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. This refined algorithm 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.

Keywords: Description logic, tractable reasoning

1. Introduction and Motivation

The quest for tractable (i.e., polynomial-time decidable) description logics (DLs) started in the 1980s after the first intractability results for DLs were shown (Brachman and Levesque, 1984; Nebel, 1988). Until recently, it was restricted to DLs that extend the basic language $\mathcal{FL}_0$, which comprises the concept constructors conjunction ($\sqcap$) and value restriction ($\forall r.C$). The main reason for this focussing was that, when clarifying the logical status of property arcs in semantic networks and slots in frames (which are the ancestors of modern DLs), the decision was taken that arcs/slots should be read as value restrictions rather than existential restrictions ($\exists r.C$).

In almost every application of DLs, it is crucial to reason with terminologies (also called TBoxes or DL ontologies), rather than with isolated concept descriptions. Unfortunately, as soon as TBoxes 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 (Nebel, 1990). If the most general form of TBoxes is admitted, which consists of general concept inclusion axioms (GCIs) as supported by all modern DL systems, then classification in $\mathcal{FL}_0$ even becomes ExpTime-complete (Baader et al., 2005).

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., (Horrocks et al., 2000)). This line of research has resulted in the availability of highly optimized DL systems for expressive DLs based on tableau algorithms (Horrocks, 1998; Haarslev and Möller, 2001), and successful applications: most notably the recommendation by the W3C of the DL-based language OWL (Horrocks et al., 2003) 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 (Baader, 2003), and this remains so even if general TBoxes with GCIs are admitted (Brandt, 2004). 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, the Systematized Nomenclature of Medicine (SNOMED), employs $\mathcal{EL}$ with an acyclic TBox (Cote et al., 1993; Spackman, 2000). The Gene Ontology (GO) (The Gene Ontology Consortium, 2000) can be seen as an acyclic $\mathcal{EL}$ TBox with one transitive role. Finally, large parts of the Galen Medical Knowledge Base (GALEN) can also be expressed in $\mathcal{EL}$ with GCIs, role hierarchy, and transitive roles (Rector and Horrocks, 1997).

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 $\mathcal{EL}^{++}$ (Baader et al., 2005), which includes transitive roles, so-called right-identities (Spackman, 2000) on roles, nominals (and thus ABoxes), and disjointness constraints on concepts. The purpose of the research presented in the present paper is 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 (Horrocks, 2003), 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 (Baader, 2003): direct implementations of both the algorithm for subsumption w.r.t. descriptive semantics (based on a reduction to satisfiability of propositional Horn formulae (Dowling and Gallier, 1984)) and the algorithm for subsumption w.r.t. greatest fixpoint semantics (based on computing the greatest simulation on a graph (Henzinger et al., 1995)) did not lead to satisfactory results on the Gene Ontology (Suntisrivaraporn, 2005).

In this paper, we consider a restriction of the polynomial-time classification algorithm for $\mathcal{EL}^{++}$ (Baader et al., 2005) 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 towards implementation. The purpose of this refinement is to remove an obvious obstacle for efficient implementation of the algorithm as given in (Baader et al., 2005): 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 CN of *concept names* and a set RN of *role names*. $\mathcal{EL}^+$ concept descriptions are formed using the three constructors shown in the upper part of Table I. An $\mathcal{EL}^+$ *ontology*

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

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

is a finite set of *general concept inclusions (GCIs)* and *role inclusions (RIs)*, whose syntax is shown in the lower part of Table I.

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 $A \in \mathsf{CN}$ to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and each role name $r \in \mathsf{RN}$ 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 I. 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 I are satisfied.

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 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

[1] despite not actually defining anything.

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

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

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 (Spackman, 2000).

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.

As an example, we consider the $\mathcal{EL}^+$ ontology $\mathcal{O}$ in Figure 1, which is motivated by Galen and expresses medical knowledge about *endocarditis* (inflammation of the lining of the heart and the heart valves) and some related concepts. In the figure, all capitalized words are concept names and all lowercase words are role names. The ontology contains five primitive concept definitions, one "real" GCI, one non-primitive concept definition, and three RIs. The latter consist of one transitivity statement, a role hierarchy statement, and a right-identity axiom. It is not hard to see that, w.r.t. this ontology, endocarditis is classified as a heart disease, i.e.,

$$\mathsf{Endocarditis} \sqsubseteq_{\mathcal{O}} \mathsf{Heartdisease}.$$

To see this, note that $\mathsf{Endocarditis}$ implies $\mathsf{Inflammation}$ and thus $\mathsf{Disease}$, which yields the first conjunct in the definition of $\mathsf{Heartdisease}$. Moreover, $\mathsf{Endocarditis}$ implies $\exists\mathsf{has\text{-}loc}.\mathsf{Endocardium}$ which can be seen to imply $\exists\mathsf{has\text{-}loc}.\exists\mathsf{cont\text{-}in}.\exists\mathsf{part\text{-}of}.\mathsf{Heart}$. In the presence of the last two RIs, this implies $\exists\mathsf{has\text{-}loc}.\mathsf{Heart}$, which is second conjunct in the definition of $\mathsf{Heartdisease}$.

