Is Tractable Reasoning in Extensions of the Description Logic $\mathcal{EL}$ Useful in Practice?

Franz Baader, Carsten Lutz, Boontawee Suntisrivaraporn

Theoretical Computer Science, TU Dresden, Germany
{baader,lutz,meng}@tcs.inf.tu-dresden.de

Abstract. Extensions of the description logic $\mathcal{EL}$ have recently been proposed as lightweight ontology languages. The most important feature of these extensions is that, despite including powerful expressive means such as general concept inclusion axioms, reasoning can be carried out in polynomial time. In this paper, we consider one of these extensions, $\mathcal{EL}^+$, and introduce a refinement of the known polynomial-time classification algorithm for this logic, which was implemented in our CEL reasoner. We describe the results of several experiments with CEL on large ontologies from practice, which show that even a relatively straightforward implementation of the described algorithm outperforms highly optimized, state-of-the-art tableau reasoners for expressive description logics.

1 Introduction and Motivation

The quest for tractable (i.e., polynomial-time decidable) description logics (DLs), which started in the 1980s after the first intractability results for DLs were shown [5, 18], was until recently restricted to DLs extending the basic language $\mathcal{FL}_0$, which allows for conjunction ($\sqcap$) and value restrictions ($\forall r.C$). The main reason was that, when clarifying the logical status of property arcs in semantic networks and slots in frames, the decision was taken that arcs/slots should be read as value restrictions rather than existential restrictions ($\exists r.C$).

Unfortunately, as soon as terminologies (also called TBoxes or DL ontologies) were taken into consideration, tractability turned out to be unattainable in $\mathcal{FL}_0$: even classifying the simplest form of TBoxes that admit only acyclic concept definitions was shown to be coNP-hard [19]. If the most general form of TBoxes is admitted, which consists of general concept inclusion axioms (GCIs) supported by all modern DL systems, then classification in $\mathcal{FL}_0$ even becomes ExpTime-complete [2].

For these reasons, and also because of the need for expressive DLs in applications, from the mid 1990s on, the DL community has mainly given up on the quest of finding tractable DLs. Instead, it investigated more and more expressive DLs, for which reasoning is worst-case intractable. The goal was then to find practical reasoning procedures, i.e., algorithms that are easy to implement and optimize, and which—though worst-case exponential or even worse—behave well in practice (see, e.g., [17]). This line of research has resulted in the availability of highly optimized DL systems for expressive DLs based on tableau algorithms [14, 10], and successful applications: most notably the recommendation by the W3C of the DL-based language OWL [16] as the ontology language for the Semantic Web.

Recently, the choice of value restrictions as a sine qua non of DLs has been reconsidered. On the one hand, it was shown that the DL $\mathcal{EL}$, which allows for conjunction and existential restrictions, has better algorithmic properties than $\mathcal{FL}_0$. Classification of both acyclic and cyclic $\mathcal{EL}$ TBoxes is tractable [1], and this remains so even if general TBoxes with GCIs are admitted [6]. On the other hand, there are applications where value restrictions are not needed, and where the expressive power of $\mathcal{EL}$ or small extensions thereof appear to be sufficient. In fact, SNOMED, the Systematized Nomenclature of Medicine, employs $\mathcal{EL}$ with an acyclic TBox [23]. Large parts of the Galen medical knowledge base can also be expressed in $\mathcal{EL}$ with GCIs and transitive roles [21]. Finally, the Gene Ontology [7] can be seen as an acyclic $\mathcal{EL}$ TBox with one transitive role.

The tractability results for $\mathcal{EL}$ together with the bio-medical applications mentioned above have motivated our research on extensions of $\mathcal{EL}$: the leitmotif for this research was to extend $\mathcal{EL}$ as far as possible by adding standard DL constructors available in ontology languages like OWL, while still retaining polynomial-time reasoning in the presence of GCIs. This has resulted in the tractable DL $\mathcal{EL}^{++}$ [2], which includes transitive roles, so-called right-identities [23] on roles, nominals (and thus ABoxes), and disjointness constraints on concepts. The purpose of the research presented in the present paper was to evaluate whether or not the polynomial-time algorithms for reasoning in $\mathcal{EL}$ and its extensions are suitable as a basis for implementing a DL reasoning system that can handle large bio-medical ontologies, and whether such a reasoner outperforms existing high-optimized DL reasoners for expressive DLs.

