Technical Report Number 2

Classifying and Querying Very Large Taxonomies

A Comparative Study to the Best of Our Knowledge

Hassan Aït-Kaci and Samir Amir

May 2013

Publication Note

Revision version:

January 9, 2015

Authors' address:

LIRIS - UFR d'Informatique
Université Claude Bernard Lyon 1
43, boulevard du 11 Novembre 1918
69622 Villeurbanne cedex
France

Email: `hassan.ait-kaci@univ-lyon1.fr`
Phone: +33 (0)4 27 46 57 08

Email: `samir.amir@univ-lyon1.fr`
Phone: +33 (0)4 27 46 57 07

CEDAR **Technical Report Number 2**

Classifying and Querying Very Large Taxonomies

A Comparative Study to the Best of Our Knowledge

Hassan Aït-Kaci and Samir Amir

`hassan.ait-kaci@univ-lyon1.fr, samir.amir@univ-lyon1.fr`

May 2013

Abstract

This document addresses the question of how efficiently the most well-known Semantic Web (SW) reasoners perform in processing (classifying and querying) taxonomies of enormous size. Using techniques that were proposed 25 years ago for implementing efficient lattice operations, we have implemented a simple taxonomic concept classification and Boolean query-answering system. We compared its performance with those of the best existing SW reasoning systems over several very large taxonomies under the exact same conditions for so-called TBox reasoning.

Keywords: Scalable Taxonomic Reasoning, Concept Classification, Boolean-Query Answering, Partial-Order Encoding, Efficient Lattice Operations, Implementation Techniques

Résumé

Ce document s'intéresse à la question de savoir avec quelle efficacité les meilleurs raisonneurs pour le Web sémantique traitent des taxonomies de taille énorme (classification et réponse aux requêtes). Utilisant des techniques proposées il y a 25 ans, nous avons développé un système élémentaire de classification de concepts taxonomiques avec réponse à des requêtes booléennes. Nous avons comparé ses performances avec celles des meilleurs raisonneurs existant sous exactement les mêmes conditions pour le raisonnement de la "TBox."

Mots-Clés: Raisonnement sur des grosses taxonomies, classification de concepts, réponse à des requêtes booléennes, encodage d'ordre partiel, opérations efficaces dans les treillis, techniques d'implémentation

Table of Contents

1 Introduction

This document addresses issues related to two topics:

- Robust and scalable knowledge management and reasoning on the Web;
- Evaluation of semantic web technologies.

Regarding the first topic, we demonstrate how a specific method for taxonomic reasoning we have implemented is both robust and scalable using a straightforward technique on very large taxonomies derived from real-life ontologies. As for the second topic, we proceed to a comparative evaluation of the best existing Semantic Web (SW) reasoners on the same taxonomies under the same conditions. This comparative study illustrates clearly the mutual stance in terms of performance for each respective reasoner with respect to one another. It also does so for our system, thus putting its performance in context with the state of the art.

We report comparative-measurement results for taxonomic reasoning over very large conceptual partial orders. This study addresses the question of how efficiently the most well-known Semantic Web (SW) reasoners perform in processing (classifying and querying) taxonomies of enormous size. We focus on Boolean queries (`and`, `or`, `not`) involving a large number of concepts. The results show that, surprisingly, authors of systems making up the state of the art in SW reasoning seem to have been unaware of techniques that were proposed over 25 years ago for implementing efficient lattice operations [3]. Using these techniques, we have implemented a simple taxonomic concept classification and Boolean query-answering system, and compared its performance with those of the best existing SW reasoning systems over several very large taxonomies under the exact same conditions for so-called TBox reasoning. The results show that our simple system is among the best for concept classification and several orders-of-magnitude more efficient in terms of response time for query-answering. We present these results in detail and comment them. We recall the simple technique we have used to achieve such performance, and how it could be even further optimized. We also discuss future work regarding how more complete ontological reasoning can be made to scale up to very large ontologies without performance degradation.

The rest of this paper is organized as follows. In Section 2, we recall the state of the art in terms of Semantic Web reasoners. In Section 3, we explain the method we have implemented for taxonomic reasoning. In Section 4, we present the results of our comparative measurements. In Section 5, we make some important remarks dealing with implementation. We conclude in Section 6 with a summary of our contribution and perspectives.

2 The State of the Art

In this section, we give a brief description of the SW reasoners that we have used for our experiments. Note that we have limited our selection to systems that are full-fledged *reasoners*, and not just *classifiers*. This is because our interest goes beyond concept classification and includes Boolean query answering as well. This rules out systems such as ELK[1] [16], CEL[2] [6], CB[3] [15], *etc.*, that do not support query answering.

We retrieved and installed the following SW reasoners:

1. FaCT++;[4]
2. HermiT;[5]
3. Pellet;[6]
4. TrOWL;[7]
5. RacerPro;[8]
6. SnoRocket.[9]

FaCT++ (Fast Classification of Terminologies) is a reasoner developed at the University of Manchester [25]. It is based on the Description Logic fragment $\mathcal{SHOIQ}$ [14]. It is implemented in C++ as a deductive tableau [18] adapted to the specifics of this logic. It is claimed to use a wide range of heuristic optimizations. FaCT++ provides TBox reasoning (subsumption, satisfiability, classification) and partial support for ABox processing (retrieval).

HermiT is also a reasoner for a (slight extension) of the Description Logic fragment $\mathcal{SHOIQ}$ (called $\mathcal{SHOIQ}^+$) [20]. It is based upon hypertableau reasoning, an optimized version of tableau reasoning [19]. It purports to provide a faster process for classifying ontologies. The main optimization of hypertableau *vs.* tableau that it tries to minimize nondeterminism in the treatment of disjunctions and is more memory-efficient. HermiT provides TBox reasoning, with the ability of checking the consistency of an ontology and inferring implicit relationships between concepts.

Pellet is a free open-source Java-based reasoner [21]. It, too, is based on the tableau algorithm and supports the Description Logic fragment $\mathcal{SHOIN}(D)$. It provides TBox reasoning (subsumption, satisfiability, and classification) and ABox reasoning (retrieval, conjunctive query answering). It uses many optimization techniques and supports entailment checks and ABox querying through its interface.

TrOWL (Tractable reasoning infrastructure for OWL 2) was developed at the University of Aberdeen [24]. This is a system that starts by transforming an ontology from OWL-DL to OWL-QL [9] in order to classify it in polynomial time. Under this transformation, conjunctive query answering and consistency checking remain the same as for OWL-DL. In addition, TrOWL can generate a database schema for storing normalized representations of OWL-QL ontologies.

[1] `http://www.cs.ox.ac.uk/isg/tools/ELK/`
[2] `https://code.google.com/p/cel/`
[3] `https://code.google.com/p/cb-reasoner/`
[4] `http://owl.cs.manchester.ac.uk/fact++/`
[5] `http://www.hermit-reasoner.com/`
[6] `http://clarkparsia.com/pellet/`
[7] `http://trowl.eu/`
[8] `http://franz.com/agraph/racer/`
[9] `http://research.ict.csiro.au/software/snorocket`

RacerPro is a commercial version of RACER (Renamed ABoxes and Concept Expression Reasoner) [12, 11]. It implements a reasoner for the description logic $\mathcal{SHIQ}$. RACER provides both TBox and ABox reasoning. It supports all the optimizations of FaCT++ as well as new techniques for dealing with number restrictions and ABoxes.

Snorocket [17] was proposed as a high-performance implementation of a polynomial-time classification algorithm for the lightweight Description Logic $\mathcal{EL}$ [5].[10] It was primarily meant to be optimized for classifying SNOMED CT. It can process only conjunctive queries.

The following section summarizes the specific method for taxonomy classification based on transitive closure that we have implemented and compared to all the above systems for Boolean query answering.

3 Our Method

In this section, we give a self-contained summary of the method we have implemented in order to measure its performance for classification of bare taxonomies and query answering of Boolean queries.

Our method is an implementation in Java of the technique described in [3]. It consists in representing the elements of a taxonomy (*i.e.*, an arbitrary poset) as bit vectors. Thus, each element has a code (a bit vector) carrying a "`1`" in the position corresponding to the index of any other elements that it subsumes. In this manner, the three Boolean operations on sorts are readily and efficiently performed as their corresponding operations on bit-vectors. However, for this to be possible, these bit vectors must be encoded as the reflexive transitive closure of the "is-a" relation obtained from subsort declarations.

How to compute such a closure has been well-known—*e.g.*, the Warshall-Strassen method using clever matrix multiplication tricks [23, 10, 7]. However, for a poset of n elements, this method has quite a large time complexity—even with the best known algorithm to date, it is $\mathcal{O}(n^{2.23727})$ [22, 28].[11] In fact, in practice, the straightforward $\mathcal{O}(n^3)$ in-place multiplication method known as Warshall's Algorithm [27, 26] is used in most cases.

