Classifying $\mathcal{ELH}$ Ontologies in SQL Databases

Vincent Delaitre[1] and Yevgeny Kazakov[2]

[1] École Normale Supérieure de Lyon, Lyon, France
`vincent.delaitre@ens-lyon.org`
[2] Oxford University Computing Laboratory, Oxford, England
`yevgeny.kazakov@comlab.ox.ac.uk`

Abstract. The current implementations of ontology classification procedures use the main memory of the computer for loading and processing ontologies, which soon can become one of the main limiting factors for very large ontologies. We describe a secondary memory implementation of a classification procedure for $\mathcal{ELH}$ ontologies using an SQL relational database management system. Although secondary memory has much slower characteristics, our preliminary experiments demonstrate that one can obtain a comparable performance to those of existing in-memory reasoners using a number of caching techniques.

1 Introduction

The ontology languages OWL and OWL 2 based on description logics (DL) are becoming increasingly popular among ontology developers, largely thanks to the availability of *ontology reasoners* which provide automated support for many ontology development tasks. One of the key ontology development task is *ontology classification*, the goal of which is to compute a hierarchical representation of the subsumption relation between classes of the ontology called *class taxonomy*.

The popularity of OWL and efficiency of ontology reasoners has resulted in the availability of large ontologies such as Snomed CT containing hundreds of thousands of classes. Modern ontology reasoners such as CEL and FaCT++ can classify Snomed CT in a matter of minutes, however scaling these reasoners to ontologies containing millions of classes is problematic due to their high memory consumption. For example, the classification of Snomed CT in CEL and FaCT++ requires almost 1GB of the main memory. In this paper we describe a secondary-memory implementation of the classification procedure for the DL $\mathcal{ELH}$—the fragment of OWL used by Snomed CT—in SQL databases. Our reasoner can classify Snomed CT in less than 20 minutes using less than 32MB of RAM.

To the best of our knowledge, secondary-memory (database) algorithms and optimizations for ontology reasoning have been studied only very recently. Lutz et al. have proposed a method for conjunctive query answering over $\mathcal{EL}$ ontologies using databases [1], but the main focus of their work is on optimizing query response time once the ontology is "compiled" into a database. The IBM SHER system (`www.alphaworks.ibm.com/tech/sher`) has a "db-backed" module, which presumably can perform secondary-memory reasoning in $\mathcal{EL}^+$, but currently not much information is available about this module.

Table 1. The syntax and semantics of $\mathcal{ELH}$

Name	Syntax	Semantics
Concepts:		
atomic concept	A	$A^{\mathcal{I}}$ (given)
top concept	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : \langle x, y\rangle \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
Axioms:		
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
role inclusion	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$

2 Preliminaries

In this section we introduce the lightweight description logic $\mathcal{ELH}$ and the classification procedure for $\mathcal{ELH}$ ontologies [2].

2.1 The Syntax and Semantics of $\mathcal{ELH}$

A description logic *vocabulary* consists of countably infinite sets N_C of *atomic concepts*, and N_R of *atomic roles*. The syntax and semantics of $\mathcal{ELH}$ is summarized in Table 1. The set of $\mathcal{ELH}$ *concepts* is recursively defined using the constructors in the upper part of Table 1, where $A \in \mathsf{N}_C$, $r \in \mathsf{N}_R$, and C, D are concepts. A *terminology* or *ontology* is a set $\mathcal{O}$ of axioms in the lower part of Table 1 where C, D are concepts and $r, s \in \mathsf{N}_R$. We use *concept equivalence* $C \equiv D$ as an abbreviation for two concept inclusion axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.

