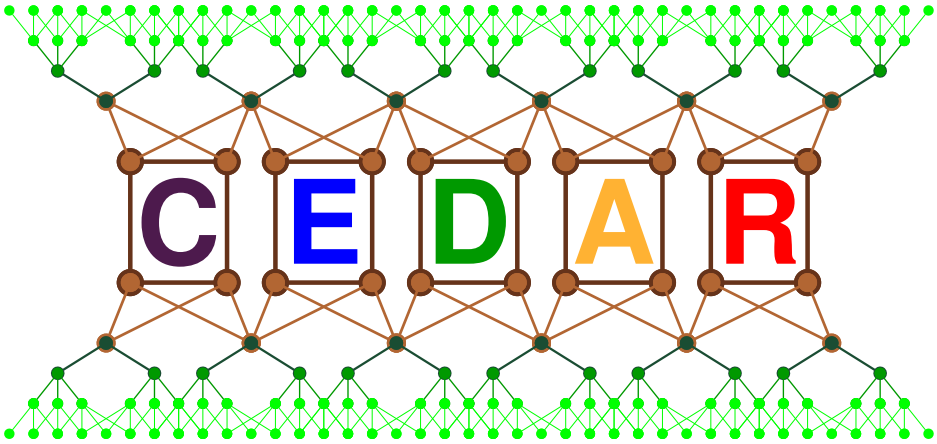


Technical Report Number 12

Design and Implementation

of an

Efficient Semantic Web Reasoner

Samir Amir and Hassan Aït-Kaci

October 2014

Publication Note

Authors' address:

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

Phone: +33 (0)4 27 46 57 08
Email: {samir.amir;hassan.ait-kaci}@univ-lyon1.fr

This work was carried out as part of the $\mathcal{CEDAR}$ Project (Constraint Event-Driven Automated Reasoning) under the *Agence Nationale de la Recherche* (ANR) Chair of Excellence grant N^o **ANR-12-CHEX-0003-01** at the *Université Claude Bernard Lyon 1* (UCBL).

CEDAR **Technical Report Number 12**

Design and Implementation of an Efficient Semantic Web Reasoner

Samir Amir and Hassan Aït-Kaci
{samir.amir;hassan.ait-kaci}@univ-lyon1.fr

October 2014

Abstract

We present an extended version of the $\mathcal{CEDAR}$ taxonomic reasoner for large ontologies. This new version provides fuller support for TBox reasoning, checking consistency, and retrieving instances. The $\mathcal{CEDAR}$ system is based upon the $\mathcal{OSF}$ formalism. It is implemented on an entirely new architecture which includes several optimization techniques. We define a bidirectional mapping between $\mathcal{OSF}$ graph structures and the Resource Description Framework (RDF) allowing a translation from $\mathcal{OSF}$ queries into SPARQL for retrieving instances from an ABox consisting of an RDF triplestore. We carried out comparative performance evaluation experiments using $\mathcal{CEDAR}$ as well as well-known Semantic Web reasoners (such as FaCT++, Pellet, HermiT, TrOWL, and RacerPro) on very large public ontologies. For the same queries on the same ontologies, the results achieved by $\mathcal{CEDAR}$ were compared to those obtained by all the other reasoners. The results of experiments show that $\mathcal{CEDAR}$ consistently performs on a par with the fastest systems for concept classification, and several orders of magnitude more efficiently in terms of response time for Boolean query-answering over attributed concepts, as well as for ABox triplestore querying. The latter result is irrespective of the triplestore management used because the $\mathcal{CEDAR}$ reasoner uses its knowledge to optimize SPARQL queries before submitting them to the triplestore.

Keywords: Ontology Classification; RDF Representation; SPARQL Query Processing; SPARQL Query Optimization.

Table of Contents

1 Introduction

For a computing device, reasoning from knowledge consists in deriving information implicit in a knowledge base. Using Semantic Web terms, this knowledge is made up of a "Terminological Box" (TBox), which contains intensional information (schema) in the form of terminological axioms describing properties of concepts, and an "Assertional Box" (ABox) which contains extensional information (data) describing specific individuals that are instances of concepts described as TBox.

There are several software systems that support some form of formal reasoning for Semantic Web (SW) applications (FaCT++,[1] HermiT,[2] Pellet,[3] TrOWL,[4] RacerPro,[5] SnoRocket,[6] to name a few). Most of these reasoners are based on Description Logic (DL) [11]. These reasoners use (some variation of) the Analytic Tableaux method as their reasoning algorithm.[7] However, as has been documented in the literature, these logics have shown their limitations in terms of their ease of use and, more importantly, as the adequacy of their inference methods for reasoning over large ontologies [15, 25, 13].

Robustness regarding scalability of reasoning is one of the issues being addressed as part of the $\mathcal{CEDAR}$ project.[8] Our approach proposes a different knowledge representation formalism from DL, and its reasoning method that is not based on Analytic Tableaux. It is based upon the Ordered-Sorted Feature ($\mathcal{OSF}$) constraint logic formalism using constrained labelled graph unification [1]. Its implementation relies on an entirely new architecture which incorporates several optimization techniques exploiting the specificity of concept taxonomies [4]. In particular, it exploits the fact that partially ordered concept taxonomies are central to all ontologies. In its initial version, the capability of the $\mathcal{CEDAR}$ reasoner was limited to pure taxonomic reasoning over unattributed concepts. Benchmarks showed that it outperformed all forecited OWL reasoners by several orders of magnitude in term of Boolean query answering on large ontologies, while its classification performance was on a par with the best among those of the tested DL-based reasoners.

The new version of the $\mathcal{CEDAR}$ reasoner we are presenting in this document is more expressive than its previous version reported in [4] and demonstrated in [10]. Most notably, while the previous version was limited to unattributed concept taxonomies, this new version allows Description Logic roles; namely, universal and existential role concepts. We demonstrate that the exact same performance improvement of the previous reasoner over the state of the art is maintained by $\mathcal{CEDAR}$ with this new expressive power.

In this new version, we have added: (1) functional attributes (called *features*); and, (2)

[1] http://owl.cs.manchester.ac.uk/fact++/
[2] http://www.hermit-reasoner.com/
[3] http://clarkparsia.com/pellet/
[4] http://trowl.eu/
[5] http://www.racer-systems.com/products/racerpro/
[6] http://research.ict.csiro.au/software/snorocket
[7] http://en.wikipedia.org/wiki/Method_of_analytic_tableaux
[8] http://cedar.liris.cnrs.fr/

set-valued features (to represent DL roles). In addition to the reasoning services provided by the previous version such as classification and cycle detection, in its current version, $\mathcal{CEDAR}$ checks as well the consistency of queries by deriving implicit necessary properties entailed by normalizing a TBox. Finally, an approach for retrieving ABox instances is proposed by compiling $\mathcal{OSF}$ queries into SPARQL. A normalized TBox is then used to optimize SPARQL queries using $\mathcal{OSF}$ reasoning.

Our experimental evaluation with the same large ontologies has shown that the new version of $\mathcal{CEDAR}$ remains among the best of all existing reasoners in term of classification and is several orders of magnitude more efficient in terms of response time for Boolean TBox reasoning on attributed concepts. $\mathcal{CEDAR}$'s reasoning method and the obtained results will be discussed in detail in this paper.

The remainder of this paper is organized as follows. In Section 2, we start with preliminaries giving an overview of relevant notions of the $\mathcal{OSF}$ formalism—especially its data structure called ψ-*term*, used to represent (semi-structured) concepts and objects. In Section 3, we describe the general architecture of the $\mathcal{CEDAR}$ reasoner for TBox reasoning and query compilation into SPARQL. In Section 4, we present and discuss the experimental results. Finally, we recapitulate our contribution and conclude in Section 5.

