Framework For an Automated Comparison of Description Logic Reasoners

Tom Gardiner, Dmitry Tsarkov, Ian Horrocks*

University of Manchester
Manchester, UK
{gardiner|tsarkov|horrocks}@cs.man.ac.uk

Abstract. OWL is an ontology language developed by the W3C, and although initially developed for the Semantic Web, OWL has rapidly become a de facto standard for ontology development in general. The design of OWL was heavily influenced by research in description logics, and the specification includes a formal semantics. One of the goals of this formal approach was to provide interoperability: different OWL reasoners should provide the same results when processing the same ontologies. In this paper we present a system that allows users: (a) to test and compare OWL reasoners using an extensible library of real-life ontologies; (b) to check the "correctness" of the reasoners by comparing the computed class hierarchy; (c) to compare the performance of the reasoners when performing this task; and (d) to use SQL queries to analyse and present the results in any way they see fit.

1 Introduction

OWL is an ontology language (or rather a family of three languages) developed by the World Wide Web Consortium (W3C) [21]. Although initially developed in order to satisfy requirements deriving from Semantic Web research [13], OWL has rapidly become a de facto standard for ontology development in general, and OWL ontologies are now under development and/or in use in areas as diverse as e-Science, medicine, biology, geography, astronomy, defence, and the automotive and aerospace industries.

The design of OWL was heavily influenced by research in description logics (DLs); investigations of (combinations of) DL language constructors provided a detailed understanding of the semantics and computational properties of, and reasoning techniques for, various ontology language designs [1, 10, 14, 15]; this understanding was used to ensure that, for two of the three OWL dialects (OWL DL and OWL Lite), key reasoning problems would be decidable. Basing the language on a suitable DL also allowed for the exploitation of existing DL implementations in order to provide reasoning services for OWL applications [11, 20, 7].

* This work was partially supported by EPSRC, project IST-2006-027269 "Sealife"

The standardisation of OWL has led to the development and adaption of a wide range of tools and services, including reasoners such as FaCT++ [26], RacerPro [7], Pellet [24] and KAON2 (`http://kaon2.semanticweb.org/`). Reasoners are often used with editing tools, e.g., Protégé [23] and Swoop [17], in order to compute the class hierarchy and alert users to problems such as inconsistent classes.

One of the key benefits that a formal language standard provides to users of web-based technologies is *interoperability*. In the case of OWL, this should mean that users can load ontologies from the internet and use them in applications, possibly answering queries against them using one of the available OWL reasoners. One of the goals of standardisation is that the result of this process is independent of the chosen reasoner. Up to now, however, relatively little attention has been given to checking if this goal is indeed satisfied by available OWL reasoners—the OWL standard includes a test suite [4], but this mainly focuses on small tests that isolate some particular language feature; it does not include many complex cases that involve interactions between different features, nor tests that use realistic ontologies.

This kind of comparison is also of great value to implementors of DL reasoning systems, who typically use testing in order to check the correctness of their implementations. This may be relatively easy for small examples, where manual checking is possible, but will usually not be feasible for realistic ontologies. In such cases, the best (perhaps the only) way to check the correctness of an implementation may be by checking for consistency with the reasoning of other systems.

Once we have confirmed that some or all reasoners are correct (or at least consistent), we may also want to compare their performance. Reasoning with expressive DLs (like those underlying OWL-DL) has high worst case complexity, and this means that, in general, reasoning is highly intractable for these logics. The hope/claim is, however, that modern highly optimised systems perform well in "realistic" ontology applications. To check the validity of this claim it is necessary to test the performance of these systems with (the widest possible range of) ontologies derived from applications.

Real-world ontologies vary considerably in their size and expressivity. While they are all valuable test cases, it is still important to understand each ontology's properties in order to provide efficient and relevant testing. For example, a user (or potential user) of OWL may have some particular application in mind, and might like to know which of the available OWL reasoners is most suitable. In this case, it would be useful if they could compare the performance of reasoners with ontologies having similar characteristics to those that will be used in their application. System developers might also find this kind of testing useful, as it can help them to identify weaknesses in their systems and to devise and test new optimisations.