The semantics of $\mathcal{ELH}$ is defined using interpretations. An *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set called *the domain of the interpretation* and $\cdot^{\mathcal{I}}$ is the *interpretation function*, which assigns to every $A \in \mathsf{N}_C$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and to every $r \in \mathsf{N}_R$ a relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation is extended to concepts according to the right column of Table 1. An interpretation $\mathcal{I}$ *satisfies an axiom* α (written $\mathcal{I} \models \alpha$) if the respective condition to the right of the axiom in Table 1 holds; $\mathcal{I}$ is a *model of an ontology* $\mathcal{O}$ (written $\mathcal{I} \models \mathcal{O}$) if $\mathcal{I}$ satisfies every axiom in $\mathcal{O}$. We say that α is a *(logical) consequence* of $\mathcal{O}$, or *is entailed* by $\mathcal{O}$ (written $\mathcal{O} \models \alpha$) if every model of $\mathcal{O}$ satisfies α.

Classification is a key reasoning problem for description logics and ontologies, which requires to compute the set of *direct subsumptions* $A \sqsubseteq B$ between atomic concepts that are entailed by $\mathcal{O}$. A subsumption $A \sqsubseteq B$ is *direct* if there is no atomic concept C such that $\mathcal{O} \models A \sqsubseteq C$ and $\mathcal{O} \models C \sqsubseteq B$, unless C is equivalent to either A or B.

2.2 Normalization of $\mathcal{ELH}$ Ontologies

Normalization is a preprocessing stage that eliminates nested occurrences of complex concept from $\mathcal{ELH}$ ontologies using auxiliary atomic concepts and

$$\textsf{IR1}\ \frac{}{A \sqsubseteq A} \qquad \textsf{IR2}\ \frac{}{A \sqsubseteq \top} \qquad \textsf{CR1}\ \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$$

$$\textsf{CR2}\ \frac{A \sqsubseteq B \quad A \sqsubseteq C}{A \sqsubseteq D} : B \sqcap C \sqsubseteq D \in \mathcal{O} \qquad \textsf{CR3}\ \frac{A \sqsubseteq B}{A \sqsubseteq \exists r.C} : B \sqsubseteq \exists r.C \in \mathcal{O}$$

$$\textsf{CR4}\ \frac{A \sqsubseteq \exists r.B}{A \sqsubseteq \exists s.B} : r \sqsubseteq s \in \mathcal{O} \qquad \textsf{CR5}\ \frac{A \sqsubseteq \exists s.B \quad B \sqsubseteq C}{A \sqsubseteq D} : \exists s.C \sqsubseteq D \in \mathcal{O}$$

Fig. 1. The Completion Rules for $\mathcal{ELH}$

axioms. The resulting ontology will contain only axioms of the forms:

$$A \sqsubseteq B, \quad A \sqcap B \sqsubseteq C, \quad A \sqsubseteq \exists r.B, \quad \exists s.B \sqsubseteq C, \quad r \sqsubseteq s, \tag{1}$$

where $A, B, C \in \mathsf{N}_C$ and $r, s \in \mathsf{N}_R$. Given an $\mathcal{ELH}$ concept C, let $\mathsf{sub}(C)$ be the set of *sub-concepts* of C recursively defined as follows: $\mathsf{sub}(A) = \{A\}$ for $A \in \mathsf{N}_C$, $\mathsf{sub}(\top) = \{\top\}$, $\mathsf{sub}(C \sqcap D) = \{C \sqcap D\} \cup \mathsf{sub}(C) \cup \mathsf{sub}(D)$, and $\mathsf{sub}(\exists r.C) = \{\exists r.C\} \cup \mathsf{sub}(C)$. Given an $\mathcal{ELH}$ ontology $\mathcal{O}$, define the set of *negative / positive / all concepts in* $\mathcal{O}$ by respectively $\mathsf{sub}^-(\mathcal{O}) = \{\mathsf{sub}(C) \mid C \sqsubseteq D \in \mathcal{O}\}$, $\mathsf{sub}^+(\mathcal{O}) = \{\mathsf{sub}(D) \mid C \sqsubseteq D \in \mathcal{O}\}$, and $\mathsf{sub}(\mathcal{O}) = \mathsf{sub}^-(\mathcal{O}) \cup \mathsf{sub}^+(\mathcal{O})$. For every $C \in \mathsf{sub}(\mathcal{O})$ we define a function $\mathsf{nf}(C)$ as follows: $\mathsf{nf}(A) = A$, if $A \in \mathsf{N}_C$, $\mathsf{nf}(\top) = \top$, $\mathsf{nf}(C \sqcap D) = A_C \sqcap A_D$, and $\mathsf{nf}(\exists r.C) = \exists r.A_C$, where A_C and A_D are fresh atomic concepts introduced for C and D.

