Benchmarking OWL Reasoners

Jürgen Bock
FZI Research Center
Karlsruhe, Germany
bock@fzi.de

Qiu Ji
Institute AIFB, University of Karlsruhe
76131 Karlsruhe, Germany
qiji@aifb.uni-karlsruhe.de

Peter Haase
Institute AIFB, University of Karlsruhe
76131 Karlsruhe, Germany
haase@aifb.uni-karlsruhe.de

Raphael Volz
FZI Research Center
Karlsruhe, Germany
volz@fzi.de

ABSTRACT
The growing popularity of semantic applications makes scalability of ontology reasoning tasks increasingly important. In this work, we firstly analyze the ontology landscape on the web, and identify typical clusters of expressivity. Secondly, we benchmark current ontology reasoners, by using representative ontologies for each cluster and a comprehensive set of queries. We point out applicability of specific reasoners to certain expressivity clusters.

1. INTRODUCTION
Semantic applications based on ontologies have become increasingly important in recent years. Yet, scalability remains one of the major obstacles in leveraging the full power of using ontologies for practical applications. Reasoning with OWL ontologies has high worst complexity, but indeed many large scale applications normally only use fragments of OWL that are rather shallow in logical terms and do not require sophisticated reasoning algorithms. Surveying the landscape of existing ontologies, we observe a broad spectrum of ontologies that differ in terms of size, complexity and their ratio between terminological and factual assertions. Keet and Rodríguez [8] point out that there is a high demand for either very expressive ontology languages to represent rather complete knowledge, or less expressive ontology languages, which are more tractable w.r.t. reasoning or other computational tasks. This issue is often called the computational cliff and leads to the problem, that often only small fragments of ontology languages are exploited, where reasoners are used that are optimized to a larger number of features, and in particular those, that are actually not used in the fragment. Today, a number of different reasoners are available that are based on quite different design decisions in addressing the tradeoff between complexity and expressiveness on the one hand and scalability on the other hand:

Classical description logic reasoners based on tableau algorithms are able to classify large, expressive ontologies as often found for example in the bio-medical domain, but they often provide limited support in dealing with large number of instances. Database-like reasoners that materialize inferred knowledge upfront are able to handle large amounts of assertional facts, but are in principle limited in terms of the logic they are able to support. Deciding for an appropriate reasoner for a given application task is far from trivial. In order to support such decisions, comparisons of reasoners based on benchmarks are required.

While a number performance evaluations for OWL reasoners have already been performed in the past, all of them so far targeted only special purpose tasks, e.g. focussing either on classical description logic reasoning tasks [2], or on answering conjunctive queries over large knowledge bases based on rather inexpressive ontologies [6]. In our work we aim to go a step further and intend to provide guidance for selecting the appropriate reasoner for a given application scenario. In order to so, we provide a survey of the ontology landscape, discuss typical reasoning tasks and define a comprehensive benchmark. Based on the benchmark results we identify which reasoners are most adequate for which classes of ontologies and corresponding reasoning tasks.

The paper is organized as follows: In Section 2 we discuss related work on benchmarking OWL reasoners. Based on an overview of the ontology landscape and relevant language fragments provided in Section 3, we define our benchmark in Section 4. In Section 5, we give a description of the reasoners selected for our benchmark. In Section 6 we report on the experiments performed. We conclude with an outlook to future work in Section 7.

2. RELATED WORK
With the availability of practical reasoners for OWL, a number of benchmarks for evaluating and comparing OWL reasoners have been proposed. The first one – the Lehigh University Benchmark (LUBM) – was proposed by Guo et al. in [6]. LUBM concentrates on the reasoning task of answering conjunctive queries over an OWL Lite ontology with an ABox of varying size. It was later pointed out that – while the ontology itself is in OWL Lite – answering the proposed queries does not require OWL Lite reasoning, but instead can be performed by realizing the ABox, i.e. computing the most specific concept(s) that each individual is an in-
stance. This indeed is performed by many system benchmarks, including for example the evaluation of RACER in [7]. LUBM was extended in [12] to the University Ontology Benchmark (UOBM), includes both OWL Lite and OWL DL ontologies covering a complete set of OWL Lite and DL constructs, respectively. In addition to measuring the performance of the reasoners, both LUBM and UOBM provide a measure for correctness of the reasoners, analyzing how many of the correct answers are returned (completeness) and how many of the returned answers are correct (soundness). However, we believe that such measures are not helpful in selecting a reasoner for a given task. Instead what is missing is a detailed analysis, which fragment of the OWL ontology language are actually needed for a given task and supported (correctly) by which reasoner.