Figure 1 gives Java code for Warshall's algorithm performed "in-place" on binary codes represented as bit-sets stored in an array `SORTS`. A bit-set object is endowed with a method `get(int)` that returns the value of its bit in the specified position as a `boolean`, and a method `set(int,boolean)` that sets its bit in the specified position to the specified Boolean value.

Now, while Warshall's algorithm may be viable for relatively small posets, it simply becomes unusable for posets of the size of the taxonomies we are considering.

Note, however, that transitive-closure methods need pay such a high performance cost

[10] Description Logics in the $\mathcal{EL}$-family are weaker versions that provide existential roles ($\exists r.C$) but no universal roles ($\forall r.C$).

[11] To the best of our knowledge, this is the latest best bound as of 2011. However, these algorithms are not implementable due to prohibitive size of constants.

```
int n = SORTS.size();
for (int k = 0; k < n; k++)
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      if (!SORTS[i].code.get(j)))
        SORTS[i].code.set(j,
                          SORTS[i].code.get(k)
                          &&
                          SORTS[k].code.get(j));
```

Figure 1: Java code for "in-place" Warshall's algorithm

only due to the fact that they are devised for arbitrary graphs. But concept taxonomies are *not* arbitrary graphs. Namely, a necessary condition for a set of partially-ordered concepts to be semantically consistent is that its graph must be *acyclic*. Thus, a consistent taxonomy must be a directed-acyclic graph (or *dag*) with a least element ($\perp$) and a highest element ($\top$). In [3], it is shown that for such a dag, an $\mathcal{O}(n)$ transitive-closure algorithm exists and is proven correct. This method is described as Algorithm 1.

Algorithm 1 Taxonomy Classification Algorithm

```
 1: procedure CLASSIFY
 2:     $L \leftarrow \mathit{Parents}(\perp)$;
 3:     while $L \neq \emptyset$ do
 4:         for all $x \in L$ do
 5:             $x.code \leftarrow 2^{x.index} \vee \bigvee_{y \in \mathit{Children}(x)} y.code$;
 6:             $x.coded \leftarrow$ true;
 7:         end for
 8:         $L \leftarrow \bigcup_{x \in L} \mathit{Parents}(x)$;
 9:         for all $x \in L$ do
10:             if $\exists y \in \mathit{Children}(x)$ and $\neg y.coded$ then
11:                 $L \leftarrow L - \{x\}$;
12:             end if
13:         end for
14:     end while
15: end procedure
```

The procedure CLASSIFY assumes that each sort s has a set *Parents(s)* and a set *Children(s)* provided (*i.e.*, from the "is-a" declarations of members of the taxonomy). Each sort is an object that has a *code* field, which is its bit vector representation (initialized to be all zeroes), a unique characteristic *index* field which is an integer (*viz.*, its number in the taxonomy), and a Boolean field *coded* initially set to **false** denoting whether this sort has been encoded or not.

Algorithm 1 computes the reflexive-transitive closure of a taxonomy [3].[12] This algorithm can be explained quite easily. It proceeds layer by layer, starting with the parents

[12] *op. cit.*, pages 125–126.

of $\perp$ (*i.e.*, the minimal sorts in the taxonomy) [Line 2], assigning a code to each element in the current layer to be the bitwise `or` of its children and also setting the bit in its index position [Line 5]. Each time an element is encoded, it is marked to be so by setting its *coded* flag to **true** [Line 6]. Then, a new layer is computed from the current one as the union of all it parents [Line 8] from which any sort that has at least one child not encoded is removed [Line 11]. Indeed, by construction, such sorts can always be reached later. This proceeds until an empty layer is obtained [Line 3]—which is when all sorts have been encoded.

This algorithm's main loop [Lines 3–14] clearly visits each sort exactly once, and is thus linear. The auxiliary computation of the next layer [Lines 8–13] has comparatively marginal cost as it can be made efficient using constant access-time data structures for the sets of parents and children, making set operations on them negligible. Also proceeding bottom up has a clear performance advantage for dags such as most concept taxonomies, where sorts tend to have many less parents than they have children.

It is this method that we have implemented, tested, and compared with the best SW reasoners we could retrieve, to run on the very large taxonomies we extracted from existing publicly accessible ontologies of enormous size. The real bonus of this method is, of course, that all three Boolean operations on sorts stay virtually $\mathcal{O}(1)$ irrespective of the size of the taxonomy nor that of the number of concepts in the query. The result of any such query is the set of sorts with codes in the set of maximal common lower-bounds of the computed code.[13] All the above claims are clearly demonstrated on all the performance graphs we are reporting in the next section.

As for incrementality, removing a sort amounts simply to erasing its index position in all codes that have it set. Adding a sort (through a new "is-a" declaration $s_1 < s_2$) is done by restarting the bottom up propagation of the loop [Lines 3–14] starting from the parents of lower of the two sorts, after having reset all its ancestors to the uncoded status. In case of several new "is-a" declarations, the same procedure is applied but starting from the union of the parents of the minimal new sorts after resetting all their ancestors to the uncoded status.

The next section reports the results of our experiments with the six reasoners that we retrieved (FaCT++, HermiT, Pellet, RacerPro, TrOWL, SnoRocket) and ours ($\mathcal{CEDAR}$). All runs for all reasoners were carried out under the exact same conditions for exactly the same queries.

4 "Just the Facts, Ma'am!"

We extracted the bare concept taxonomies gathered from four very large ontologies. Here they are, listed in order of increasing sizes:

1. Wikipedia—this is an ontology derived from the Wikipedia online database (size: 111,599 sorts);[14]

[13] See Section 5.

[14] `http://www.h-its.org/english/research/nlp/download/wikitaxonomy.php`

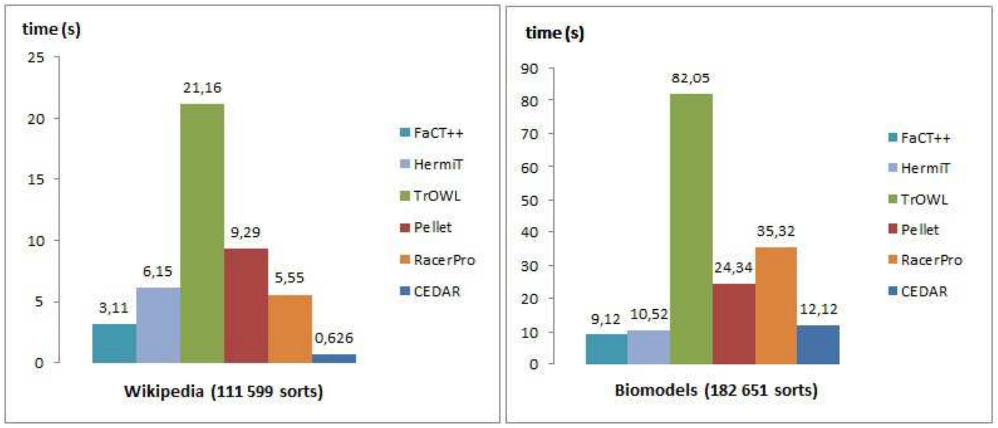


Figure 2: Classification time per reasoner for the "Wikipedia" and "BioModels" taxonomies

2. BioModels—this is an ontology of various biological models (size: 182,651 sorts);[15]
3. MeSH—(Medical Subject Headings) this is an ontology of the National Library of Medicine (size: 286,381 sorts);[16]
4. NCBI—this is the National Center for Biotechnology Information's ontology of all known living organisms (size: 903,617 sorts).[17]

We focused only on bare conceptual taxonomic reasoning.[18] That is, we considered no roles, just sorts—a sort being defined as a monadic concept in a partially ordered "is-a" taxonomy. The "reasoning" on such sorts amounts to propositional logic. In other words, this boils down to computing Boolean expressions consisting of sorts and the three operations: `and`, `or`, `not`. Seen another (equivalent) way, these operations applied to set-denoting expressions are interpreted respectively as set intersection, union, and complementation. The topmost sort $\top$ denotes the set of all things, and the bottommost sort $\bot$ denotes the empty set—*i.e.*, the set of no thing.

4.1 Classification

For each reasoner, Figures 2 and 3 show the comparative classification time performances on each of the large taxonomies we have selected. This makes six out of the seven reasoners. We did not consider SnoRocket in these classification-performance graphs because we realized that the system does not actually perform any preliminary classification, but does so on demand at query time.

While our system ($\mathcal{CEDAR}$—the rightmost on all graphs) is not always the best performance for classification, it is always among the 3 best out of six, the worst being

[15] http://bioportal.bioontology.org/ontologies/3022
[16] http://www.nlm.nih.gov/mesh/meshhome.html
[17] http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/
[18] See Section 4.3 for a discussion concerning this point.