In this paper we present a system that allows users:

- to test and compare OWL reasoners using an extensible library of real-life ontologies;

– to check the "correctness" of the reasoners by comparing the computed class hierarchy;
– to compare the performance of the reasoners when performing this task;
– to use SQL queries to analyse and present the results in any way they see fit.

2 Background and Related Work

There is extensive existing work on benchmarking DL (as well as modal logic) reasoners. E.g., the TANCS comparisons and benchmark suites [18], the DL comparisons and benchmark suite [12], the OWL benchmark suite and test results, and various test results from papers describing systems such as M-SPASS [16], FaCT and DLP [9, 8], FaCT++ [25], KAON2, Pellet [24], Racer [6], Vampire [27], etc.

Due to the fact that relatively few (large and/or interesting) ontologies were available, earlier tests often used artificially generated test data. For some tests of this kind (e.g. the DL-98 tests, [12]) are hand crafted, or constructed according to a pattern, in such a way that a correct answer is known; in this case they can be used both to test correctness and to measure the performance of reasoners on a certain class of tasks. Other artificial benchmarks (like [22]) are randomly generated, so no correct answer is known for them; in this case they can only be used for performance testing (or for comparing results from more than one reasoner). The Lehigh University Benchmark [5] has been developed specifically for testing OWL reasoners, and uses a synthetic ontology and randomly generated data to test their capabilities using specific weightings to compare systems on characteristics of interest. Results from such tests are, however, of doubtful relevance when gauging performance on real-life ontologies. The popularity of OWL means that many more real-life ontologies are now available, and recent benchmarking work has focused on testing performance with such ontologies.

One such example involved benchmarking of a number of reasoners against a broad range of realistic ontologies [19]. Note that only performance was tested in that work; no results regarding any comparison of the outputs of the reasoner are known. Additionally, not all reasoners used in that comparison supported OWL as an input language, so quantitative comparison of performance would have been difficult/un-justified. This latter problem is eased by the popularity of the DIG (DL Implementation Group) interface [2], which is widely used by application developers and has thus been implemented in most modern DL Reasoners.

Our work builds on these earlier efforts, taking advantage of the DIG standard to provide a generic benchmarking suite that allows the automatic quantitative testing and comparison of DL Reasoners on real-world ontologies with a wide range of different characteristics, e.g., with respect to size and the subset of OWL actually used in the ontology. We aim to make the testing process as automatic as possible, taking care, for example, of (re)starting and stopping reasoners as necessary, to make the results available as and when required by

storing them in a database, and to make the analysis of results as easy and flexible as possible by allowing for arbitrary SQL queries against the collected data. We also aim to provide (in the form of a publicly available resource) a library of test ontologies where each ontology has been checked for expressivity (i.e., the subset of OWL actually used) and syntactic conformance, translated into DIG syntax (which is much easier to work with for the benchmarking purposes than OWL's RDF/XML syntax), and includes (where possible) results (such as the class hierarchy) that can be used for testing the correctness of reasoning systems.

3 Methodology

The system we present here has three main functions. The first is to collect real-world OWL ontologies (e.g., from the Web), process them and add them to a library of ontologies which can be used as a test suite. The second is to automatically run benchmarking tests for one or more reasoners, using the ontology library and storing the results (both performance data and reasoning results) in a database. The third is to allow users to analyse and compare saved results for one or more reasoners.

When processing ontologies, the system takes as an input a list of OWL ontology URIs. Before they can be used in testing, some preprocessing of these ontologies is required. The process involves generating of valuable meta-data about each ontology, as well as converting each of the OWL ontologies into DIG.

The meta-data is generated using code written for SWOOP [17], and specifies some details w.r.t. the expressivity (i.e. the constructs present in the ontology) together with the number of classes, object properties, data properties, individuals, class axioms, property axioms and individual axioms present. This is invaluable information in helping to understand and analyse the results obtained through testing; it can be used, e.g., to study the strengths and weaknesses of particular systems, or to identify ontologies with similar characteristics to those that will be used in a given application.