[2] presents a system for comparing DL reasoners 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. Again, this benchmarking system is only targeted to classical DL reasoning tasks, disregarding many other practical applications of OWL.

[10] provides a comparison of reasoning techniques with a focus on querying large Description Logic ABoxes. The results show that, on knowledge bases with large ABoxes but with simple TBoxes, the KAON2 algorithms for reducing a DL knowledge base to a disjunctive datalog program show good performance; in contrast, on knowledge bases with large and complex TBoxes, existing techniques still perform better.

In [13] the authors pointed out some deficiencies with existing benchmarks and formulated requirements they would like to see met by new benchmarks. These requirements were driven by two major use cases: (1) Frequent ABox changes (situation classification) and (2) rare ABox changes (social networks). While in our work we rather concentrate on the dimensions of the classes of ontologies and reasoning tasks, we take most of the defined requirements into account (c.f. Section 4.4).

3. OVERVIEW OF THE WEB ONTOLOGY LANDSCAPE

Ontologies on the web are becoming increasingly numerous. These ontologies differ significantly in their expressivity, as well as in the size of their TBox and ABox. In order to identify a representative picture of the ontology landscape, we analyzed 289 ontologies with particular respect to their expressivity. In this context, we identified four main expressivity clusters, namely the RDFS fragment of description logics\(^1\), OWL DLP, OWL Lite, and OWL DL. Hereby, the clusters OWL DL and OWL Lite comprise ontologies that fall into description logics of at most $\mathbf{SHOIN}D$ for OWL DL and $\mathbf{SHIT}D$ for OWL Lite, respectively. In addition to these, we included the tractable fragment OWL DLP [4], since it returns a number of interesting features of OWL Lite while keeping the complexity low to a certain degree. Theoretical investigations proved the combined complexity for standard reasoning tasks in the DLP fragment to be EXP-TIME, while the data complexity for both standard reasoning tasks and conjunctive query answering remains PTIME.

\(^1\)We will call this fragment RDFS(DL) subsequently.

The most light-weight complexity cluster we identified was RDFS(DL), which is, in terms of expressivity, sufficient for representing many taxonomy-style ontologies. Due to its simplicity it seems to be popular for many use cases of ontologies in current web based applications.

Table 1 illustrates the set of ontologies partitioned according to the different fragments, and some characteristics. As one can see, the RDFS(DL) cluster is the largest cluster in the ontology landscape, where ontologies make use of only the basic, taxonomic features, such as classes and basic properties. In DLP we go clearly beyond RDFS(DL) by allowing more specified properties, and a restricted form of intersection, universal, and existential quantification. These features do not occur very frequently in the ontologies we analyzed, however, about a third of the ontologies in the DLP cluster make use of restrictions, which entail significantly more expressiveness than plain RDFS(DL). The DLP cluster was the smallest cluster we analyzed. This may be due to the more cautious ontology design that has to be followed, when it comes to the use of specific features that are not supported in DLP. The fact that DLP has only very recently been identified as a tractable fragment, also explains the smaller number of ontologies that have been found, since most DLP ontologies in our corpus have been identified to be only incidentally in the DLP fragment, but few might be explicitly designed as such.

The OWL Lite cluster is determined by a stronger axiomatization of classes, as the frequent use of restrictions in nearly all ontologies demonstrates.

In the OWL DL cluster we can find the full range of available OWL features, including nominals and for instance a relatively frequent usage of unions and disjointness axioms (both used in more than 40% of all OWL DL ontologies).