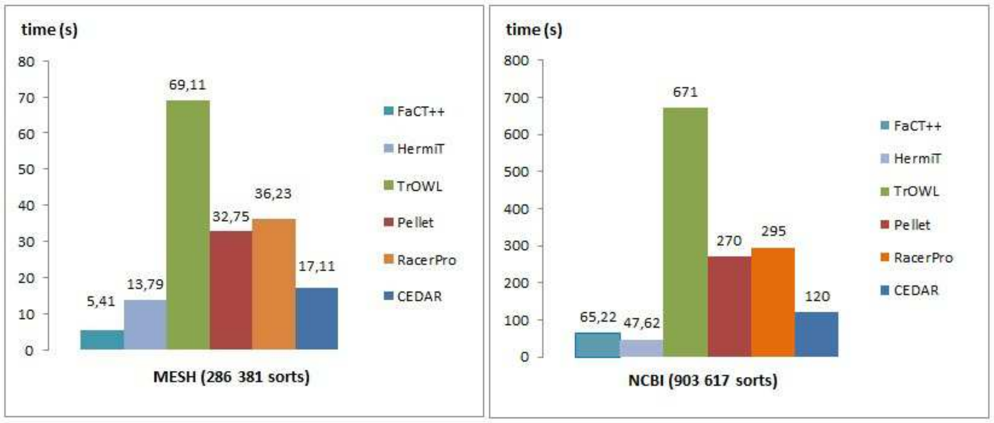


Figure 3: Classification time per reasoner for the "MeSH" and "NCBI" taxonomies

systematically TrOWL. This latter point may be due to the fact that it involves a preliminary compilation from DL to QL. Be that as it may, one can accept a longer classification time if it means faster query answering. In this regard, as illustrated next, our system definitely keeps a huge margin over all the others. It is to be noted also that TrOWL is faster at query answering relatively to the others (although still quite worse than our system on all tested taxonomies). This, again, may be justified by the longer classification time. On the other hand, HermiT, that is among the better classifiers, is always the worst for query answering. This is shown next.

4.2 Querying

For each of the seven reasoners, Figures 4–11 show the comparative query response time performances for two kinds of queries. If some reasoners are missing on some of these graphs, it is because they could not provide an answer before a time-out period that we set to 30 minutes.

The first series of graphs (Figures 4–7) are for mixed conjunctive and disjunctive queries, of the form: $s_1\&\ldots\&s_{n/2}\&(s_{n/2+1}|\ldots|s_n)$, for $n = 10, 20, \ldots, 100$.[19] We made one exception for SnoRocket, for which the queries were all conjunctive since the latest system available does not, to the best of our knowledge, support disjunctive queries.

The second series of graphs (Figures 8–11) are for purely disjunctive queries of the form: $s_1|\ldots|s_n$, for $n = 10, 20, \ldots, 100$. We did not include SnoRocket in this series of tests since, again, the system we retrieved does not support disjunctive queries.

In both series of graphs, it is clear that our system ($\mathcal{CEDAR}$) systematically achieves the best performance. Moreover, it does so by several orders of magnitude (recall that the scale of time is logarithmic).

[19] We use "&" to denote "`and`," and "|" to denote "`or`."