The OWL-to-DIG conversion uses the OWL-API (`http://sourceforge.net/projects/owlapi`). This process is far from being trivial as OWL's RDF syntax is complex, and it is easy to (inadvertently) cause ontologies to fall outside OWL-DL, e.g., by simply forgetting to explicitly type every object. Moreover, the current DIG interface supports only the most basic of datatypes, such as `<xsd:string>` and `<xsd:integer>`.[1] The result is that many of the available OWL ontologies we found could not be successfully converted to DIG, due to either being OWL-Full or to using more expressive data types than those that are allowed in DIG. In the former case, i.e., OWL-Full ontologies, it is almost *always* the case that they are OWL-Full only as the result of some trivial syntax error; usually missing typing information. Such ontologies can easily be "repaired" and added to the library.

[1] A new DIG standard, DIG 2.0, is currently under development, and will provide support for all OWL compatible datatypes.

Local copies of both the OWL ontology and the DIG version are stored in a database. This is done not only for efficiency during the testing, but also to ensure consistency (as online ontologies rarely remain static). Moreover, this allows us to fix trivial errors (like missing type information for an object) in OWL ontologies in order to ensure that they are in OWL-DL. This can be done by using a technique described in [3]. Such "repaired" ontologies can be successfully translated into DIG, and thus can be used for testing purposes. These files, together with their properties/meta-data, are stored as database entries for easy access and manipulation.

The main function of the benchmark suite itself is to gather the classification information for each ontology and each reasoner. This information includes the time spent by a reasoner in performing certain queries, and the query answer returned by the reasoner.

To promote fairness, each reasoner is terminated and then restarted before loading each ontology. This ensures that every reasoner is in the same "state" when working on a given ontology, regardless of how successful previous attempts were. This also simplifies the management of cases for which the time-out limit was exceeded.

One problem that arises when trying to compare the performance of different reasoners is that they may perform reasoning tasks in different ways. For example, some may take an "eager" approach, fully classifying the whole ontology and caching the results as soon as it is received; others may take a "lazy" approach, only performing reasoning tasks as required in order to answer queries. To try to get around this problem, we use a five step test, for each ontology, that forces reasoners to fully classify the ontology, whether eagerly or lazily. The steps are as follows:

1. Load the ontology into the reasoner;
2. Query the reasoner for all the named (atomic) classes in the ontology;
3. Query the reasoner for the consistency of the ontology by checking the satisfiability of the class owl:Thing;
4. Query the reasoner for the satisfiability of each of the named classes in the ontology;
5. Query the reasoner for the ontology taxonomy (i.e., the parents and children of all named classes).

Each of these individual steps is timed, providing interesting information about when different reasoners do their work. It is, however, the total time for this complete (classification) test that is probably of most interest.

Each test can end in one of three ways. It can either complete successfully, fail due to lack of resources (either time or memory), or fail because the reasoner could not parse/process the ontology and/or query successfully. In case of success, the answers given by the reasoner are saved in the database. These answers can be used for testing the correctness of reasoning (or at least comparing results with those obtained from other reasoners).

The benchmarking process is fully automatic, dealing with most errors autonomously, meaning that the testing can be left to run over-night or over a

week-end (which may be necessary when using a large time-out). All data is recorded in a database, making it easy for the user to view and analyse the data in a variety of ways.

As discussed in Section 1, in order to get a clearer indication of how DL Reasoners perform in the real world, we aim to build a large library of OWL ontologies from those that are publicly available. Currently, our library contains over 300 OWL ontologies, but so far only 172 of these have been successfully converted to DIG. This has, however, provided us with a total of just under 72,000 classes and over 30,000 individuals in a DIG format. Only 18% of the ontologies had the expressivity corresponding to the DL $\mathcal{ALC}$ or higher, which suggests that the majority of real-world ontologies are not, in fact, very complex, but it also means that we have a useful number of "interesting" examples.