4. BENCHMARK DEFINITION

4.1 Reasoning Tasks

Many reasoning tasks for OWL correspond to standard description logic reasoning tasks, i.e. tasks that allow to draw new conclusions about the knowledge base or check its consistency. Theoretically it is possible to reduce all reasoning tasks to the task of checking KB consistency. However in practice this is not necessarily the fastest way of reasoning and various optimizations are taken for different tasks. We therefore analyze reasoning tasks separately.

**TBox reasoning tasks.** Reasoning tasks typically considered for TBoxes are the following:

- **Satisfiability** checks whether a class $C$ can have instances according to the current ontology.
- **Subsumption** checks whether a class $D$ subsumes a class $C$ according to the current ontology. Property subsumption is defined analogously.

For our benchmarks we consider the classification of the ontology, i.e. computing the complete subsumption hierarchy of the ontology.

**ABox reasoning tasks.** ABox reasoning tasks usually come into play at runtime of the ontology. Reasoning tasks typically considered for ABoxes are the following:
Table 1: Partitions of the ontology landscape into 4 clusters along with number and percentage of occurrences of particular OWL features.

<table>
<thead>
<tr>
<th>Fragment</th>
<th>RDFS (DL)</th>
<th>OWL DLP</th>
<th>OWL Lite</th>
<th>OWL DL</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>113</td>
<td>31</td>
<td>57</td>
<td>88</td>
<td>289</td>
</tr>
<tr>
<td>Class</td>
<td>105 (92.92%)</td>
<td>31 (100.0%)</td>
<td>57 (100.0%)</td>
<td>88 (100.0%)</td>
<td>275 (95.16%)</td>
</tr>
<tr>
<td>ObjectProperty</td>
<td>58 (51.33%)</td>
<td>23 (74.19%)</td>
<td>55 (96.49%)</td>
<td>74 (84.09%)</td>
<td>210 (72.66%)</td>
</tr>
<tr>
<td>DatatypeProperty</td>
<td>38 (33.63%)</td>
<td>15 (48.39%)</td>
<td>12 (21.05%)</td>
<td>49 (55.68%)</td>
<td>114 (39.45%)</td>
</tr>
<tr>
<td>EquivClass</td>
<td>3 (5.26%)</td>
<td>2 (3.51%)</td>
<td>24 (8.30%)</td>
<td></td>
<td>31 (10.73%)</td>
</tr>
<tr>
<td>EquivProp</td>
<td>8 (14.04%)</td>
<td>1 (1.14%)</td>
<td>9 (3.11%)</td>
<td></td>
<td>18 (6.23%)</td>
</tr>
<tr>
<td>FunProp</td>
<td>8 (25.81%)</td>
<td>3 (5.26%)</td>
<td>32 (36.36%)</td>
<td>43 (8.30%)</td>
<td>86 (17.05%)</td>
</tr>
<tr>
<td>InvFunProp</td>
<td></td>
<td></td>
<td></td>
<td>7 (2.42%)</td>
<td></td>
</tr>
<tr>
<td>SynProp</td>
<td>2 (6.45%)</td>
<td>0 (0.00%)</td>
<td>5 (15.62%)</td>
<td>14 (7.30%)</td>
<td>21 (7.22%)</td>
</tr>
<tr>
<td>TranProp</td>
<td>3 (9.68%)</td>
<td>7 (12.28%)</td>
<td>14 (15.91%)</td>
<td></td>
<td>24 (8.30%)</td>
</tr>
<tr>
<td>InverseOf</td>
<td>12 (38.71%)</td>
<td>13 (22.81%)</td>
<td>26 (29.55%)</td>
<td></td>
<td>51 (17.65%)</td>
</tr>
<tr>
<td>Restriction OnProp</td>
<td>9 (29.03%)</td>
<td>52 (91.23%)</td>
<td>50 (56.82%)</td>
<td></td>
<td>111 (38.41%)</td>
</tr>
<tr>
<td>IntersectOf</td>
<td>3 (9.68%)</td>
<td>2 (3.51%)</td>
<td>22 (25.00%)</td>
<td></td>
<td>27 (9.34%)</td>
</tr>
<tr>
<td>AllValuesFrom</td>
<td>6 (19.35%)</td>
<td>3 (5.26%)</td>
<td>22 (25.00%)</td>
<td></td>
<td>31 (10.73%)</td>
</tr>
<tr>
<td>SomeValuesFrom</td>
<td>1 (3.23%)</td>
<td>34 (59.65%)</td>
<td>23 (20.14%)</td>
<td></td>
<td>58 (20.07%)</td>
</tr>
<tr>
<td>MaxCard</td>
<td>2 (6.45%)</td>
<td>10 (17.54%)</td>
<td>9 (10.23%)</td>
<td></td>
<td>21 (7.27%)</td>
</tr>
<tr>
<td>MinCard</td>
<td>6 (10.53%)</td>
<td>23 (20.14%)</td>
<td>29 (30.03%)</td>
<td></td>
<td>58 (20.07%)</td>
</tr>
<tr>
<td>Cardinality</td>
<td>16 (28.07%)</td>
<td>19 (21.59%)</td>
<td>35 (12.11%)</td>
<td></td>
<td>86 (17.05%)</td>
</tr>
<tr>
<td>HasValue</td>
<td>18 (20.45%)</td>
<td>23 (26.14%)</td>
<td>31 (10.73%)</td>
<td></td>
<td>86 (17.05%)</td>
</tr>
<tr>
<td>OneOf</td>
<td>15 (17.05%)</td>
<td>15 (17.05%)</td>
<td>15 (17.05%)</td>
<td></td>
<td>45 (14.94%)</td>
</tr>
<tr>
<td>DiffFrom</td>
<td>4 (4.55%)</td>
<td>4 (4.55%)</td>
<td>4 (4.55%)</td>
<td></td>
<td>12 (4.19%)</td>
</tr>
<tr>
<td>UnionOf</td>
<td>37 (42.05%)</td>
<td>37 (42.05%)</td>
<td>37 (42.05%)</td>
<td></td>
<td>111 (38.41%)</td>
</tr>
<tr>
<td>DisjointWith</td>
<td>41 (46.59%)</td>
<td>41 (46.59%)</td>
<td>41 (46.59%)</td>
<td></td>
<td>123 (40.12%)</td>
</tr>
<tr>
<td>ComplementOf</td>
<td>6 (6.82%)</td>
<td>6 (6.82%)</td>
<td>6 (6.82%)</td>
<td></td>
<td>18 (6.23%)</td>
</tr>
<tr>
<td>SameAs</td>
<td>2 (2.27%)</td>
<td>2 (2.27%)</td>
<td>4 (4.55%)</td>
<td></td>
<td>8 (2.75%)</td>
</tr>
</tbody>
</table>