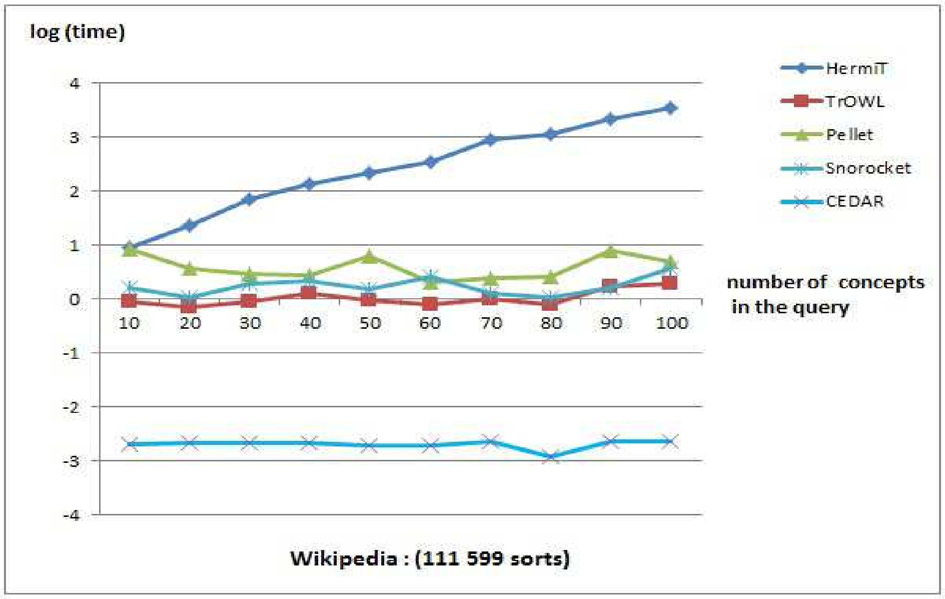


Figure 4: Mixed-query response time per reasoner for the "Wikipedia" taxonomy

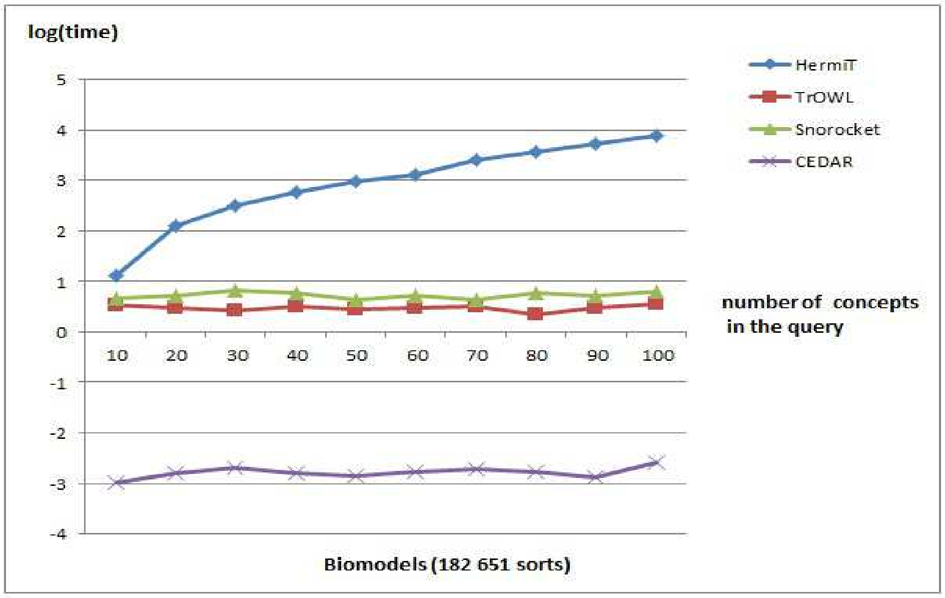


Figure 5: Mixed-query response time per reasoner for the "BioModels" taxonomy

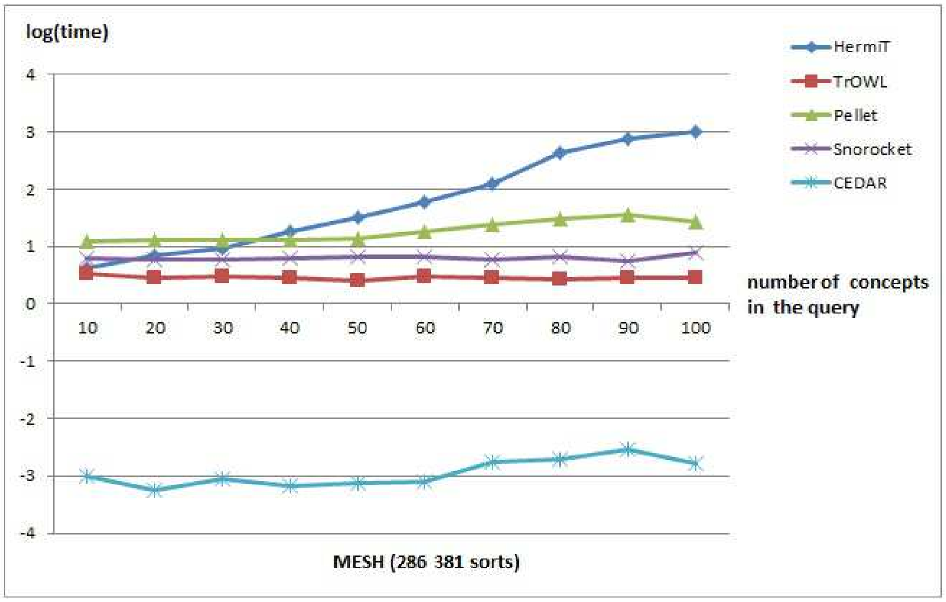


Figure 6: Mixed-query response time per reasoner for the "MeSH" taxonomy

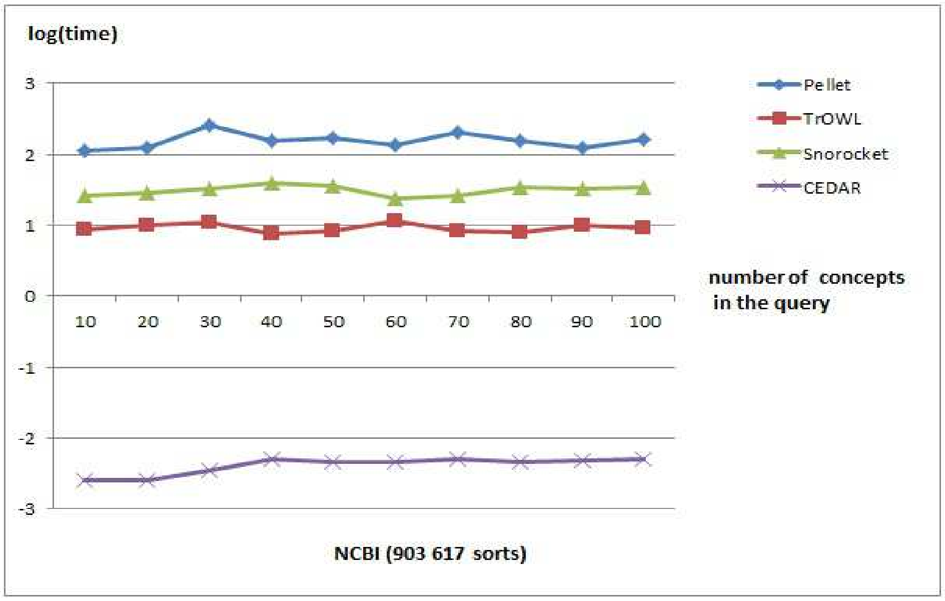


Figure 7: Mixed-query response time per reasoner for the "NCBI" taxonomy

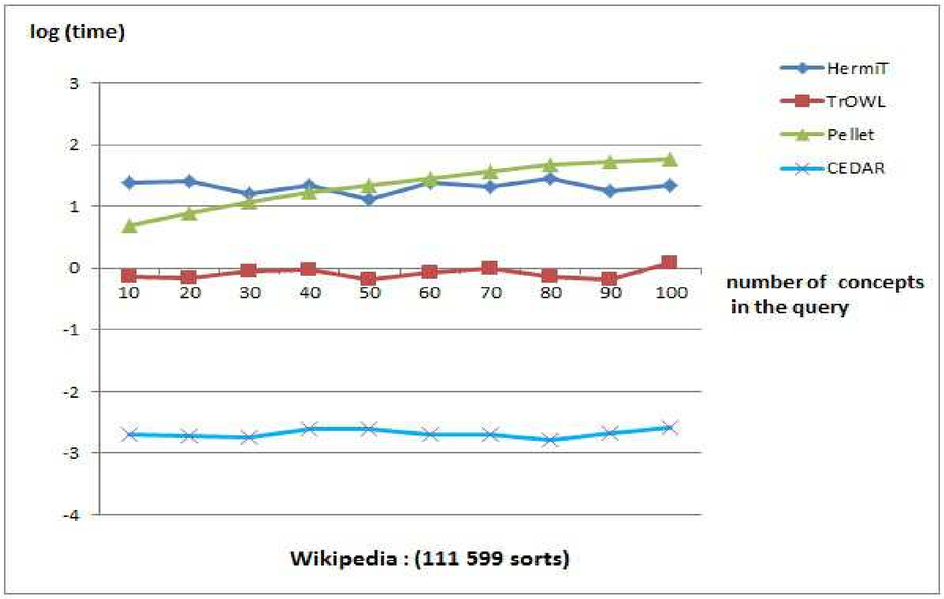


Figure 8: Disjunctive-query response time per reasoner for the “Wikipedia” taxonomy

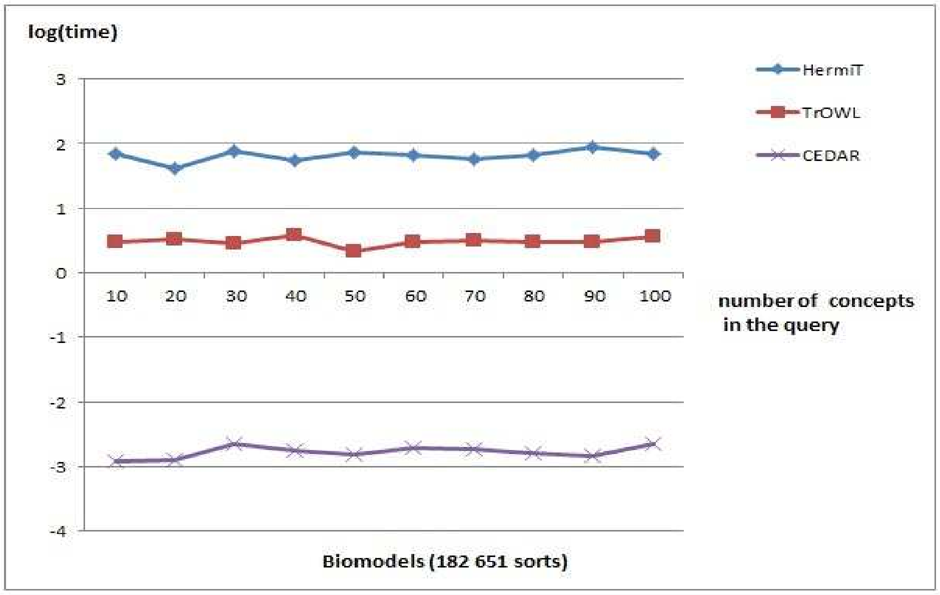


Figure 9: Disjunctive-query response time per reasoner for the “BioModels” taxonomy