The result of applying *normalization* to $\mathcal{O}$ is the ontology $\mathcal{O}'$ consisting of the following axioms: (*i*) $A_C \sqsubseteq A_D$ for $C \sqsubseteq D \in \mathcal{O}$, (*ii*) $r \sqsubseteq s$ for $r \sqsubseteq s \in \mathcal{O}$, (*iii*) $\mathsf{nf}(C) \sqsubseteq A_C$ for $C \in \mathsf{sub}^-(\mathcal{O})$, and (*iv*) $A_D \sqsubseteq \mathsf{nf}(D)$ for $D \in \mathsf{sub}^+(\mathcal{O})$, where the axioms of the form $A \sqsubseteq B \sqcap C$ are replaced with a pair of axioms $A \sqsubseteq B$ and $A \sqsubseteq C$. It has been shown [2] that this transformation preserves all original subsumptions between atomic concepts in $\mathcal{O}$.

2.3 Completion Rules

In order to classify a normalized $\mathcal{ELH}$ ontology $\mathcal{O}$, the procedure applies the completion rules in Fig. 1. The rules derive new axioms of the form $A \sqsubseteq B$ and $C \sqsubseteq \exists r.D$ which are logical consequences of $\mathcal{O}$, where A, B, C, and D are atomic concepts or $\top$, and r, s atomic roles. Rules **IR1** and **IR2** derive trivial axioms for $A \in \mathsf{N}_C \cap \mathsf{sub}(\mathcal{O})$. The remaining rules are applied to already derived axioms and use the normalized axioms in $\mathcal{O}$ as side conditions. The completion rules are applied until no new axiom can be derived, i.e., the resulting set of axioms is closed under all inference rules. It has been shown [2] that the rules **IR1**–**CR5** are *sound* and *complete*, that is, a concept subsumption $A \sqsubseteq B$ is entailed by $\mathcal{O}$ if and only if it is derivable by these rules.

3 Implementing $\mathcal{ELH}$ Classification in a Database

In this section we describe the basic idea of our database implementation for the $\mathcal{ELH}$ classification procedure, the performance problems we face with, and our

at_conc

name	id
MuscularOrgan	1
Organ	2
MuscularSystem	4
Heart	6
CirculatorySystem	10

at_role

name	id
isPartOf	3
belongsTo	8

conj

$A\ \sqcap$	B	pos	neg	id
2	5	✓	✓	1
4	10	✓		11
2	12	✓		13

exis

$\exists\ r.$	B	pos	neg	id
3	4	✓	✓	5
8	11	✓		12

ax_c_incl

$A\ \sqsubseteq$	B
1	2
1	5
11	4
11	10
13	2
13	12
6	13

ax_r_incl

$r\ \sqsubseteq$	s
8	3

ax_conj

$A\ \sqcap$	$B\ \sqsubseteq$	C
2	5	1

ax_exis_pos

$A\ \sqsubseteq\ \exists$	$r.$	B
5	3	4
12	8	11

ax_exis_neg

$\exists\ s.$	$B\ \sqsubseteq$	C
3	4	5

Fig. 2. Storing $\mathcal{ELH}$ concepts and normalized axioms in a database: the first part of every table is introduced for axiom (2), the second part is introduced for axiom (3).

solutions to these problems. Although our presentation is specific to the MySQL syntax, the procedure can be implemented in any other SQL database system.

