Efficient Reasoning in $\mathcal{EL}^+$

Franz Baader, Carsten Lutz, Boontawee Suntisrivaraporn
Institute for Theoretical Computer Science
TU Dresden, Germany
{baader,clu,meng}@tcs.inf.tu-dresden.de

1 Introduction

The early dream of a description logic (DL) system that offers both sound and complete polynomial-time algorithms and expressive means that allow its use in real-world applications has since the 1990ies largely been considered to be a pipe dream. This was, on the one hand, due to complexity results showing intractability even in very inexpressive DLs [5], in particular in the presence of TBoxes [13]. On the other hand, many of the applications considered then required more expressive power rather than less, which led to the development of more and more expressive DLs. The use of such intractable DLs in applications was made possible by the fact that highly-optimized tableau-based reasoners for them behaved quite well in practice.

However, more recent developments regarding the $\mathcal{EL}$ family of DLs have shed a new light on the realizability of this dream.[1] On the one hand, theoretical results [1, 6, 2] have shown that reasoning in $\mathcal{EL}$ and several of its extensions remains tractable in the presence of TBoxes and even of general concept inclusions (GCIs). On the other hand, it has turned out that, despite its relatively low expressivity, the $\mathcal{EL}$ family is highly relevant for a number of important applications, in particular in the bio-medical domain: for example, medical terminologies such as the Systematized Nomenclature of Medicine (Snomed) [9] and the Galen Medical Knowledge Base (Galen) [14] are formulated in $\mathcal{EL}$ or small extensions thereof, and the Gene Ontology (Go) [8] used in bioinformatics can also be viewed as an $\mathcal{EL}$ TBox.

In this paper, we address the question of whether the polynomial-time algorithms for reasoning in $\mathcal{EL}$ and its extensions really behave better in practice than intractable, but highly-optimized tableau-based algorithms. To this end, we have implemented a refined version of the algorithm described in [2] in our

[1] An alternative contender for a usable tractable DL is the one introduced in [7].

$$
\begin{array}{rcl}
\mathsf{Endocardium} & \sqsubseteq & \mathsf{Tissue} \sqcap \exists \mathsf{cont\text{-}in}.\mathsf{HeartWall} \sqcap \\
 & & \qquad \exists \mathsf{cont\text{-}in}.\mathsf{HeartValve} \\
\mathsf{HeartWall} & \sqsubseteq & \mathsf{BodyWall} \sqcap \exists \mathsf{part\text{-}of}.\mathsf{Heart} \\
\mathsf{HeartValve} & \sqsubseteq & \mathsf{BodyValve} \sqcap \exists \mathsf{part\text{-}of}.\mathsf{Heart} \\
\mathsf{Endocarditis} & \sqsubseteq & \mathsf{Inflammation} \sqcap \exists \mathsf{has\text{-}loc}.\mathsf{Endocardium} \\
\mathsf{Inflammation} & \sqsubseteq & \mathsf{Disease} \sqcap \exists \mathsf{acts\text{-}on}.\mathsf{Tissue} \\
\mathsf{Heartdisease} \sqcap \exists \mathsf{has\text{-}loc}.\mathsf{HeartValve} & \sqsubseteq & \mathsf{CriticalDisease} \\
\mathsf{Heartdisease} & \doteq & \mathsf{Disease} \sqcap \exists \mathsf{has\text{-}loc}.\mathsf{Heart} \\
\mathsf{part\text{-}of} \circ \mathsf{part\text{-}of} & \sqsubseteq & \mathsf{part\text{-}of} \\
\mathsf{part\text{-}of} & \sqsubseteq & \mathsf{cont\text{-}in} \\
\mathsf{has\text{-}loc} \circ \mathsf{cont\text{-}in} & \sqsubseteq & \mathsf{has\text{-}loc}
\end{array}
$$

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

CEL reasoner,[2] and compared its performance on the large bio-medical ontologies Snomed, Go, and a trimmed-down version of Galen with the performance of the most advanced tableau-based reasoners FaCT^{++},[3] RacerMaster,[4] and Pellet[5] on these ontologies.