As a second example, it can also be seen that endocarditis is classified as a critical disease, i.e., $\mathsf{Endocarditis} \sqsubseteq_\mathcal{O} \mathsf{CriticalDisease}$.

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

A polynomial-time algorithm for classification in $\mathcal{EL}$ with GCIs and role hierarchies has been proposed in (Brandt, 2004), and this algorithm was extended to the more powerful DL $\mathcal{EL}^{++}$ in (Baader et al., 2005). We introduce the restriction of the algorithm from (Baader et al., 2005) 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 number of subsumption tests needed to compute the whole hierarchy (Baader et al., 1994). In contrast, the classification algorithm in (Baader et al., 2005) simultaneously computes the subsumption relationships between *all* pairs of concept names in the input ontology.

3.1. A Normal Form for $\mathcal{EL}^+$ Ontologies

Before we can describe the polynomial-time classification algorithm for $\mathcal{EL}^+$, we must introduce an appropriate normal form for $\mathcal{EL}^+$ ontologies. Given an ontology $\mathcal{O}$, we write $\mathsf{CN}^\top_\mathcal{O}$ and $\mathsf{CN}_\mathcal{O}$ to denote the sets of concept names occurring in $\mathcal{O}$ with and without the top concept, respectively. Then, $\mathcal{O}$ is in *normal form* if

1. all GCIs in $\mathcal{O}$ have one of the following forms, where $A_i \in \mathsf{CN}^\top_\mathcal{O}$ and $B \in \mathsf{CN}_\mathcal{O}$:
$$\begin{aligned} A_1 \sqcap \cdots \sqcap A_n &\sqsubseteq B, \\ A_1 &\sqsubseteq \exists r.A_2, \\ \exists r.A_1 &\sqsubseteq B. \end{aligned}$$
2. all role inclusions are of the form $r \sqsubseteq s$ or $r_1 \circ r_2 \sqsubseteq s$.

By introducing new concept and role names, any $\mathcal{EL}^+$ ontology $\mathcal{O}$ can be turned into a normalized ontology $\mathcal{O}'$ that is a *conservative extension* of $\mathcal{O}$, i.e., we have $C \sqsubseteq_\mathcal{O} D$ iff $C \sqsubseteq_{\mathcal{O}'} D$ for all concept descriptions C, D that are constructed from concept and role names occurring in $\mathcal{O}$.

Lemma 1. *Subsumption w.r.t. ontologies in $\mathcal{EL}^+$ can be reduced in linear time to subsumption w.r.t. normalized ontologies in $\mathcal{EL}^+$.*

NF1	$r_1 \circ \cdots \circ r_k \sqsubseteq s$	$\rightsquigarrow$	$r_1 \circ \cdots \circ r_{k-1} \sqsubseteq u,\ u \circ r_k \sqsubseteq s$
NF2	$C_1 \sqcap \ldots \sqcap \hat{C} \sqcap \ldots \sqcap C_n \sqsubseteq D$	$\rightsquigarrow$	$\hat{C} \sqsubseteq A, C_1 \sqcap \ldots \sqcap A \sqcap \ldots \sqcap C_n \sqsubseteq D$
NF3	$\exists r.\hat{C} \sqsubseteq D$	$\rightsquigarrow$	$\hat{C} \sqsubseteq A, \exists r.A \sqsubseteq D$
NF4	$\hat{C} \sqsubseteq \hat{D}$	$\rightsquigarrow$	$\hat{C} \sqsubseteq A, A \sqsubseteq \hat{D}$
NF5	$B \sqsubseteq \exists r.\hat{C}$	$\rightsquigarrow$	$B \sqsubseteq \exists r.A, A \sqsubseteq \hat{C}$
NF6	$B \sqsubseteq C \sqcap D$	$\rightsquigarrow$	$B \sqsubseteq C, B \sqsubseteq D$

where $\hat{C}, \hat{D} \notin \mathsf{CN}_{\mathcal{O}}^{\top}$, C_i, C, D are arbitrary concept descriptions, $B \in \mathsf{CN}_{\mathcal{O}}^{\top}$, u denotes a new role name, and A denotes a new concept name.

Figure 2. Normalization rules

Proof sketch: An $\mathcal{EL}^+$ ontology $\mathcal{O}$ can be converted into normal form using the rewrite rules shown in Figure 2 in two phases:

1. exhaustively apply rules **NF1** to **NF3**;

2. exhaustively apply rules **NF4** to **NF6**.