2 Preliminaries

2.1 Order-sorted feature constraints

In this section, we recall briefly ψ-terms that represent the basis of a logic of record structures called $\mathcal{OSF}$ Logic. The syntax of ψ-terms were proposed in [7] as flexible record structures generalizing Prolog terms and how they are used in Logic Programming, while enabling both more expressive and efficient queries using order-sorted graph unification. Indeed, ψ-terms can be seen as a generalization of first-order terms. The easiest way to describe a ψ-term is with an example, as shown in Figure 1.

```
X:professor(name ⇒ string
           ,age ⇒ 30
           ,school ⇒ Y:university(location ⇒ string
                                 ,established ⇒ 1988
                                 )
           )
```

Figure 1: Example of a ψ-term pattern in $\mathcal{OSF}$ Logic syntax

This is a ψ-term that could be used to depict a structure pattern for a "professor" object. Namely, it specifies a `professor` object pattern with and `age` of value `30`, with a `name` (a `string`), and a `school` of type `university`. This `university` object pattern specifies two parts: a `location`, of type `string`, and `established`

with value **1988**. This expression looks like a record structure. Like a typical record, it has field names; *viz.*, the symbols on the left of $\Rightarrow$. We call these *feature* symbols. In contrast with conventional records, however, ψ-terms can carry more information. Namely, they can be nested, and the fields are attached to *sort* symbols (*e.g.*, **professor**, **name**, **string**, **30**, *etc.*).[9] These sorts may indifferently denote individual values (*e.g.*, **30**) or sets of values (*e.g.*, **university**, **string**). In fact, values are assimilated to singleton-denoting sorts. Sorts are partially ordered so as to reflect set inclusion; "**professor is-a employee**" means that all instances of the sort **professor** are also instances of the sort **employee**. Node reference *tags* (*i.e.*, *variables*) such as **X** and **Y** are used in a ψ-term to indicate structure sharing or to indicate the desired bindings in queries as in Logic Programming [6]. These variables are capitalized as in Prolog. More details about $\mathcal{OSF}$ algebra and constraint-solving can be found in [1, 8, 9].

2.2 $\mathcal{OSF}$ Logic for the Semantic Web

The Resource Description Framework (RDF) is a standard notation proposed by the World Wide Web Consortium (W3C) to represent semi-structured schemas and data on the Web. An RDF structure essentially represents (sets of) labelled graphs as collections of the individual directed edges comprising them. These edges are called RDF "*triples*" since such an edge is a triplet $\langle$`subject`, `predicate`, `object`$\rangle$. The subject and object are nodes labelled by URIs, or literals also for objects; both can also be unlabelled blank nodes. The predicate is a URI labelling the link from a subject node to an object object. These triples are stored in repositories, possibly distributed over the Internet, called "*triplestores*." Therefore, an RDF dataset forms a directed, possibly distributed, labelled graph, where subjects and objects are vertices and predicates are labels on the directed edges.

As Figure 1 in the previous section depicts, the $\mathcal{OSF}$ formalism's ψ-term syntax describes data that takes the form of rooted labelled graphs. Since a ψ-term is a notation for a labelled graph, an RDF notation for it can be readily derived. Such a representation must account for differences between conventions of both notations. The essential difference is that, whereas in a ψ-term all nodes are labelled uniformly with sort symbols or values, RDF makes a difference between nodes that are labelled with URIs, blank nodes, and value nodes, and arcs are labelled with URIs. Also, rather than labelling typed nodes with their types, it uses an arc labelled with the specific URI **rdf:type** pointing to such a node (which is then a unique representation for a type, and thus shared by all so-typed nodes). Hence, an RDF representation for a ψ-term is obtained as a straightforward adaptation accounting for these differences. This is done using a set of transformation rules given explicitly in [2]. Figure 2 shows the RDF graph corresponding to the ψ-term of Figure 1.

[9] We use "sort" as a synonym of atomic "class" or "concept." In particular, sorts are partially ordered type symbols, where the ordering ("`is-a`") denotes set inclusion.

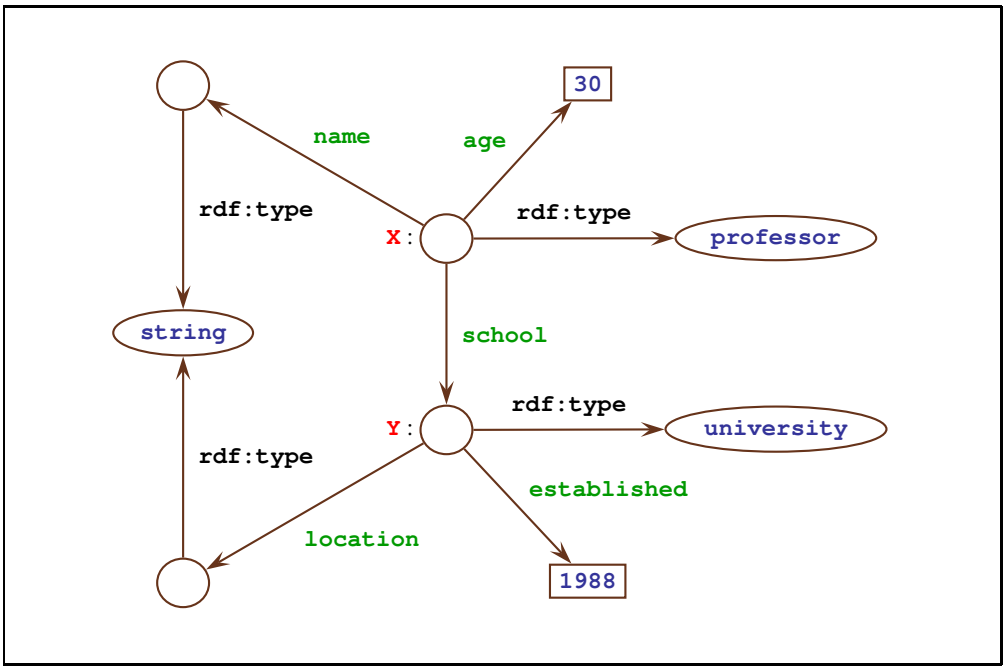


Figure 2: RDF representation for ψ-term of Figure 1

3 The $\mathcal{CEDAR}$ Reasoner

The $\mathcal{CEDAR}$ reasoner is an implementation of a limited form $\mathcal{OSF}$ Logic [1]. In its previous version, $\mathcal{CEDAR}$ was demonstrated to be among the fastest reasoners in term of classification [10]. Moreover, for Boolean query answering, $\mathcal{CEDAR}$ outperformed all OWL reasoners by several orders of magnitude [4]. Such efficiency can be explained essentially by order-sorted reasoning using binary encoding of taxonomies [5].

In the new version presented in this paper, the $\mathcal{CEDAR}$ reasoner has been extended by adding *features*, *domain/range* specifications for such features, and *aggregate sorts* denoting collections of instances of specific sorts. The specification of each constructs is discussed in Section 3.1 where a comparison with DL-based constructs is provided. $\mathcal{CEDAR}$'s architecture is shown in Figure 3. Therefore, this version is a much more complete reasoner that can process both TBox and ABox information. It starts with a classification phase performed on the TBox which includes cycle detection, transitive closure of the "`is-a`" taxonomic ordering, and feature domain/range constraint propagation down the taxonomy, and normalization of all such information [2]. Then, the encoded TBox is saved on secondary storage independently of any ABox. It is then used for query normalization and consistency checking. Finally, for retrieving instances, normalized queries are translated into SPARQL form that can be optimized using TBox knowledge. In Section 3.2, we discuss the classification and the query normalization steps and give an example of SPARQL query generation process from ψ-term query forms.