- **Consistency** checks whether the ABox is consistent with respect to the TBox.
- **Instance checking** checks whether an assertion is entailed by the ABox.
- **Retrieval problem** retrieves all individuals that instantiate a class C, dually we can find all named classes C that an individual a belongs to.
- **Property fillers** retrieves, given a property R and an individual i, all individuals x which are related with i via R. Similarly we can retrieve the set of all named properties R between two individuals i and j, ask whether the pair (i, j) is a filler of P or ask for all pairs (i, j) that are fillers of P.
- **Conjunctive Queries** Conjunctive queries are a popular query formalism capable of expressing the class of selection/projection/join/renaming relational queries.

As ABox reasoning task we focus on answering conjunctive queries, as the vast majority of query languages for many data models used in practice fall into this fragment and conjunctive queries have been found useful in diverse practical applications.

### 4.2 Performance Measures

Our primary performance measure is response time, i.e. the time (in milliseconds) that is needed to solve the given reasoning task. This means that we ignore the utilization of system resources, which could be another interesting measure for performance.

In our benchmarks we separate prepare time and query time to fairly compare the performance of the reasoners w.r.t. the different test ontologies:

- **Setup Time** (P): Includes the time to do some important preparation before querying, e.g. load ontologies and check ABox consistency.
- **Response Time** (Q): Starts with executing the query and ends when all the query results were stored into a local variable. Usually, the query time means when a query is executed while not including the time for iterating the results.

#### 4.3 Datasets and Queries

For the language fragments identified in section 3, we chose a representative ontology for each fragment. The ontologies were chosen for two reasons. Firstly, they are popular, well established ontologies, which have been used in previous benchmarks. Secondly, they represent the cluster of ontologies as identified in section 3 in terms of size and ontological features used.