Here, "rule application" means that the axiom on the left-hand side is replaced with the axioms on the right-hand side. Note that the concepts $\hat{C}$ and $\hat{D}$ denote complex concepts, i.e., concepts that are not simply a concept name or the top concept. It is easy to see that this normalization generates a conservative extension $\mathcal{O}'$ of the original ontology $\mathcal{O}$ in the sense that a subsumption relationship between concept names occurring in $\mathcal{O}$ holds w.r.t. $\mathcal{O}'$ iff it holds w.r.t. $\mathcal{O}$. In addition, it is not difficult to verify that the size of the normalized ontology $\mathcal{O}'$ is linear in $|\mathcal{O}|$, and that the computation needs only linear time. A detailed proof can be found in (Suntisrivaraporn, 2005). □

Note that the normal form introduced here slightly differs from the one presented in (Baader et al., 2005; Suntisrivaraporn, 2005) in that it admits n-ary conjunction of concept names on the left-hand side of GCIs rather than only binary conjunction. The reason for this modification is to avoid the introduction of $n-2$ new concept names for each n-ary conjunction of concept names during normalization, and thus to decrease the number of concept names in the normalized ontology. This decrease can be quite large: for example, the normalized version of the ontology SNOMED [2] (which before normalization contains 379,691 concept names) contains 401,830 additional new concept names when

[2] SNOMED is discussed in more detail in Section 4.

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 $S(X) := S(X) \cup \{B\}$

R2 If $A \in S(X)$, $A \sqsubseteq \exists r.B \in \mathcal{O}$, and $(X, B) \notin R(r)$
then $R(r) := R(r) \cup \{(X, B)\}$

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 $S(X) := S(X) \cup \{B\}$

R4 If $(X, Y) \in R(r)$, $r \sqsubseteq s \in \mathcal{O}$, and $(X, Y) \notin R(s)$
then $R(s) := R(s) \cup \{(X, Y)\}$

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 3. Completion rules

the original normal form is used, and "only" 114,658 new concept names with the modified one introduced above.

3.2. The Abstract Algorithm

In 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 $\mathcal{O}$. The algorithm computes

- a mapping S assigning to each element A of $\mathsf{CN}_\mathcal{O}$ a subset $S(A)$ of $\mathsf{CN}^\top_\mathcal{O}$, and

- a mapping R assigning to each element r of $\mathsf{RN}_\mathcal{O}$ a binary relation $R(r)$ on $\mathsf{CN}^\top_\mathcal{O}$.

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 $A \in \mathsf{CN}_\mathcal{O}$ 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 Figure 3 until no more rule applies. The algorithm has been shown to be sound and complete in (Baader et al., 2005) in the sense that, after termination, we have $B \in S(A)$ iff $A \sqsubseteq_\mathcal{O} B$ (for all $A, B \in \mathsf{CN}^\top_\mathcal{O}$). It has also been proved that the algorithm always terminates in time polynomial in the size of the input ontology.

3.3. The Refined Algorithm

One of the main problems to be solved when implementing the described algorithm is to develop a good approach for finding the next completion rule to be applied. If this is realized by a naïve brute-force search, 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 (Dowling and Gallier, 1984). 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

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

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 $B_1 \sqcap \cdots \sqcap B_n \to B'$ as B'. Intuitively,

- an entry $B_1 \sqcap \cdots \sqcap B_n \to B'$ 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
$$A_1 \sqcap \cdots \sqcap A_{i-1} \sqcap A_{i+1} \sqcap \cdots \sqcap A_n \to B \in \widehat{\mathcal{O}}(A);$$
- 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 $B \in \widehat{\mathcal{O}}(\exists r.A)$.