At first sight, one might think that a polynomial-time algorithm is always better suited for implementation than worst-case exponential-time algorithms such as the ones underlying modern DL reasoners. However, due to the plethora of sophisticated optimization techniques that have been developed for tableau algorithms over the last decade [15], it is far from obvious whether a straightforward implementation of the polynomial-time algorithm can compete with highly-optimized implementations of tableau algorithms. A case in point is our experience with implementing the polynomial-time classification algorithms for cyclic $\mathcal{EL}$ TBoxes introduced in [1]: direct implementations of both the algorithm for subsumption w.r.t. descriptive semantics (based on a reduction to satisfiability of propositional Horn formulae [9]) and the algorithm for subsumption w.r.t. greatest fixpoint semantics (based on computing the greatest simulation on a graph [11]) did not lead to satisfactory results on the Gene Ontology [24].

In this paper, we consider a restriction of the polynomial-time classification algorithm for $\mathcal{EL}^{++}$ [2] to the fragment $\mathcal{EL}^{+}$ of $\mathcal{EL}^{++}$. This fragment differs from $\mathcal{EL}^{++}$ in that nominals and the bottom concept are disallowed. The reason for considering this fragment was that none of the bio-medical ontologies mentioned above use nominals or the bottom concept. We describe a refined version of this algorithm that is tailored toward implementation. The purpose of this refinement is to remove an obvious obstacle for efficient implementation of the algorithm as given in [2]: the uninformed, brute-force search for applicable completion rules. With (almost) no further optimizations, we have implemented the refined algorithm in our CEL (Classifier for EL) reasoner. We have performed several experiments to compare the performance of CEL with the performance of state-of-the-art DL systems based on tableau algorithms. It turns out that CEL can compete with modern DL systems and often outperforms them. We view these results as a serious encouragement for further research into *optimized* implementations of DL reasoners based on polynomial-time algorithms for the $\mathcal{EL}$ family of DLs.

2 The Description Logic $\mathcal{EL}^+$

In DLs, *concept descriptions* are inductively defined with the help of a set of *constructors*, starting with a set NC of *concept names* and a set NR of *role names*. $\mathcal{EL}^+$ concept descriptions are formed using the constructors shown in the upper part of Table 1. An $\mathcal{EL}^+$ *ontology* is a finite set of *general concept inclusions (GCIs)* and *role inclusions (RIs)*, whose syntax is shown in the lower part of Table 1.

The semantics of $\mathcal{EL}^+$ is defined in terms of *interpretations* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where the domain $\Delta^\mathcal{I}$ is a non-empty set of individuals, and the interpretation function $\cdot^\mathcal{I}$ maps each concept name $A \in \mathsf{NC}$ to a subset $A^\mathcal{I}$ of $\Delta^\mathcal{I}$ and each role name $r \in \mathsf{NR}$ to a binary relation $r^\mathcal{I}$ on $\Delta^\mathcal{I}$. The extension of $\cdot^\mathcal{I}$ to arbitrary concept descriptions is inductively defined as shown in the semantics column of Table 1. An interpretation $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ if, for each inclusion in $\mathcal{O}$, the conditions given in the semantics column of Table 1 are satisfied.

Name	Syntax	Semantics
top	$\top$	$\Delta^\mathcal{I}$
conjunction	$C \sqcap D$	$C^\mathcal{I} \cap D^\mathcal{I}$
existential restriction	$\exists r.C$	$\{x \in \Delta^\mathcal{I} \mid \exists y \in \Delta^\mathcal{I} : (x, y) \in r^\mathcal{I} \wedge y \in C^\mathcal{I}\}$
general concept inclusion	$C \sqsubseteq D$	$C^\mathcal{I} \subseteq D^\mathcal{I}$
role inclusion	$r_1 \circ \cdots \circ r_n \sqsubseteq s$	$r_1^\mathcal{I} \circ \cdots \circ r_n^\mathcal{I} \subseteq s^\mathcal{I}$

Table 1. Syntax and semantics of $\mathcal{EL}^+$.

One main use of GCIs in $\mathcal{EL}^+$ is to give definitions of concept names in terms of complex concept descriptions. Therefore, we introduce *concept definitions* $A \equiv C$, with A a concept name, as an abbreviation for the two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$. Intuitively, C describes the *necessary and sufficient* conditions for being an instance of A. GCIs of the form $A \sqsubseteq C$, with A a concept

$$
\begin{aligned}
\mathsf{Pericardium} &\sqsubseteq \mathsf{Tissue} \sqcap \exists \mathsf{contained\text{-}in}.\mathsf{Heart} \\
\mathsf{Pericarditis} &\sqsubseteq \mathsf{Inflammation} \sqcap \exists \mathsf{has\text{-}location}.\mathsf{Pericardium} \\
\mathsf{Inflammation} &\sqsubseteq \mathsf{Disease} \sqcap \exists \mathsf{acts\text{-}on}.\mathsf{Tissue} \\
\mathsf{Heartdisease} &\doteq \mathsf{Disease} \sqcap \exists \mathsf{has\text{-}location}.\mathsf{Heart} \\
\mathsf{Heartdisease} &\sqsubseteq \exists \mathsf{has\text{-}state}.\mathsf{NeedsTreatment} \\
\mathsf{has\text{-}location} \circ \mathsf{contained\text{-}in} &\sqsubseteq \mathsf{has\text{-}location}
\end{aligned}
$$