2 $\mathcal{EL}^+$ and the algorithm implemented in CEL

The CEL system currently supports the DL $\mathcal{EL}^+$, whose concepts are formed according to the syntax rule

$$C ::= A \mid \top \mid C \sqcap D \mid \exists r.C,$$

where A ranges over concept names, r over role names, and C, D over concepts. Thus, the concept language of $\mathcal{EL}^+$ is identical to that of $\mathcal{EL}$. The $\cdot^+$ in its name refers to the more powerful ontology formalism. An $\mathcal{EL}^+$ *ontology* is a finite set of *general concept inclusions (GCIs)* $C \sqsubseteq D$ and *role inclusions (RIs)* $r_1 \circ \cdots \circ r_n \sqsubseteq r$. Note that $\mathcal{EL}^+$ ontologies can thus express transitive roles, role hierarchies, and so-called right-identities on roles ($r \circ s \sqsubseteq s$), which are very useful in medical ontologies [15, 11]. The semantics of concepts and ontologies is defined in the usual way (see, e.g., [2]). We write $C \sqsubseteq_{\mathcal{O}} D$ if the concept C is subsumed by the concept D w.r.t. the ontology $\mathcal{O}$. Figure 1 shows a small medical ontology formulated in $\mathcal{EL}^+$. Based on this ontology, it is not hard to see that the

[2] CEL can be downloaded from `http://lat.inf.tu-dresden.de/systems/cel/`

[3] See `http://owl.man.ac.uk/factplusplus/`

[4] See `http://www.racer-systems.com/`

[5] See `http://www.mindswap.org/2003/pellet/`

Normal Form	Completion Rules
$A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$	**R1** 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 $S(X) := S(X) \cup \{B\}$
$A \sqsubseteq \exists r.B$	**R2** If $A \in S(X)$, $A \sqsubseteq \exists r.B \in \mathcal{O}$, and $(X, B) \notin R(r)$ then $R(r) := R(r) \cup \{(X, B)\}$
$\exists r.A \sqsubseteq B$	**R3** If $(X, Y) \in R(r)$, $A \in S(Y)$, $\exists r.A \sqsubseteq B \in \mathcal{O}$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$
$r \sqsubseteq s$	**R4** If $(X, Y) \in R(r)$, $r \sqsubseteq s \in \mathcal{O}$, and $(X, Y) \notin R(s)$ then $R(s) := R(s) \cup \{(X, Y)\}$
$r \circ s \sqsubseteq t$	**R5** 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 $R(t) := R(t) \cup \{(X, Z)\}$

Figure 2: Normal form and completion rules

concept Endocarditis is subsumed by the concept Heartdisease and CriticalDisease (i.e. Endocarditis $\sqsubseteq_{\mathcal{O}}$ Heartdisease and Endocarditis $\sqsubseteq_{\mathcal{O}}$ CriticalDisease).

The main reasoning service that CEL offers is classification, i.e., the computation of the complete subsumption hierarchy between all concept names occurring in the input ontology $\mathcal{O}$. The CEL reasoner is based on the polynomial time algorithm developed in [2] for the extension $\mathcal{EL}^{++}$ of $\mathcal{EL}^{+}$, which can additionally handle the bottom concept (and thus disjoint concepts), restricted concrete domains, and nonimals (and thus ABoxes). In the following, we briefly introduce the restriction of this algorithm to $\mathcal{EL}^{+}$, and then describe the refined version of this algorithm implemented in our CEL reasoner.

To classify an ontology, the algorithm first transforms it into *normal form*, which requires that all concept and role inclusions 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 simple transformation rules, any ontology $\mathcal{O}$ can be transformed in linear time into a normalized one such that subsumption between the concept names occurring in $\mathcal{O}$ is preserved [16, 2].

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

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

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 $A \in \mathsf{CN}_{\mathcal{O}}^{\top}$ and $R(r) := \emptyset$ for each $r \in \mathsf{RN}_{\mathcal{O}}$. 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.

In [2], it is shown that this algorithm always terminates in time polynomial in the size of the input ontology, and that it is sound and complete in the following sense: after termination we have $B \in S(A)$ iff $A \sqsubseteq_{\mathcal{O}} B$, for all $A, B \in \mathsf{CN}_{\mathcal{O}}^{\top}$. Thus, the sets $S(A)$ yield a complete representation of the subsumption relation $\sqsubseteq_{\mathcal{O}}$ on the concept names from $\mathcal{O}$ (including $\top$).