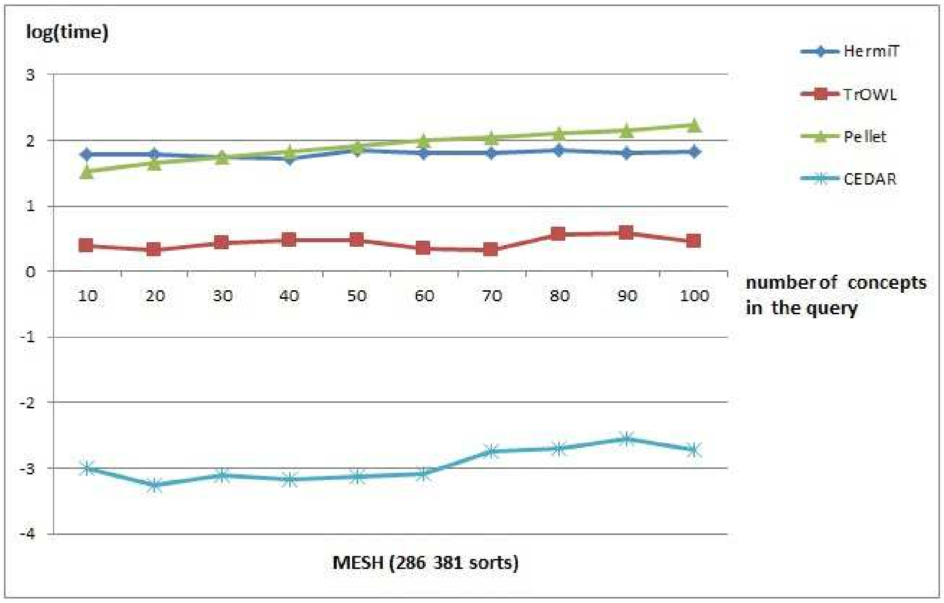


Figure 10: Disjunctive-query response time per reasoner for the "MeSH" taxonomy

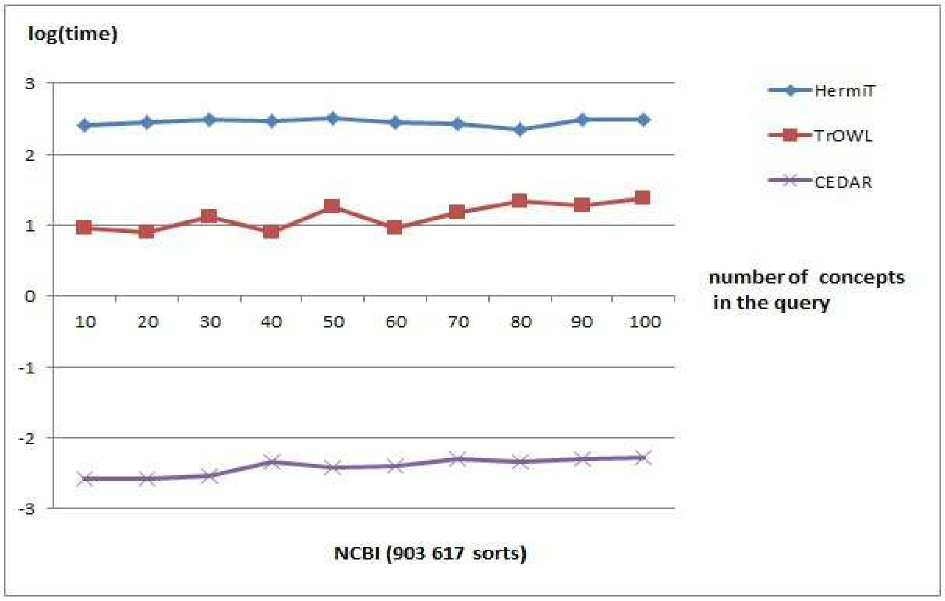


Figure 11: Disjunctive-query response time per reasoner for the "NCBI" taxonomy

Taxonomy	FaCT++	HermiT	TrOWL	Pellet	RacerPro	SnoRocket	$\mathcal{CEDAR}$
Wikipedia		100	0.13	0.51		0.21	0.000233
BioModels		100	0.13	Error		0.24	0.000074
MeSH		100	1.17	8.29		2.60	0.000530
NCBI			5.78	100		19.75	0.002627

Table 1: Relative normalized average percentiles of performance times for mixed queries

Taxonomy	FaCT++	HermiT	TrOWL	Pellet	RacerPro	SnoRocket	$\mathcal{CEDAR}$
Wikipedia		74.65	2.86	100		N/A	0.00719
BioModels		100	4.68	Error		N/A	0.00258
MeSH		67.50	3.01	100		N/A	0.00141
NCBI		100	5.10			N/A	0.00141

Table 2: Relative normalized average percentiles of performance times for disjunctive queries

4.3 Discussion

Although the graphs in Figures 4–11 speak for themselves, it is interesting to get an appreciation of the relative performances for query answering of all the reasoners we have tested. In order to do so, Tables 1 and 2 sum up the facts displayed in the graphs by taking the average over all query sizes (*viz.*, from 10 to 100 concepts), giving the maximum of these averages the value 100, and showing all the other averages as percent values.

Empty cells mean that the reasoner was never able to provide an answer within our time-out limit (which, again, was set to 30 minutes). On the "BioModels" taxonomy, Pellet stumbles into a Java runtime error for some unknown reason.

Table 1 shows these figures for the mixed conjunctive and disjunctive queries. Table 2 shows these figures for purely disjunctive queries. Again, SnoRocket does not appear in the latter because it could not be tested on disjunctive queries (hence the N/A entries).

The reason why we limited our study to bare propositional reasoning is that this is (or ought to be) the most basic capability of any ontological reasoner. Any further capability (*e.g.*, reasoning with roles—existential and/or universal, cardinality constraints, *etc.*.) must be conjugated with the basic propositional sort reasoning. In fact, in such systems, a "complex" concept is typically a sort conjoined with some additional role-related expression. Thus, one can see bare Boolean taxonomic sort reasoning as sheer abstract interpretation of complex-concept reasoning [8].

Now, in terms of implementing such reasoners, it comes as evidence that efficiency must start with the simpler form of reasoning since it is part of any further reasoning. In fact, taking advantage of commutativity and associativity of conjunction, it may even be a technique of optimizing the process of ontological reasoning. Indeed, taking a complex expression such as involving a Boolean combination of expressions such of the form s & *s-properties*, ignoring the *s-properties* parts as a first pass will narrow the original expression to its essential remaining maximal sorts.

In order to carry out these experiments, we also developed a tool with an easy-to-use GUI that lets a user run these tests for any listed reasoner and taxonomy.[20] This tool is available for download from the $\mathcal{CEDAR}$ Project's website for anyone to verify our results.[21] A video clip of these demos showing these experiments *in vivo*, and a web service for running these demos on line are to be posted as well. In this way, our results may hopefully not have to be taken on faith, but could be verified *de visu* by anyone who might wish to check them on their own.

Finally, it is worth pointing out that once a taxonomy has been classified, it may be saved on disk to be reloaded without any penalty and reused over and over.[22] This is akin to compiling a program and not needing to recompile it for each use. In this regard, we do not understand the relative importance given in SW literature to performance of ontology classification as opposed to that of query answering. In programming-language technology, what is of prime importance is runtime performance, not compile-time performance.

5 Implementation

In this section, we discuss implementation issues dealing with detecting and identifying potential cycles in a taxonomy being encoded, and decoding codes into sorts.

5.1 Detecting Cycles

This section adds some information concerning the detection and identification of potential cycles in a set of "is-a" declarations specifying a taxonomy.

Problem

Since it expresses a subset ordering among concepts, a taxonomy is supposed to be a directed acyclic graph (dag). However, it may be possible that mistakes occur whereby the "is-a" declarations among concepts contain inconsistencies in the form of cycles. To detect such cycles is thus useful for a classification tool to allow reporting them for correction to the user.

If the general $\mathcal{O}(n^3)$ Warshall algorithm is used to compute transitive-closure codes for all sorts in the taxonomy, then an existing maximal cycle will necessarily imply that all its elements are given equal codes. Indeed, by transitivity, the code of an element denotes the set of all its descendants. But all the elements of a cycle have the same set of descendants, and so their codes must be equal. Such a cycle is in fact an equivalence class for the least equivalence relation containing the declared "is-a" pairs.

One could eliminate all such cycles by collapsing them into a single sort (the class representative), obtaining the quotient set, which is then a dag. However, this is not

[20]See Section A for an overview.

[21]http://cedar.liris.cnrs.fr/data/CEDAR-V1.0.zip

[22]We implemented such a facility, of course.

desirable since such cycles are in all cases errors resulting from inconsistent declarations. In this case, they should be flagged as errors and their contents identified.

Reporting such cycles efficiently can be done by performing a topological reordering of the taxonomy according to the codes that would guarantee that sorts of equal codes are contiguous. Thus, a maximal cycle must be a maximal contiguous sequence of equal-coded sorts in this topological reordering of the taxonomy.

However, for the reasons discussed earlier in this paper, it is not feasible to use Warshall's algorithm to compute the transitive closure on very large taxonomies. In addition, reordering such a very large taxonomy using QuickSort will be on average $\mathcal{O}(n \log n)$ with a prohibitive, although very rare, $\mathcal{O}(n^2)$ worst case [13].[23]

Note that using our bottom-up encoding to compute the transitive closure of a taxonomy is correct only if it is a dag. If there are cycles in it, it will necessarily terminate with some of its elements left without code. This is because in Algorithm 1 a layer computed from a previous one [Line 8] removes any sort that has at least one child not encoded [Line 11]. This is correct for a dag since such sorts will always be reached later through a different longer path from $\bot$. But the existence of a cycle will make this assumption incorrect. For example, declaring both $s_1 \prec s_2$ and $s_2 \prec s_1$ will cause both s_1 and s_2 (and all their ancestors) to be removed from any layer to be encoded.

Therefore, the best we can expect with the bottom-up encoding method is that it always terminate in at most n iterations for a taxonomy of n elements. If the taxonomy is indeed a dag, all sorts will be correctly encoded. But if there are cycles, it will detect this to be the case (by checking that there remain non-encoded sorts upon termination). However, it does not have any possiblity to identify how many maximal cycles there are and which sorts compose them. It is because all it knows is that bottom-up encoding left some elements non-encoded—which happens if and only if there are cycles. But it does not have specific information allowing identification of which exact (maximal) cycle(s) they are.

Solution

There is a simple and feasible way to proceed. It is enough to collect the non-encoded sorts after a bottom-up encoding in a new set to be classified using Warshall's method and topologically reordered. This is feasible then because such a set is relatively much smaller than the full declared taxonomy. In this way, all cycles can be identified as maximal contiguous elements and reported as errors.

Let us now define such a topological ordering. Recall that a taxonomy of n sorts is represented as an array of size n of `Sort` objects that are characterized by three fields; the sort's:

1. index—its offset in the array;
2. code—its bit-vector encoding;
3. name—its name.