Fig. 1. An example $\mathcal{EL}^+$ ontology.

name, are called *primitive concept definitions*.[1] They give only necessary (but no sufficient) conditions for being an instance of A. In DL, a finite set of GCIs is commonly called a *general TBox*, and a finite set of (possibly primitive) concept definitions with unique left-hand sides is called a *TBox*. We call a TBox *primitive* if it contains only primitive concept definitions and *acyclic* if there are no concept names $A_0, \ldots, A_{n-1}$ such that $A_{(i+1) \bmod n}$ occurs on the right hand of the (possibly primitive) concept definition of A_i, for all $i < n$.

It is worthwhile to note that the role inclusions available in $\mathcal{EL}^+$ generalize a number of standard expressive means: role inclusions of the form $r \sqsubseteq s$ are commonly called *role hierarchies*; *transitivity* of a role r can be expressed by writing $r \circ r \sqsubseteq r$; finally, RIs can express *right-identity rules* $r \circ s \sqsubseteq r$, which play an important role in medical ontologies [23]. An example ontology formulated in $\mathcal{EL}^+$ can be found in Figure 1, where all uppercase words are concept names, and all lowercase words are role names.

The basic inference problem for DL concept descriptions is *concept subsumption*: a concept C is subsumed by a concept D w.r.t. an ontology $\mathcal{O}$ (written $C \sqsubseteq_{\mathcal{O}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in every model $\mathcal{I}$ of $\mathcal{O}$. The basic inference problem for DL ontologies is *classification*: compute the subsumption hierarchy of all concept names occurring in the ontology $\mathcal{O}$. In our example ontology, it is not hard to see that Pericarditis is classified as Heartdisease (i.e., Pericarditis $\sqsubseteq_{\mathcal{O}}$ Heartdisease), and thus needs treatment.

3 Classifying an $\mathcal{EL}^+$ Ontology

A polynomial-time algorithm for classification in $\mathcal{EL}$ with GCIs and role hierarchies has been proposed in [6], and this algorithm was extended to the more powerful DL $\mathcal{EL}^{++}$ in [2]. We introduce the restriction of the algorithm from [2] to $\mathcal{EL}^+$, and then propose a refined version for implementation purposes.

Both in tableau-based DL systems and in earlier DL systems based on structural subsumption algorithms, the subsumption hierarchy is computed by performing multiple subsumption tests. In addition to optimizing the single subsumption tests, such systems can also be optimized by trying to minimize the

[1] despite not actually defining anything.

number of subsumption tests needed to compute the whole hierarchy [3]. In contrast, the classification algorithm in [2] simultaneously computes the subsumption relationships between *all* pairs of concept names in the input ontology.

To classify an ontology, the algorithm first transforms it into *normal form*, which requires that all GCIs and RIs are of one of the forms shown in the left part of Figure 2. By introducing new concept and role names and applying a number of straightforward rewriting rules, any $\mathcal{EL}^+$ ontology $\mathcal{O}$ can be transformed into a normalized one such that subsumption between the concept names occurring in $\mathcal{O}$ is preserved. The normalization can be carried out in linear time, yielding an ontology whose size is linear in the size of the original one [2, 24].

Normal Form	Completion Rules
$A \sqsubseteq B$	**CR1** If $A \in S(X)$, $A \sqsubseteq B \in \mathcal{O}$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$
$A_1 \sqcap A_2 \sqsubseteq B$	**CR2** If $A_1, A_2 \in S(X)$, $A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{O}$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$
$A \sqsubseteq \exists r.B$	**CR3** If $A \in S(X)$, $A \sqsubseteq \exists r.B \in \mathcal{O}$, and $(X, B) \notin R(r)$ then $R(r) := R(r) \cup \{(X, B)\}$
$\exists r.A \sqsubseteq B$	**CR4** If $(X, Y) \in R(r)$, $A \in S(Y)$, $\exists r.A \sqsubseteq B \in \mathcal{O}$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$
$r \sqsubseteq s$	**CR5** If $(X, Y) \in R(r)$, $r \sqsubseteq s \in \mathcal{O}$, and $(X, Y) \notin R(s)$ then $R(s) := R(s) \cup \{(X, Y)\}$
$r \circ s \sqsubseteq t$	**CR6** If $(X, Y) \in R(r)$, $(Y, Z) \in R(s)$, $r \circ s \sqsubseteq t \in \mathcal{O}$, and $(X, Z) \notin R(t)$ then $R(t) := R(t) \cup \{(X, Z)\}$