It is obvious that, when implementing this algorithm, an efficient approach for finding an applicable rule must be developed. Though a naïve brute-force search for applicable rules would still yield a polynomial time algorithm, its complexity on very large knowledge bases would be prohibitive. As a solution to this problem, we use a refined version of the algorithm, which is inspired by the linear-time algorithm for satisfiability of propositional Horn formulas proposed in [10]. More precisely, our algorithm uses a set of queues, one for each element of $\mathsf{CN}_{\mathcal{O}}^{\top}$, to guide the application of completion rules. Intuitively, the queues list additions to $S(A)$ and $R(r)$ that still have to be carried out. The possible entries of the queues are of the form

$$B_1 \sqcap \cdots \sqcap B_n \rightarrow B' \quad \text{and} \quad \exists r.B$$

with $B_1, \ldots, B_n$, B, and B' concept names, r a role name, and $n \geq 0$. For the case $n = 0$, we simply write the queue entry $B_1 \sqcap \cdots \sqcap B_n \rightarrow B'$ as B'. Intuitively,

- an entry $B_1 \sqcap \cdots \sqcap B_n \rightarrow B'$ in $\mathsf{queue}(A)$ means that B' has to be added to $S(A)$ if $S(A)$ already contains $B_1, \ldots, B_n$, 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 $A \in \mathsf{CN}_{\mathcal{O}}^{\top}$, $\widehat{\mathcal{O}}(A)$ is the minimal set of queue entries such that

- if $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B \in \mathcal{O}$ and $A = A_i$, then

$$A_1 \sqcap \cdots \sqcap A_{i-1} \sqcap A_{i+1} \sqcap \cdots \sqcap A_n \rightarrow B \in \widehat{\mathcal{O}}(A);$$

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

```
procedure process(A, X)
begin
    if X = B_1, ..., B_n → B and B ∉ S(A) then
        if {B_1, ..., B_n} ⊆ S(A) then
            S(A) := S(A) ∪ {B};
            queue(A) := queue(A) ∪ Ô(B);
            for all concept names A' and role names r
                    with (A', A) ∈ R(r) do
                queue(A') := queue(A') ∪ Ô(∃r.B);
    if X = ∃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;
```

Figure 3: Processing the queue entries

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 $B \in \widehat{\mathcal{O}}(\exists r.A)$.

In the modified algorithm, the queues are used as follows: 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 queues and process them using the procedure process displayed in Figure 3. To be more precise, $\mathsf{process}(A, X)$ is called when the queue of A was non-empty and we fetched the queue entry X from $\mathsf{queue}(A)$ to be treated next. Observe that the first if-clause of the procedure process implements **R1** and (part of) **R3**, and the second if-clause implements **R2**, (the rest of) **R3**, as well as **R4** and **R5**. The procedure $\mathsf{process\text{-}new\text{-}edge}(A, r, B)$ is called by process to handle 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 statements $r \sqsubseteq s$ occurring in $\mathcal{O}$. 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.

With a relatively straightforward implementation (in Common LISP) of this algorithm, we were able to classify the large SNOMED ontology (see below) in less than 4 hours (see [4] for this and other experimental results). Since then, however, we have further improved the implementation by changing the strategy of rule applications, changing the encoding of concept and role names, and by using low-level optimizations of the underlying data structure. These optimizations have enhanced the performance of CEL on large real-world ontologies. In particular, CEL can now classify SNOMED in less than half an hour (see below).

3 The experimental results

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) [8], the Galen Medical Knowledge Base (GALEN) [14], and the Systematized Nomenclature of Medicine (SNOMED) [9]. These ontologies provide us with the following benchmark $\mathcal{EL}^+$ ontologies:

- $\mathcal{O}^{\text{Go}}$, which is the latest OWL version of GO;
- $\mathcal{O}^{\text{Galen}}$, which is a stripped-down version of GALEN obtained by removing inverse role axioms and treating functional roles as ordinary ones;
- $\mathcal{O}^{\text{Snomed}}$, which is the complete SNOMED ontology;
- $\mathcal{O}^{\text{Snomed}}_{\text{core}}$, which is the definitorial fragment of $\mathcal{O}^{\text{Snomed}}$, obtained by keeping only complete concept definitions $(A \equiv C)$, but not the primitive ones $(A \sqsubseteq C)$. We consider this fragment in order to obtain another benchmark that is easier to tackle for standard DL reasoners.

Some information about the size and structure of these ontologies is shown in the upper part of Table 1, where CDefs stands for complete concept definitions $(A \equiv C)$, PCDefs for primitive concept definitions $(A \sqsubseteq C)$, and GCIs for concept inclusions that are neither CDefs nor PCDefs. It is interesting to note that all ontologies except for $\mathcal{O}^{\text{Galen}}$ are actually acyclic TBoxes.