For each ontology, we used different datasets with increasing ABox size. Apart from one ontology (LUBM), which comes with its own ABox generator, the datasets are generated by duplicating originally existing ABox axioms.

For each ontology we used test queries that were either adopted from previous benchmarks, or explicitly defined for this benchmark. In particular we focused on conjunctive queries, as these are crucial in terms of complexity and response time.
VICODI. As a representative of the RDFS(DL) fragment, we used the VICODI ontology\(^2\), and the following two ABox queries adopted from Motik and Sattler [10]:

\[
Q_{v_1}(x) \equiv \text{Individual}(x) \quad (1)
\]
\[
Q_{v_2}(x, y, z) \equiv \text{Military-Person}(x) \quad \text{hasRole}(y, x), \text{related}(x, z) \quad (2)
\]

SWRC. As a representative for the DLP fragment, we used the Semantic Web for Research Communities (SWRC) ontology\(^3\). The ontology is clearly settled above the RDFS(DL) fragment (in terms of expressiveness), since it contains universal quantification. However, this occurs only in the superclass description of class expressions, which keeps the ontology in the DLP fragment [4]. The following queries have been used for our benchmark:

\[
Q_{s_1}(x) \equiv \text{PhDStudent}(x) \quad (3)
\]
\[
Q_{s_2}(x, y) \equiv \text{ResearchTopic}(x),
\]
\[
\text{isWorkedOnBy}(y, x, \text{“id2042instance”}),
\]
\[
\text{dealtWithIn}(x, y), \quad (4)
\]

LUBM. The Lehigh University Benchmark (LUBM)\(^4\) [5] was explicitly designed for OWL benchmarks. It models a scenario of the university domain and comes with its own ABox generator and a set of queries. Due to existential restriction on the right side of class expressions, the LUBM ontology is in OWL Lite and just beyond the DL fragment [4]. We used the following three queries out of the LUBM queries:

\[
Q_{l_1}(x, y_1, y_2, y_3) \equiv \text{Professor}(x),
\]
\[
\text{worksFor}(x, \text{“University0.edu”}),
\]
\[
\text{mastersDegreeFrom}(x, y_1),
\]
\[
\text{undergraduateDegreeFrom}(x, y_2),
\]
\[
\text{teacherOf}(x, y_3) \quad (5)
\]
\[
Q_{l_2}(x) \equiv \text{Person}(x),
\]
\[
\text{memberOf}(x, \text{“University0.edu”}) \quad (6)
\]
\[
Q_{l_3}(x, z) \equiv \text{Student}(x)
\]
\[
\text{Department}(y),
\]
\[
\text{memberOf}(x, y),
\]
\[
\text{subOrganizationOf}(y, \text{“University0.edu”}) \quad (7)
\]

Wine. The Wine ontology\(^5\) is a prominent example of an OWL DL ontology. Since some reasoners are not able to handle nominals, we used the same datasets for the Wine ontology, as in previous benchmarks (cf. [10]), where nominals have been removed. We defined the following three ABox queries:

\[
Q_{w_1}(x) \equiv \text{SemillonOrSauvignonBlanc}(x) \quad (8)
\]
\[
Q_{w_2}(x) \equiv \text{DessertWine}(x),
\]
\[
\text{locatedIn}(x, \text{“GermanyRegion”}) \quad (9)
\]
\[
Q_{w_3}(x) \equiv \text{hasFlavor}(x, \text{“Strong”}),
\]
\[
\text{locatedIn}(x, \text{“NewZealandRegion”}),
\]
\[
\text{hasSugar}(x, \text{“Dry”}) \quad (10)
\]

