In the Search of Improvements to the $\mathcal{EL}^+$ Classification Algorithm

Barış Sertkaya

sertkaya.baris@googlemail.com

Abstract. We investigate possible improvements to the existing algorithm for classifying $\mathcal{EL}^+$ TBoxes. We present a modified algorithm based on the well-known linear closure algorithm from relational databases. Despite its better worst-case complexity, surprisingly it turns out that this algorithm does not perform well in practice. We discuss optimizations to the existing algorithm and evaluate them using our prototypical reasoner CHEETAH on several large bio-medical knowledge bases.

1 Introduction

In [9, 8] Brandt has shown that the tractability result in [2] for subsumption w.r.t. cyclic $\mathcal{EL}$ TBoxes can be extended to the DL $\mathcal{ELH}$, which in addition to $\mathcal{EL}$ allows for general concept inclusion axioms and role hierarchies. Later in [3] Baader et. al. have shown that the tractability result can even be further extended to the DL $\mathcal{EL}^{++}$ which in addition to $\mathcal{ELH}$ allows for the bottom concept, nominals, role inclusion axioms, and a restricted form of concrete domains. In addition to these promising theoretical results, it turned out that despite their relatively low expressivity, these fragments are still expressive enough for the well-known bio-medical knowledge bases SNOMED [12] and (large parts of) Galen [19], and the Gene Ontology GO [11]. In [4, 6, 20] the practical usability of these fragments on large knowledge bases has been investigated. The CEL Reasoner [18] was as a result of these studies the first reasoner that could classify the mentioned knowledge bases from life sciences domain in reasonable times.

Successful applications of the $\mathcal{EL}$ family increased investment of further work in this direction. The $\mathcal{EL}$ family now provides the basis for the profile OWL2 EL[1]. Moreover, there are now a few other reasoners specifically tailored for the $\mathcal{EL}$ family, like Snorocket [16] and TrOWL [21], and CB [15], which extends the $\mathcal{EL}^{++}$ algorithm to Horn $\mathcal{SHIQ}$. A comprehensive study comparing the performace of several reasoners on large bio-medical knowledge bases has been presented in [13].

In the present work we investigate possible improvements to the existing classification algorithm for the $\mathcal{EL}$ family. We present a modified algorithm based on the well-known linear closure algorithm [7] from relational databases [17]. We evaluate both the modified algorithm and the implementation of the simple

[1] http://www.w3.org/TR/owl2-profiles/#OWL_2_EL

algorithm in our prototypical $\mathcal{EL}^+$ reasoner CHEETAH on several large knowledge bases from life sciences. Surprisingly, it turns out that despite its better worst-case complexity, the modified algorithm performs worse than the simple algorithm in practice. In Section 2 we introduce the linear closure algorithm from relational databases. In Section 3 we present our modified algorithm based on linear closure, and in Section 4 we present our experimental results.

2 Computing Closure Under Functional Dependencies

In relational databases [17], specification of constraints on data is of crucial importance for correct modelling of the world and correct design of the database schemas. One way of specifying constraints is using functional dependencies introduced in [10]. A functional dependency occurs when the values of a tuple on one set of attributes uniquely determine the values on another set of attributes. Formally, given a relation r and a set of attribute names R, a *functional dependency* (FD) is a pair of sets $X, Y \subseteq R$ written as $X \to Y$. The relation r *satisfies* the FD $X \to Y$ if the tuples with equal X-values also have equal Y-values. In this case, one says that the set of attributes X *functionally determine* the set of attributes Y.

Given a set of FDs $\mathcal{F}$ and an FD $X \to Y$, one interesting question is whether $\mathcal{F}$ *implies* $X \to Y$, i.e., whether every relation that satisfies all FDs in $\mathcal{F}$ also satisfy $X \to Y$, which we denote as $\mathcal{F} \models X \to Y$. In order to answer this, one can compute the smallest set of all FDs that $\mathcal{F}$ implies by using a set of inference axioms called *Armstrong's axioms* [1] and check whether the mentioned FD is an element of this set. However, the set of FDs that $\mathcal{F}$ implies can be considerably larger than $\mathcal{F}$ and costly to compute. Thus one is interested in answering this question without computing this set. Instead, one computes the so-called closure of X under $\mathcal{F}$ and checks whether it contains Y. The *closure* of a set of attributes $X \subseteq R$ under a set of FDs $\mathcal{F}$ is the smallest subset X^+ of R such that $X \subseteq X^+$ and for every $A \to B \in \mathcal{F}$ if $A \subseteq X^+$ holds, then $B \subseteq X^+$ holds as well. [2]