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

```
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
(P1)                S(A) := S(A) ∪ {B};
(Q1)                queue(A) := queue(A) ∪ Ô(B);
                    for all concept names A' and role names r
                            with (A', A) ∈ R(r) do
(Q2)                    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
(P2)          R(s) := R(s) ∪ {(A, B)};
(Q3)          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 4. Processing the queue entries

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 4. To be more precise, process(A, X) is called when the queue of A was non-empty and we fetched the queue entry X from 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 process-new-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 role hierarchy axioms 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.

Theorem 2. *The refined algorithm runs in polynomial time, and it is sound and complete, i.e., after it terminates on input $\mathcal{O}$ we have, for all $A, B \in \mathsf{CN}_{\mathcal{O}}^{\top}$, that $B \in S(A)$ iff $A \sqsubseteq_{\mathcal{O}} B$.*

The proof of this theorem can be found in Appendix A.

4. Implementation and Evaluation

Modern DL reasoners are usually based on tableau-based subsumption algorithms for expressive DLs (Baader and Sattler, 2001). Although such algorithms are (at least) 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 (Patel-Schneider and Swartout, 1993). For details about using the system, we refer to the CEL manual. Together with the reasoner, it is available for download at

http://lat.inf.tu-dresden.de/systems/cel/.

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 (The Gene Ontology Consortium, 2000), the Galen medical knowledge base (Rector and Horrocks, 1997), and the Systematized Nomenclature of Medicine Snomed (Cote et al., 1993; Spackman, 2000).

Go. The Gene Ontology provides a controlled vocabulary to describe genes and gene products in any organism. It currently consists of 20,465 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 that plays 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 $\mathcal{O}^{\mathrm{Go}}$, 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.

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 (Horrocks, 1997). 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 $\mathcal{O}^{\mathrm{Galen}}$ 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. This acronym stands for 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 $\mathcal{O}^{\mathrm{Snomed}}$. 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, but dropping both the primitive concept definitions and the role inclusions. We call the resulting ontology $\mathcal{O}^{\mathrm{Snomed}}_{\mathsf{core}}$.

The most important facts about our benchmark ontologies are summarized in the upper part of Table II, where the first line gives the number of non-primitive concept definitions, the second line the number of primitive concept definitions, and the third line the number of GCIs

[3] Actually, Snomed needs transitive roles, but since the Apelon reasoner used by the developers cannot handle transitivity, they have simulated it in an incomplete way using the approach described in (Schulz et al., 1998).

Table II. Benchmarks and Evaluation Results (runtime given in seconds)

	$\mathcal{O}^{\textsc{Go}}$	$\mathcal{O}^{\textsc{Galen}}$	$\mathcal{O}^{\textsc{Snomed}}_{\mathsf{core}}$	$\mathcal{O}^{\textsc{Snomed}}$
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*

that are not a concept definition. Lines five and six give the number of concept names and role names, respectively. All these figures concern the ontologies prior to normalization.

We have compared the performance of CEL on these ontologies with that of the three most advanced tableau-based reasoning systems: FaCT^{++} (v1.1.0), RacerMaster (v1.9.0), and Pellet (v1.3b). All these systems implement tableau algorithms for expressive DLs in which subsumption is ExpTime-hard. The experiments have been performed on a PC with 2.8GHz Intel Pentium 4 processor and 512MB memory running Linux v2.6.14. 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 such axioms. The results of our experiments are summarized in the lower part of Table II, where all classification times are shown in seconds and *unattainable* means the reasoner failed due to memory exhaustion.

We would like to highlight the following outcomes of our experimental results:

1. CEL outperforms all the reasoners in all benchmarks (except for the case of $\mathcal{O}^{\text{GALEN}}$, where CEL and RacerMaster show the same performance);

2. CEL and FaCT^{++} are the only reasoners that can classify $\mathcal{O}^{\text{SNOMED}}$, with CEL being more than twice as fast as FaCT^{++}. In contrast, RacerMaster and Pellet fail to classify this ontology. Pellet and the original version of FaCT (not shown in the table) even fail to classify $\mathcal{O}^{\text{SNOMED}}_{\text{core}}$.