4.4 Discussion

We built our benchmark on several requirements, motivated by the work of Weithöner et al. [13]. Firstly, we distinguish between setup time and response time. This distinction is necessary to demonstrate strengths and weaknesses of reasoners, that follow different paradigms, in particular materialization of inferred ABox assertions in the setup stage (e.g. Sesame, OWLIM). Our benchmark demonstrates how these approaches work on fairly simple ontologies (w.r.t. TBox complexity), as well as more complex ontologies, which these reasoners are unable to load at all. Secondly, we evaluate these measurements w.r.t. differently scaled ABoxes, but constant TBox. By doing so, we picked up methods, that have been used in previous benchmarks, which scale up the ABox by duplicating existing ABox assertions. While we evaluated reasoning tasks on four different expressivity (complexity) classes regarding TBox complexity, we did not put any effort in increasing TBox complexity for given ontologies. We rather focused on representative ontologies of given complexity for different classes of ontologies. We used only native interfaces of the different reasoners, and did not consider higher abstraction layers such as DIG. The influence of different interfaces is negligible anyway for querying large ontologies [13]. We did not evaluate cache influence or ABox changes on consecutive query requests, which goes beyond the scope of this paper. We also did not consider different serializations of ontologies, as we focus on well established ontologies, that do not occur in different serializations.

5. OVERVIEW OF REASONERS

In this section we provide a short overview of the reasoners we used for our evaluations. Roughly, the reasoners can be grouped according to the employed reasoning techniques into three groups: In the first class of traditional DL reasoners (e.g. Racer, Pellet), tableau-based algorithms are used to implement the inference calculus. A second alternative relies on the reuse of the techniques of deductive databases, based on a transformation of an OWL ontology into a disjunctive datalog program and to the utilization of a disjunctive datalog engine for reasoning as implemented in KAON2. A final class of reasoners – including Sesame and OWLIM – use standard rule engine to reason with OWL. Often the consequences are materialized when the ontology is loaded. However, this procedure is in principle limited to less expressive language fragments.

Table 2 shows an overview of the reasoners along with the language fragments they support.

5.1 Sesame
Sesame\(^6\) is an open source repository for storing and querying RDF and RDFS information. OWL ontologies are simply treated on the level of RDF graphs. Sesame enables the connection to DBMS (currently MySQL, PostgreSQL and Oracle) through the SAIL (the Storage and Inference Layer) module, and also offers a very efficient direct to disk Sail called Native Sail. Sesame provides RDFS inferencing and allows querying through SesRQL, RQL, RDQL and SPARQL. Via the SAIL, it is also possible to extend the inferencing capabilities of the system. (In fact, this is how the OWLIM reasoner is realized.) The main ways to communicate with the Sesame modules are through the Sesame API or through the Sesame Server, running within a Java Servlet Container.

5.2 OWLIM

OWLIM is semantic repository and reasoner, packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database. OWLIM uses the TRREE engine to perform RDFS, and OWL DLP reasoning. It performs forward-chaining of entailment rules on top of RDF graphs and employs a reasoning strategy, which can be described as total materialization. OWLIM offers configurable reasoning support and performance. In the "standard" version of OWLIM (referred to as SwiftOWLIM) reasoning and query evaluation are performed in-memory, while a reliable persistence strategy assures data preservation, consistency and integrity.

5.3 KAON2

KAON2 is a free (free for non-commercial usage) Java reasoner for \(\mathcal{SHIQ}(D)\) extended with the DL-safe fragment of SWRL. It implements a resolution-based decision procedure for general TBoxes (subsumption, satisfiability, classification) and ABoxes (retrieval, conjunctive query answering). It comes with its own, Java-based interface, and supports the DIG-API.

5.4 Racer

The Racer system \([7]\) is an optimized tableau reasoner for the description logic \(\mathcal{SHIQ}(D)\). For concrete domains, it supports integers and real numbers, as well as various polynomial equations over those, and strings with equality checks. It can handle several TBoxes and several ABoxes and treats individuals under the unique name assumption. Besides basic reasoning tasks as satisfiability and subsumption, it offers ABox querying based on the nRQL optimizations. It is implemented in the Common Lisp programming language. It supports the OWL-API and the DIG-API and comes with numerous other features. Recently, Racer has been turned into the commercial (free trials and research licenses available) RacerPro\(^7\) system, which we used for our experiments.