Fig. 2. Normal Form and Completion Rules

For the rest of this section, we assume without loss of generality that the input ontology $\mathcal{O}$ is in normal form. Let $\mathsf{NC}_{\mathcal{O}}$ be the set of all concept names occurring in $\mathcal{O}$ including $\top$, and $\mathsf{NR}_{\mathcal{O}}$ be the set of all role names occurring in $\mathcal{O}$. The algorithm computes

- a mapping S assigning to each element of $\mathsf{NC}_{\mathcal{O}}$ a subset of $\mathsf{NC}_{\mathcal{O}}$, and
- a mapping R assigning to each element of $\mathsf{NR}_{\mathcal{O}}$ a binary relation on $\mathsf{NC}_{\mathcal{O}}$.

The intuition is that these mappings make implicit subsumption relationships explicit in the sense that $B \in S(A)$ implies $A \sqsubseteq_{\mathcal{O}} B$, and $(A, B) \in R(r)$ implies $A \sqsubseteq_{\mathcal{O}} \exists r.B$. The mappings are initialized by setting $S(A) := \{A, \top\}$ for each $A \in \mathsf{NC}_{\mathcal{O}}$ and $R(r) := \emptyset$ for each $r \in \mathsf{NR}_{\mathcal{O}}$. Then the sets $S(A)$ and $R(r)$ are extended by applying the completion rules shown in the right part of Figure 2 until no more rule applies. The algorithm has been proved sound and complete in [2], i.e., after termination we have $B \in S(A)$ iff $A \sqsubseteq_{\mathcal{O}} B$, for all concept names A, B occurring in $\mathcal{O}$. It has also been proved that the algorithm always terminates in time polynomial in the size of the input ontology.

One of the main problems to be solved when implementing the described algorithm is to develop a good approach for finding the next rule to be applied. If this is realized by a naïve brute-force search for applicable completion rules, then one cannot expect an acceptable runtime behavior on large inputs. As a solution to this problem, we propose a refined version of the algorithm, which is inspired by the linear-time algorithm for satisfiability of propositional Horn formulas proposed in [9]. This version uses a set of queues, one for each concept name appearing in the input ontology, to guide the application of completion rules. Intuitively, the queues list modifications to the data structure (i.e. to the sets $S(A)$ and $R(r)$) that still have to be carried out. The possible entries of the queues are of the form

$$B, \quad B \to B', \qquad \text{and} \qquad \exists r.B$$

with B and B' concept names, and r a role name. The entry $B \in \mathsf{queue}(A)$ means that the concept name B has to be added to $S(A)$. Similarly, $B \to B' \in \mathsf{queue}(A)$ means that B' has to be added to $S(A)$ if $S(A)$ already contains B, and $\exists r.B \in \mathsf{queue}(A)$ means that (A, B) has to be added to $R(r)$. The fact that such an addition triggers other rules will be taken into account by appropriately extending the queues when the addition is performed.

To facilitate describing the manipulation of the queues, we view the (normalized) input ontology $\mathcal{O}$ as a mapping $\widehat{\mathcal{O}}$ from concepts to sets of queue entries as follows: for each concept name A (including the case $A = \top$), $\widehat{\mathcal{O}}(A)$ is the minimal set of queue entries such that

- if $A \sqsubseteq B \in \mathcal{O}$, then $B \in \widehat{\mathcal{O}}(A)$;
- if $A \sqcap A' \sqsubseteq B \in \mathcal{O}$ or $A' \sqcap A \sqsubseteq B \in \mathcal{O}$, then $A' \to B \in \widehat{\mathcal{O}}(A)$;
- if $A \sqsubseteq \exists r.B \in \mathcal{O}$, then $\exists r.B \in \widehat{\mathcal{O}}(A)$.

Likewise, for each concept $\exists r.A$, $\widehat{\mathcal{O}}(\exists r.A)$ is the minimal set of queue entries such that, if $\exists r.A \sqsubseteq B \in \mathcal{O}$, then $B \in \widehat{\mathcal{O}}(\exists r.A)$.