2.1 The Linear Closure Algorithm

In [7] Beeri and Bernstein have given an algorithm for efficiently computing closure under a set of FDs. Briefly, for each attribute the algorithm keeps an index pointing to the set of FDs whose left handsides contain that attribute. Additionally, for each FD it keeps a counter whose value is initally the size of the left handside of that FD. Initialization of these data structures is shown in the procedure `Initialization` in Algorithm 1.

For computing the closure of a set of attributes `x` under a set of FDs `F` it keeps a queue `update` which is initally equal to `x`. In the procedure `Closure` it repeatedly fetches and removes an attribute from `update` and decrements the

[2] Note that, from the viewpoint of logic, computing closure is computing consequences in propositional Horn logic. In fact, the notions we have defined can easily be reformulated in propositional logic when we view the attributes as propositional variables.

Algorithm 1 The Linear Closure Algorithm

```
Procedure: Initialization
 1: for all FD $W \to Z \in F$ do
 2:     $count[W \to Z] := |W|$
 3:     for all attribute $A \in W$ do
 4:         $list[A] := list[A] \cup \{W \to Z\}$
 5:     end for
 6: end for
 7: $newdep := update := x$
Procedure: Closure
 1: while $update \neq \emptyset$ do
 2:     choose an $A$ from $update$
 3:     $update := update \setminus \{A\}$
 4:     for all FD $W \to Z \in list[A]$ do
 5:         $count[W \to Z] := count[W \to Z] - 1$
 6:         if $count[W \to Z] = 0$ then
 7:             $add := Z \setminus newdep$
 8:             $newdep := newdep \cup add$
 9:             $update := update \cup add$
10:         end if
11:     end for
12: end while
    return $newdep$
```

counters of FDs that contain this attribute in the left handside. Once a counter becomes zero, it extends the queue `update` and the closure `newdep` with the new attributes on the right handside of that FD. This continues until the queue `update` becomes empty.

Note that the initialization takes at most $|F|.|W|$ time, which is linear in the size of the input. Now consider the closure computation: Each attribute can enter `update` at most once. For each attribute `A` fetched from `update` the counters of the FDs in `list[A]` are decremented, which is performed at most $\Sigma_{W \to Z \in F}|W|$ times. If the counter of any FD $W \to Z$ becomes 0, then the new attributes in `Z` are added to `update` and `newdep`. If the involved sets are represented as bit vectors, this operation takes time proportional to $\Sigma_{W \to Z \in F}|Z|$.

Since all steps of the algorithm can be performed in time linear in the sizes of FDs `F` and the set of attributes, the algorithm has complexity $O(n)$.

3 A Modified Algorithm for classifying $\mathcal{EL}^+$ TBoxes

In the present section we present an $\mathcal{EL}^+$ classification algorithm based on the linear closure algorithm introduced in the previous section. $\mathcal{EL}^+$ is the DL allowing for the top concept $\top$, conjunction $C \sqcap D$, existential restriction $\exists r.A$, general concept inclusion axioms (GCIs) $C \sqsubseteq D$ and role inclusion axioms (RIs) $r_1 \circ \cdots \circ r_n \sqsubseteq s$, where A is an atomic concept, r an atomic role, and C, D

concept descriptions. RIs are interpreted as $r_1^\mathcal{I} \circ \cdots \circ r_n^\mathcal{I} \subseteq s^\mathcal{I}$, where $\circ$ denotes composition of binary relations.

In [9, 8] Brandt has shown that the tractability result in [2] for subsumption w.r.t. cyclic TBoxes can be extended to the DL $\mathcal{ELH}$, which in addition to $\mathcal{EL}$ allows for GCIs and simple RIs, i.e., RIs with an atomic role on the left handside. Later in [3] Baader et. al. have shown that the tractability result can even be further extended to the DL $\mathcal{EL}^{++}$ which in addition to $\mathcal{EL}^+$ allows for the bottom concept $\bot$, nominals $\{a\}$ and a restricted form of concrete domains.