Regarding the second point, it is interesting to observe that $\mathcal{O}^{\text{SNOMED}}$ is only an acyclic TBox and, in particular, contains no real GCIs. The relatively good performance of FaCT^{++} on this ontology seems to be due to this fact: if we extend $\mathcal{O}^{\text{SNOMED}}$ with additional GCIs of the simple form $\exists r.A \sqsubseteq B$, where A and B are concept names randomly chosen from $\mathcal{O}^{\text{SNOMED}}$ and r is a role name randomly chosen from $\mathcal{O}^{\text{SNOMED}}$, then FaCT^{++} needs about 3,000 more seconds to classify $\mathcal{O}^{\text{SNOMED}}$ *for each additional GCI*. In contrast, the performance of CEL does not change noticeably if we add such GCIs. We view this as an indication that CEL's computational behavior is more robust than that of FaCT^{++}.

5. 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 (Baader et al., 1994) 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 $DS(A) := SS(A) \setminus (\bigcup_{B \in SS(A)} SS(B))$ 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 (Baader et al., 1994). 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 5 shows a pseudo code representation of our algorithm. The sets $\mathsf{parents}(A)$ are used to store the direct subsumers of A, the set $\mathsf{children}(A)$ 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 (Baader et al., 1994) using told subsumer information and some of the optimizations described in (Horrocks and Tsarkov, 2005) 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 II, 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 (Baader et al., 1994). 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 computed parts of the mappings $S(\cdot)$ and $R(\cdot)$ for such repeated calls.

procedure $\mathsf{compute\text{-}dag}$
    for all concept names $X \in \mathsf{CN}_{\mathcal{O}}^{\top}$ **do**
        $\mathsf{classified}(X) :=$ **false**
        $\mathsf{parents}(X) := \mathsf{children}(X) := \mathsf{equivalents}(A) := \emptyset$
    for each concept name $A \in \mathsf{CN}_{\mathcal{O}}^{\top}$ **do**
        if not $\mathsf{classified}(A)$ **then**
            $\mathsf{dag\text{-}classify}(A)$;
end;

procedure $\mathsf{dag\text{-}classify}(A)$
    $\mathsf{candidates} := \emptyset$;
    for all subsumers $B \in S(A)$ **do**
        if $A \in S(B)$ **then**
            $\mathsf{classified}(B) :=$ **true**;
            $\mathsf{equivalents}(A) := \mathsf{equivalents}(A) \cup \{B\}$;
        else
            if not $\mathsf{classified}(B)$ **then**
                $\mathsf{dag\text{-}classify}(B)$;
            $\mathsf{candidates} := \mathsf{candidates} \cup \{B\}$;
    $\mathsf{dag\text{-}insert}(A, \mathsf{candidates})$;
    $\mathsf{classified}(B) :=$ **true**;
end;

procedure $\mathsf{dag\text{-}insert}(A, \mathsf{candidates})$
    $\mathsf{marked}(X) :=$ **false** for all $X \in \mathsf{CN}_{\mathcal{O}}^{\top}$;
    for all $B \in \mathsf{candidates}$ **do**
        for all $X \in \mathsf{parents}(B)$ **do**
            $\mathsf{marked}(X) :=$ **true**
    while there are $X, Y \in \mathsf{CN}_{\mathcal{O}}^{\top}$ with $\mathsf{marked}(X)$, $Y \in \mathsf{parents}(X)$, and
          not $\mathsf{marked}(Y)$ **do**
        $\mathsf{marked}(Y) :=$ **true**
    $\mathsf{parents}(A) := \{B \in \mathsf{candidates} \mid \mathsf{marked}(B) = \textbf{false}\}$;
    for all $B \in \mathsf{parents}(A)$ **do**
        $\mathsf{children}(B) := \mathsf{children}(B) \cup \{A\}$;
end;

Figure 5. Computing the DAG from the subsumer sets

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).

6. 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 $\mathcal{EL}^{++}$ (Baader et al., 2005), with which one can express, among other things, *nominals* and *disjoint concepts*. We also plan to implement DIG and OWL interfaces,[4] so that CEL can be used as a backend reasoner for ontology editors like OilEd[5] and Protègè,[6] which would also make their more sophisticated graphical user-interfaces available to users of CEL.

References

Baader, F.: 2003, 'Terminological Cycles in a Description Logic with Existential Restrictions'. In: G. Gottlob and T. Walsh (eds.): *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. pp. 325–330.

Baader, F., S. Brandt, and C. Lutz: 2005, 'Pushing the $\mathcal{EL}$ Envelope'. In: *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*. Edinburgh, UK.

Baader, F., E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich: 1994, 'An Empirical Analysis of Optimization Techniques for Terminological Representation

[4] See http://dl.kr.org/dig/ and http://www.w3.org/2004/OWL/
[5] see http://oiled.man.ac.uk/
[6] See http://protege.stanford.edu/

Systems or: Making KRIS get a move on'. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management* **4**, 109–132.

Baader, F. and U. Sattler: 2001, 'An Overview of Tableau Algorithms for Description Logics'. *Studia Logica* **69**, 5–40.

Brachman, R. J. and H. J. Levesque: 1984, 'The Tractability of Subsumption in Frame-Based Description Languages'. In: *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI'84)*. pp. 34–37.