Now, we can describe how the queues are used: since the sets $S(A)$ are initialized with $\{A, \top\}$, we initialize $\mathsf{queue}(A)$ with $\widehat{\mathcal{O}}(A) \cup \widehat{\mathcal{O}}(\top)$, i.e., we add to the queues the *immediate* consequences of being an instance of A and $\top$. Then, we repeatedly fetch (and thereby remove) entries from the non-empty queues and process them using the procedure process displayed in Figure 3. To be more precise, $\mathsf{process}(A, X)$ is called when we are currently treating the concept name A, and X is the next element on $\mathsf{queue}(A)$. Observe that the second if-clause implements **CR1** and (part of) **CR4**, the first if-clause is a pre-processing step addressing the **CR2** rule and delegating the real work to the second if-clause, and the third if-clause implements **CR3**, (the other part of) **CR4**, as well as **CR5** and **CR6**. The procedure $\mathsf{process\text{-}new\text{-}edge}(A, r, B)$ handles the effects of adding a new pair (A, B) to $R(r)$. The notation $\sqsubseteq^*_{\mathcal{O}}$ used in its top-most **for**-loop stands for the reflexive-transitive closure of the role hierarchy statements.

```
procedure process(A, X)
begin
    if X = B → B′ then
        if B ∈ S(A) then
            continue with X := B′;
        else return; {do nothing}
    if X is a concept name and X ∉ S(A) then
        S(A) := S(A) ∪ {X};
        queue(A) := queue(A) ∪ Ô(X);
        for all concept names B and role names r with (B, A) ∈ R(r) do
            queue(B) = queue(B) ∪ Ô(∃r.X);
    if X is an existential restriction ∃r.B and (A, B) ∉ R(r) then
        process-new-edge(A, r, B)
end;
procedure process-new-edge(A, r, B)
begin
    for all role names s with r ⊑*_O s do
        R(s) := R(s) ∪ {(A, B)};
        queue(A) := queue(A) ∪ ⋃_{B′ | B′ ∈ S(B)} Ô(∃s.B′);
        for all concept names A′ and role names t, u with
          t ∘ s ⊑ u ∈ O and (A′, A) ∈ R(t) and (A′, B) ∉ R(u) do
          process-new-edge(A′, u, B);
        for all concept names B′ and role names t, u with
          s ∘ t ⊑ u ∈ O and (B, B′) ∈ R(t) and (A, B′) ∉ R(u) do
          process-new-edge(A, u, B′);
end;
```

Fig. 3. Processing the queue

Queue processing is continued until all queues are empty. Observe that the refined algorithm need not perform *any* search to check which completion rules are applicable. It can be proved that the refined algorithm is still sound and complete, and that it terminates in polynomial time.

4 Implementation and Evaluation

Modern DL reasoners are usually based on tableau-based subsumption algorithms [4]. Although such algorithms are exponential in the worst case, the development of a whole plethora of sophisticated optimization techniques has led to a quite good runtime behavior in practice. In this section we will show that, nevertheless, even a relatively naïve implementation of the refined algorithm described above can compete with, and even outperform, modern tableau-based DL systems.

We have implemented the refined algorithm described in the previous section in the CEL reasoner. CEL is written in Common LISP and accepts input based on a small extension of the KRSS syntax [20]. For details about the use of the system, refer to the CEL manual.[2] To test whether CEL can compete with modern tableau-based reasoners, we have conducted a number of experiments based on three important bio-medical ontologies: the Gene Ontology (Go) [7], the Galen medical knowledge base (Galen) [21], and the Systematized Nomenclature of Medicine (Snomed) [8].

Go. The Gene Ontology provides a controlled vocabulary to describe gene and gene product attributes in any organism. It currently consists of 16,803 concepts and a single, transitive role "part-of". The original distribution of Go used a frame-like formalism without formal semantics. For example, the concept Polarisome is described as follows:

```
[Term]
id: GO:0000133
name: polarisome
namespace: cellular_component
def: "Protein complex playing a role in determining cell polarity"
is_a: GO:0043234 ! protein complex
relationship: part_of GO:0005938 ! cell cortex
relationship: part_of GO:0030427 ! site of polarized growth
```

The most natural approach to translate Go concept definitions into an $\mathcal{EL}^+$ ontology is to use primitive concept definitions. For example, the above Go concept would be defined as

$$\mathsf{GO0000133} \sqsubseteq \mathsf{GO0043234} \sqcap \exists \mathsf{part\_of}.\mathsf{GO0005938} \sqcap \exists \mathsf{part\_of}.\mathsf{GO0030427}.$$