We have compared the performance of CEL with three of the most advanced tableau-based reasoning systems: FaCT^{++} (v1.1.0), RacerMaster (v1.9.0), and Pellet (v1.3b). All these systems implement expressive DLs in which subsumption is EXPTIME-complete. The experiments have been performed on a PC with 2.8GHz Intel Pentium 4 processor and 512MB memory running Linux v2.6.14.

	$\mathcal{O}^{\text{Go}}$	$\mathcal{O}^{\text{Galen}}$	$\mathcal{O}^{\text{Snomed}}_{\text{core}}$	$\mathcal{O}^{\text{Snomed}}$
No. of CDefs.	0	699	38,719	38,719
No. of PCDefs.	20,465	2041	0	340,972
No. of GCIs	0	1214	0	0
No. of role axioms	1	438	0	$11 + 1$
$\lvert \mathsf{CN}_{\mathcal{O}} \rvert$	20,465	2,740	53,234	379,691
$\lvert \mathsf{RN}_{\mathcal{O}} \rvert$	1	413	52	52
CEL	5.8	14	95	1,782
FaCT^{++}	6.9	50	740	3,859
RacerMaster	19	14	34,709	*unattainable*
Pellet	1,357	75	*unattainable*	*unattainable*

Table 1: Benchmarks and Evaluation Results

For Pellet, we used JVM v1.5 and set the Java heap space to 256MB (as recommended by the implementor). In the case of Galen, for the sake of fairness also the tableau reasoners have been used with the restricted version of Galen that includes neither functional nor inverse roles. In the case of Snomed, the only existing right-identity rule was passed to CEL, but not to the other reasoners as they do not support right identities. The results of our experiments are summarized in the lower part of Table 1, where all classification times are shown in seconds and *unattainable* means the reasoner failed due to memory exhaustion. Notably, CEL outperforms all the reasoners in all benchmarks except for $\mathcal{O}^{\text{Galen}}$, where CEL and RacerMaster show the same performance.

CEL and FaCT^{++} are the only reasoners that can classify $\mathcal{O}^{\text{Snomed}}$, whereas RacerMaster and Pellet fail. Pellet and the original version of FaCT (not shown in the table) also fail already to classify $\mathcal{O}^{\text{Snomed}}_{\text{core}}$. It seems worth noting that the performance of FaCT^{++} on $\mathcal{O}^{\text{Snomed}}$ degrades dramatically if $\mathcal{O}^{\text{Snomed}}$ is extended with real GCIs. For instance, FaCT^{++} needs about 3,000 more seconds to classify $\mathcal{O}^{\text{Snomed}}$ for each additional randomly generated GCI of the form $\exists r.C \sqsubseteq D$, whereas the performance of CEL does not change noticeably if we add such GCIs.

4 Computing the Subsumption DAG

The innate classification output of CEL is simply the computed sets $S(A)$ for all concept names A. We call these sets *subsumer sets* in what follows. In contrast, tableau-based reasoners usually employ the enhanced traversal method from [3] to generate a directed acyclic graph (DAG) describing the *direct* subsumption relationships, i.e., for every concept name A they compute the sets of its direct subsumers and subsumees, which are the sets of concept names B such that $A \sqsubseteq_{\mathcal{O}} B$ ($B \sqsubseteq_{\mathcal{O}} A$) and there is no concept name $B' \notin \{A, B\}$ with $A \sqsubseteq_{\mathcal{O}} B' \sqsubseteq_{\mathcal{O}} B$ ($B \sqsubseteq_{\mathcal{O}} B' \sqsubseteq_{\mathcal{O}} A$). We will call this graph the *subsumption DAG*. Since the subsumption relation is a quasi-order rather than a partial order (i.e., in general not antisymmetric), one node of the DAG actually corresponds to an equivalence class of concept names rather than a single concept name. The advantage of using subsumption DAGs over subsumer sets is that this format is more compact, and it directly supports browsing the subsumption hierarchy by going from a concept name to its direct subsumers or subsumees. The disadvantage is that answering a subsumption question $A \sqsubseteq^?_{\mathcal{O}} B$ then requires to test reachability of B from A in the DAG, and not just a look-up in the subsumer set $S(A)$.

Since many applications require subsumption DAGs rather than (or in addition to) subsumer sets, CEL allows to construct the former from the latter in an efficient way. In principle, converting subsumer sets into a subsumption DAG is easy. We can simply compute, for each concept name A,

- the set $SS(A) := \{B \in S(A) \mid A \notin S(B)\}$ of strict subsumers of A, i.e., subsumers of A that are not equivalent to A;
- the set $DS(A) := SS(A) \setminus \left(\bigcup_{B \in SS(A)} SS(B)\right)$ of direct subsumers of A;
- the set $DS^-(A) := \{B \mid A \in DS(B)\}$ of direct subsumees of A.