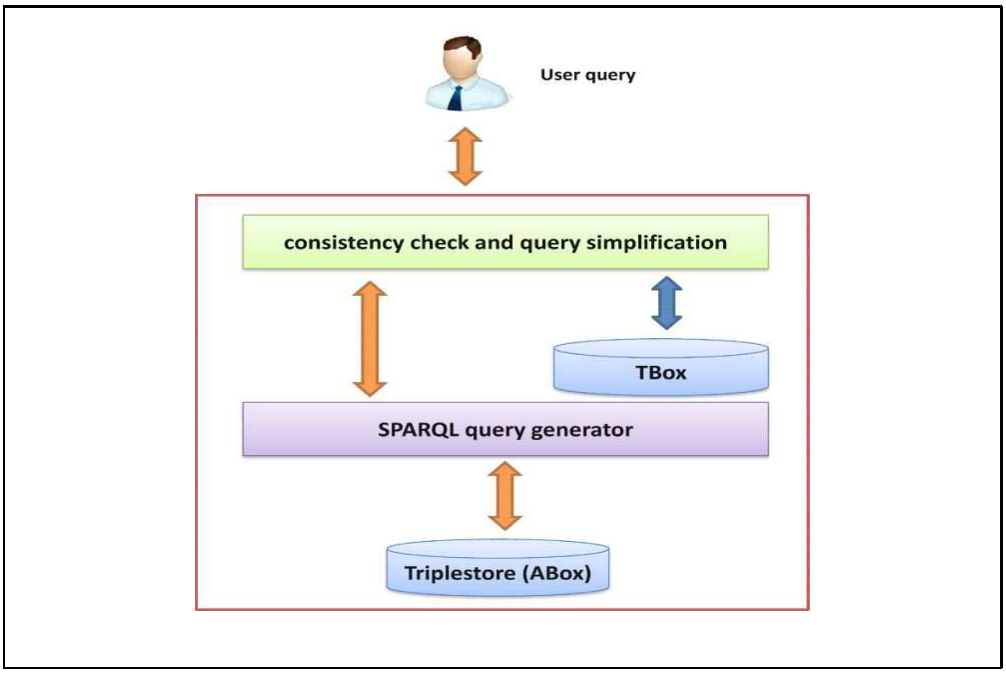


Figure 3: The $\mathcal{CEDAR}$ reasoner's general architecture

3.1 Expressing DL constructs in $\mathcal{OSF}$ Logic

3.1.1 Roles as features

The $\mathcal{OSF}$ formalism uses functional features while DL uses relational roles. A role is a binary relation. As such, it can be equivalently defined as a set-valued function. In other words, the multi values spanned by a role may be aggregated into sets. For instance, given a concept *professor* denoting a set of professors, and a concept "*course*" denoting a set of courses. Using DL semantics, we can define the role "*teaches*" with domain "*professor*" and range "*course*" to denote the set of all the pairs of professors and the courses they teach; *i.e.*, $[\![teaches]\!] \subseteq [\![professor]\!] \times [\![course]\!]$).[10]

In $\mathcal{OSF}$ Logic, the fact that a feature symbol denotes a function is crucial because $\mathcal{OSF}$ inference rules, based on graph unification, rely on it in order to be correct [9]. It is for this reason that we propose to represent a relational role as a set-valued function. Indeed, for any sets S and S', a binary relation $r \subseteq S \times S'$ can as well be seen as a set-valued function $f_r : S \to \mathcal{P}(S')$, where $\mathcal{P}(S')$ is the powerset of S', *i.e.*, the set of all subsets of S'. Formally,

$$\forall x \in S, \quad f_r(x) \stackrel{\text{def}}{=} \{\ y \in S' \mid \langle x, y \rangle \in r\ \}.$$

In order to accommodate set-valued features, basic $\mathcal{OSF}$ Logic is extended with a new syntactic construct denoting the powerset of a concept. Given a concept c, we write a

[10] The notation $[\![\ldots]\!]$ denotes the formal meaning of whatever "…" is.

set-of(c) the concept denoting the sets of subsets of $[\![c]\!]$; *i.e.*,

$$[\![\textbf{set-of}(c)]\!] \stackrel{\text{def}}{=} \mathcal{P}([\![c]\!]).$$

Considering the role "*teaches*" with domain "*professor*" and range "*course*," for example, it can also be expressed as the functional feature:

$$f_{teaches} \ : \ professor \rightarrow \textbf{set-of}(course)$$

to denote the function:

$$[\![f_{teaches}]\!] \ : \ [\![professor]\!] \rightarrow \mathcal{P}([\![course]\!])$$

that associates to a professor the set of course s/he teaches. More details on the semantics of set-valued features and can be found in [3].

3.1.2 Role-based concepts

In Description Logic, in addition to concept names comprising a taxonomy, one can also express two kinds of nameless concepts involving roles: (1) *universal role* concepts; and, (2) *existential role* concepts. Formally, given a role R and a concept C, this semantics of universal and existential roles is expressed in DL as, respectively:

$$[\![\forall R.C]\!] \stackrel{\text{def}}{=} \{\ x \in \mathcal{D} \mid \{y \in \mathcal{D} \mid \langle x, y\rangle \in [\![R]\!]\} \subseteq [\![C]\!]\ \}$$

$$[\![\exists R.C]\!] \stackrel{\text{def}}{=} \{\ x \in \mathcal{D} \mid \{y \in \mathcal{D} \mid \langle x, y\rangle \in [\![R]\!]\} \cap [\![C]\!] \neq \varnothing\ \}$$

where $\mathcal{D}$ is the domain of interpretation of all concept instances (*i.e.*, the universe of discourse). For example, the universal-role concept $\forall$*teaches*.*cs-course* will denote the set of professors that teach *only* computer-science courses. Similarly, the existential-role concept $\exists$*teaches*.*cs-course* will denote the set of professors that teach *at least one* computer-science course.

Using set-valued functional features instead of roles, these are expressed in $\mathcal{OSF}$ Logic as, respectively:

$$[\![\forall R.C]\!] \stackrel{\text{def}}{=} \{\ x \in \mathcal{D} \mid f_R(x) \subseteq [\![C]\!]\ \}$$

$$[\![\exists R.C]\!] \stackrel{\text{def}}{=} \{\ x \in \mathcal{D} \mid f_R(x) \cap [\![C]\!] \neq \varnothing\ \}.$$

3.2 TBox consistency check and query normalization

In previous work, we showed how bit-vector encoding of partially ordered sorts boosts the performance of taxonomic reasoning [4]. This was done using an efficient classification algorithm calculating the transitive closure of the partial order on sorts [5]. As demonstrated in [10], these codes yield extremely fast evaluation of Boolean concept

expressions finding the Maximal Lower Bounds (MLBs) of all concepts subsumed by such expressions. In this paper, we illustrate how they can also be used for efficient ψ-term normalization since the features refer to the binary codes corresponding to their domains and ranges.

3.2.1 Taxonomy consistency normalization

The idea is simple: given a taxonomy defining a partial order on sorts that have been encoded as bit-vectors, a feature's f domain/range declaration of the form $f : d \to r$ is propagated to a subsort s of d as follows:

- if there is no declaration for f for the sort s, then we simply install the declaration $f : s \to r$ for the sort s, and iterate the process for subsorts of s;
- if there is already a declaration for $f : s \to r'$ for the sort s, then we normalize it to be $f : s \to r \wedge r'$, where $r \wedge r'$ is the (binary code of) the conjunction of (the binary codes of) the sorts r and r'. If this code is all 0's, this means that the feature declaration is inconsistent, and so is the taxonomy.

Clearly, this process always terminates: it is in fact linear in the number of declared features and the number of subsorts of their domains. If no inconsistency is found, the resulting taxonomy is then normalized into a consistent set of feature declarations.

For example, assume that the sort ordering on sorts is such that:

researchScientist **is-a** **researcher**
researchScientist **is-a** **scientist**
scientificResearch **is-a** **research**
scientificResearch **is-a** **science**

and that we have the feature declarations:

interestedIn : **researcher** $\to$ **research**
interestedIn : **scientist** $\to$ **science**

Feature propagation brings these declarations for feature **interestedIn** from the sorts **researcher** and **scientist** down to the sort **researchScientist**, for which they are normalized into the single declaration:

interestedIn : **researchScientist** $\to$ **scientificResearch**.