Brandt, S.: 2004, 'Polynomial Time Reasoning in a Description Logic with Existential Restrictions, GCI Axioms, and—What Else?'. In: R. L. de Mantáras and L. Saitta (eds.): *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*. pp. 298–302.

Cote, R., D. Rothwell, J. Palotay, R. Beckett, and L. Brochu: 1993, 'The Systematized Nomenclature of Human and Veterinary Medicine'. Technical report, SNOMED International, Northfield, IL: College of American Pathologists.

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

Haarslev, V. and R. Möller: 2001, 'RACER System Description'. In: *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*.

Henzinger, M. R., T. A. Henzinge, and P. W. Kopke: 1995, 'Computing simulations on finite and infinite graphs'. In: *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*. pp. 453–462.

Horrocks, I.: 1997, 'Optimising Tableaux Decision Procedures for Description Logics'. Ph.D. thesis, University of Manchester.

Horrocks, I.: 1998, 'Using an Expressive Description Logic: FaCT or Fiction?'. In: *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*. pp. 636–647.

Horrocks, I.: 2003, 'Implementation and Optimization Techniques'. In: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider (eds.): *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, pp. 306–346.

Horrocks, I., P. F. Patel-Schneider, and F. van Harmelen: 2003, 'From SHIQ and RDF to OWL: The Making of a Web Ontology Language'. *Journal of Web Semantics* **1**(1), 7–26.

Horrocks, I., U. Sattler, and S. Tobies: 2000, 'Practical Reasoning for Very Expressive Description Logics'. *J. of the Interest Group in Pure and Applied Logic* **8**(3), 239–264.

Horrocks, I. and D. Tsarkov: 2005, 'Optimised Classification for Taxonomic Knowledge Bases'. In: *Proceedings of the 2005 International Workshop on Description Logics (DL'05)*. pp. 184–191.

Nebel, B.: 1988, 'Computational Complexity of Terminological Reasoning in BACK'. *Artificial Intelligence* **34**(3), 371–383.

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

Patel-Schneider, P. and B. Swartout: 1993, '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.

Rector, A. and I. Horrocks: 1997, '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.

Schulz, S., M. Romacker, and U. Hahn: 1998, 'Part-Whole Reasoning in Medical Ontologies Revisited: Introducing SEP Triplets into Classification-Based Description Logics'. *Journal of the American Medical Informatics Association (JAMIA)* pp. 830–834.

Spackman, K.: 2000, 'Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT'. *Journal of the American Medical Informatics Association (JAMIA).* Fall Symposium Special Issue.

Suntisrivaraporn, B.: 2005, '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.

The Gene Ontology Consortium: 2000, 'Gene Ontology: Tool for the Unification of Biology'. *Nature Genetics* **25**, 25–29.

Appendix

A. Correctness Proofs of the Refined Algorithm

Theorem 2 is an immediate consequence of the following three lemmas.

Lemma 3 (Soundness). *After the refined algorithm terminates on an ontology $\mathcal{O}$, the following holds: if $B \in S(A)$, then $A \sqsubseteq_{\mathcal{O}} B$.*

Proof. We introduce a number of invariants on the three interdependent data structures of the refined algorithm and then show that the invariants hold throughout the computation.

INV1 If $B \in S(A)$, then $A \sqsubseteq_{\mathcal{O}} B$.

INV2 If $(A, B) \in R(r)$, then $A \sqsubseteq_{\mathcal{O}} \exists r.B$

INV3 If $\exists r.B \in \mathsf{queue}(A)$, then $A \sqsubseteq_{\mathcal{O}} \exists r.B$

INV4 If $B_1, \ldots, B_n \rightarrow B \in \mathsf{queue}(A)$, then

$$A \sqsubseteq_{\mathcal{O}} B_1 \wedge \ldots \wedge A \sqsubseteq_{\mathcal{O}} B_n \text{ implies } A \sqsubseteq_{\mathcal{O}} B.$$

Observe that, in case that $n = 0$, **INV4** degenerates to "if $B \in \mathsf{queue}(A)$, then $A \sqsubseteq_{\mathcal{O}} B$". Also note that satisfaction of **INV1** throughout (and after) the computation implies Lemma 3.

We start with showing that the invariants hold after the initialization of the refined algorithm. Since the sets $S(A)$ is initialized with $\{A, \top\}$ and the sets $R(A)$ with $\emptyset$, **INV1** and **INV2** are clearly satisfied. Now suppose that $\exists r.B \in \mathsf{queue}(A)$. Since $\mathsf{queue}(A)$ is initialized as $\widehat{\mathcal{O}}(A)$, the GCI $A \sqsubseteq \exists r.B$ belongs to $\mathcal{O}$. This implies that $A \sqsubseteq_{\mathcal{O}} \exists r.B$ and thus **INV3** is satisfied. For **INV4**, assume that $B_1, \ldots, B_n \rightarrow B \in \mathsf{queue}(A) = \widehat{\mathcal{O}}(A)$. Then there exists (up to commutativity of $\sqcap$) a GCI $B_1 \sqcap \cdots \sqcap B_n \sqcap A \sqsubseteq B$ in $\mathcal{O}$. Thus, **INV4** is satisfied.

After the initialization, the data structures are manipulated by the procedures process and process-new-edge in the lines marked **P1**, **P2**, **Q1**, **Q2**, and **Q3** in Figure 4. We show that each manipulation preserves the invariants.

