Proc. 23rd Int. Workshop on Description Logics (DL2010), CEUR-WS 573, Waterloo, Canada, 2010.

Fast Classification in Protégé: Snorocket as an OWL2 $\mathcal{EL}$ Reasoner

Michael J. Lawley and Cyril Bousquet

The Australian e-Health Research Centre
ICT Centre, CSIRO
Brisbane, Queensland
{michael.lawley,cyril.bousquet}@csiro.au

Abstract. Snorocket is a high-performance implementation of the polynomial-time classification algorithm for the lightweight DL $\mathcal{EL}^{++}$. Until recently it has only been generally available in a form suitable for classifying the biomedical ontology Snomed ct. To make this cross-platform reasoner more generally available to the DL community, Snorocket now implements the OWL API and is thus available as a Protégé plugin. Additionally, through the use of OWL Annotations we have enabled support for the partial incremental classification functionality of Snorocket in order to greatly ease the use of Protégé for manipulating very large ontologies. Benchmarking results are presented for both full and incremental classification demonstrating that Snorocket is the only reasoner capable of handling all the test ontologies and is least an order of magnitude faster than the fastest other Protégé reasoner tested.

1 Introduction

The development of a polynomial-time algorithm [1] for the lightweight description logics in the $\mathcal{EL}$ family has been a significant recent advance, leading to the first academic DL system capable of classifying the very large clinical ontology Snomed ct. Subsequently, there has been a renewed interest in classifiers capable of working with a range of biomedical ontologies that exhibit particular structure amenable to specific optimisations.

However, the algorithms implementation in CEL[1] faces several barriers to wider use, not least of which is the limitation to Linux-based operating systems only.

This paper describes and presents performance measures for Snorocket, another implementation of the algorithm, this time in Java, which is cross-platform, is at least an order of magnitude faster than CEL, has been licensed by the International Health Terminology Standards Development Organisation (IHTSDO) for use in their Snomed ct Workbench, and has recently been adapted as a Protégé plugin including support for (partial) incremental classification.

[1] http://cel.googlecode.com/

2 Background

The initial motivation for Snorocket was the desire for a classifier that could process extended versions of Snomed ct [2]. The major problem with existing classifiers was one of scale. The July 2009 international release of Snomed ct consists of approximately 310,000 active Concepts, 63 Roles, and 370,000 axioms, until quite recently beyond the capabilities of most reasoners (e.g., FaCT++ and RacerPro). A second issue is that Snomed ct is not represented directly in a standard description logic form, but uses its own representation that comprises:

– a *concepts file* identifying each of the Concepts and Roles, and whether a Concept is primitive or not,
– a *relationships file* describing SubClassOf and ObjectSomeValuesFrom expressions, and
– a *descriptions file* detailing the labels and synonyms.

Snomed ct also includes a mechanism known as *role grouping* [3] that, although it can be transformed away to produce a corresponding description logic representation, requires special handling when interpreting classification results.

Prior to the January 2009 International Release of Snomed ct, its stated form was only made available on a limited basis for academic research, along with an unofficial Perl script to transform it into to either a KRSS or OWL/XML representation. Possibly because of this, many publications use older, out-of-date versions of Snomed ct rather than the most recently available one. However, the stated form and the Perl script are now part of the official standard release and are thus much more easily accessible (usually from a national release centre).

With the recent adoption of the OWL 2 Web Ontology Language (OWL2) as a W3C Recommendation [4], there is now a standard profile, OWL2 EL [5], that corresponds to $\mathcal{EL}^{++}$. This profile (a fragment or sub-language of OWL2) is summarised in Table 1.

Table 1. Constructs in the OWL2 EL Profile, Snorocket supported constructs in bold.

Thing	**Nothing**
ObjectSomeValuesFrom	DataSomeValuesFrom
ObjectIntersectionOf	DataIntersectionOf
SubClassOf	**EquivalentClasses**
DisjointClasses	
SubObjectPropertyOf	SubDataPropertyOf
EquivalentObjectProperties	EquivalentDataProperties
TransitiveObjectProperty	**ReflexiveObjectProperty**
ObjectHasValue	DataHasValue
ObjectHasSelf	ClassAssertion
ObjectOneOf	DataOneOf
ObjectPropertyDomain	DataPropertyDomain
ObjectPropertyRange	DataPropertyRange
SameIndividual	DifferentIndividuals
ObjectPropertyAssertion	DataPropertyAssertion
NegativeObjectPropertyAssertion	NegativeDataPropertyAssertion
FunctionalDataProperty	HasKey

It is a relatively simple exercise to update the aforementioned Perl script to produce an OWL2 Functional Syntax representation of SNOMED CT that conforms to this profile. The result can then be loaded, albeit slowly, into tools such as Protégé.