3.1 The Outline of the Approach

In this section we give a high level description of the $\mathcal{ELH}$ classification procedure using databases. We use the $\mathcal{ELH}$ ontology $\mathcal{O}$ consisting of axioms (2)–(4) as a (rather artificial but simple) running example.

$$\mathsf{MuscularOrgan} \equiv \mathsf{Organ} \sqcap \exists \mathsf{isPartOf}.\mathsf{MuscularSystem} \tag{2}$$

$$\mathsf{Heart} \sqsubseteq \mathsf{Organ} \sqcap \exists \mathsf{belongsTo}.(\mathsf{MuscularSystem} \sqcap \mathsf{CirculatorySystem}) \tag{3}$$

$$\mathsf{belongsTo} \sqsubseteq \mathsf{isPartOf} \tag{4}$$

The first step of the procedure is to assign integer identifiers (ids) to every concept and role occurring in the ontology, in order to use them for computing normalized axioms (1). In order to represent all information about concepts, it is sufficient to store, for every concept, its topmost constructor together with the ids of its direct sub-concepts. Thus, all information about concepts and roles can be represented in a database using four tables corresponding to possible types of constructors shown in Fig. 2, left: `at_conc` for atomic concepts, `at_role` for atomic roles, `conj` for conjunctions, and `exis` for existential restrictions. The assignment of ids is optimized so that the same ids are assigned to concepts that are structurally equivalent or declared equivalent using axioms such as (2).

Apart from the ids, for every complex concept we store flags indicating whether the concept occurs positively and/or negatively in the ontology. The polarities of concepts are used consequently to identify normal forms of axioms which are stored in five tables corresponding to five types of normal forms (1) (see Fig. 2, right). Table `ax_c_incl` stores inclusions between concepts: for every row (A, B, pos, neg, id) in table `conj` with pos = `true`, table `ac_c_incl` contains inclusions between id and A as well as between id and B; for every concept inclusion $C \sqsubseteq D$ in $\mathcal{O}$ the table also contains the inclusion between the id for C and the id for D. Similarly, table `ax_r_incl` contains inclusion between ids for role inclusions $r \sqsubseteq s$ in $\mathcal{O}$. Table `ax_conj` is obtained from the rows of table `conj` with negative flag. Likewise, tables `ax_exis_pos` and `ax_exis_neg` are obtained from the rows of `exis` with respectively positive and negative flags.

Once the tables for normal form of the axioms are computed, it is possible to apply inference rules in Fig. 1 using SQL joins to derive new subsumptions. In the next sections we describe both steps of the classification procedure in detail.

3.2 Normalization

The first problem appears when trying to assign the same ids to the same concept occurring in the ontology. Since our goal is to design a scalable procedure which can deal with very large ontologies, we cannot load the whole ontology into the main memory before assigning ids. Hence all tables in Fig. 2 should be constructed "on the fly" while reading the ontology.

We have divided every table in Fig. 2 in two parts: the first is constructed after reading axiom (2), the second is constructed after reading axiom (3), except for table `ax_r_incl` which is constructed after reading axiom (4). Suppose we have just processed axiom (2) and are now reading axiom (3). The concept Heart did not occur before, so we need to assign it with a fresh id = 6 and add a row into the table `at_conc`. The concept Organ, however, has been added before and we need to reuse the old id = 2. Thus table `at_conc` acts as a lookup table for atomic concepts: if a new atomic concept occurs in this table then we reuse its id, otherwise, we introduce a fresh id and add a row into this table. This strategy works quite well for in-memory lookup. However it is hopelessly slow for external memory lookup due to the considerably slow disc access time. Moreover, since executing a query in a database management system, such as MySQL, involves an overhead on opening connection, performing transaction, and parsing the query, executing a query one by one for every atomic concept is hardly practical.

In order to solve this problem, we use temporary in-memory tables to buffer the newly added concepts. We assign fresh ids regardless of whether the concept has been read before or not, and later restore uniqueness of the ids using a series of SQL queries. The tables are similar to those used for storing original ids, except that the tables for conjunctions and existential restrictions have an additional column `depth` representing the nesting depth of the concepts. In addition, there is a new table `map` which is used for resolving ids in case of duplicates. The sizes of temporary tables can be tuned for best speed/memory performance.