Clearly, the sets $DS(A)$ and $DS^-(A)$ yield a representation of the subsumption DAG.

However, we do not use this direct construction since computing the sets $DS^-(A)$ is expensive (it needs quadratic time) and it is possible to avoid the direct computation of these sets according to the above definition by using an approach that is inspired by the enhanced traversal method in [3]. Another virtue of our alternative approach is that the potentially costly set operations in the computation of $DS(A)$ are replaced by an inexpensive marking algorithm.

In order to explain the main idea underlying our algorithm, assume that we have already computed a restriction of the subsumption DAG to some subset of the concept names, and that we now want to insert the concept name A into this DAG. We start by computing the set $SS(A)$ of strict subsumers according to the definition given above. The elements of $S(A) \setminus SS(A)$ are the concepts that are equivalent to A. To find all the *direct* subsumers of A among the elements of $SS(A)$, we proceed as follows. If all elements of $SS(A)$ belong to the already computed DAG, we can find the direct subsumers by using a simple graph traversal algorithm to mark all the strict subsumers of elements of $SS(A)$ in the DAG. The direct subsumers of A are then those elements of $SS(A)$ that are not marked. If there are elements of $SS(A)$ that do not belong to the already computed DAG, then we simply first insert these elements into the DAG (by issuing recursive calls of the insertion procedure) before inserting A. By following this strategy, we ensure that, when inserting a concept name A into the DAG, all subsumers of A are already in the DAG, but no subsumee of A is. Hence, our algorithm need not compute the direct subsumees explicitly. Instead, it is enough to extend the set of direct subsumees of B by A in case B is found to be a direct subsumer of A.

Figure 4 shows a pseudo code representation of our algorithm. The sets $\mathsf{parents}(A)$ are used to store the direct subsumers of A, the sets $\mathsf{children}(A)$ are used to store the direct subsumees of A, and the sets $\mathsf{equivalents}(A)$ are used to store the concepts that are equivalent to A. Note that the description of the algorithm is a bit sloppy in that we do not distinguish between a concept name and the node in the DAG representing (the equivalence class of) this name.

An algorithm similar to ours is obtained if we describe the subsumer sets as a primitive TBox, i.e. a set of primitive concept definitions $A \sqsubseteq \sqcap_{B_i \in S(A)} B_i$ for each concept name A, and then employ a simplified version of the enhanced traversal method [3] using told subsumer information and some of the optimizations described in [12] to compute the subsumption DAG from the resulting TBox.

The time required by CEL for computing subsumption DAGs is very small. For example, even in the case of $\mathcal{O}^{\text{Snomed}}$, which has almost 380,000 concepts and huge subsumer sets, it takes only 9 seconds. This is negligible compared to the time needed to compute the subsumer sets. In particular, if we add this time to CEL's run-time on $\mathcal{O}^{\text{Snomed}}$ in Table 1, CEL is still more than twice as fast as FaCT^{++}.

There is an obvious alternative to first computing the full subsumer sets, and only then deriving the subsumption DAG from them: we could modify our classification algorithm, which computes the whole subsumption hierarchy, into a subsumption algorithm that answers only a single subsumption query, and then use this subsumption algorithm inside a standard enhanced traversal algorithm as described in [3]. We have experimented with this strategy, which is closer to the approach employed by tableau-based systems. To turn our algorithm into a subsumption algorithm, we have developed a goal-directed variant of it, which is based on activating a concept name if computing its subsumer set is required for answering the subsumption question at hand. If the aim is to answer the subsumption query $A \sqsubseteq^?_{\mathcal{O}} B$, then initially only A is activated. Intuitively, completion rules are only applied to activated names. We activate a concept name B' whenever B' is the second component of a tuple added to some $R(r)$. The set $S(A')$ and the queue of A' is initialized only when the concept name becomes activated, and thus the subsumer sets of concept names that do not become activated are not populated by the algorithm. During the construction of the whole subsumption DAG, the enhanced traversal procedure makes repeated calls to the subsumption algorithm. To avoid redoing work, we retain the already

```
procedure compute-dag
    for all concept names X ∈ CN_O^⊤ do
        classified(X) := false
        parents(X) := children(X) := equivalents(A) := ∅
    for each concept name A ∈ CN_O^⊤ do
        if not classified(A) then
            dag-classify(A);
end;

procedure dag-classify(A)
    candidates := ∅;
    for all subsumers B ∈ S(A) do
        if A ∈ S(B) then
            classified(B) := true;
            equivalents(A) := equivalents(A) ∪ {B};
        else
            if not classified(B) then
                dag-classify(B);
            candidates := candidates ∪ {B};
    dag-insert(A, candidates);
    classified(B) := true;
end;

procedure dag-insert(A, candidates)
    marked(X) := false for all X ∈ CN_O^⊤;
    for all B ∈ candidates do
        for all X ∈ parents(B) do
            marked(X) := true
    while there are X, Y ∈ CN_O^⊤ with marked(X), Y ∈ parents(X), and
         not marked(Y) do
        marked(Y) := true
    parents(A) := {B ∈ candidates | marked(B) = false};
    for all B ∈ parents(A) do
        children(B) := children(B) ∪ {A};
end;
```