4 Data storage

As we have mentioned on several occasions, the database is used to store all processed data. We have used a database as it provides persistence, and allows the user to quickly summarize data and to analyse it in any way that is deemed appropriate. The database contains a wealth of information about both the ontologies and the behaviour of the reasoners, allowing users to produce high level summaries or to focus on and investigate results of particular interest.

Moreover, the database is designed so that any data obtained through our tests is stored with a normalised layout. For example, the responses to the queries in our 5 step test are returned by the reasoners as large XML documents (in DIG format) which can represent the same information in a number of different ways. Our system parses these responses and records them as simple class names, along with information such as satisfiability status (i.e., satisfiable or not), parents (i.e., named classes that are direct subsumers) and children (i.e., named classes that are direct subsumees). This makes it easy to use SQL queries to look for similarities or differences between different reasoner's responses.

The advantages of our approach are demonstrated in Section 5 below, where we show examples of the kind of query that we could issue and the resulting information that would be extracted from the database.

5 Testing

Our system as it stands is fully automatic and runs the classification tests successfully through our whole library. The times taken by each reasoner, for each step and for each ontology are recorded, together with the parsed and normalised version of the responses returned by each reasoner.

We have performed tests using our system with several state-of-the-art DIG reasoners, and we provide here some examples of the kinds of information that the system can produce. It is important to note that we simply set up our benchmarking system to run overnight for each ontology with each reasoner.

All the information described in the following sub-sections was then obtained by querying the resulting database to extract the information that we were interested in.

FaCT++ v1.1.3, KAON2, Pellet v1.3 and RacerPro v1.8.1 are four of the most widely used OWL/DIG reasoners, and we therefore decided to use these to test the current capabilities of our system. The tests were performed using an Intel Pentium-M processor 1.60 GHz and 1Gb of main memory on Windows XP. The time-out period was set to 10 minutes (in real time). Pellet and KAON2 are Java applications, and for these tests were run with a maximum heap space of 200Mb. RacerPro and FaCT++ were left to run at their default settings. Our system does not try to optimise the performance of the reasoners for particular ontologies, as we believe this is the job of the reasoners themselves: users of OWL reasoners cannot be expected to be experts in how to tune them in order to optimise their performance.

5.1 Correctness

Every step of our benchmarking is timed to allow performance comparison of the reasoners in question. However, this data only becomes relevant if we can confirm the correctness of the responses returned by these systems when answering the queries we put to them (test steps 2-5): a reasoner that quickly but incorrectly answers queries is of little use (or at least cannot be fairly compared to one that gives correct answers).

When reasoner implementors test the correctness of their systems, they typically use small artificially generated examples and manually check the correctness of their system's responses. Due to the sheer size of the ontologies in our library, it is not feasible to check responses by hand, so we needed a way of automating this task.

It is impossible to say any one reasoner is universally correct, and we are therefore unable to base correctness on any one reasoner's answers. Our solution was to base our measure of correctness on tests of consistency of different reasoner's answers. Our claim is that consistency between multiple reasoners implies a high probability of correctness, especially when the reasoners have been designed and implemented independently, and in some cases even use different reasoning techniques.[2]

With normalised entries of the parsed responses stored in our database, checking for consistency was a simple matter of writing a few short SQL queries to see if each reasoner had symmetrical entries for each of the DIG queries. Naturally, this required that at least two reasoners had successfully completed the 5-step classification test. Of our 172 DIG ontologies, 148 had this property; of the remaining 24 ontologies, more than half were not successfully classified by *any* of the reasoners.

[2] KAON2 uses a resolution based technique; the other reasoners tested here use a tableaux based technique.

We started by checking for 100% consistency on each of the classification steps, where 100% consistency meant that all reasoners that successfully completed the test gave the same answers w.r.t. the class hierarchy. Where there were conflicts, we used more specific SQL queries to analyse the reason in detail, allowing us to see exactly how big the differences were.