This translation gives us $\mathcal{O}^{\text{Go}}_{\mathsf{prim}}$, an acyclic, primitive TBox with one role inclusion $\mathsf{part\_of} \circ \mathsf{part\_of} \sqsubseteq \mathsf{part\_of}$. It coincides with the OWL version of Go contained in recent distributions of the gene ontology. However, as an ontology for testing DL systems, $\mathcal{O}^{\text{Go}}_{\mathsf{prim}}$ is not well-suited since it is too simple. Actually, computing subsumption in an acyclic, primitive TBox is relatively easy: it simply means computing the transitive closure over the explicitly told subsumption relationships. In the above example, $\mathsf{GO0000133} \sqsubseteq \mathsf{GO0043234}$ is such an explicit relationship. It should be noted however, that neither tableau-based systems nor our implementation directly use this fact.

To obtain an ontology based on Go that is harder to classify, we also consider the translation of Go into non-primitive concept definitions, i.e., the above Go concept would be defined as

$$\mathsf{GO0000133} \equiv \mathsf{GO0043234} \sqcap \exists \mathsf{part\_of}.\mathsf{GO0005938} \sqcap \exists \mathsf{part\_of}.\mathsf{GO0030427}.$$

The result $\mathcal{O}^{\text{Go}}_{\mathsf{def}}$ is an acyclic TBox with the same additional role inclusion as above. In $\mathcal{O}^{\text{Go}}_{\mathsf{def}}$, subsumption cannot be computed as the transitive closure of told subsumptions. However, $\mathcal{O}^{\text{Go}}_{\mathsf{def}}$ is a relatively unusual ontology: due to the

[2] The CEL reasoner and the manual are available at http://lat.inf.tu-dresden.de/systems/cel

assumption that concept definitions also provide sufficient conditions, which has not been intended by the Go developers, there are large groups of concepts that turn out to be equivalent (i.e., subsume each other).

Galen. This ontology aims to promote the sharing and re-use of medical data. It was originally formulated in the language Grail and has been translated into description logic by Ian Horrocks [12]. In that translation, Galen is formulated in $\mathcal{EL}$ extended with GCIs, role hierarchies, transitive roles, functional roles, and inverse roles. Since $\mathcal{EL}^+$ and CEL do not support inverse roles and functional roles, we have removed inverse role axioms and treat functional roles as ordinary ones. In this way, we obtain the $\mathcal{EL}^+$ ontology $\mathcal{O}^{\text{Galen}}$ that contains 1,214 GCIs as well as 2,041 primitive and 699 non-primitive concept definitions. It refers to 413 roles, of which 26 are declared transitive. Moreover, there are 412 role hierarchy axioms.

Snomed is the systematized nomenclature of medicine, a standardization of medical terminology used in the health systems of the US and the UK. The current version of Snomed comprises 379,691 concept and 52 role names. Snomed is formulated as an acyclic $\mathcal{EL}$ TBox that contains 38,719 concept definitions and 340,972 primitive concept definitions. There are no transitive roles,[3] but 11 role hierarchy statements and one right-identity rule of the form $r \circ s \sqsubseteq r$. This gives us the ontology $\mathcal{O}^{\text{Snomed}}$. To get a smaller version of Snomed that can be handled by standard DL reasoners, we also consider the fragment that is obtained by keeping only the non-primitive concept definitions. We call the resulting ontology $\mathcal{O}^{\text{Snomed}}_{\mathsf{core}}$.

The CEL reasoner reads the input ontology, converts each concept definition (if any) into a pair of GCIs, normalizes the resulting ontology, and then starts the refined algorithm described in the previous section. When performing experiments with the initial version of CEL, we found that it could classify only the ontologies $\mathcal{O}^{\text{Go}}_{\mathsf{prim}}$, $\mathcal{O}^{\text{Galen}}$, and $\mathcal{O}^{\text{Snomed}}_{\mathsf{core}}$, but not $\mathcal{O}^{\text{Go}}_{\mathsf{def}}$ and $\mathcal{O}^{\text{Snomed}}$. In the latter two cases, the problem is that the data structures become too big due to too many computed subsumption relationships. A more careful analysis revealed that the problem has different origins for the two ontologies. In the case of $\mathcal{O}^{\text{Go}}_{\mathsf{def}}$, the many subsumption relationships are due to the presence of large groups of equivalent concepts: this is problematic as n equivalent concepts give rise to n^2 subsumption relationships. In addition, if there is a subsumption between two equivalence classes of size n and m, then we must store $n \cdot m$ subsumption relationships. In the case of $\mathcal{O}^{\text{Snomed}}$, many additional concept names are introduced during the normalization phase, for which we also compute all subsumption relationships. These observations led us to adopting the following improvements of the original implementation.