In [4, 5] Baader et. al. have considered a restriction of the polynomial-time classification algorithm in [3] to $\mathcal{EL}^+$ and have given a refined version of the algorithm tailored for efficient implementations of it. This algorithm initially turns the input TBox into a normalized TBox by applying a series of normalization rules. Afterwards, it applies a set of completion rules to compute a mapping S assigning to each concept name a subset of the concept names occurring in the original TBox. The completion rules are repeatedly applied until no rule applies any more. Consequently, the mapping S maps every concept name A to its set of subsumers. That is, $B \in S(A)$ implies that the subsumption relation $A \sqsubseteq B$ holds in the original TBox. What is important here is a clever strategy for finding the next completion rule to be applied. Because if this is done by a brute-force approach, even though still polynomial, the algorithm will not perform well in practice for large real life TBoxes. In order to avoid this, the "refined" algorithm suggested in [4, 5] uses a modification of the approach used in [14] for checking satisfiability of propositional Horn formulae. As shown in [6, 20], the refined classification algorithm makes at most $O(n^4)$ additions to the mapping S and to the other data structures used.

In the following, we present an algorithm based on the linear closure algorithm introduced in the previous section. We exploit the similarity between computing closure under a set of functional dependencies and computing the set of subsumers of a concept. In its simplest form, one can view a GCI that consists of conjunctions of concept names on both sides as an FD. In this case, computing the subsumers of a concept w.r.t. a set of such GCIs trivially boils down to computing the closure of that concept under that set of GCIs, and classifying the TBox boils down to computing the *closure of every concept name* occurring in the TBox. For the general case, where TBox contains RIs, and where GCIs contain existential restrictions, the inferences due to these should of course also be taken into account.

As in the existing algorithm, we first transform the TBox into a normal form. Our normal form slightly differs from the original one introduced in [9, 3]. Instead of only binary conjunctions on the left handsides of GCIs, it allows for conjunctions of arbitary size. This kind of GCIs have already been used in the normal form in [4, 6]. There it was reported that for large knowledge bases like SNOMED [12], this minor change considerably reduces the number of newly introduced concept names, and thus reduces the size of the normalized knowledge base. Here, in addition to the left handside, we also allow conjunctions of arbitrary size on the right handside of GCIs. Of course theoretically this does not make any difference but it in the implemtation of the algorithm it allows a compact representation of the axioms.

3.1 The Normal Form

Given a TBox $\mathcal{T}$ we write $\mathrm{CN}_{\mathcal{T}}$ and $\mathrm{CN}_{\mathcal{T}}^{\top}$ to denote the sets of concept names occurring in $\mathcal{T}$ with and without the top concept, respectively. Likewise we write $\mathrm{RN}_{\mathcal{T}}$ to denote the set of role names occurring in $\mathcal{T}$. We say that $\mathcal{T}$ is in *normal form* if

1. all GCIs in $\mathcal{T}$ are of the form
$$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D_1 \sqcap \ldots \sqcap D_m$$
where C_i is either a concept name from $\mathrm{CN}_{\mathcal{T}}^{\top}$ or is of the form $\exists r.A$, and D_j is either a concept name from $\mathrm{CN}_{\mathcal{T}}$ or is of the form $\exists r.A$ where $A \in \mathrm{CN}_{\mathcal{T}}^{\top}$ and $r \in \mathrm{RN}_{\mathcal{T}}$.
2. all role inclusions are of the form $r \sqsubseteq s$ or $r_1 \circ r_2 \sqsubseteq s$ where $r_1, r_2, s \in \mathrm{RN}_{\mathcal{T}}$.

Basically, a normalized GCI consists of conjunctions on both sides where conjuncts are either concept names or existentially quantified concept names. Role inclusion axioms are normalized exactly the same way as in [4, 6]. Note that

NF1	$r_1 \circ \ldots \circ r_k \sqsubseteq s$	$\rightsquigarrow$	$r_1 \circ \ldots \circ r_{k-1} \sqsubseteq u,\ u \circ r_k \sqsubseteq s$
NF2	$C_1 \sqcap \ldots \sqcap \exists r.\hat{C} \sqcap \ldots \sqcap C_n \sqsubseteq D$	$\rightsquigarrow$	$\hat{C} \sqsubseteq A,\ C_1 \sqcap \ldots \sqcap \exists r.A \sqcap \ldots \sqcap C_n \sqsubseteq D$
NF3	$C \sqsubseteq D_1 \sqcap \ldots \sqcap \exists r.\hat{C} \sqcap \ldots \sqcap D_m$	$\rightsquigarrow$	$C \sqsubseteq D_1 \sqcap \ldots \sqcap \exists r.A \sqcap \ldots \sqcap D_m,\ A \sqsubseteq \hat{C}$

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

Fig. 1. Normalization rules

our normalization rules shown in Figure 1 are a "stripped down" version of the original normalization rules. Therefore the linear upper bound on the size of the normalized TBox shown in [4, 20] also holds for our normalization rules.