3 Snorocket implementation

While Snorocket was originally implemented as a cross-platform classifier specifically targeted at SNOMED CT, the developments detailed in the previous section have motivated the development of a Protégé plugin form of Snorocket. Note that, at this time, Snorocket does not support *individuals* or domain and range constraints; the supported constructs in in bold face in Table 1.

The implementation of Snorocket is organised into several components:

`au.csiro.snorocket.core`
Contains the core algorithm implementation including support for incremental classification of additions only.

`au.csiro.snorocket.snapi`
Contains the significant extra code to deal with SNOMED CT-specific functionality including:
- pre- and post-processing of *role grouping* [3, 6] including incremental classification support,
- SNOMED CT-specific constants and meta-data, and
- post-processing routines to generate SNOMED CT distribution-format results.

It has a well-defined Java API modelled around the SNOMED CT standard distribution file formats.

`au.csiro.snorocket`
Contains the command-line wrapper to drive the classifier along with parsers for various input file formats: two variants of KRSS, the SNOMED CT distribution format, and OWL2 Functional Syntax [4].

`au.csiro.snorocket.ace`
A thin wrapper around `au.csiro.snorocket.snapi` to interface to the IHTSDO Workbench.

`au.csiro.snorocket.protege`
Contains an OWL API implementation of `OWLReasoner` that wraps the core implementation, `au.csiro.snorocket.core`, and converts from the OWL API representation of axioms to that of Snorocket.

3.1 Experimental results with Protégé reasoners

A set of experiments were performed on a computer equipped with a 3 GHz Intel® Core 2 Duo processor, 4 GB of physical memory, and running Ubuntu Linux. Protégé was run with Java 6 and a maximum heap size of 1900 MB. All the experiments described in this part use the CPU time as an indicator. To

Table 2. Profiles of the various test ontologies.

	#Concepts	#Roles	#Axioms			
Ontology			PCDef	CDef	GCI	RI
Go	26270	6	63426	0	0	0
Nci	27652	70	46800	0	0	0
Fma	78990	113	82343	0	0	0
NotGalen	2748	413	3238	699	357	416
FullGalen	23141	950	25563	9968	1951	957
Snomed ct (stated)	386965	63	443725	59182	0	11
Snomed ct (dist.)	386965	63	595182	59182	0	11

achieve comparable measurement, external timing has been used to check the validity of the results.

The four Protégé reasoners tested were Fact++, Pellet, CEL and Snorocket. These plugins vary in the sense that they have been implemented using different algorithms and different technologies. The core of Fact++ is implemented in C++, Pellet and Snorocket in Java, and the core of CEL in Common Lisp.

The set of ontologies[2] chosen for testing all come from the life sciences domain, and have been used in other performance comparisons [8, 9]. They include the Gene Ontology (Go); a large classification ontology from the US National Cancer Institute (Nci); the Foundational Model of Anatomy (Fma)[3]; two versions of the Galen medical knowledge base (Galen)[4]; and the Systematized Nomenclature of Medicine, Clinical Terms (Snomed ct). In addition, we have included both the original stated form of Snomed ct as well as the distribution form which is derived from the inferred form, excluding redundant subsumption expressions. Table 2 gives some details of these ontologies. For more background on these ontologies see [8].

For each ontology we ran the reasoners five times and threw away the longest time (in most cases, this was from the initial run) before computing an average time. These averages are displayed in Table 3. If the reasoner failed due to an out-of-memory exception, it is indicated by *o/m*, and if it crashed it is indicated by *exit*.

These results show clearly how much faster Snorocket is, especially for the larger ontologies where the order of magnitude speed difference is especially apparent to the user (six seconds vs more than one minute for FullGalen, and one minute vs more than ten minutes for Snomed ct).

[2] Most are available at: `http://lat.inf.tu-dresden.de/~meng/ontologies/`

[3] This is a different version to that used in [8, 9] and is available from `http://www.bioontology.org/projects/ontologies/fma/fmaOwlFullComponent_2_0.owl`

[4] Role inverses and functionalities were excluded from Galen.

Table 3. Average classification times using various reasoners in Protégé on Linux.