[23] http://en.wikipedia.org/wiki/Quicksort

Using this precedence test on sorts, we can thus obtain a unique topological (total) ordering of a taxonomy whereby a sort s is said to *precede* another sort s' iff, in this order:[24]

1. $s.\texttt{code} < s'.\texttt{code}$; or,
2. $s.\texttt{code} = s'.\texttt{code}$ and $s.\texttt{index} < s'.\texttt{index}$; or,
3. s and s' are unrelated, and:
   - $|s.\texttt{code}| < |s'.\texttt{code}|$; or,
   - $|s.\texttt{code}| = |s'.\texttt{code}|$ and, $\texttt{firstDiff}(s.\texttt{code}, s'.\texttt{code}) < \texttt{firstDiff}(s'.\texttt{code}, s.\texttt{code})$.

The expression $|c|$ for a code c denotes this code's cardinality (*i.e.*, its number of bits set to true). The expression $\texttt{firstDiff}(c, c')$ for two codes of equal cardinality c and c' denotes the lowest `1`-bit position in c that is `0` in c'. So the last condition ranks sorts according to the number of descendants, and when such are equal, according to the descendant of lowest differing index.

This ordering on sorts will keep lesser sorts and sorts of lesser cardinality at lower ranks. Same-cardinality codes (*i.e.*, sort with same number of subsorts) are ranked according to lowest index of the subsort contained in one but not the other. For example, code `1000111` ($\{0, 1, 2, 6\}$) is toplogically less than `0101011` ($\{0, 1, 3, 5\}$) because $2 < 3$.

It is not difficult to see that such a topological reordering will always end up with equally encoded elements being contiguous, while sorts with a greater number of lower bounds will be at higher ranks. In this manner, it is easy to identify all cycles in one single sweep of the taxonomy array as maximal sequences of contiguous equal codes of length at least 2.

As we shall see next, such a topological reordering of the taxonomy is also useful for decoding codes into sorts.

5.2 Decoding

We now give a succint description of a few points regarding decoding—*i.e.*, relating bit-vector codes to the sorts they denote.

By construction of transitive closure, Algorithm 1, the bit vector of a sort at index i ($0 \leq i \leq n - 1$) has a `1` in position j ($0 \leq j \leq n - 1$) if and only if the sort at index j is its subsort. Therefore, a sort's bit vector has `1`'s in all and only the positions of its descendant sorts. For example, the taxonomy shown as Figure 12 containing 12 sorts (other than $\top$ and $\bot$) will result in the encoding shown as Table 3. Since there are 12 sorts in this taxonomy, all codes in this taxonomy have 12 bits. The top and bottom elements ($\top$ and $\bot$) are implicit both in Figure 12 and in Table 3. So the code for bottom is all `0`'s, and the code for $\top$ is all `1`'s.

[24]The ordering on bit-vector codes is simply defined as $c_1 \leq c_2$ iff $c_1 = c_1 \& c_2$.

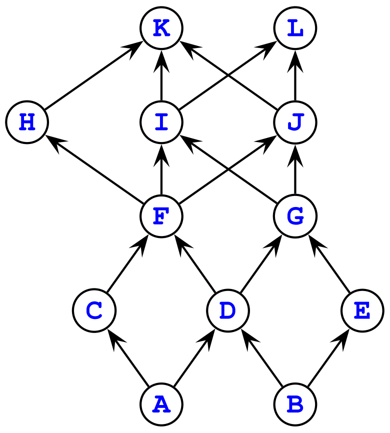


Figure 12: Example of a small “is-a” taxonomy

Index	Code	Sort
11	101110111111	L
10	011111111111	K
9	001010111111	I
8	000110111111	J
7	000010011011	G
6	000001101111	H
5	000000101111	F
4	000000011000	E
3	000000001000	B
2	000000000101	C
1	000000001011	D
0	000000000001	A

Table 3: Transitive-closure codes for sorts in Figure 12

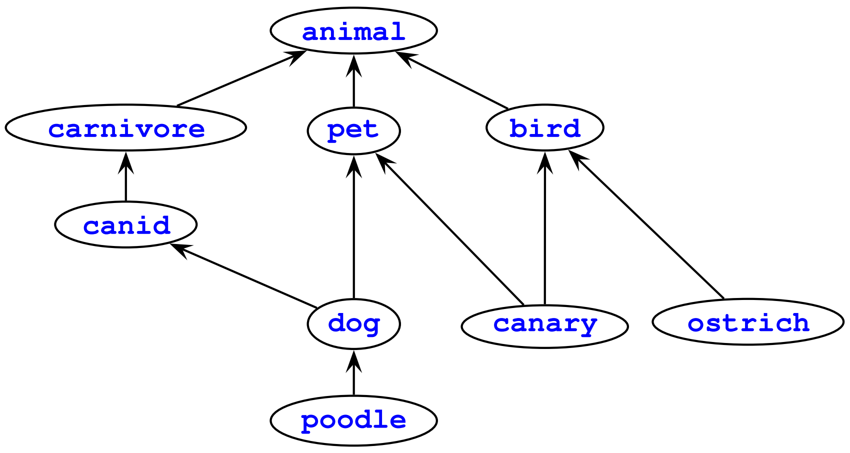


Figure 13: Example of a small animal "is-a" taxonomy

Let us first consider codes obtained without using negation. In other words, let us first restrict ourselves to decoding the result of only *positive* queries—*i.e.*, ones involving sorts of an encoded taxonomy using a Boolean expression of its sorts' bit-vector codes using bitwise `and`, `or`—but not `not`. This always results in a bit vector. In order to determine what sorts this resulting bit vector corresponds to, there are two cases: either the resulting bit vector is that of an existing sort, or it is not.

In the first case, in order to speed up determining the sort of the bit vector, all codes are stored in a hash table mapping a code to its sort. In this way, evaluating for example "`F&G`" in the taxonomy of Figure 12, which results in the bit-vector code `000000001011`, the sort can be retrieved in this hash table to be associated with the code—sort `D` in our example.

In the second case, the code resulting from a query evaluation does not correspond to an existing sort. For example evaluating "`I&J`" in the taxonomy of Figure 12 yields the code `000010111111`, which does not correspond to any specific sort in Table 3. However, semantically, this code is necessarily a minimal upper bound of the set denoted by the resulting sort if it existed. Hence, if we wish to express the resulting sort in terms of existing sorts, it must be the union of all the sorts whose codes are *maximal lower bounds* of the resulting code. In order to compute what sorts are in this set of maximal upper bounds, it suffices to retrieve all the sorts at index i such that there is a `1` at position i in the resulting code and keep only the maximal ones. In our example, the code `000010111111` has a `1` in positions $0, 1, 2, 3, 4, 5$ and 7. This means that its subsorts are `A`, `D`, `C`, `B`, `E`, `F`, and `G`. However, among these, only `F` and `G` are maximal. Therefore, the result of the query "`I&J`" is the *disjunctive* sort $\{$`F;G`$\}$.

While the above decoding scheme is correct for positive queries, it is not so however if the query made use of negation. To see this, let us consider the taxonomy shown as Figure 13 and its encoding: shown as Table 4.

Let us now consider the (negative) query: "`!canid`." The code resulting from eval-

Index	**Code**	**Sort**
8	`100000000`	`poodle`
7	`010000000`	`canary`
6	`001000000`	`ostrich`
5	`100100000`	`dog`
4	`100110000`	`canid`
3	`110101000`	`pet`
2	`100110100`	`carnivore`
1	`111111111`	`animal`
0	`011000001`	`bird`

Table 4: Transitive-closure codes for sorts in Figure 13

Rank	**Index**	**Code**	**Sort**
8	1	`111111111`	`animal`
7	3	`110101000`	`pet`
6	2	`100110100`	`carnivore`
5	4	`100110000`	`canid`
4	0	`011000001`	`bird`
3	5	`100100000`	`dog`
2	8	`100000000`	`poodle`
1	7	`010000000`	`canary`
0	6	`001000000`	`ostrich`

Table 5: Ordered transitive-closure codes for sorts in Figure 13

uating this query is the complement of the sort "`canid`"—namely, `011001111`. The decoding method that we used above will yield the maximal elements in index set $\{0, 1, 2, 3, 6, 7\}$—*viz.*, $\{$`bird`, `animal`, `pet`, `ostrich`, `canary`, `poodle`$\}$; namely, $\{$`animal`$\}$, which is obviously wrong. This decoding is incorrect because a negated code can no longer be interpreted as having a '`1`' in a position corresponding to a subsort. Indeed, for such a code, a '`1`' in position i means that sort of index i is either a supersort or unrelated. (This is because it comes from negating '`0`' at position i in the complement where that meant "sort of index i is *not a subsort of* this sort.")

Note however that, in all cases, a guaranteed-correct way of decoding a code c is to sweep all the taxonomy and keep any maximal sort whose code c' is such that $c' \leq c$. In order to make this sweep more efficient, it is convenient to reorder the taxonomy using a topological precedence ordering respecting the code subsumption ordering and giving a lower rank to a sort of lower index for incomparable codes. For example, such a total reordering for the taxonomy in Figure 13 is given by the Table 5, using the sort precedence ordering defined in the previous section.