3.2 The Modified Classification Algorithm

Like the linear closure algorithm, our classification algorithm maintains a set of counters in order to decide when to apply a GCI. However we do not maintain only one counter per GCI, but for every concept name we have a counter for every GCI. This is because we want to compute the subsumer list of every concept name occurring in the input TBox, and not only one concept name. The counters initally contain the size (number of conjuncts) of the left handsides of the GCIs. For every concept name we maintain a stack that keeps track of the concepts still to be processed for that concept name. Note that our algorithm differs from the original classification algorithm in [4, 6] here in the sense that instead of keeping track of axioms to be processed we keep track of concepts to be processed. Our possible stack entries are concept names $A \in \mathrm{CN}_{\mathcal{T}}^{\top}$, or existentially restricted concept names $\exists r.A$ where $A \in \mathrm{CN}_{\mathcal{T}}^{\top}$. The counters are kept in the two-dimensional array `count[`A`][`$C \sqsubseteq D$`]` and the stacks are stored as `q(`A`)` for $A \in \mathrm{CN}_{\mathcal{T}}^{\top}$ and $C \sqsubseteq D \in \mathcal{T}$.

In addition to these, for every concept name we keep an index pointing to the list of GCIs that contain this concept name on the left handside. This information is stored in `list(`A`)` for $A \in \mathrm{CN}_{\mathcal{T}}$. As in the original algorithm we keep a subsumer list $S(B)$ for each concept name B. Unlike the original algorithm in addition to concept names this list also contains concept descriptions of the form $\exists r.A$ where $A \in \mathrm{CN}_{\mathcal{T}}^{\top}$. Therefore we do not have the data structure $R(\cdot,\cdot)$ in the original algorithm for storing relations.

Having explained the data structures we are now ready to give the algorithm. The first procedure properly initializes the data structures `count[·][·]`, `list(·)`, `q(·)`, and `S(·)`.

Algorithm 2 initialize the data structures

Procedure: Initialization

```
 1: for all GCI $\alpha = \bigsqcap\{C_1,\dots,C_n\} \sqsubseteq \bigsqcap\{D_1,\dots,D_m\} \in \mathcal{T}$ do
 2:     for all $A \in \mathrm{CN}_{\mathcal{T}}$ do
 3:         count[A][$\alpha$] = $n$
 4:     end for
 5:     for all $C \in \{C_1,\dots,C_n\}$ do
 6:         list(C) = list(C) $\cup\{\alpha\}$
 7:     end for
 8: end for
 9: for all $A \in \mathrm{CN}_{\mathcal{T}}$ do
10:     $q(A) = \{A, \top\}$
11:     $S(A) = \{A, \top\}$
12: end for
```

Next we describe the processing of the stacks. Upon popping an entry (a normalized concept description) C from $q(A)$ we call `process-concept-name` if C is a concept name, and `process-existential-restriction` if C is an existential restriction. Later we traverse the GCIs that have C on the left handside and decrement the counters for A. If the counter $count[A][\bigsqcap\{C_1,\dots,C_n\} \sqsubseteq \bigsqcap\{D_1,\dots,D_m\}]$ becomes zero, we extend $S(A)$ and $q(A)$ with the new concept descriptions in $\{D_1,\dots,D_m\}$.

