.
23. S. Prediger and G. Stumme. Theory-driven logical scaling: Conceptual information systems meet description logics. In E. Franconi and M. Kifer, editors, *Proceedings of the 6th International Workshop on Knowledge Representation meets Databases (KRDB'99)*, 1999.
24. S. Rudolf. Exploring relational structures via FLE. In K. E. Wolff, H. D. Pfeiffer, and H. S. Delugach, editors, *Proceedings of the 12th International Conference on Conceptual Structures, (ICCS 2004)*, volume 3127 of *Lecture Notes in Computer Science*, pages 196–212. Springer-Verlag, 2004.
25. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
26. G. Schopfer, A. Yang, L. von Wedel, and W. Marquardt. Cheops: A tool-integration platform for chemical process modelling and simulation. *International Journal on Software Tools for Technology Transfer*, 6(3):186–202, 2004.
27. G. Stumme and R. Wille. *Begriffliche Wissensverarbeitung – Methoden und Anwendungen.* Springer-Verlag, Heidelberg, Germany, 2000.