Fig. 3 presents the contents of the temporary tables after reading axiom (3) assuming that axiom (2) is already processed. In order to resolve ids for all buffered atomic concept we perform the following SQL queries:

```
1: INSERT IGNORE INTO at_conc SELECT * FROM tmp_at_conc;
2: INSERT INTO map SELECT tmp_at_conc.id, at_conc.id
       FROM tmp_at_conc JOIN at_conc USING (name);
```

tmp_at_conc

name	id
Heart	6
Organ	7
MuscularSystem	9
CirculatorySystem	10

tmp_at_role

name	id
belongsTo	8

tmp_conj

A	$\sqcap$ B	pos	neg	id	depth
9/4	10	✓		11	1
7/2	12	✓		13	2

tmp_exis

$\exists$ $r.$	B	pos	neg	id	depth
8	11	✓		12	1

map

tmp_id	orig_id
7	2
9	4

Fig. 3. Temporary tables after reading axiom (3). The entries $id1/id2$ represent ids before/after mapping temporary ids to original ids using table `map`.

The first query inserts all buffered atomic concepts into the main table; duplicate insertions are ignored using a uniqueness constraint on the filed `name` in table `at_conc`. The second query creates a mapping between temporary ids and ogirinal ids to be used for replacement of the ids in complex concepts containing atomic ones. The query can be optimized to avoid trivial mappings.

Replacement of ids and resolving of duplicate ids in complex concepts is done recursively over the depth d of concepts starting from $d = 1$. Processing conjunctions of the depth d can be done as follows. First, for every row in table `tmp_conj` with the depth d the original ids of columns A and B are restored using the table `map`. Second, the resulting rows are inserted into the main conjunction table `conj`, or update the polarities in case the table has already entries with the same values of A and B. Finally, new mapping between ids for corresponding rows in `conj` and `tmp_conj` are added to table `map`. The ids for existential restrictions are resolved analogously. More details can be found in the technical report [3].

3.3 Completion under Rules CR1 and CR2

We apply the completion rules from Fig. 1 in the order of priority in which they are listed. That is, we exhaustively apply rule **CR1** before applying rule **CR2**, and perform closure under the rules **CR1** and **CR2** before applying the remaining rules **CR3**–**CR5**. This decision stems from an observation that closure under rules **CR1** and **CR2** can be computed efficiently using a temporary in-memory table. Indeed, the closure under rules **CR1** and **CR2** of the form $A \sqsubseteq X$ for a fixed A can be computed independently of other subsumptions since the premises and the conclusions of these rules share the same concept A. The main idea is to use a temporary in-memory table to compute a closure for a bounded number of concepts A, and then output the union of the results into the main on-disc table.

Let `tmp_subs` be a temporary in-memory table with columns A, B representing subsumptions between A and B, and a column `step` representing the step at which each subsumption has been added. We use a global variable `step` to indicate that rule **CR1** has been already applied for all subsumptions in `tmp_subs` with smaller values of `tmp_subs.step`. Then the closure of `tmp_subs` under **CR1** can be computed using the following procedure:

```
REPEAT
  SET @size = (SELECT COUNT(*) FROM tmp_subs);
  INSERT IGNORE INTO tmp_subs
      SELECT tmp_subs.A, ax_c_incl.B, (step + 1)
      FROM tmp_subs JOIN ax_c_incl ON tmp_subs.B = ax_c_incl.A
      WHERE tmp_subs.step = step;
  SET step = step + 1;
UNTIL @size = (SELECT COUNT(*) FROM tmp_subs)   -- nothing has changed
END REPEAT;
```

The procedure repeatedly performs joins of the temporary table `tmp_subs` with the table containing concept inclusions `ac_c_incl` and inserts the result back into `tmp_subs` with the increased step. We assume that `tmp_subs` has a uniqueness constraint on the pair (A, B), so that duplicate subsumptions are ignored.

The same idea can be used to perform closure under rule **CR2** with the only difference that **CR2** has two premises and therefore requires a more complex join. Please refer to the technical report [3] for more details.