P1 This line adds B to $S(A)$ when $\mathsf{process}(A, B_1, \ldots, B_n \rightarrow B)$ is invoked and we have $\{B_1, \ldots, B_n\} \subseteq S(A)$. The former means that $B_1, \ldots, B_n \rightarrow B$ was in the queue of A. By Invariant **INV4**, we have that $A \sqsubseteq_{\mathcal{O}} B_1 \wedge \ldots \wedge A \sqsubseteq_{\mathcal{O}} B_n$ implies $A \sqsubseteq_{\mathcal{O}} B$. Because of $\{B_1, \ldots, B_n\} \subseteq S(A)$ and **INV1**, this yields $A \sqsubseteq_{\mathcal{O}} B$. Thus, **INV1** (the only invariant that could be invalidated by **P1**) is preserved.

P2 This line adds (A, B) to $R(s)$ when $\mathsf{process\text{-}new\text{-}edge}(A, r, B)$ is invoked and $r \sqsubseteq^*_{\mathcal{O}} s$. We make a case distinction according to the three possible reasons for which process-new-edge can be invoked.

First, assume that $\mathsf{process\text{-}new\text{-}edge}(A, r, B)$ is called by the process procedure. Then $\exists r.B$ was in the queue of A and **INV4** yields $A \sqsubseteq_{\mathcal{O}} \exists r.B$. Since $r \sqsubseteq^*_{\mathcal{O}} s$, this implies $A \sqsubseteq_{\mathcal{O}} \exists s.B$ and **INV2** is preserved.

Second, assume that $\mathsf{process\text{-}new\text{-}edge}(A, r, B)$ is called recursively by process-new-edge in the first inner **for** loop. Then there are role names t, u and a concept name A' such that $t \circ u \sqsubseteq r \in \mathcal{O}$, $(A, A') \in R(t)$ and $(A', B) \in R(u)$. By **INV2**, we have $A \sqsubseteq_{\mathcal{O}} \exists t.A'$ and $A' \sqsubseteq_{\mathcal{O}} \exists u.B$. Together with $t \circ u \sqsubseteq r \in \mathcal{O}$ and $r \sqsubseteq^*_{\mathcal{O}} s$, we get $A \sqsubseteq_{\mathcal{O}} \exists s.B$ and thus **INV2** is preserved.

Third, assume that $\mathsf{process\text{-}new\text{-}edge}(A, r, B)$ is called recursively by process-new-edge in the second inner **for** loop. Then there are role names t, u and a concept name B' such that $u \circ t \sqsubseteq r$, $(B', B) \in R(t)$, and $(A, B') \in R(u)$. We can continue in a similar fashion to the previous case.

Q1 This line adds all the elements of $\widehat{\mathcal{O}}(B)$ to $\mathsf{queue}(A)$ when process is invoked with arguments $(A, B_1, \ldots, B_n \rightarrow B)$. We have that $\{B_1, \ldots, B_n\} \subseteq S(A)$, and B has been added to $S(A)$ in line **P1**. By **INV1**, the latter implies $A \sqsubseteq_{\mathcal{O}} B$. Thus, we may argue as for the initialization step that **INV3** and **INV4** are preserved.

Q2 This line adds all the elements of $\widehat{\mathcal{O}}(\exists r.B)$ to $\mathsf{queue}(A')$ only if $B \in S(A)$ and $(A', A) \in R(r)$ for some concept name A' and role name r. By Invariants **INV1** and **INV2**, we have $A \sqsubseteq_{\mathcal{O}} B$ and $A' \sqsubseteq_{\mathcal{O}} \exists r.A$, implying $A' \sqsubseteq_{\mathcal{O}} \exists r.B$. The existence of an element $B' \in \widehat{\mathcal{O}}(\exists r.B)$ implies that the GCI $\exists r.B \sqsubseteq B'$ belongs to $\mathcal{O}$. Hence, we have $A' \sqsubseteq_{\mathcal{O}} B'$ and adding B' to $\mathsf{queue}(A')$ preserves (the $n = 0$ case of) **INV4**.

Q3 This line adds the elements of $\widehat{\mathcal{O}}(\exists s.B')$ to $\mathsf{queue}(A)$ only if $(A, B) \in R(s)$ and $B' \in S(B)$. With the same arguments as that in the previous case, we have $A \sqsubseteq_{\mathcal{O}} \exists s.B \sqsubseteq_{\mathcal{O}} \exists s.B' \sqsubseteq_{\mathcal{O}} B''$ for all $B'' \in \widehat{\mathcal{O}}(\exists s.B')$. Thus, **INV4** is preserved.

$\square$

Lemma 4 (Completeness). *If $A \sqsubseteq_{\mathcal{O}} B$ for some $A, B \in \mathsf{CN}^{\top}_{\mathcal{O}}$, then we have $B \in S(A)$ after termination of the refined algorithm.*

Proof. Instead of proving completeness of the refined algorithm from scratch, we want to use the completeness proof for the abstract algorithm given in (Baader et al., 2005). This completeness proof shows the following: if $\mathcal{O}$ is an ontology and S, R are mappings such that

– S assigning to each element A of $\mathsf{CN}^{\top}_{\mathcal{O}}$ a subset $S(A)$ of $\mathsf{CN}^{\top}_{\mathcal{O}}$,

– R assigning to each element r of $\mathsf{RN}_{\mathcal{O}}$ a binary relation $R(r)$ on $\mathsf{CN}^{\top}_{\mathcal{O}}$,

– $S(A)$ contains $\top$ and A for all $A \in \mathsf{CN}^{\top}_{\mathcal{O}}$,

– the completion rules of Figure 3 do not apply to $S, R, \mathcal{O}$,

then $A \sqsubseteq_{\mathcal{O}} B$ implies $B \in S(A)$.

Since the our refined algorithm initializes he sets $S(A)$ with $\{A, \top\}$ and never removes elements from $S(A)$, it is thus sufficient to show that, after termination of the refined algorithm, none of the completion rules is applicable. Assume, to the contrary of what has to be proved, that there exists an applicable completion rule after termination of the refined We make a case distinction according to the rule type, and show that in each case our assumption leads to a contradiction.