Our findings were surprisingly impressive, with only 7 of the 148 ontologies (that were fully classified by at least two reasoners) not being 100% consistent on all tests. This reflects very positively on the OWL standardisation process (and on the developers of these reasoners), and shows that the OWL standard really does result in a high degree of interoperability.

The inconsistencies between reasoners on the seven aforementioned ontologies raised some interesting issues.

Starting with step 2 (querying for the list of classes in each ontology), there were only three ontologies on which there were inconsistencies. The reason for the inconsistencies was due to the existence of nominals (i.e., extensionally defined classes resulting, e.g., from the use of the OWL *oneOf* or *hasValue* constructors). RacerPro was actually returning nominals as classes, while the other three reasoners were not. (These three ontologies were also the only three ontologies containing nominals that RacerPro successfully classified). We assume that this happens because RacerPro does not claim to be sound and complete in the presence of nominals, but instead tries to approximate them using classes to represent them.

Step 3 (querying for the satisfiability of owl:Thing) was 100% consistent for all ontologies. This is, however, not surprising as owl:Thing is satisfiable for all of the ontologies in the library.

Step 4 (querying for the satisfiability of each named class in the ontology) found only two ontologies with inconsistent answers. Interestingly, they were both only successfully classified by FaCT++ and Pellet, and on one of the ontologies they disagreed about the satisfiability of over 2/3 of the classes present.

The first ontology was in $\mathcal{SHIN}$ (Tambis) and the other was in DL-Lite with Datatypes and Inverse roles. Other properties of these ontologies suggested no obvious challenges. We therefore selected a few of the classes on which there was disagreement, and used SWOOP to analyse them. Using this tool we found that FaCT++ was clearly inconsistent in its reasoning w.r.t. these classes.

Taking one class as an example: FaCT++ answered that the class *RNA* was satisfiable, while Pellet disagreed. SWOOP showed that the definition of *RNA* consisted of the intersection of the class *Macromolecular-compound* and a number of other classes. However, FaCT++ and Pellet both agreed that *Macromolecular-compound* was not satisfiable, hence implying that *RNA* was definitely unsatisfiable. As a result of this investigation, a bug in FaCT++ has been identified and is now in the process of being fixed. This demonstrates how valuable the information provided by our system can be to system developers as well as to users of OWL reasoners.

As hoped, step 5 (querying for the parents and children of each named class in the ontology) found that the taxonomies defined by the parent and children

relations were consistent in the vast majority of cases, and there were only three ontologies on which the reasoners were not in agreement. In each of these cases, FaCT++ was not in agreement with the other reasoners. Due to the detailed information recorded by our system holds, we are easily able to identify the classes that are causing this problem, and thus investigate it more closely. In this way we can not only find bugs that have not been discovered before, but the detailed analysis allows a system implementor to quickly find exactly what is causing the bug, and hopefully to fix it.

5.2 Efficiency

Table 1. Sample of Overall Performance for 100% Consistent Ontologies

Type	Status	FaCT++	KAON2	Pellet	RacerPro
All	Success	137	43	143	105
All	CouldNotProcess	24	119	20	60
All	ResourcesExceeded	4	3	2	0
Nominals	Success	4	0	2	0
Nominals	CouldNotProcess	0	5	3	5
Nominals	ResourcesExceeded	1	0	0	0
TransRoles	Success	9	5	9	6
TransRoles	CouldNotProcess	2	6	3	7
TransRoles	ResourcesExceeded	2	2	1	0
Datatypes	Success	91	0	98	62
Datatypes	CouldNotProcess	19	112	14	50
Datatypes	ResourcesExceeded	2	0	0	0
OWL-Lite	Success	43	41	42	43
OWL-Lite	CouldNotProcess	5	6	6	7
OWL-Lite	ResourcesExceeded	2	3	2	0

We present here some examples of the kinds of performance related information that can be extracted from the database using suitable SQL queries. Table 1 provides an overview of the performance of the four reasoners: it shows how many of the test ontologies they were able to classify within the specified time limit, and then breaks this information down by focussing on sets of ontologies using particular language features. Finally, it also shows their performance on OWL-Lite ontologies, i.e., all those with expressivity up to and including $\mathcal{SHIF}$. Note that only 100% consistent ontologies are compared here; as we mentioned before, the performance analysis is of doubtful value when different reasoners do not agree on query results.