3.4 Completion under Rules CR3–CR5

Direct execution of rules **CR3**–**CR5** requires performing many complicated joins. We combine these rules into one inference rule with multiple side conditions:

$$\frac{A \sqsubseteq B}{\exists r.A \sqsubseteq \exists s.B} : r \sqsubseteq s \in \mathcal{O},\ \exists r.A \in \mathsf{sub}^{+}(\mathcal{O}),\ \exists s.B \in \mathsf{sub}^{-}(\mathcal{O}) \qquad (5)$$

Rule (5) derives new inclusions between (ids of) existentially restricted concepts occurring in the ontology using previously derived subsumptions. The main idea is to repeatedly derive these new inclusions, adding them into table `ax_c_incl`, and performing incremental closure of the letter under rules **CR1** and **CR2**. The interested reader can find more details in the technical report [3].

3.5 Transitive Reduction

The output of the classification algorithm is not the set of all subsumptions between sub-concepts of an ontology, but a *taxonomy* which contains only *direct* subsumptions between atomic concepts. In order to compute the taxonomy, the set of all subsumptions between atomic concepts should be transitively reduced. Transitive reduction of a transitive subsumption relation can be done by applying one step of transitive closure and marking the resulting relations as "not-direct", thus finding the remaining set of "direct" subsumptions. Although this can be easily implemented in databases with just one join, we found that this approach has a poor performance because it requires to make a large number of on-disc updates, namely, marking subsumptions as non-direct. In contrast, the number of direct subsumptions is usually much smaller. A solution to this problem involves the usage of a temporary in-memory table. We repeatedly fetch into the table all subsumption relations $A \sqsubseteq X$ for a bounded number of atomic concepts A, perform transitive reduction according to the method described above, and

Table 2. Comparison with other reasoners. Time is in seconds.

Reasoner	NCI	GO	Galen^{-}	Snomed
DB	35.51	20.36	100.86	1183.80
CB	7.64	1.23	3.36	45.17
CEL v.1.0	3.60	1.02	169.23	1302.18
FaCT++ v.1.3.0	4.60	10.50	—	965.84
HermiT v.0.9.3	70.23	92.76	—	—

output the direct subsumptions into the on-disk taxonomy. This algorithm can be also extended to handle equivalence classes of atomic concepts.

4 Experimental Evaluation

We have implemented a prototype reasoner DB[3] in MySQL according to the procedure described in Section 3 (with some small additional optimizations).

In order to evaluate the performance of the reasoner and compare it with existing ontology reasoners, we have performed a series of experiments with four large bio-medical ontologies of various sizes and complexity. The Gene Ontology (GO) (`www.geneontology.org`) and National Cancer Institute (NCI) ontology (`www.cancer.gov`) are quite big ontologies containing respectively 20,465 and 27,652 classes. Galen^{-} containing 23,136 classes is obtain from a version of the OpenGalen ontology (`www.co-ode.org/galen`) by removing role functionalities, transitivities, and inverses. The largest tested ontology Snomed based on Snomed CT (`www.ihtsdo.org`) contains 315,489 classes. We ran the experiments on a PC with a 2GHz Intel® Core™ 2 Duo processor, a 5400 RPM 2.5" hard drive, and 4GB RAM operated by Ubuntu Linux v.9.04 with MySQL v.5.1.31.

In Table 2, we compare the performance of our reasoner with existing in-memory ontology reasoners CB (`cb-reasoner.googlecode.com`), CEL (`lat.inf.tu-dresden.de/systems/cel`), FaCT++ (`owl.man.ac.uk/factplusplus`), and Hermit (`hermit-reasoner.com`). We can see that the performance of DB is on par with the in-memory reasoners and, in particular, that DB outperforms CEL on Galen^{-} and SNOMED, FaCT++ on Galen^{-}, and HermiT on all ontologies. More experimental results are presented in the technical report [3].

References

1. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in the description logic EL using a relational database system. In: IJCAI, AAAI Press (2009)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the EL envelope. In: IJCAI, Professional Book Center (2005) 364–369
3. Delaitre, V., Kazakov, Y.: Classifying ELH ontologies in SQL databases. Technical report, The University of Oxford (2009)

[3] The reasoner is available open source from `db-reasoner.googlecode.com`