R1 If this rule is applicable, then there exists an $X \in \mathsf{CN}^{\top}_{\mathcal{O}}$ and an $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$ in $\mathcal{O}$ such that $A_1, \ldots, A_n \in S(X)$ and $B \notin S(X)$. Let $\ell \leq n$ be such that A_ℓ had been added to $S(X)$ most recently among the $A_1, \ldots, A_n$. When A_ℓ was added, the entry $A_1, \ldots, A_{\ell-1}, A_{\ell+1}, \ldots, A_n \to B \in \widehat{\mathcal{O}}(A_\ell)$ was added to $\mathsf{queue}(X)$, due to the presence of the GCI $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$ in $\mathcal{O}$. Since $B \notin S(X)$, the conditional entry has not yet been processed w.r.t. X, implying that $\mathsf{queue}(X)$ is non-empty, which is a contradiction to our assumption that the algorithm has terminated.

R2 If this rule is applicable, there exists an $X \in \mathsf{CN}^{\top}_{\mathcal{O}}$ and a GCI $A \sqsubseteq \exists r.B$ in $\mathcal{O}$ such that $A \in S(X)$ and $(X, B) \notin R(r)$. When A was added to $S(X)$, the entry $\exists r.B \in \widehat{\mathcal{O}}(A)$ was added to $\mathsf{queue}(X)$. However, $(X, B) \notin R(r)$ means that this entry has not yet been processed, and thus $\mathsf{queue}(X)$ is non-empty.

R3 If this rule is applicable, there are $X, Y \in \mathsf{CN}^{\top}_{\mathcal{O}}$ and a GCI $\exists r.A \sqsubseteq B$ in $\mathcal{O}$ such that $(X, Y) \in R(r)$, $A \in S(Y)$, and $B \notin S(X)$. We distinguish the following two cases:

– (X, Y) had been added to $R(r)$ before A was added to $S(Y)$. Then, the entry $B \in \widehat{\mathcal{O}}(\exists r.A)$ was added to $\mathsf{queue}(X)$ when A was added to $S(Y)$. However, $B \notin S(X)$ means that this entry was not yet processed, implying the queue of X to be non-empty.

– A had been added to $S(Y)$ before (X,Y) was added to $R(r)$. Then, the entry $B \in \widehat{\mathcal{O}}(\exists r.A)$ was added to $\mathsf{queue}(X)$ when (X,Y) was added to $R(r)$, and we can continue as in the previous case.

R4 If this rule is applicable, there are $X, Y \in \mathsf{CN}_{\mathcal{O}}^{\top}$ and an $r \sqsubseteq s$ in $\mathcal{O}$ such that $(X,Y) \in R(r) \setminus R(s)$. The addition of (X,Y) to $R(r)$ took place in the process-new-edge procedure that was started with arguments (X,t,Y) for some t such that $t \sqsubseteq_{\mathcal{O}}^{*} r$. Since $r \sqsubseteq s$ in $\mathcal{O}$, we have $t \sqsubseteq_{\mathcal{O}}^{*} s$, and (X,Y) is added to $R(s)$ in the same call to process-new-edge. We derive a contradiction to the applicability of **R4**.

R5 If this rule is applicable, there are $X, Y, Z \in \mathsf{CN}_{\mathcal{O}}^{\top}$ and a role inclusion axiom $r \circ s \sqsubseteq t$ in $\mathcal{O}$ such that $(X,Y) \in R(r)$, $(Y,Z) \in R(s)$, and $(X,Z) \notin R(t)$. We distinguish two cases:

– (X,Y) had been added to $R(r)$ before (Y,Z) was added to $R(s)$. The pair (Y,Z) is added to $R(s)$ in the procedure process-new-edge. Then the first inner **for** loop and the recursive call to process-new-edge inside this loop ensure that (X,Z) is added to $R(t)$. This is a contradiction to the fact that $(X,Z) \notin R(t)$.

– (Y,Z) had been added to $R(s)$ before (X,Y) was added to $R(r)$. This case is analogous to the previous one with the second inner **for** loop in place of the first.

□

If $\mathcal{O}$ is an ontology, we use $|\mathcal{O}|$ to denote its *size*, i.e., the number of symbols needed to write it.

Lemma 5 (Termination). *If the refined algorithm is applied to an ontology $\mathcal{O}$ with $|\mathcal{O}| = n$, then it terminates after $O(n^4)$ additions to the data structures $S(\cdot)$, $R(\cdot)$, and* $\mathsf{queue}(\cdot)$.

Proof. To show termination, it suffices to show that there are at most n^4 additions to the data structure since every infinite run of the algorithm must clearly make infinitely many additions to the data structures. For the sets S, R, every element can be added at most once and no element is deleted. In the case of the $\mathsf{queue}(\cdot)$ structure, however, entries can be deleted, and the same entry can be added to a specific queue several times.

We start with additions to the sets $S(\cdot)$. Since there are at most n such sets, each set can contain at most n elements, and elements are never deleted, there can be at most n^2 additions. Analogously, we can show that there can be at most n^3 additions for the sets $R(\cdot)$. Now, consider additions to $\mathsf{queue}(\cdot)$. These are only made together with an addition to $S(\cdot)$ or $R(\cdot)$. More precisely, each single addition to $S(\cdot)$ comes together with at most n^2 additions to $\mathsf{queue}(\cdot)$: the algorithm makes additions to at most number of concept names many queues (which is bounded by n), and for each such queue a linear number of additions is made. Each single addition to $R(\cdot)$ comes together with at most n additions to $\mathsf{queue}(\cdot)$: only one queue is extended, and again the number of items added is linear in n.

Overall, this implies the bound of $O(n^4)$ for the total number of additions. $\square$