A concept name fetched from the stack of A is processed as in Algorithm 4, and an existential restriction popped from the stack of A is processed as shown in Algorithm 5. Here $\sqsubseteq_{\mathcal{T}}^{*}$ denotes the reflexive transitive closure of the role hierarchy axioms in the normalized TBox. Processing of the stacks continues until

Algorithm 3 process stack entry C popped from $q(A)$

```
Procedure: process-stack-entry(A,C)
 1: if $C \in \mathrm{CN}_\mathcal{T}$ then
 2:     process-concept-name(A,C)
 3: end if
 4: if $C = \exists r.E$ then
 5:     process-existential-restriction(A,$\exists r.E$)
 6: end if
 7: for all GCI $\alpha = \bigsqcap\{C_1, \dots, C_n\} \sqsubseteq \bigsqcap\{D_1, \dots, D_m\} \in$ list(C) do
 8:     count[A][$\alpha$] = count[A][$\alpha$] - 1
 9:     if count[A][$\alpha$] = 0 then
10:         $q(A) = q(A) \cup \{D_i \mid D_i \notin S(A)\}$
11:         $S(A) = S(A) \cup \{D_i \mid D_i \notin S(A)\}$
12:     end if
13: end for
```

Algorithm 4 process the concept name B popped from $q(A)$

```
Procedure: process-concept-name(A,B)
 1: for all $D \in \mathrm{CN}_\mathcal{T}$ s.t. $\exists r.A \in S(D)$ and $\exists r.B \notin S(D)$ do
 2:     $q(D) = q(D) \cup \{\exists r.B\}$
 3:     $S(D) = S(D) \cup \{\exists r.B\}$
 4: end for
```

all stacks `q(·)` are empty. Note that our algorithm differs from the "refined" algorithm introduced in [5, 6] only in the way how the stacks (there queues) are processed, and how axioms that apply at a particular step are detected. In principle it still performs exactly the same operations in the "abstract" algorithm introduced there. That is, it still performs the *completion rules* in [5, 6]. In fact, the lines 9-11 of Algorithm 3 implement the completion rules R1, R2 and part of R3 in [6]. Lines 2-4 of Algorithm 5 implement rest of R3, and the whole `process-existential-restricton` procedure implement rules R4 and R5. Since we do not modify the original abstract algorithm in [5, 6] we do not need to give proof of correcteness of our algorithm here.

It has been shown in [6, 20] that the refined algorithm there makes at most n^4 additions to the subsumer list $S(\cdot)$ and to the queues used, where n is the size of the normalized TBox. For every addition to $S(\cdot)$ this algorithm performs a subset check in order to decide whether the fetched axiom from the queue is applicable at that step or not. This subset check brings an overhead which in the worst-case is n, thus the overall runtime of the original algorithm is $O(n^5)$.

The counters used by our algorithm allow us to check whether an axiom applies without doing the subset check mentioned above, thus avoid the n-step overhead in the worst-case. Basically, this is how our algorithm achieves a better worst-case complexity $O(n^4)$ instead of the $O(n^5)$ worst-case complexity of the original algorithm.

Algorithm 5 process the existential restriction $\exists r.E$ fetched from $q(A)$

Procedure: process-existential-restriction(A,$\exists r.E$)

```
1: for all $s \in \mathrm{RN}_\mathcal{T}$ s.t. $r \sqsubseteq^*_\mathcal{T} s$ do
2:     for all $D \in \mathrm{CN}_\mathcal{T}$ s.t. $D \in S(E)$ and $\exists s.D \notin S(A)$ do
3:         $q(A) = q(A) \cup \{\exists s.D\}$
4:         $S(A) = S(A) \cup \{\exists s.D\}$
5:     end for
6:     for all $D \in \mathrm{CN}_\mathcal{T}$ s.t. $\exists x.A \in S(D)$ and $\exists y.E \notin S(D)$ and $x, y \in \mathrm{RN}_\mathcal{T}$ s.t. $x \circ s \sqsubseteq y \in \mathcal{T}$ do
7:         $q(D) = q(D) \cup \{\exists y.E\}$
8:         $S(D) = S(D) \cup \{\exists y.E\}$
9:     end for
10:    for all $D \in \mathrm{CN}_\mathcal{T}$ s.t. $\exists x.D \in S(E)$ and $\exists y.D \notin S(A)$ and $x, y \in \mathrm{RN}_\mathcal{T}$ s.t. $s \circ x \sqsubseteq y \in \mathcal{T}$ do
11:        $q(A) = q(A) \cup \{\exists y.D\}$
12:        $S(A) = S(A) \cup \{\exists y.D\}$
13:    end for
14: end for
```