It is important to note that "Could not process" most often means that the reasoner does not support the constructs present within that particular ontology (and does not claim to either).

The SQL statement below shows how easily we filled the transitive-roles part of Table 1. Here, "name" refers to the name of the reasoner, "status" is a choice of "Succes", "CouldNotProcess" or "ResourcesExceeded" and the COUNT function returns a count of the number of ontologies that meet the given criteria.

```
SELECT name, status, COUNT(status)
FROM resultsview
WHERE rplus
AND ontology IN
(
  /*Get the list of ontologies that had full consistency on all steps*/
  SELECT ontology
  FROM consistency
  WHERE clist AND topsat AND allsat AND parents
)
GROUP BY name, status;
```

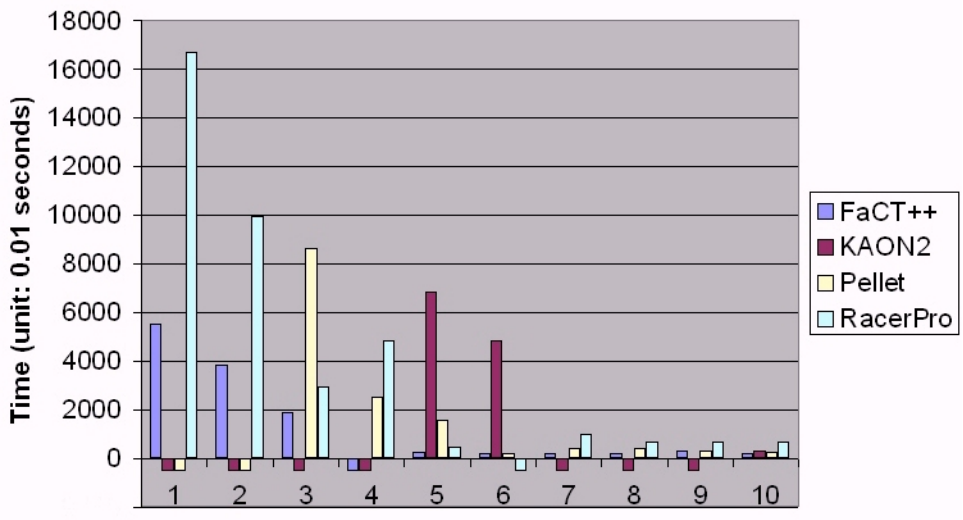

Fig. 1. Comparison of Reasoners on the Top 10 Most Challenging Ontologies

Table 2 presents some information describing the features of the the 10 most "challenging" (w.r.t. reasoning) ontologies in the library. We did this by selecting all those ontologies that were successfully classified by at least two reasoners, and then ordering these ontologies by the average classification time for those reasoners that successfully classified them. Note that this is just an example of the way in which the data can be analysed, and we do not claim this to be the "correct" way to select challenging ontologies.

Table 2. Properties of Top 10 Most Challenging Ontologies

	Expressivity	nClass	nIndiv	Ontology
1	DLLite	27652	0	NCI
2	$\mathcal{ALR}+$	20526	0	GeneOntology
3	$\mathcal{SHF}$	2749	0	Galen
4	RDFS(DL)	1108	3635	WorldFactBook
5	RDFS(DL)	1514	0	DataCenter
6	$\mathcal{SHIF}$	37	0	DolceLite
7	$\mathcal{ALR}+\mathcal{HI}(\mathcal{D})$	5	2744	HydrolicUnits2003
8	RDFS(DL)	382	1872	Tambis
9	RDFS(DL)	98	0	MovieDatabase
10	RDFS(DL)	6	2744	HydrolicUnits