This normalization results in a consistent taxonomy by coercing the range sort of the feature declaration **interestedIn** on the domain sort **researchScientist** to the most general sort that is compatible with the declarations of this feature inherited from its supersorts. If there had been no compatible range sorts for this feature, this normalization would have reported an inconsistent feature declaration. The normalization rules for taxonomy consistency check are described more formally in [2].

3.2.2 Query normalization for efficient ABox instance retrieval

Checking the consistency of a taxonomy and normalization of queries are two primordial steps that must be performed prior to submitting queries for execution—*i.e.*, before actual ABox instance retrieval. Ensuring taxonomy consistency is a process called *classification*. It computes all implicit subsumptions relationships and properties. An efficient classification algorithm was proposed in our previous work [4]. As explained above, the current version extends the classification algorithm to check also feature domain/range consistent by propagating feature declarations down the taxonomy and normalizing these declarations to be consistent, or reporting inconsistencies. A consistent normalized taxonomy (or TBox) can then be used to optimize ABox instance retrieval by normalizing queries. Here is how this is done.

Once classification is performed, the encoded ontology is saved on disk once and for all. There are three important advantages for proceeding so:

1. a saved classified ontology can be reused without the need to be reclassified for every new query sessions;
2. checking a query's consistency with respect to a classified ontology before executing it prevents useless scanning of the ABox for instance retrieval if the query is not consistent; and,
3. normalizing a query with respect to a classified ontology drastically reduces the ABox retrieval search space focusing only on relevant instances.

Let us illustrate this on an example. Figure 4 shows an ontology describing academic workers and institutions. Besides the concept taxonomy, it shows two set-valued feature declarations:

$$\texttt{teachesAt} : \texttt{teacher} \rightarrow \textbf{set-of}(\texttt{university})$$

and:

$$\texttt{worksAt} : \texttt{researcher} \rightarrow \textbf{set-of}(\texttt{researchCenter})$$

The binary codes corresponding to the taxonomy of Figure 4 are shown in Table 1.

In order to understand the process, let us consider the query Q_1 corresponding to the ψ-term:

$$\begin{aligned}\texttt{X} : \texttt{person}\,(\ &\texttt{worksAt} \Rightarrow \textbf{set-of}(\texttt{researchCenter})\\ ,\ &\texttt{teachesAt} \Rightarrow \textbf{set-of}(\texttt{university})\\ ).\end{aligned}$$

This query aims to retrieve all instances of persons teaching at a university, and working at a research center as well. Rather than submitting this query as is for retrieving instances from the ABox that verify it, we first proceed to normalize it to be consistent with the knowledge in the TBox. Doing so, we identify the sort `professor` as being

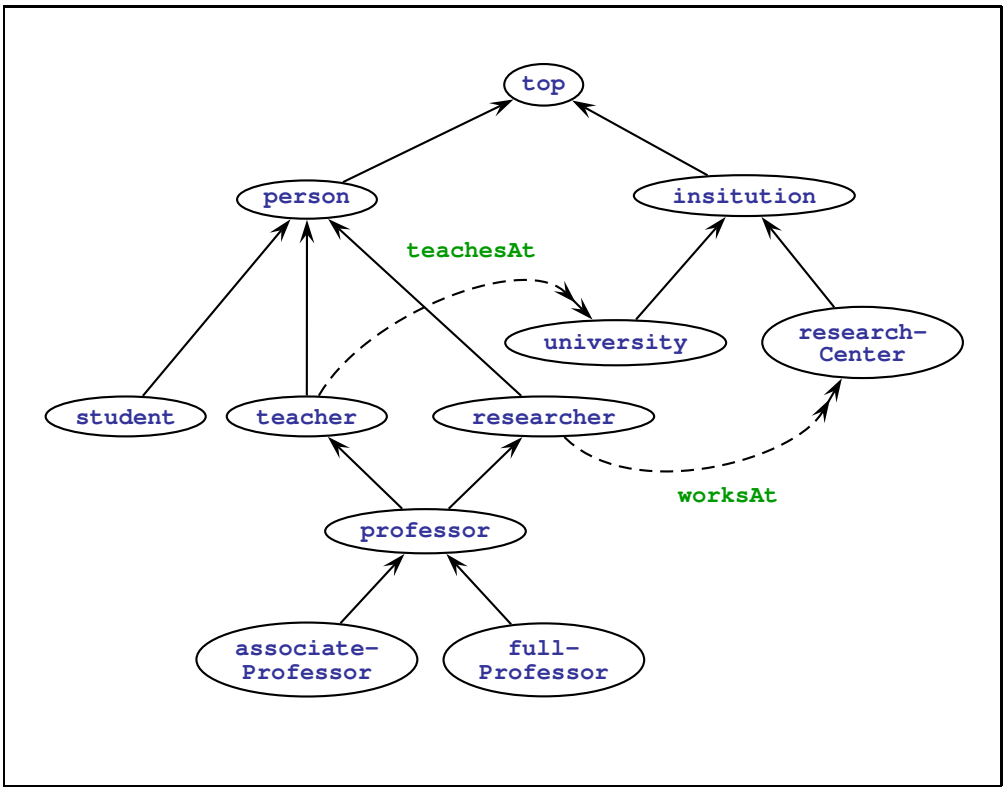


Figure 4: Example of a taxonomy with attribute features

Sort	Code
top	11111111111
institution	01110000000
university	00100000000
researchCenter	00010000000
person	00001111111
researcher	00000100111
teacher	00000010111
student	00000001000
professor	00000000111
associateProfessor	00000000010
fullProfessor	00000000001

Table 1: Binary codes for the poset shown in Figure 4

the most specific one. In fact, the sort corresponding to **professor** is the MLB of the root sort (**person**) and the sorts which are the domains of the two features presents in Q_1; namely, **researcher** and **teacher**. The MLB is the intersection (conjunction) of the binary code corresponding to **person**, **teacher** and **researcher** which are

represented by `00001111111`, `00000010111`, and `00000100111`. The result of that intersection is `00000000111`, which corresponds to the sort **professor**. Thus, the normalized query is:

```
X : professor ( worksAt ⇒ set-of(researchCenter)
              , teachesAt ⇒ set-of(university)
              ).
```

Following the same reasoning, normalizing the query Q_2 expressed as the ψ-term:

```
X : student ( worksAt ⇒ set-of(researchCenter)
            )
```

yields the inconsistent (*i.e.*, empty) sort represented by the code `00000000000`. In this case, the query is considered inconsistent with the TBox because the sort **student** has no subsort that is compatible with a known domain for the feature **worksAt**. Thus, there is no need to search the ABox for any instance of this query.

3.3 SPARQL query generation

Once a query expressed as a a ψ-term is normalized with respect to a TBox and found consistent, it is compiled into SPARQL for efficient instance retrieval. To appreciate the advantage of normalizing a query with respect to a TBox *before* submitting it for evaluation explained in Section 3.2, it is informative to to see an example.

Figure 5 shows the SPARQL query corresponding to the query Q_1 without prior normalization, and Figure 6 shows the SPARQL query for the same query after normalization.

```
SELECT  ?x
 WHERE
      {
        ?x  rdf:type   person.
        ?x  worksAt    ?y.
        ?y  rdf:type   researchCenter.
        ?x  teachesAt  ?z.
        ?z  rdf:type   university.
      }
```

Figure 5: Generated SPARQL from Q_1 without normalization

One can clearly see that the SPARQL query in the normalized format has many less constraints than the first one. Not only could the `rdf:type` of query variable **?x** be narrowed to the more specific sort **professor**, but also the domain/range feature constraints could be eliminated altogether! This is because they were already

```
SELECT  ?x
 WHERE
      {
         ?x  rdf:type  professor.
      }
```

Figure 6: Generated SPARQL from Q_1 with normalization

verified to be consistent by normalization, and since all instances in the ABox are necessarily consistent with the knowledge of the TBox (in the same manner as data in a database obey its schema), it can be safely assumed that all relevant instances of sort **professor** in the ABox *already abide by those feature constraints!* This not only reduces the search space in the ABox, but also greatly improves query evaluation by removing useless costly joins. In addition, evaluating this query can be made even more efficient if a datatype indexing is already performed by the triplestore.