4 Implementation and Evaluation

In order to evaluate the runtime behaviour of our modified algorithm, we have implemented it and performed a series of tests on large bio-medical knowledge bases: Foundational Model of Anatomy[3] (FMA) is a large but simple TBox that contains 75139 concept names. Similarly, the Gene Ontology [4] (GO) and National Cancer Institute Thesaurus [5] (NCI) are large knowledge bases with shallow hierarchies. The GO contains 25070 concept names, and the NCI contains 27652 concept names. We have stripped down Galen [6] by removing functionalities and inverse roles to obtain the knowledge base Galen$^-$, which contains 23136 concept names. Finally we have also used the very large knowledge base SNOMED [7], which contains 293707 concept names. We have implemented the algorithm in the C programming language due to its speed and efficient use of the memory, which is important for dealing with these large knowledge bases. Currently our implementation can only read $\mathcal{EL}^+$ knowledge bases written in OWL 2 Functional-Style Syntax. We have implemented the parser for this syntax using the tools lex and yacc, which are used for generating lexical analyzer and parser for a given grammar.

In order to evaluate its performance, we have implemented the modified algorithm presented above, in our prototypical reasoner CHEETAH[8]. We have

[3] http://sig.biostr.washington.edu/projects/fm/AboutFM.html
[4] http://www.geneontology.org/
[5] http://www.cancer.gov/cancertopics/cancerlibrary/terminologyresources
[6] http://www.co-ode.org/galen/
[7] http://www.nlm.nih.gov/research/umls/Snomed/snomed_main.html
[8] http://code.google.com/p/cheetah

	FMA	GO	NCI	SNOMED	Galen$^-$
CHEETAH	4.78	1.37	1.29	179.57	21.41
CHEETAH*	4.03	1.00	0.88	92.23	12.32
CB	6.59	3.17	2.13	46.88	3.20

Table 1. Comparison of runtimes in seconds

compared its performance with the performance of the simple algorithm that performs subset checks instead of maintaining counters in order to decide when to apply an axiom. The simple algorithm still uses the normalization rules introduced in Section 3.1, and stacks for keeping track of concepts still to be processed, but does not use counters. Instead, for a concept name B popped from `q(A)`, it compares the left handsides of axioms containing B with the current subsumers of A, and applies an axiom if the former is a subset of the later. In our comparison we have also involved the CB Reasoner[9] that has been introduced in [15]. The underlying algorithm of the CB Reasoner extends the $\mathcal{EL}^{++}$ classification algorithm in [4] to the much more expressive fragment Horn $\mathcal{SHIQ}$, for which reasoning is not tractable any more. CB is able to classify the medical knowledge base Galen [19] that uses this expressivity, and as reported in [13], it outperforms all other reasoners for several other large bio-medical knowledge bases as well.

The results of our experiments were surprising. Despite its better worst-case complexity, in practice our modified algorithm performed worse than our implementation of the simple algorithm. The results of our experiments are shown in Table 1, where CHEETAH represents the implementation of our modified algorithm, and CHEETAH* the implementation of the simple algorithm. The experiments are run on a computer with an Intel Core i3 processor running at 2.1 GHz, 8 GB of main memory and on the Linux operating system with 2.6.38 kernel. As seen in Table 1, the modified algorithm performs always worse than the simple algorithm. A closer look into the input knowledge bases reveals that the worst-case, i.e., axioms with very long conjunctions on the left handsides do not occur in practice. In fact, in SNOMED the longest conjunction on the left handside of a GCI is 12, for the Galen version we used it is 5, for FMA, GO and NCI it is just 1. When it comes to average size of conjunctions on the left hand side, for SNOMED it is 1.30 and for Galen$^-$ it is 1.29, which are very small compared to the number of concept names occurring in these knowledge bases. This explains the poor performance of the modified algorithm. The worst-case, i.e., large conjunctions on the left handsides, does not occur in any of the knowledge bases we have used in our experiments. In practice the conjunctions on the left handsides are so small that even plain subset check is fast enough compared to the overhead of maintaining the counters.

According to the table the CB Reasoner performs in general better than both CHEETAH and CHEETAH*. This is due to our unoptimized implementation of the processing of stacks. CB spends a big portion of the runtime for loading

[9] http://code.google.com/p/cb-reasoner

and normalizing the knowledge base however it is very efficient in computing the subsumer lists. For instance for SNOMED in our experiments it took CB 13.99 seconds to load and normalize the knowledge base, and only 21.47 seconds to compute the subsumer lists. On the other hand for CHEETAH* loading and normalizing the knowledge base took 3.9 seconds, and computing the subsumer lists took 84.13 seconds.