5.5 Pellet

Pellet\(^8\) \([11]\) is a free open-source Java-based reasoner for \(\mathcal{SROIQ}\) with simple data types (i.e., for OWL 1.1). It implements a tableau-based decision procedure for general TBoxes (subsumption, satisfiability, and classification) and ABoxes (retrieval, conjunctive query answering). Pellet employs many of the optimizations for standard DL reasoning as other state-of-the-art DL reasoners. It directly supports entailment checks and optimised ABox querying through its interface. Pellet supports the OWL-API, the DIG-API, and Jena interface.

<table>
<thead>
<tr>
<th>Table 2: Overview of Reasoners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fragment</td>
</tr>
<tr>
<td>Example</td>
</tr>
<tr>
<td>Sesame</td>
</tr>
<tr>
<td>OWLIM</td>
</tr>
<tr>
<td>KAON2</td>
</tr>
<tr>
<td>Racer</td>
</tr>
<tr>
<td>Pellet</td>
</tr>
</tbody>
</table>

\(^{a}\)Except for nominals

6. EXPERIMENTS

In this section we report on the benchmarking experiments performed.

6.1 Test Settings

Our tests were performed on a laptop with a 1.69 GHz Intel Pentium-M processor and 1 GB of RAM using Windows XP Service Pack 2. For each reasoning task, the time-out period was assigned 5 minutes. Sun’s Java 1.5.0 Update 6 was used for Java-based tools and the maximum heap space was set to 800MB. More details about test settings for different reasoners are described as followings.

For our tests, we chose the default OWL reasoner for Jena which implements an incomplete implementation of OWL-FULL. The version 2.5.2 is used. It is noted that the results from queries are streamed. That is, they are not all in-memory at the same time and not when the query is executed. The execution of a query only sets up the query and creates the required iterators. While the real reasoning about a query will be done when iterating the results, which is different from other reasoners. So Jena spends very little time on executing a query but much more time for the iteration of the results.

We use version 1.9.1 Beta for RacerPro and 1.3 for Pellet. Specifically, Java-based API for RacerPro (JRacer 1.8) is used to connect RacerPro server and OWL API is used to access Pellet. The two reasoners usually take much longer time to check ABox consistency before answering the first query than computing query results.

As for Sesame, we also used the default reasoner rather but not the custom one, since we did not expect the users are experts to configure the reasoner to optimize the performance of the reasoner. The default reasoner of Sesame only does standard RDF Schema reasoning. Thus, this will influence the results of queries in some aspects.

OWLIM 2.9 with Sesame 1.2.6 was used, since it has not been tested with the newest version of Sesame. As it is assumed that we did not know which kind of rule set fits the ontology to be tested, no rule set was used for our tests. Besides, both Sesame and OWLIM are main-memory based reasoning, which take much longer time to load the ontologies into main-memory while spending quite little time to do queries.

For each reasoning task, a new instance of the reasoner...
is created and the testing ontology is loaded. No methods about optimization for the reasoners are called and we just use the default settings by assuming the general users have no too much knowledge to make the optimization.

6.2 The analysis of test results

6.2.1 Loading

Figure 1: Average load time for all the systems

In Figure 1, the average loading time of all the ontologies and systems to be tested for executing A-Box queries. Generally, for each data set with the same T-Box but different size of A-Box, the load time increases when the size of A-Box is increasing. KAON2, OWLIM and Sesame can load all the ontologies. However, jena can not load all the Wine ontologies. Racer fails to check A-Box consistency, so the load time is marked as time out. Jena, Pellet and Racer also can not load the ontologies with a relatively complex TBox and quite large A-Box, such as lubm_3, lubm_4 and wine_10. It is noted that KAON2 performs best to load Wine ontology.

Figure 2: Load Time comparison between the systems

To deeply analyze the loading time, Figure 2 only compares systems on those ontologies which can be processed and loaded by all systems. Figure 2 shows the time needed by the systems to load a given ontology and perform a first A-Box consistency check. The load time obviously depends on the size of the ontology. KAON2 is consistently the fastest system and scales more gracefully than all other systems. Sesame’s high load time is due to optimizations calculated at load time to speed up query processing. RacerPro additionally has problems with executing the A-Box consistency check for large ontologies9.