Figure 4: Computing the DAG from the subsumer sets

computed parts of the mappings $S(\cdot)$ and $R(\cdot)$ for such repeated calls.

However, our current implementation of this idea cannot compete with the runtime of the original CEL implementation described before. For example, the classification of $\mathcal{O}^{\text{Snomed}}$ takes 3,750 seconds. This is still slightly better than the performance of FaCT^{++}, but more than twice of the 1,791 seconds needed when first computing the subsumer sets and then constructing the subsumption DAG. The reason is probably that, in sum, the single subsumption tests do the same work as the full classification algorithm, but then there is the additional overhead of the enhanced traversal method (which is more complicated than the simplified version employed to compute the subsumption DAG from the subsumer sets).

5 Conclusion

The performance evaluations show that our tractable reasoner CEL outperforms modern reasoners for intractable DLs based on tableau algorithms. It should be noted that the good performance of CEL is achieved with a relatively straightforward implementation of the tractable algorithm, whereas the tableau-based systems are the result of many years of research into optimization techniques. The robustness and scalability of tractable reasoning is visible in the case of SNOMED, which comprises almost 380,000 concept definitions. Only CEL and FaCT^{++} can classify this terminology, whereas RacerMaster, Pellet, and the original version of FaCT fail. Additionally, FaCT^{++} shows a significant degradation in performance if SNOMED, which is an acyclic TBox, is extended with GCIs. In contrast, the runtime of CEL is not noticeably affected by such an extension.

Developing CEL is ongoing work. We plan to extend its capabilities to the DL $\mathcal{EL}^{++}$ [2], which includes, among other things, *nominals* and *the bottom concept* (thus one can express *ABoxes* and *disjoint concepts*). We also plan to implement DIG and OWL interfaces,[6] so that CEL can be used as a backend reasoner for ontology editors like OilEd[7] and Protégé,[8] which would also make their sophisticated graphical user-interfaces available to users of CEL.

References

[1] F. Baader. Terminological cycles in a description logic with existential restrictions. In *Proc. IJCAI'03*, 2003.

[2] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. IJCAI'05*, 2005.

[3] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence*, 4:109–132, 1994.

[6] See `http://dl.kr.org/dig/` and `http://www.w3.org/2004/OWL/`

[7] See `http://oiled.man.ac.uk/`

[8] See `http://protege.stanford.edu/`

[4] F. Baader, C. Lutz, and B. Suntisrivaraporn. Is tractable reasoning in extensions of the description logic $\mathcal{EL}$ useful in practice? In *Proc. M4M'05*, 2005.

[5] R. J. Brachman and H. J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. AAAI'84*, 1984.

[6] S. Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In *Proc. ECAI'04*, 2004.

[7] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. AAAI'05*, 2005.

[8] The Gene Ontology Consortium. Gene Ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[9] R. Cote, D. Rothwell, J. Palotay, R. Beckett, and L. Brochu. The systematized nomenclature of human and veterinary medicine. Technical report, SNOMED International, Northfield, IL: College of American Pathologists, 1993.

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

[11] I. Horrocks and U. Sattler. Decidability of SHIQ with complex role inclusion axioms. *Artificial Intelligence*, 160(1–2):79–104, 2004.

[12] I. Horrocks and D. Tsarkov. Optimised classification for taxonomic knowledge bases. In *Proc. DL'05*, 2005.

[13] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[14] A. Rector and I. Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proc. Ontological Engineering Workshop, AAAI Spring Symposium*, 1997.

[15] K.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.

[16] B. 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.