This table is useful in helping us understand what makes these particular Ontologies so time-consuming to reason over. In the case of the NCI and Gene Ontologies (1st and 2nd), it can be clearly seen that it is their sheer size that provides the challenge. The Hydrolic Units ontologies (7th and 10th) have very few classes (only 5 and 6 respectively), but relatively large numbers of individuals. The world-fact-book ontology (4th) uses only a minimal subset of the ontology language (no more than the DL subset of RDFS), but has a reasonably large number of both classes and individuals. Finally, the Galen ontology (2nd) has a moderately large number of classes, and also uses a relatively rich subset of OWL. This demonstrates the kinds of insight that can be achieved using suitable queries. In this case we examined just three properties (specifically expressivity, number of classes and number of individuals) of our chosen ontologies; we could easily have extended our analysis to include available information such as the number of object/data properties, kinds of axiom occurring, etc.

In Figure 1, we present a graphical view of the amount of time each Reasoner took to classify the 10 most challenging ontologies according to the above mentioned measure (where negative time represents unsuccessful classification). It is interesting to note that there is no clear "winner" here; for example, FaCT++ performs well on the ontologies with very large numbers of classes (the NCI and Gene Ontologies), but relatively poorly on some ontologies with large numbers of individuals (e.g., the world-fact-book ontology).

Regarding the ontologies that include large numbers of individuals, it is important to note that our testing procedure (the 5-step classification) does not yet include any ABox queries (adding this kind of testing will be part of future work). This is clearly disadvantageous to systems such as KAON2 that are mainly designed to optimise ABox query answering rather than classification.

Finally, in Table 3, we present the average proportion of the classification time that the reasoners spent on each of the five steps. This shows, for example, that Pellet performs a lot of work as soon as it receives the Ontology (the `Load` step), while FaCT++ does relatively little work until the first query (the `ClassList` step).

Table 3. Average Division of Task Time

Reasoner	`Load`	`ClassList`	`SatofTop`	`SatOfClasses`	`Hierarchy`
FaCT++	16%	26%	16%	21%	21%
KAON2	48%	44%	1%	2%	5%
Pellet	69%	21%	1%	2%	7%
RacerPro	57%	10%	4%	9%	19%

Note that the reason for the `Load` step taking such a large proportion of the total time may be the result of the relatively high overhead of loading an ontology into a reasoner via the DIG interface; it is not necessarily due to time taken performing "eager" reasoning.

6 Discussion

As we mentioned in Section 1, testing is useful for reasoner and tool developers as well as for application users. Building on existing work, we have developed a system for testing reasoners with real-life ontologies. The benefits of our approach include autonomous testing, flexible analysis of results, correctness/consistency checking and the development of a test library that should be a valuable resource for both the DL and ontology community. We will continue to extend the library, and will publish the computed class hierarchy in case a consistent answer is obtained. We will also continue to analyse the reasons for the inconsistencies we have found, and would eventually like to analyse which implementation strategies and optimisations seem to be most effective for particular kinds of ontology and reasoning problems.

While there are an increasingly large array of OWL ontologies available for public use, other Ontology formats (e.g. OBO: the Open Biomedical Ontologies, `http://obo.sourceforge.net`) are still in widespread use, and would make a valuable addition to our test examples. It is also the case, as described in [3], that a large proportion of the available OWL-Full ontologies, could relatively easily be "repaired" so as to be OWL-DL; adding a few extra typing statements is all that is typically required. In the future we hope to use semi-automated repair of OWL-Full ontologies and translation from formats such as OBO to increase the size and scope of our ontology library.

So far we have focused on testing TBox reasoning (classification). Although the use of nominals in OWL-DL blurs the separation between TBox and ABox, it would still be useful to explicitly test ABox reasoning, e.g., by asking for the instances of some query class. In fact, for ontologies that include far more individuals than classes (such as the world-fact-book, Tambis and Hydrolic Units ontologies), it makes little sense to test classification and not to test query answering. Focusing on classification also fails to give a fair picture of the performance of systems such as KAON2 that aim to optimise query answering. Extending the testing regime to include querying will be part of our future work.