6.2.2 Classification

Figure 3: Classification for all the systems

For classification, we consider Classification as a representative task for T-Box reasoning tasks. The load time here only means the time to load ontology and no A-Box consistency need to be checked. So we can see that the load time for classification for Pellet and Racer is obviously less than that for A-Box queries. It is obviously observed from Figure 3 that Jena spent quite a lot time on classifying the ontologies which can be loaded, comparing with other systems. Figure 4 compares the results for classification among the expressive ontologies that can be processed by all systems except Jena10.

Both OWLIM and Sesame are incomplete, i.e. they cannot handle the classification task and return explicitly stated subsumption relationships only, which explains their speed. The Tableau-based RacerPro is the fastest system to compute the classification, leveraging optimizations computed at load time. The Resolution-based KAON2 system is significantly slower but still much better than Pellet and actually the only system that can classify all ontologies (see Figure 3). Pellet additionally does not scale well with the increasing size of the ontology, which shows that implementation usually matters more than actually algorithmic technique

9 vicodi_4, lubm_3, lubm_4 and all of the SWRC ontologies
10 Unfortunately, Jena can not load LUBM and Wine ontology, RacerPro has difficulties with processing the SWRC ontology and Pellet cannot handle the largest WIN...
6.2.3 Conjunctive Queries

Figure 5 gives an overall view of the average time for conjunctive queries. More details about some specific results are discussed below.

Sesame and OWLIM are the clear winners with respect to processing conjunctive queries on RDFS(DL) ontologies as Figure 6 shows. Apparently there is a clear trade-off between loading and query time as Sesame is the slowest system to load but the fastest system to respond.

KAON2 still shows favorable performance on the query that plays nice with the loading time and scalability while Pellet and RacerPro are not only significantly slower but also significantly less scalable with respect to size of the A-Box.

Turning to more expressive ontologies, where only KAON2, RacerPro and Pellet are able to process the semantics of the ontology language. Figure 7 shows that RacerPro outperforms KAON2 for small ontologies but is significantly less scalable than KAON2 resulting in a time out for the largest WINE ontology. The performance of Pellet lags behind and produces time outs as soon as the A-Box produces medium size.

For KAON2, the same performance developed observed for classification, i.e also holds for A-Box queries, with an average response time of 7.17. Beyond RDF(S) ontologies, Sesame and OWLIM deliver only incomplete results on DLP, OWL Lite and OWL DL ontologies. As we could see in the setup test Pellet cannot process ontologies with larger size (fails on lubm_4, all of wine etc.). However, this results also holds for RacerPro which can process even fewer ontologies.

6.2.4 Recommendation

The detailed results show that there is no clear winner for all types of queries and ontologies even though KAON2 provides a reasonable general purpose reasoner that behaves well in all cases.

Figure 8 depicts a differentiated case along two main dimensions language complexity and A-Box size. While RacerPro is the system of choice in settings with high complexity and small A-Boxes, OWLIM can be generally recommended for low complexity settings while KAON2 is the best alternative for all other cases.
7. CONCLUSION

In today’s landscape of ontologies, we observe a wide spectrum of ontologies that differ in terms of the expressiveness of their used language fragments, as well as their complexity in terms of their size of TBox and ABox. A number of rather different reasoning techniques are implemented in state-of-the-art OWL reasoners. In our benchmarks we have shown that it is important to understand the strengths and weaknesses of the different approaches in order to select an adequate reasoner for a given reasoning task. It does not come as a surprise that there is no clear winner that performs well for all types of ontologies and reasoning tasks.

As general conclusions we can summarize our results in that (1) reasoners that employ a simple rule engine scale very well for large A-Boxes, but are in principle very limited to light-weight language fragments, (2) classical tableau reasoners scale well for complex T-Box reasoning tasks, but are limited with respect to their support for large A-Boxes, and (3) the reasoning techniques based on reduction to disjunctive datalog as implemented in KAON2 scale well for large A-Boxes, while at the same time they support rich language fragment.

An important current research topic is the investigation of tractable fragments of the OWL language [9] and the development of reasoners specialized for these fragments. As future work, we will extend our analysis taking these new fragments into account.

8. REFERENCES