Note that it is not always possible to eliminate feature constraints from a generated SPARQL query. This is the case in particular when a feature constraint in a query specifies a *value* as opposed to a sort as the range of a feature. In that case, these features must be included in the generated SPARQL query. Consider for example the query Q_3 expressed as:

```
X : person ( school ⇒ "Stanford"
           )
```

(assuming feature declaration **school** : **student** $\rightarrow$ **string**). Then, the generated SPARQL query shown in Figure 7 does normalize the sort **person** to **student**, but it must keep the feature **name** with specific value **"Stanford"**.

```
SELECT  ?x
 WHERE
      {
         ?x  rdf:type  student.
         ?x  school    "Stanford"
      }
```

Figure 7: Generated SPARQL query Q_3 with normalization and valued feature

Finally, it is important to mention that for each sort occurring in the query, sending its binary code along with the query to the triplestore allows efficient filtering of eligible answer instances. For example, for the sort **professor**, the binary code `00000000111` allows to filter instances of the sorts **associateProfessor** and **fullProfessor** as eligible answers since their sorts are subsorts of **professor**.

4 Experimental Evaluation

A comparative evaluation was conducted to assess the performance of $\mathcal{CEDAR}$ over reasoners based on the OWL API.[11] We evaluated our classification and query normalization algorithms comparing their performance with those of these reasoners, under the exact same conditions, over medium-size to very large ontologies. The characteristics in term of number of sorts and properties of the ontologies used in our experiments are shown in Table 2. We removed all axioms besides the taxonomy itself and properties attached to its concepts. Since Amphibian and NCBI are bare taxonomies, the properties for these two ontologies were generated.

Ontology	**Reference**	**# of Sorts**	**# of Properties**
Amphibian	[19]	6,135	30
MoleculeRole	[26]	9,127	7
FMA	[22]	83,283	77
CPO	[17]	136,006	55
MeSH	[20]	286,380	32
NCBI	[14]	903,617	30

Table 2: Ontologies used as benchmarks in our experiments

The ontologies described in Table 2 can only be used for TBox reasoning (classification and query normalization) because they do not contain instances. Therefore, in order to evaluate our instance-retrieval approach (explained in Section 4.3), we used the Lehigh University Benchmark suite (LUBM) [16]. The LUBM benchmark's triple generator uses a predefined ontology as a set of constraints (containing around 43 classes and 31 properties). In order to perform an evaluation using universal and existential roles, we created a new version of LUBM ontology by replacing some existential roles with universal roles.

4.1 Classification

Figure 8 shows the comparative classification time performances for each reasoner on each of the ontologies described above. Note that the SnoRocket reasoner was excluded from this comparison because it does not support the `owl:allValuesFrom` constraint [18].

From Figure 8, it can be seen that for classification $\mathcal{CEDAR}$ does not always achieve the best performance, although in five cases out six, it consistently ranks among the three reasoners that do. This can be observed particularly on the largest tested ontologies where it is roughly ten time faster than TrOWL. This latter reasoner's classification performance is incidentally always the worst of the tested systems. We suppose that this must be due to the fact that TrOWL preprocesses ontologies by first compiling them from OWL2 to OWL-QL [21]. As for FaCT++, it systematically outperforms

[11] http://owlapi.sourceforge.net

HermiT on all our tested ontologies, contrary to what is claimed in [23]. This is perhaps due to the relatively limited number of role axioms available in these ontologies. Finally, RacerPro and Pellet consistently rank as average performers compared to the rest of reasoners.

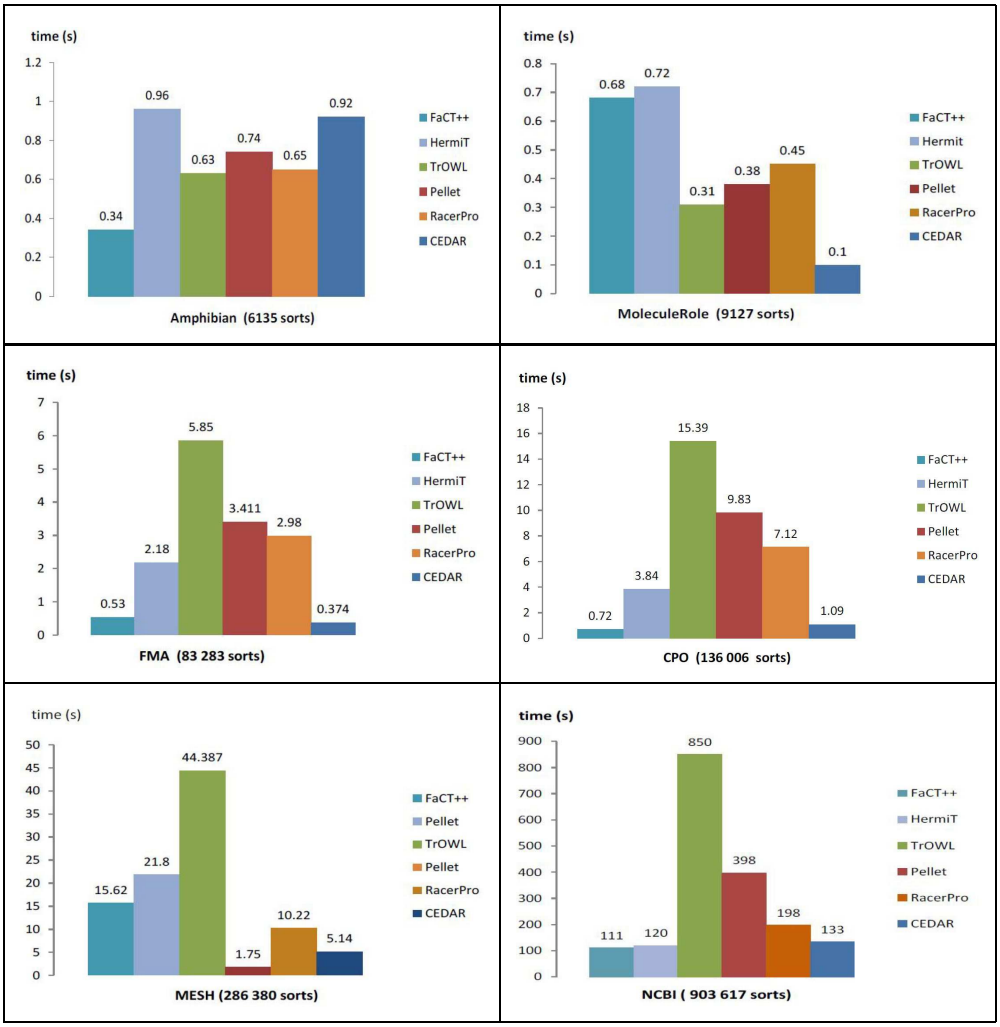


Figure 8: Classification time for all reasoners per ontology

4.2 TBox reasoning

In order to perform the evaluation of the TBox reasoning step which is necessary for query normalization, we have randomly generated a set of queries, of the form:

$$\mathtt{X} : \mathtt{s}\,(\,\mathtt{f}_1 \Rightarrow \mathtt{s}_1, \ldots, \mathtt{f}_n \Rightarrow \mathtt{s}_n\,) \text{ for } 10 \leq n \leq 100.$$

Figure 9 shows query-response time performance using logarithmic scale. One can see clearly that the $\mathcal{CEDAR}$ reasoner is systematically orders-of-magnitude faster than all

Figure 9: TBox query response time for all reasoners per ontology

the other reasoners. It can be seen also that TrOWL has a much better performance compared to those of the other Description Logic reasoners. According to our understanding, this could be explained by the fact that its preprocessing from OWL2 to OWL-QL mentioned before is not guaranteed to be faithful in that it applies some syntactic and semantic approximations [21]. Hermit, FaCT++, RacerPro and Pellet show very similar performances on medium-size ontologies. However, FaCT++ could not provide an answer before 30 minutes for large ontologies (FMA, CPO, MESH and NCBI). For NCBI, which is the largest ontology, all reasoners except $\mathcal{CEDAR}$, Pellet, and TrOWL, exceeded 30 minutes.