	Go	Nci	Fma	NotGalen	FullGalen	Snomed ct (stated)	Snomed ct (distribution)
FACT++	7.69	3.82	*exit*	2.78	*o/m*	597	1287
Pellet	1.54	5.68	*o/m*	1.20	100	*o/m*	*o/m*
CEL	1.07	2.53	*o/m*	2.26	141	760	*o/m*
Snorocket	0.06	0.19	2.68	0.15	6.37	58.3	54.8

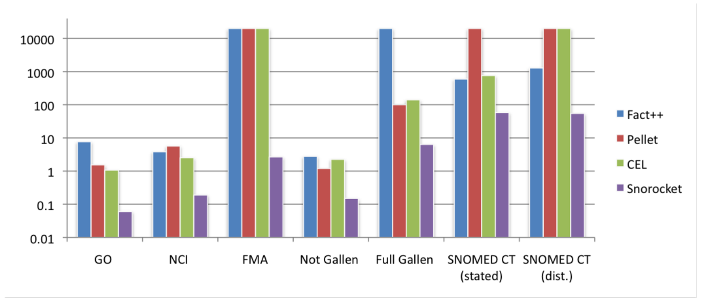

Fig. 1. Performance comparison

4 Incremental classification

For the special case of additions (only) to an ontology that has already been classified, it is possible to re-use the previously computed classification information and to compute the newly inferable relationships [7]. This functionality is especially useful for very large ontologies such as Snomed ct. The Snorocket core implementation supports this kind of incremental classification and is also able to save and later restore its state after classifying an ontology.

Through the use of specific *Annotations* to an ontology, the Snorocket plugin for Protégé is able to restore a previous state and then perform partial incremental classification. This provides a usable environment for either building extensions to large ontologies such as Snomed ct or editing pre-selected fragments after first partitioning the ontology into a large fixed set of axioms and a smaller changeable set.

To do this, one would classify the fixed ontology `tag:fixed.owl` and store it in a file `file:base.txt`. This only needs to be done once. The editable ontology `tag:ediable.owl` would then *Import* `tag:fixed.owl` and include two annotations, one indicating the URI of the pre-classified ontology (`tag:fixed.owl`), and the second indicating the location of the state file (`file:base.txt`). Figure 2 illustrates an example extension of a subset of Snomed ct.

This information is then used by the Snorocket OWL API reasoner to determine where to load the pre-computed state from, and which set of axioms it

```
Namespace(owl=<http://www.w3.org/2002/07/owl#> )
Namespace(owl2annotation=<http://aehrc.com/snorocket/owl2annotation#>)
Namespace(=<http://www.ihtsdo.org/> )
Namespace(base=<http://www.ihtsdo.org/> )

Ontology(<http://www.ihtsdo.org/SCT7>
  Annotation(owl2annotation:BaseUri "http://www.aehrc.com/ontologies/A.owl")
  Annotation(owl2annotation:BaseState "file:A.owlSnorocketProtegeState.txt")

  Import(<http://www.aehrc.com/ontologies/A.owl>)

  EquivalentClasses( SCT_162957009
    ObjectIntersectionOf(
      SCT_366136002
      SCT_272016000
      ObjectSomeValuesFrom( RoleGroup ObjectSomeValuesFrom( SCT_363698007 SCT_51185008))
      ObjectSomeValuesFrom( RoleGroup ObjectSomeValuesFrom( SCT_363698007 SCT_82094008))
      ObjectSomeValuesFrom( RoleGroup ObjectSomeValuesFrom( SCT_418775008 SCT_37931006))
      ObjectSomeValuesFrom( RoleGroup ObjectSomeValuesFrom( SCT_419066007 SCT_420158005))
      ObjectSomeValuesFrom( RoleGroup ObjectSomeValuesFrom( SCT_363714003 SCT_79466001))
    )
  )
)
```

Fig. 2. Example annotated extension of a SNOMED CT subset.

does not need to process (because they are already represented in the restored state).

Note, because of the `Import(<http://www.aehrc.com/ontologies/A.owl>)` clause, Protégé will still load the (large) set of axioms in the referenced ontology even though Snorocket does not need to. To avoid Protégé slowing down due to excessive RAM usage, it is possible to omit this clause without affecting the behaviour of Snorocket.

4.1 Experimental results for incremental classification

We performed two separate evaluations of incremental classification times for Snorocket. First, we created a fixed subset of SNOMED CT consisting of 351261 concepts and an approximately equal number of axioms. We then measured the time it took to incrementally classify some number of additional axioms. The results are given in Table 4. Note that these figures do not include the time taken to load the pre-computed state (approximately 12 seconds) since this is only necessary for the very first classification.

In the second evaluation we used the same fixed subset of SNOMED CT and five different sets of additional axioms of roughly the same size. The results indicate that the incremental classification time is essentially stable for a reasonably sized addition of axioms.