Synonyms identification. Told synonyms are concept names A, B that are explicitly stated to be equivalent by a concept definition $A \equiv B$. We improved

[3] Actually, Snomed needs transitive roles, but since the commercial reasoner used by the developers cannot handle transitivity, they have simulated it in an incomplete way using the approach described in [22].

the algorithm by choosing and keeping only a single representative for each equivalence class of told synonyms. All the dropped synonyms are stored in a cost-effective *union-find* data structure to allow the identification of their representative. This is needed for answering subsumption queries that involve dropped synonyms. The removal of synonyms reduces the number of concept names by more than half in the case of $\mathcal{O}^{\text{Go}}_{\mathsf{def}}$ and enables classification of this ontology by CEL.

Improved normal form. Here, the goal is to reduce the number of additional concept names introduced through normalization. This is achieved in two ways. First, the purpose of additional concept names introduced during normalization is to replace complex concept descriptions in GCIs. The general pattern is that we introduce a concept name A as replacement for a complex concept C, and then add one of $A \sqsubseteq C$ and $A \sqsupseteq C$ to the ontology, depending on whether C was replaced on the left-hand side or on the right-hand side of a GCI. In the improved normalization, we keep track of the concept descriptions C for which an abbreviation has already been introduced, and avoid the introduction of multiple concept names for the same concept description.

Second, in the improved normal form we admit n-ary conjunction on the left-hand side of GCIs, thus avoiding the introduction of $n-1$ concept names for each left-hand side n-ary conjunction. Of course, the Completion Rule **CR2** has to be generalized accordingly:

CR2 If $A_1, \ldots, A_n \in S(X)$, $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B \in \mathcal{O}$, and $B \notin S(X)$
then $S(X) := S(X) \cup \{B\}$

Also, queue entries of the form $B \to B'$ are now replaced by the more general entries $B_1, \cdots, B_n \to B'$ with the obvious meaning. It is straightforward to generalize the process function given in Figure 3 to the new type of queue entries. These two modifications had a drastic effect when normalizing $\mathcal{O}^{\text{Snomed}}$: whereas the original normalization approach introduced 401,830 new concepts, the improved one introduced "only" 114,658. We have compared the perfor-

	$\mathcal{O}^{\text{Go}}_{\mathsf{prim}}$	$\mathcal{O}^{\text{Go}}_{\mathsf{def}}$	$\mathcal{O}^{\text{Galen}}$	$\mathcal{O}^{\text{Snomed}}_{\mathsf{core}}$	$\mathcal{O}^{\text{Snomed}}$
No. of CDefs.	0	16,803	699	38,719	38,719
No. of PCDefs.	16,803	0	2041	0	340,972
No. of GCIs	0	0	1214	0	0
No. of role axioms	1	1	438	0	12
$\lvert\mathsf{CN}_{\mathcal{O}}\rvert$	16,806	16,806	2,740	53,234	379,691
$\lvert\mathsf{RN}_{\mathcal{O}}\rvert$	1	1	413	52	52
CEL	25	10,833	16	1,360	12,772
FaCT	117	11	19	*unattainable*	*unattainable*
RACER	36	*unattainable*	25	84,917	*unattainable*

Table 2. Benchmarks and Evaluation Results

mance of CEL with two of the most advanced tableau-based reasoning systems: FaCT (v2.32.17) and RACER (v1.7.6). Both systems implement expressive DLs in which subsumption is ExpTime-complete. All experiments have been performed on a PC with 2.4GHz Intel Pentium 4 processor and 2GB memory running RedHat Linux 7.2. In the case of Galen, for the sake of fairness also FaCT and RACER have been used with the restricted version of Galen that includes neither functional nor inverse roles. In the case of Snomed, the right-identity rule was passed to CEL, but not to FaCT and RACER, as the latter do not support right identities.