5 Concluding Remarks and Future Work

We have investigated a modification to the $\mathcal{EL}^+$ classification algorithm for improving its worst-case complexity. It turned out the modified algorithm performs worse in practice. However, there is some room for further improvement of both the modified and the simple algorithm. During the execution of both algorithms, some axioms are applied several times, which in principle could be avoided. For instance if we are processing the stack of concept name A and find out that A is subsumed by B, we can skip the axioms already applied while computing the subsumers of B and just extend the subsumer list of A with that of B. This would bring the overhead of keeping track of which axioms have already been applied for which concept name, but save the effort of applying those axioms again. One other possible improvement is to make use of the axioms that have only one concept name or existential restriction, i.e., no conjuction on the left handside. One can apply such axioms immediately before the execution of the algorithm, thus pre-filling the stacks and subsumer lists with told-subsumers appropriately.

As future work we are going to implement and test these further improvements. In addition to this, we are going to extend the CHEETAH reasoner to support the OWL2 EL Profile and improve its usability by providing a platform independent Java interface and a Protege plugin for it.

Acknowledgements: We would like to thank Yevgeny Kazakov for giving his remarks and providing information about the CB Reasoner.

References

1. W. W. Armstrong. Dependency structures of data base relationships. *Proceedings of the Information Processing Congress 74, (IFIP 74)*, pages 580–583. North-Holland, 1974.
2. F. Baader. Terminological cycles in a description logic with existential restrictions. *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 325–330. Morgan Kaufmann, 2003.
3. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, (IJCAI 05)*, pages 364–369. Professional Book Center, 2005.
4. F. Baader, C. Lutz, and B. Suntisrivaraporn. Is tractable reasoning in extensions of the description logic $\mathcal{EL}$ useful in practice? In *Proceedings of the Methods for Modalities Workshop (M4M-05)*, 2005.

5. F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.
6. F. Baader, C. Lutz, and B. Suntisrivaraporn. Is tractable reasoning in extensions of the description logic $\mathcal{EL}$ useful in practice? In *Journal of Logic, Language and Information, Special Issue on Method for Modality (M4M)*, 2007. To appear.
7. C. Beeri and P. A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems*, 4(1):30–59, 1979.
8. S. Brandt. On subsumption and instance problem in $\mathcal{ELH}$ w.r.t. general tboxes. *Proceedings of the 2004 International Workshop on Description Logics, (DL2004)*, volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.
9. S. Brandt. Polynomial time reasoning in a description logic with existential restrictions, gci axioms, and - what else? *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, (ECAI 2004)*, pages 298–302. IOS Press, 2004.
10. E. F. Codd. A relational model of data for large shared data banks. *Communications of ACM*, 13(6):377–387, 1970.
11. T. G. O. Consortium. Gene ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.
12. R. Cote, D. Rothwell, J. Palotay, R. Beckett, and L. Brochu. The systematized nomenclature of human and veterinary medicine. Technical report, International, Northfield, IL: College of American Pathologists, 1993.
13. K. Dentler, R. Cornet, A. ten Teije, and N. de Keizer. Comparison of reasoners for large ontologies in the owl 2 el profile. *Semantic Web Journal*, 2011. To appear.
14. W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.
15. Y. Kazakov. Consequence-driven reasoning for horn shiq ontologies. *Proceedings of the 21st International Joint Conference on Artificial Intelligence, (IJCAI 2009)*, pages 2040–2045, 2009.
16. M. Lawley and C. Bousque. Fast classification in protege: Snorocket as an owl2 el reasoner. In *Proceedings of Australasian Ontology Workshop*, 2010.
17. D. Maier. *The Theory of Relational Databases*. Computer Science Press, Maryland, 1983.
18. J. Mendez and B. Suntisrivaraporn. Reintroducing CEL as an OWL 2 EL reasoner. *Proceedings of the 22nd International Workshop on Description Logics (DL 2009)*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.
19. A. Rector and I. Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*. AAAI Press, 1997.
20. B. Suntisrivaraporn. *Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies*. Ph.D. dissertation, Institute for Theoretical Computer Science, TU Dresden, Germany, 2009.
21. E. Thomas, J. Z. Pan, and Y. Ren. Trowl: Tractable owl 2 reasoning infrastructure. *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, (ESWC 2010)*, volume 6089 of *Lecture Notes in Computer Science*, pages 431–435. Springer-Verlag, 2010.