To compute such a set of maximal lower bounds of a given code from a topologically reordered taxonomy, we sweep it upwards starting from rank 0. As we proceed upwards, we collect lower bounds of the given code in a set, keeping only maximal elements. Note that there is no need to go higher than the rank of the first sort whose code is the given code or that is a supersort of the given code (because the rank order of the topologically reordered array ensures that all sorts of higher rank are greater or unrelated).

In our example above, decoding the code `011001111` resulting from the evaluation of "`!canid`," we need only sweep no higher than rank `4`. This will collect the set of lower bounds: {`ostrich`, `canary`, `bird`}. Keeping only maximal elements, this yield the (correct) answer: "`bird`."

For efficiency reasons, once a bit-vector code has been decoded, it is stored in a cache (a hash table) associating the code to the set of sorts whose codes are its maximal lower bounds. In this way, should the same code appear again as a result, it is first looked up in this cache to avoid the need to compute again its set of maximal lower bound sorts.

Finally, note that decoding a bit vector is only necessary for extracting the end result of a query in terms of defined sorts. All intermediate computation need not refer at all to the sorts and deal only with bit vectors, whether or not they correspond to defined sorts. There is no loss of information doing so as the encoding plunges the taxonomy in the minimal Boolean lattice containing it [3].

6 Conclusion

In this paper, we have demonstrated how the state of the art in the best exisiting Semantic Web reasoners, when applied to very large taxonomies, fail to live up to performances that can be easily achieved using bit-vector encoding. We have implemented such a reasoner, and have compared its performances to those of six among the most renown reasoners.[25] Focusing only on pure Boolean taxonomic reasoning—which is as the core of any SW reasoner—the results of our measurements show that our system achieves performances for Boolean query answering that are several orders of magnitude better than those of the state of the art.

There are several optimizations that may still be performed to our basic method. The technique known as *code modulation* (explained in detail in [3]) can take advantage of a taxonomy's specific shape to minimize drastically code space. Code modulation being independent of the encoding technique, it can be applied to any method. Each module can in fact use different encoding methods each adapted to its specific topology. Another optimization to minimize classification time could be to perform lazy encoding.[26] In other words, one could only encode the sorts relevant to a query and cache intermediate results. The price to pay would be at query time, although only the first time a subset of the concepts it involves are used.

As for future work, we are extending this work to unification-based Knowledge Representation known as Order-Sorted Feature (OSF) constraint logic [2]. While OSF logic uses functional features, we can use them to represent roles using aggregates. The advantage is that role-based reasoning is thus made simpler since it relies on Logic-

[25]To the best of our knowledge.

[26]In the same manner as we have noticed that SnoRocket does.

Programming unification technology made possible by functional attributes [4, 1]. This is akin to compiling DL-based relational roles into aggregate-valued functional features. OSF sorts have also a "memoizing" effect whereby no property needs to be proven again once it has been established for any supersort [2].

Appendix

This appendix gives an overview of the software tool we developed to run our experiments. This tool is publicly available on the $\mathcal{CEDAR}$ project website.[27]

A The $\mathcal{CEDAR}$ Taxonomic Reasoner Evaluation Tool

Figures 14 and 15 show a display of the Graphical User Interface (GUI) of a tool that we developed to ease experimenting with several Semantic Web reasoners and taxonomies.

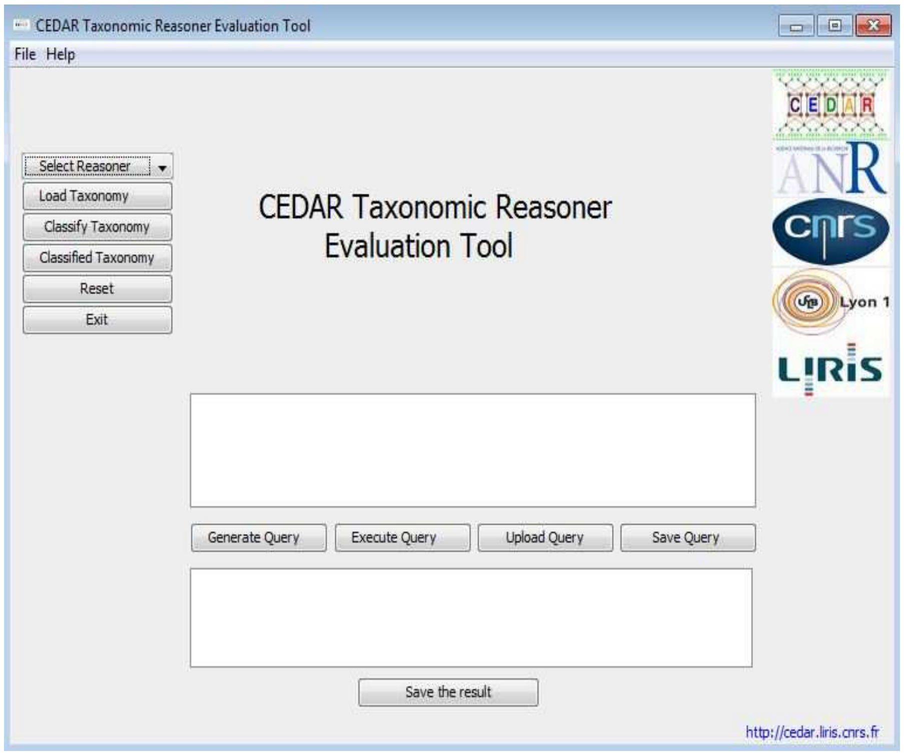


Figure 14: Screenshot of the $\mathcal{CEDAR}$ Taxonomic Reasoner Evaluation Tool: Basic GUI

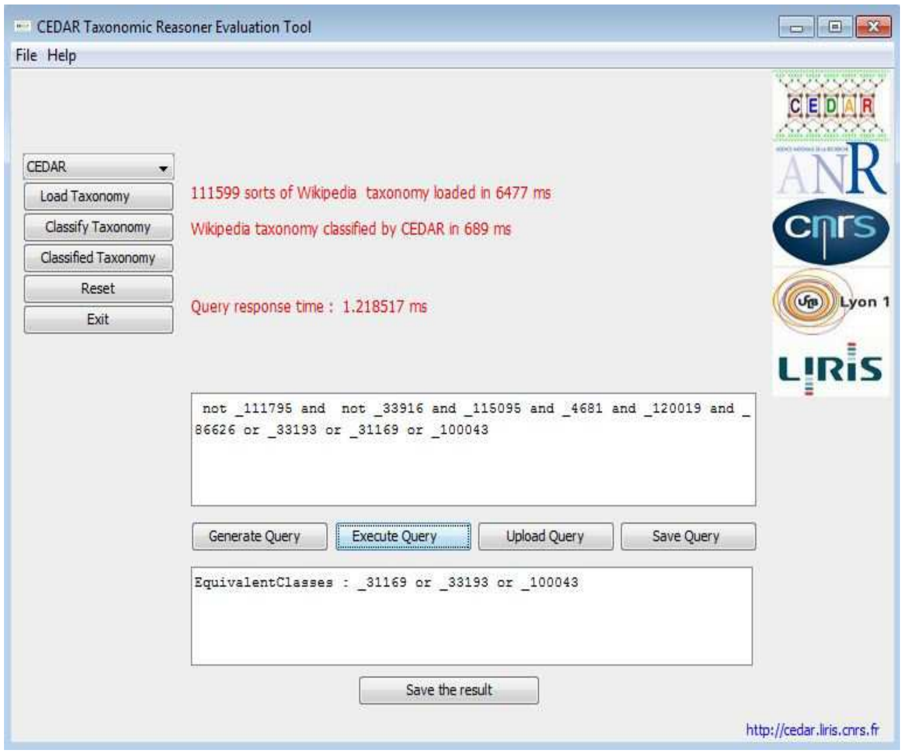
Using the GUI displayed as Figure 14, one can:

- select a reasoner from a menu;

[27] cedar.liris.cnrs.fr

- select a taxonomy to be loaded from a file located in a specifiable directory;
- classify a loaded taxonomy;
- if the reasoner is ours ($\mathcal{CEDAR}$), the option is offered to save a classified taxonomy on disk;
- one can also, again in the case of the $\mathcal{CEDAR}$ reasoner, load a taxonomy that was previously classified and saved;[28]
- reset the context (*i.e.*, clear the selected reasoner and loaded taxonomy) and the current query;
- exit the system.