Although the graphs in Figure 9 speak for themselves, it is informative to get an appreciation of the relative performances for query answering of all the reasoners we have tested. Table 3 sums up the facts displayed in the graphs of Figure 9 by taking the average time over all query sizes (*viz.*, from 10 to 100 concepts), giving the maximum

Ontology	FaCT++	HermiT	TrOWL	Pellet	RacerPro	$\mathcal{CEDAR}$
Amphibian	100.00%	75.74%	3.96%	83.91%	61.61%	0.0864%
MoleculeRole	100.00%	81.61%	4.90%	31.29%	30.90%	0.0165%
FMA	40.19%	91.00%	14.76%	100.00%	15.79%	0.0073%
CPO	91.27%	100.00%	7.78%	94.02%	9.59%	0.0075%
MeSH		87.83%	9.35%	100.00%		0.0017%
NCBI			7.31%	100.00%		0.0012%

Table 3: Relative normalized average percentiles of performance times for TBox queries

of these averages for all reasoners 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). Note that, as shown in the (log-scaled) graphs, the larger the size of the ontology, the more notable the difference of performances.

4.3 ABox query answering

As for query answering, this requires both TBox reasoning and ABox instance retrieval. We selected a set of available tools for the evaluation of this step. We compared $\mathcal{CEDAR}$ with all OWL-API reasoners, as well as with other reasoners such as SPARQL-DL [24],[12] and Jena.[13] We conducted our test of ABox querying on the following two queries we defined according to LUBM ontology.

Query 1:

$$\mathtt{X} : \mathtt{person}\,(\,\mathtt{takesCourse} \Rightarrow \textbf{set-of}(\mathtt{course}))$$

Query 2:

$$\begin{aligned}\mathtt{X} : \mathtt{person}\,(\,&\mathtt{worksFor} \Rightarrow \textbf{set-of}(\mathtt{organization})\\ ,\,&\mathtt{headOf} \Rightarrow \textbf{set-of}(\mathtt{department})\\ ).&\end{aligned}$$

Figures 10 and 11 show query-answering times per query and reasoner. Each figure shows the performance of the reasoners for ABox querying on a medium-size dataset up to 100 thousand triples (the graphs on the left), and on a large dataset up to 5 million triples (the graphs on the right). Due to the significant difference in response time between HermiT and the rest of the SW reasoners, the graphs corresponding to the medium-size dataset (on the left-hand side) use a logarithmic scale. For the large-size dataset, none of the OWL-API reasoners could handle more that 1 million triples. Only three could hold the load up to 5 million triples: (1) Jena with its own reasoner; (2) Jena with the $\mathcal{CEDAR}$ reasoner; and (3) $\mathcal{CEDAR}$ with its own instance retrieval method using a type-indexed triplestore, as explained in Section 4.4.

[12] http://www.derivo.de/en/resources/sparql-dl-api.html
[13] http://jena.apache.org/

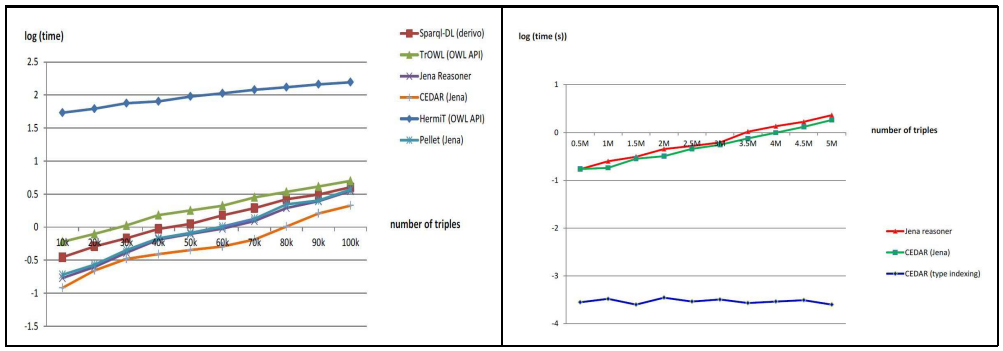


Figure 10: Response time for the first ABox query

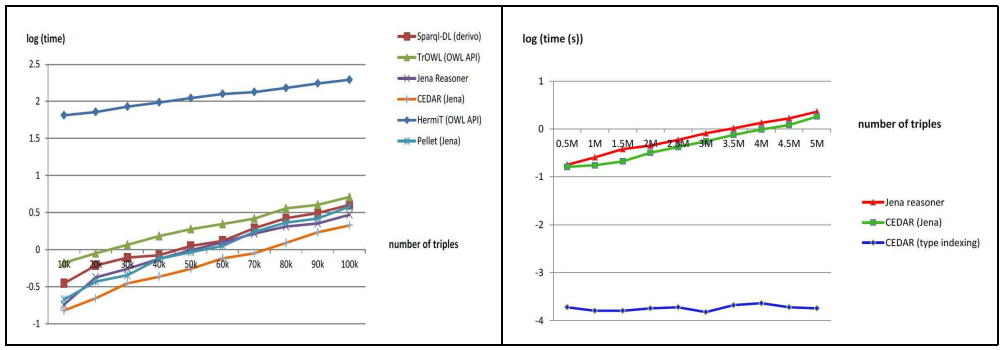


Figure 11: Response time for the second ABox query

These figures show that HermiT has the poorest perfomance for ABox query answering, even among OWL-API reasoners. It took more than 60 seconds for a dataset whose size is 10 thousand of triples and could not provide an answer before a time-out period that we set to 30 minutes. TrOWL is the best among OWL-API reasoners in term of performance on the medium-size dataset. However, it failed at the classification step for the large ontologies where the cost in term of memory was around 15Gb for only 0.5 million of triples. For the same ontologies, other OWL-API reasoners (FaCT++ and Pellet) failed to give an answer. SPARQL-DL also showed its limits for the large dataset. Such a result was not surprising since that SPARQL-DL engine also uses the OWL-API; hence its behavior similar to the other OWL-API reasoners.

Pellet coupled with Jena, on the other hand, showed relatively good performance, Jena for the large dataset, its performances are close to those of Jena internal reasoner. This may be due to the fact that Pellet is implemented differently when used with Jena. Jena with its own reasoner is the best among all tested, but never outperforms the $\mathcal{CEDAR}$ reasoner.

Finally, as can be clearly seen in Figures 10 and 11, the $\mathcal{CEDAR}$ reasoner and SPARQL query generator, using Jena as SPARQL query evaluator, systematically achieves the best performances of all the systems tested in term of query answering time. This is true even without indexing the set of triples (*i.e.*, by fully relying on Jena's triplestore

ABox Query	**Jena Reasoner (Jena Query Manager)**	$\mathcal{CEDAR}$ **Reasoner (Jena Query Manager)**	$\mathcal{CEDAR}$ **Reasoner (Type-Indexed Triples)**
Query 1	100.00%	78.8%	0.042%
Query 2	100.00%	93.1%	0.019%

Table 4: Relative normalized average percentiles of performance times for ABox queries

manager). However, just to have an idea, we implemented our own method taking advantage of encoded sorts to preprocess the set of triples to be queried into a type-indexed partitioned set. As shown in the figures, the results were simply staggeringly better. The comparative normalized relative percentiles for the right-hand side graphs are shown in Table 4. We did not report the OWL-API's performance in these percentiles since the difference would simply be too great.