Apart from the future work described above, there are a number of extensions to our benchmarking system that would enhance its utility. Allowing users to define their own customised tests would help reasoner developers to test specific optimisations and implementations as they are developed. It would also be useful to be able to investigate how multiple concurrent queries affect reasoner performance, and whether reasoners perform better or worse if they are *not* restarted between tests.

Both the testing system and the ontology library are publicly available resources, and can be downloaded from `http://www.cs.man.ac.uk/~horrocks/testing/`.

References

1. Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1):5–40, October 2001.
2. Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG description logic interface. In *Proceedings of DL2003 International Workshop on Description Logics*, September 2003.
3. Sean Bechhofer, Raphael Volz, and Phillip Lord. Cooking the semantic web with the OWL API. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science. Springer, 2003.
4. Jeremy J. Carroll and Jos De Roo. OWL web ontology language test cases. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-test/`.
5. Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. An evaluation of knowledge base systems for large OWL datasets. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proc. of the 2004 International Semantic Web Conference (ISWC 2004)*, number 3298 in Lecture Notes in Computer Science, pages 274–288. Springer, 2004.
6. Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 161–168, 2001.
7. Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.
8. I. Horrocks. Benchmark analysis with FaCT. In *Proc. of the 4th Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX 2000)*, number 1847 in Lecture Notes in Artificial Intelligence, pages 62–66. Springer-Verlag, 2000.
9. I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. In *Proc. of Tableaux'98*, pages 27–30, 1998.
10. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.
11. Ian Horrocks. The FaCT system. In Harrie de Swart, editor, *Proc. of the 2nd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'98)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 307–312. Springer, 1998.

12. Ian Horrocks and Peter F. Patel-Schneider. DL systems comparison. In *Proc. of the 1998 Description Logic Workshop (DL'98)*, pages 55–57. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-11/`, 1998.
13. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.
14. Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204, 2001.
15. Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005.
16. U. Hustadt and R. A. Schmidt. Using resolution for testing modal satisfiability and building models. In I. P. Gent, H. van Maaren, and T. Walsh, editors, *SAT 2000: Highlights of Satisfiability Research in the Year 2000*, volume 63 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, 2000. Also to appear in a special issue of *Journal of Automated Reasoning*.
17. A. Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca-Grau, and James Hendler. SWOOP: a web ontology editing browser. *J. of Web Semantics*, 4(2), 2005.
18. Fabio Massacci and Francesco M. Donini. Design and results of TANCS-00. In R. Dyckhoff, editor, *Proc. of the 4th Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.
19. Zhengxiang Pan. Benchmarking DL reasoners using realistic ontologies. In *Proc. of the First OWL Experiences and Directions Workshop*, 2005.
20. P. F. Patel-Schneider. DLP system description. In *Proc. of the 1998 Description Logic Workshop (DL'98)*, pages 87–89. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-11/`, 1998.
21. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL web ontology language semantics and abstract syntax. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-semantics/`.
22. Peter F. Patel-Schneider and Roberto Sebastiani. A new general method to generate random modal formulae for testing decision procedures. *J. of Artificial Intelligence Research*, 18:351–389, 2003.
23. Protégé. `http://protege.stanford.edu/`, 2003.
24. E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. Submitted for publication to Journal of Web Semantics, 2005.
25. Dmitry Tsarkov and Ian Horrocks. Ordering heuristics for description logic reasoning. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 609–614, 2005.
26. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297, 2006.
27. Dmitry Tsarkov, Alexandre Riazanov, Sean Bechhofer, and Ian Horrocks. Using Vampire to reason with OWL. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proc. of the 2004 International Semantic Web Conference (ISWC 2004)*, number 3298 in Lecture Notes in Computer Science, pages 471–485. Springer, 2004.