There are also two windows for querying. In the upper one, one can either type in a query, load a previously saved query, or automatically generate a new query. In the latter case, a prompt window will ask to specify how many conjunctions, disjunctions, and negations one wishes the query to contain. The generated query can then be edited, executed, and/or saved for later reuse. Executing the query shown in the upper window will display the result in the lower window. This result can also be saved if wished.



```
111599 sorts of Wikipedia  taxonomy loaded in 6477 ms
Wikipedia taxonomy classified by CEDAR in 689 ms

Query response time :  1.218517 ms
```

```
 not _111795 and  not _33916 and _115095 and _4681 and _120019 and _
86626 or _33193 or _31169 or _100043
```

```
EquivalentClasses : _31169 or _33193 or _100043
```

Figure 15: Screenshot of the $\mathcal{CEDAR}$ Taxonomic Reasoner Evaluation Tool: Query Test

Such a usage is illustrated in Figure 15, which shows what the GUI looks like after selecting the $\mathcal{CEDAR}$ reasoner, loading the taxonomy "Wikipedia," generating a query

[28]This feature and the previous one may not be used for other reasoners because they do not offer such a capability.

with two 'not' operations, five 'and' operations, and three 'or' operations. The result of executing this query (*i.e.*, the union of sorts in the taxonomy that are maximal lower bounds of this sort expression) is displayed in the lower window.

References

[1] Hassan Aït-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction*. The MIT Press, Cambridge, MA, USA, 1991. [See online[29]].

[2] Hassan Aït-Kaci. Data models as constraint systems—A key to the Semantic Web. *Constraint Processsing Letters*, 1(1):33–88, November 2007. [See online[30]].

[3] Hassan Aït-Kaci, Robert Boyer, Patrick Lincoln, and Roger Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, January 1989. [See online[31]].

[4] Hassan Aït-Kaci and Roberto Di Cosmo. Compiling order-sorted feature term unification. PRL Technical Note 7, Digital Paris Research Lab, Rueil-Malmaison, France, December 1993. [See online[32]].

[5] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 364–369, Edinburgh, Scotland, UK, July-August 2005. IJCAI'05, Morgan Kaufmann Publishers. [See online[33]].

[6] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, pages 287–291, Seattle, WA, USA, August 2006. IJCAR'06, Springer-Verlag LNAI Vol. 4130. [See online[34]].

[7] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, March 1990. [See online[35]].

[8] Patrick Cousot. Abstract interpretation. *ACM Computing Surveys—Symposium on Models of Programming Languages and Computation*, 28(2):324–328, June 1996. Tutorial summary:[See online[36]].

[9] Richard Fikes, Patrick Hayes, and Ian Horrocks. OWL-QL—a language for deductive query answering on the Semantic Web. *Journal of Web Semantics*, 2(1):19–29, December 2004. [See online[37]].

[10] Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory*, SWAT'71, pages 129–131, Washington, DC, USA, 1971. IEEE Computer Society. [See online[38]].

[29] http://wambook.sourceforge.net/
[30] http://www.cs.brown.edu/people/pvh/CPL/Papers/v1/hak.pdf
[31] http://www.hassan-ait-kaci.net/pdf/encoding-toplas-89.pdf
[32] http://www.hassan-ait-kaci.net/pdf/PRL-TN-7.pdf
[33] http://www.ijcai.org/papers/0372.pdf
[34] http://www.informatik.uni-bremen.de/~clu/papers/archive/ijcar06.pdf
[35] http://www.sciencedirect.com/science/article/pii/S0747717108800132
[36] http://www.di.ens.fr/~cousot/AI/IntroAbsInt.html
[37] http://www.sciencedirect.com/science/article/pii/S1570826804000137
[38] http://rjlipton.files.wordpress.com/2009/10/matrix1971.pdf

[11] Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The RacerPro knowledge representation and reasoning system. *Semantic Web Journal*, 1:1–11, March 2011. [See online[39]].

[12] Volker Haarslev and Ralf Möller. RACER system description. In Rajeev Gore, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the 1st International Joint Conference on Automated Reasoning*, pages 701–706, Siena, Italy, June 2001. IJCAR'01, Springer-Verlag. [See online[40]].

[13] Charles Antony Richard Hoare. Algorithm 63: Partition, Algorithm 64: Quicksort. *Communications of the ACM*, 4(7):321–321, July 1961. [See online[41]].

[14] Ian Horrocks and Ulrike Sattler. A tableau decision procedure for $\mathcal{SHOIQ}$. *Journal of Automated Reasoning*, 39(3):249–276, July 2007. [See online[42]].

[15] Yevgeny Kazakov. Consequence-driven reasoning for horn $\mathcal{SHIQ}$ ontologies. In Craig Boutilier, editor, *Proceedings of the 21st International Conference on Artificial Intelligence*, pages 2040–2045, Pasadena, CA, USA, July 2009. IJCAI'09, Association for the Advancement of Artificial Intelligence. [See online[43]].

[16] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. Unchain my $\mathcal{EL}$ reasoner. In Riccardo Rosati, Sebastian Rudolph, and Michael Zakharyaschev, editors, *Proceedings of the 24th International Workshop on Description Logics*, Barcelona, Spain, July 2011. DL'11, CEUR Workshop Proceedings. [See online[44]].

[17] Michael J. Lawley and Cyril Bousquet. Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In Thomas Meyer, Mehmet A. Orgun, and Kerry Taylor, editors, *Proceedings of the 2nd Australasian Ontology Workshop: Advances in Ontologies*, pages 45–50, Adelaide, Australia, December 2010. AOW'10, ACS. [See online[45]].

[18] Zohar Manna and Richard Waldinger. Fundamentals of deductive program synthesis. In Alberto Apostolico and Zvi Galil, editors, *Combinatorial Algorithms on Words*, NATO ISI Series. Springer-Verlag, 1991. [See online[46]].

[19] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, 36(1):165–228, September 2009. [See online[47]].

[20] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A highly-efficient OWL reasoner. In Ulrike Sattler and Cathy Dolbear, editors, *Proceedings of the 5th International Workshop on OWL Experiences and Directions*, Karlsruhe, Germany, October 2008. OWLED'08, CEUR Workshop Proceedings. [See online[48]].

[21] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, June 2007. This is a summary; full paper: [See online[49]].

[39] http://www.semantic-web-journal.net/sites/default/files/swj109_3.pdf
[40] http://www.racer-systems.com/technology/contributions/2001/HaMo01e.pdf
[41] http://comjnl.oxfordjournals.org/content/5/1/10.full.pdf
[42] http://link.springer.com/article/10.1007/s10817-007-9079-9
[43] http://ijcai.org/papers09/Papers/IJCAI09-336.pdf
[44] http://ceur-ws.org/Vol-745/paper_54.pdf
[45] http://krr.meraka.org.za/~aow2010/Lawley-etal.pdf
[46] http://citeseer.ist.psu.edu/manna92fundamentals.html
[47] https://www.jair.org/media/2811/live-2811-4689-jair.pdf
[48] http://www.cs.ox.ac.uk/ian.horrocks/Publications/download/2008/ShMH08b.pdf
[49] http://pellet.owldl.com/papers/sirin05pellet.pdf

[22] Andrew Stothers. *On the Complexity of Matrix Multiplication*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, UK, 2010. [See online[50]].

[23] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[24] Edward Thomas, Jeff Z. Pan, and Yuan Ren. TrOWL: Tractable OWL 2 reasoning infrastructure. In Lora Aroyo, Grigoris Antoniou, Eero Hyvnen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *Proceedings of the 7th Extended Semantic Web Conference*, pages 431–435, Heraklion, Greece, May-June 2010. ESWC'10, Springer-Verlag. [See online[51]].

[25] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the 3rd International Joint conference on Automated Reasoning*, pages 292–297, Seattle, WA, USA, August 2006. IJCAR'06, Springer-Verlag. [See online[52]].

[26] Henry S. Warren Jr. A modification of Warshall's algorithm for the transitive closure of binary relations. *Communications of the ACM*, 18(4):218–220, April 1975. [See online[53]].

[27] Stephen Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9(1):11–12, January 1962.

[28] Virginia Vassilevska Williams. Breaking the Coppersmith-Winograd barrier. University of California at Berkeley and Stanford University, 2011. [See online[54]].

Acknowledgements: The authors wishes to thank Mohand-Saïd Hacid for his kind proof-reading and comments. All remaining mistakes, if any, are the authors' responsibility.

[50] `http://www.maths.ed.ac.uk/pg/thesis/stothers.pdf`
[51] `http://homepages.abdn.ac.uk/jeff.z.pan/pages/pub/TPR2010.pdf`
[52] `http://www.cs.ox.ac.uk/Ian.Horrocks/Publications/download/2006/TsHo06a.pdf`
[53] `http://dl.acm.org/citation.cfm?id=360746`
[54] `http://www.cs.berkeley.edu/~virgi/matrixmult.pdf`

Technical Report Number 2

Classifying and Querying
Very Large Taxonomies

Hassan Aït-Kaci and Samir Amir

May 2013