4.4 Discussion

The results reported in this document show that $\mathcal{CEDAR}$ performs well for classification, and very well for TBox reasoning and query answering. In its first version, the $\mathcal{CEDAR}$ taxonomic reasoner already showed very good performance in terms of classification. In the current version extending taxonomic reasoning to include functional and relational attributes, classification also propagates feature domain/range declarations down the taxonomy and normalizes them to be consistent. As demonstrated, this new capability turns out to be not so expensive in terms of time since it takes advantage of the fact that this is performed on the taxonomy that has already been encoded. For each feature, its domain sorts are identified by the binary codes of the original sort for which this feature was declared as well as its subsorts. Static propagation and normalization is performed once an for all and no lookup is ever needed thereafter for a feature's domain and range.

The results reported in Section 4.3 show that all reasoners based on the OWL-API fail on the large dataset we used in our tests. This was not surprising to us because this confirmed similar evaluations done by others (*e.g.*, [25]). We are convinced that TBox-based query normalization enabled by our attributed-taxonomy classification technique is the key boosting the efficiency of our reasoner. The generated SPARQL queries are thus optimized using reasoning based on the knowledge represented in the TBox, itself made efficient by feature propagation and normalization. This reasoning enables generating queries so that a SPARQL engine executing them can sidestep generic constraint-checking which is done statically once and for all, narrowing the search to relevant triples only. This is in contrast with the fact that other reasoners, even when coupled with an efficient SPARQL engine like Jena, actually use their deductive power to *generate more facts* from already existing triples, thereby increasing the number of facts to retrieve from the triplestore before filtering the relevant ones. Finally, $\mathcal{CEDAR}$ performs its TBox normalization statically and saves it on disk once and for all. This is an ideal solution for its use in a dynamic environment where the ABox changes regularly while the TBox does not.

While there are many triplestores available in the world, we restricted our study to

those using the OWL API, SPARQL-DL, and Jena. This is because our goal was not to evaluate triplestores, but how existing Semantic Web reasoners perform with triplestores. In fact, these triplestores do not vary much in the TBox reasoners they rely on, but they do when it comes to optimizing queries by applying several techniques such as indexing, hashing, *etc.*, . . . , irrespectively of the reasoners using them. For our instance retrieval tests, we used Jena as the SPARQL engine to use with the $\mathcal{CEDAR}$ reasoner since it is widely used by many other Semantic Web reasoners.

In order to explore the magnitude of improvement yet possible if the underlying triplestore had the means to index its sets of triples taking advantage of the taxonomic knowledge available at query-generation time, we experimented with our own method for indexing a set of triples to be queried using sort encoding. Thus, we implemented a simple type-indexing scheme taking advantage of the bit-vector encoding. We organized the ABox in memory so that triples of a given sort ("root" triples of this sort) are all stored contiguously. Then, in the taxonomy array containing each sort and its properties (such as name, binary code, *etc.*), we added two integer fields: one indicating the ABox index of the first triple of this sort, and the other indicating the last such index. In this way, it is possible to iterate only over those triples in the ABox sorted with subsorts of a given query simply by using the binary code of the query's root sort. This is possible since its "`1`" bits' positions correspond to the indices of its subsorts in the taxonomy array [4]. Thus, the relevant ranges of triples stored at these indices in this array are readily accessible. Such an indexing scheme is illustrated in Figure 12.

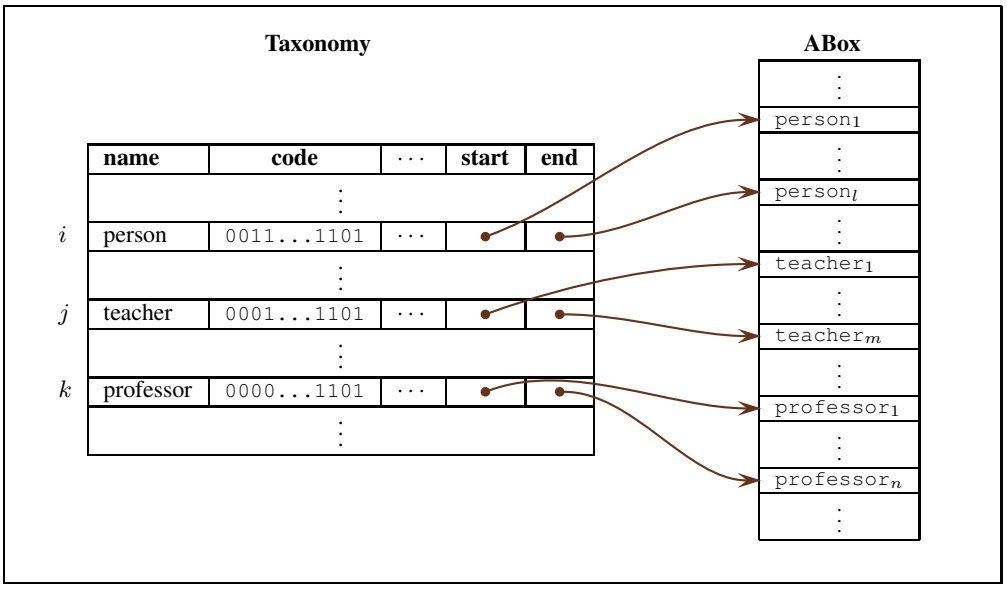


Figure 12: ABox indexing scheme using sort encoding

Of course, when used, indexing is performed once and for all as an offline step, to be reused on the same ABox as often as needed. By using such a scheme, the results we obtained in this experiment for the same queries on the same (indexed) set of triples showed that performance of query processing could be further divided by a factor in the

order of thousands. This indicates that building type-indexing using encoded sorts in an actual triplestore management is likely to provide similar results—even if lessened to a factor of tens for complex queries.

5 Conclusion

In this document, we have presented an implemention of a new version of the $\mathcal{CEDAR}$ Semantic Web reasoner supporting reasoning with taxonomies with simple and set-valued functional features, the latter expressing relational roles as used in Description Logic. A comparative evaluation of $\mathcal{CEDAR}$ was carried out along with several of the most reputed Semantic Web reasoners. Experiments showed that $\mathcal{CEDAR}$'s performance is consistently among the best SW reasoners in terms of classification, and several orders-of-magnitude better in terms of TBox reasoning. $\mathcal{CEDAR}$ exploits the fact that its underlying logic, Order-Sorted Feature Logic [9], is based on labelled graph structures—called ψ-terms—that can be straightforwardly mapped into RDF format. We illustrated how to transform ψ-term queries that are normalized using a TBox knowledge, itself normalized, into optimized SPARQL queries. This TBox normalization ensures consistency of the TBox knowldedge, which in turn can be used to normalize the generated SPARQL queries to reduce the search space of the SPARQL engine.

As for the future, we are extending this work to enable $\mathcal{CEDAR}$ to support more complex ontologies and queries such as disjunction or filtering. Since full $\mathcal{OSF}$ Logic is a powerful knowledge description and reasoning formalism that goes beyond simple conjunctive queries, the main part of the work to be done is to adapt its operational semantics to the context of Semantic Web. We are also working on developing our own distributed triplestore management system for efficient instance retrieval and advanced indexing methods from secondary-storage or networked data [12].

References

[1] Hassan Aït-Kaci. Data models as constraint systems—a key to the Semantic Web. *Constraint Processing Letters*, 1:33–88, November 2007. [See online[14]].

[2] Hassan Aït-Kaci. $\mathcal{HOOT}$: A language for expressing and querying hierarchical ontologies, objects, and types—a specification. CEDAR Technical Report Number 16, CEDAR Project, LIRIS, Département d'Informatique, Université Claude Bernard Lyon 1, Villeurbanne, France, December 2014. [See online[15]].

[3] Hassan Aït-Kaci. A set-complete domain construction for order-sorted set-valued features. CEDAR Technical Report Number 11, CEDAR Project, LIRIS, Département d'Informatique, Université Claude Bernard Lyon 1, Villeurbanne, France, October 2014. [See online[16]].