The results of our experiments are summarized in the lower part of Table 2, where *unattainable* means that the reasoner failed due to exhaustion of memory. Notably, CEL outperforms both FaCT and RACER in all benchmarks except $\mathcal{O}^{\mathrm{Go}}_{\mathsf{def}}$, where FaCT needs only 11 seconds (we currently have no explanation for the good behavior of FaCT on $\mathcal{O}^{\mathrm{Go}}_{\mathsf{def}}$). Moreover, CEL was the only reasoner that was able to classify the whole Snomed ontology.[4]

5 Conclusion

We have proposed a refined classification algorithm for the description logic $\mathcal{EL}^+$, implemented it in the CEL reasoner, and then used CEL to show that even a relatively straightforward implementation of this algorithm can compete with, and usually even outperform, highly optimized tableau-based DL reasoners. We view this result as a strong argument for the use of tractable DLs, provided that their expressive power is sufficient for the application in question. We also believe that, when more powerful optimization techniques are developed, reasoners based on the refined algorithm presented in this paper can be made much more efficient than our current implementation CEL. Such additional optimizations may include the following:

Currently, we explicitly compute and store all the subsumption relationships between concepts names, rather than just computing and storing the Hasse-diagram of the subsumption hierarchy, as done by the other DL systems. Storing only the Hasse-diagram will definitely save space, but it is not yet clear how much time overhead (for computing subsumption relationships from this representation) this will cause.

Closely related to the previous point is the question whether we can use the well-known techniques for avoiding calls to the subsumption algorithm used by other DL systems while computing the Hasse-diagram (see [3]). With the current algorithm, this does not really make sense since a single call to our subsumption algorithm already computes the whole hierarchy. Thus, one must first modify our classification algorithm into a goal-directed subsumption algorithm that checks, with minimum efforts, only whether two given concept names are in a subsumption relationship.

[4] We have recently heard that the optimizations described in [13] have the effect that FaCT^{++} is now also capable of classifying $\mathcal{O}^{\mathrm{Snomed}}$. We were, however, not yet able to verify this claim with the version of FaCT^{++} available for download.

Another technique used by other DL systems is to determine some obvious (non-)subsumption relationships (such as told subsumers) before calling the actual subsumption algorithm. It is not clear yet how this kind of information can be used in our type of algorithm, but one could try to adopt the approach described in [13].

As shown by our experiments, the reduction of the number of concept names is a crucial issue for the efficiency of our algorithm. Therefore, any techniques that make such a reduction is promising candidates for further optimizations. It is clear that the two simple optimizations described in this paper can further be improved, but there may also be other, more sophisticated approaches to reduce the number of concepts to be considered.

Finally, we plan to extend our implementation CEL to the description logic $\mathcal{EL}^{++}$ [2].

References

1. Franz Baader. Terminological cycles in a description logic with existential restrictions. In *Proc. IJCAI'03*, 2003.
2. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. IJCAI'05*, 2005.
3. Franz Baader, Enrico Franconi, Bernhard Hollunder, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.
4. Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
5. Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. AAAI'84*, 1984.
6. Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In *Proc. ECAI'04*, 2004.
7. The Gene Ontology Consortium. Gene Ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.
8. R.A. Cote, D.J. Rothwell, J.L. Palotay, R.S. Beckett, and L. Brochu. The systematized nomenclature of human and veterinary medicine. Technical report, SNOMED International, Northfield, IL: College of American Pathologists, 1993.
9. William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 1(3):267–284, 1984.
10. Volker Haarslev and Ralf Möller. RACER system description. In *Proc. IJCAR'01*, 2001.
11. Monika R. Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *36th Annual Symposium on Foundations of Computer Science*, 1995.
12. Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics.* PhD thesis, University of Manchester, 1997.
13. Dmitry Tsarkov and Ian Horrocks. Optimised classification for taxonomic knowledge bases. In *Proc. DL'05*, 2005.

14. Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. KR'98*, 1998.
15. Ian Horrocks. Implementation and optimization techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.
16. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
17. Ian Horrocks, Ulrike Sattler, and Stefan Tobies. Practical reasoning for very expressive description logics. *J. of the IGPL*, 8(3):239–264, 2000.
18. Bernhard Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.
19. Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.
20. Peter F. Patel-Schneider and Bill Swartout. Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort. Technical report, DARPA Knowledge Representation System Specification (KRSS) Group of the Knowledge Sharing Initiative, 1993.
21. Alan Rector and Ian Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*, Stanford, CA, 1997. AAAI Press.
22. Stefan Schulz, Martin Romacker, and Udo Hahn. Part-whole reasoning in medical ontologies revisited: Introducing SEP triplets into classification-based description logics. *Journal of the American Medical Informatics Association (JAMIA)*, pages 830–834, 1998.
23. Kent A. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. *J. of the American Medical Informatics Association*, 2000. Fall Symposium Special Issue.
24. Boontawee Suntisrivaraporn. Optimization and implementation of subsumption algorithms for the description logic $\mathcal{EL}$ with cyclic TBoxes and general concept inclusion axioms. Master thesis, TU Dresden, Germany, 2005.