Table 4. Incremental classification times

#Additional axioms	Time (seconds)
2	0.8
10	0.9
30	1.2
50	1.1
100	1.2
200	2.0
400	2.2
800	5.8
1000	7.0
2000	10.8
4000	20.0

Table 5. Incremental classification times for different extensions

	#Concepts	#Axioms		Time
		PCDef	CDef	(seconds)
A1	171	103	16	3.2
A2	212	187	20	2.3
A3	203	150	29	3.3
A4	210	154	23	1.7
A5	219	110	27	1.8

5 Concluding remarks

The experimental results from the previous section in conjunction with comparable results in benchmarking other Protégé reasoners clearly show that Snorocket offers the best performance for OWL2 EL ontologies.

The size of ontologies such as SNOMED CT still present performance problems for editing tools such as Protégé since they load slowly, consume large amounts of RAM leading to sluggish behaviour, and still take a significant amount of time to classify. However, exploiting the incremental classification capabilities of Snorocket the classification time can be reduiced to a more acceptable level.

A further advantage is that, by omitting the *Import* statement for the ontology containing the fixed axioms, one can avoid Protégé loading the fixed axioms, yet the concepts contained therein and the inferred relationships will be displayed in Protégé after classification, but only in a lazy, on-demand manner, thus alleviating the slow load time and excessive RAM consumption.

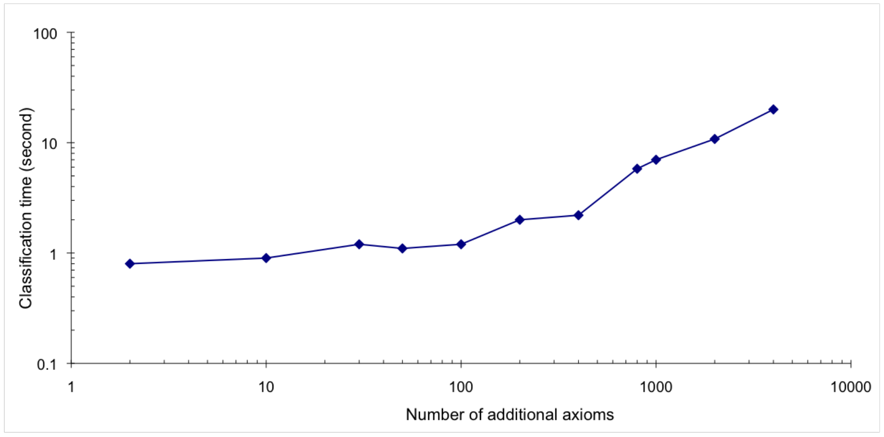

Fig. 3. Incremental classification times for additional axioms

References

1. Baader, F., Lutz, C., Suntisrivaraporn, B.: `CEL`—a polynomial-time reasoner for life science ontologies. In Furbach, U., Shankar, N., eds.: Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06). Volume 4130 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2006) 287–291
2. Lawley, M.: Exploiting fast classification of SNOMED CT for query and integration of health data. In: KR-MED. Volume 410 of CEUR Workshop Proceedings., CEUR-WS.org (2008)
3. Spackman, K., Dionne, R., Mays, E., Weis, J.: Role grouping as an extension to the description logic of Ontylog, motivated by concept modeling in SNOMED. In: Proceedings AMIA Symposium. Volume 712. (2002)
4. World Wide Web Consortium: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. (2009) `http://www.w3.org/TR/owl2-syntax/`.
5. World Wide Web Consortium: OWL 2 Web Ontology Language: Profiles. (2009) `http://www.w3.org/TR/owl2-profiles/`.
6. The International Health Terminology Standards Development Organisation: SNOMED Clinical Terms® Technical Reference Guide. (January 2009)
7. Suntisrivaraporn, B.: Module extraction and incremental classification: A pragmatic approach for $\mathcal{EL}^+$ ontologies. In Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M., eds.: Proceedings of the 5th European Semantic Web Conference (ESWC'08). Volume 5021/2008 of Lecture Notes in Computer Science., Springer-Verlag (2008) 230–244
8. Suntisrivaraporn, B.: Empirical evaluation of reasoning in lightweight DLs on life science ontologies. In: Proceedings of the 2nd Mahasarakham International Workshop on AI (MIWAI'08). (2008)
9. Mendez, J., Suntisrivaraporn, B.: Reintroducing CEL as an OWL 2 EL Reasoner. In Grau, B., Horrocks, I., Motik, B., Sattler, U., eds.: Proceedings of the 22nd International Workshop on Description Logics (DL 2009). Volume 477 of CEUR Workshop Proceedings. (July 2009)