[14] http://cs.brown.edu/people/pvh/CPL/Papers/v1/hak.pdf
[15] http://cedar.liris.cnrs.fr/papers/ctr16.pdf
[16] http://cedar.liris.cnrs.fr/papers/ctr11.pdf

[4] Hassan Aït-Kaci and Samir Amir. Classifying and querying very large taxonomies—a comparative study to the best of our knowledge. CEDAR Technical Report Number 2, $\mathcal{CEDAR}$ Project, LIRIS, *Département d'Informatique, Université Claude Bernard Lyon 1*, Villeurbanne, France, May 2013. [See online[17]].

[5] 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[18]].

[6] Hassan Aït-Kaci, Bruno Dumant, Richard Meyer, Andreas Podelski, and Peter Van Roy. The Wild LIFE handbook (prepublication edition). [See online[19]], 1994.

[7] Hassan Aït-Kaci and Roger Nasr. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986. [See online[20]].

[8] Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. In Jan Maluszyński and Martin Wirsing, editors, *Proceedings of the 3rd International Symposium on Programming Language Implementation and Logic Programming (Passau, Germany)*, number 528 in Lecture Notes in Computer Science, pages 255–274. Springer-Verlag, August 1991. [See online[21]].

[9] Hassan Aït-Kaci and Andreas Podelski. Logic programming with functions over order-sorted feature terms. In Evelina Lamma and Paola Mello, editors, *Proceedings of the 3rd International Workshop on Extensions of Logic Programming (Bologna, Italy)*, pages 100–119. LNAI 660, Springer-Verlag, February 1992. [See online[22]].

[10] Samir Amir and Hassan Aït-Kaci. $\mathcal{CEDAR}$: a fast taxonomic reasoner based on lattice operations—system demonstration. In Eva Blomqvist and Tudor Groza, editors, *Proceedings of the Posters & Demonstrations Track of the 12th International Semantic Web Conference*, pages 9–12, Sydney, Australia, October 2013. CEUR Workshop Proceedings. [See online[23]].

[11] Franz Baader, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003. [See online[24]].

[12] Minwei Chen, Rafiqul Haque, and Mohand-Saïd Hacid. CedTMart—a triplestore for storing and querying blinked data. CEDAR Technical Report Number 7, $\mathcal{CEDAR}$ Project, LIRIS, *Département d'Informatique, Université Claude Bernard Lyon 1*, Villeurbanne, France, July 2014. [See online[25]].

[13] Kathrin Dentler, Ronald Cornet, Annette ten Teije, and Nicolette de Keizer. Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semantic Web Journal*, 2(2):1–5, April 2011. [See online[26]].

[14] Scott Federhen. The NCBI taxonomy database. *Nucleic Acids Research*, 40:136–143, January 2012. [See online[27]].

[17] http://cedar.liris.cnrs.fr/papers/ctr2.pdf
[18] http://www.hassan-ait-kaci.net/pdf/encoding-toplas-89.pdf
[19] http://hassan-ait-kaci.net/pdf/WildLIFE-HANDBOOK.pdf
[20] http://hassan-ait-kaci.net/pdf/login-jlp-86.pdf
[21] http://hassan-ait-kaci.net/pdf/meaningoflife.pdf
[22] http://hassan-ait-kaci.net/pdf/elp-92.pdf
[23] http://ceur-ws.org/Vol-1035/iswc2013_demo_3.pdf
[24] http://cdn.preterhuman.net/.../The Description Logic Handbook...(2003).pdf
[25] http://cedar.liris.cnrs.fr/interns/MinweiChen/ctr7.pdf
[26] http://www.semantic-web-journal.net/sites/default/files/swj120_2.pdf
[27] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3245000/pdf/gkr1178.pdf

[15] John Goodwin. Experiences of using OWL at the ordnance survey. In Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Peter Patel-Schneider, editors, *Proceedings of the OWLED'05 Workshop on OWL: Experiences and Directions*, Galway, Ireland, November 11–12, 2005. CEUR Workshop Proceedings Vol. 188. [See online[28]].

[16] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2–3):158–182, July 2005. [See online[29]].

[17] Robert Hoehndorf, Midori A. Harris, Heinrich Herre, Gabriella Rustici, and Georgios V. Gkoutos. Semantic integration of physiology phenotypes with an application to the Cellular Phenotype Ontology. *Bioinformatics*, 28(13):1783–1789, April 2012. [See online[30]].

[18] 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 (AOW 2010)*, pages 45–50, Adelaide, Australia, December 2010. ACS. [See online[31]].

[19] Anne M. Maglia, Jennifer L. Leopold, L. Analía Pugener, and Susan Gauch. An anatomical ontology for amphibians. In Russ B. Altman, A. Keith Dunker, Lawrence Hunter, Tiffany Murray, and Teri E. Klein, editors, *Pacific Symposium on Biocomputing*, pages 367–378. World Scientific, January 2007. [See online[32]].

[20] U.S. National Library of Medicine. Medical subject headings MeSH. Web Site. [See online[33]].

[21] Jeff Z. Pan, Yuan Ren, Nophadol Jekjantuk, and Jhonatan Garcia. Reasoning the FMA ontologies with TrOWL. In Samantha Bail, Birte Glimm, Rafael S. Gonçalves, Ernesto Jiménez-Ruiz, Yevgeny Kazakov, Nicolas Matentzoglu, and Bijan Parsia, editors, *Proceedings of 2nd International Workshop on OWL Reasoner Evaluation (ORE 2013)*, pages 107–113. CEUR Workshop Proceedings, July 2013. [See online[34]].

[22] Cornelius Rosse and José L.V. Mejino Jr. A reference ontology for biomedical informatics: the foundational model of anatomy. *Journal of Biomedical Informatics*, 36(6):478–500, December 2003. [See online[35]].

[23] 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[36]].

[24] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL query for OWL-DL. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *Proceedings of the Workshop on OWL: Experiences and Directions (OWLED 2007)*, Innsbruck, Austria, June 2007. CEUR Workshop Proceedings. [See online[37]].

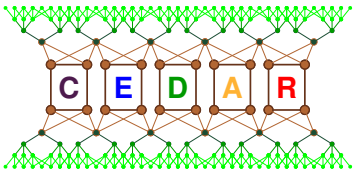
[28] http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-188/sub17.pdf
[29] http://www.websemanticsjournal.org/index.php/ps/article/download/70/68
[30] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3381966/pdf/bts250.pdf
[31] http://krr.meraka.org.za/~aow2010/Lawley-etal.pdf
[32] http://psb.stanford.edu/psb-online/proceedings/psb07/maglia.pdf
[33] http://www.nlm.nih.gov/mesh/meshhome.html
[34] http://ceur-ws.org/Vol-1015/paper_18.pdf
[35] http://www.j-biomed-inform.com/article/S1532-0464(03)00127-8/pdf
[36] http://www.cs.ox.ac.uk/ian.horrocks/Publications/download/2008/ShMH08b.pdf
[37] http://ceur-ws.org/Vol-258/paper14.pdf

[25] Kavitha Srinivas. OWL reasoning in the real world: Searching for Godot. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proceedings of the 22nd International Workshop on Description Logics (DL 2009)*, Oxford, United Kindgom, July 27–30 2009. CEUR Workshop Proceedings. Invited lecture [See online[38]].

[26] Satoko Yamamoto, Takao Asanuma, Toshihisa Takagi, and Ken Ichiro Fukuda. The Molecule Role Ontology: An ontology for annotation of signal transduction pathway molecules in the scientific literature. *Comparative and Functional Genomics*, 5(6–7):528–536, October 2004. [See online[39]].

Acknowledgements: The authors thank Mohand-Saïd Hacid and Rafiqul Haque for their kind proofreading and comments. All remaining mistakes, if any, are the authors' responsibility.

[38] http://www.cs.ox.ac.uk/DL2009/proceedings/invited/Srinivas.pdf
[39] http://downloads.hindawi.com/journals/ijg/2004/439679.pdf



Technical Report Number 12

Design and Implementation
of an
Efficient Semantic Web Reasoner

Samir Amir and Hassan Aït-Kaci

October 2014