# Proceedings of the 2nd OWL Reasoner Evaluation Workshop (ORE 2013)

Collocated with DL 2013 Workshop
July 22nd, Ulm, Germany

Volume 1015 of CEUR-WS.org: http://ceur-ws.org/Vol-1015/

Workshop webpage: http://ore2013.cs.manchester.ac.uk/

**Edited by**
Samantha Bail, Birte Glimm, Rafael Gonçalves, Ernesto Jiménez-Ruiz,
Yevgeny Kazakov, Nicolas Matentzoglu and Bijan Parsia

# Preface

OWL is a logic-based ontology language standard designed to promote interoperability, particularly in the context of the (Semantic) Web. The standard has encouraged the development of numerous OWL reasoning systems, and such systems are already key components of many applications.

The goal of this workshop is to bring together both developers and users of reasoners for (subsets of) OWL, including systems focusing on both intensional (ontology) and extensional (data) query answering.

This volume contains the papers presented at ORE 2013: The 2nd International Workshop on OWL Reasoner Evaluation, held in Ulm, Germany, on July 22, 2013. ORE 2013 was collocated with the 26th edition of the DL workshop. The workshop received a 18 submissions (14 system papers and 4 ontology/benchmark papers) each of which was reviewed by at least three members of the program committee or additional reviewers. Since there was not any off-topic submission, we accepted all submission, following the inclusive tradition of DL, for oral presentation at the workshop.

In addition to workshop paper submissions, ORE 2013 also included a competition in which OWL reasoners were faced with different reasoning task, such as ontology classification, consistency checking, and satisfiability checking of concepts. The tasks were performed on several large corpora of real-life OWL ontologies obtained from the web, as well as user-submitted ontologies which were found to be challenging for reasoners. The proceedings also contains a short report summarizing the main results of the competition.

Fourteen OWL reasoners participated in the ORE 2013 competition:

- BaseVISor `http://vistology.com/basevisor/basevisor.html`
- TrOWL `http://trowl.eu/`
- Konclude `http://www.derivo.de/en/produkte/konclude/`
- ELepHant `https://code.google.com/p/elephant-reasoner/`
- TReasoner `https://code.google.com/p/treasoner/`
- HermiT `http://www.hermit-reasoner.com/`
- MORe `http://code.google.com/p/more-reasoner/`
- ELK `http://code.google.com/p/elk-reasoner/`
- jcel `http://jcel.sourceforge.net/`
- SnoRocket `http://research.ict.csiro.au/software/snorocket`
- FaCT++ `http://code.google.com/p/factplusplus/`
- Jfact `http://sourceforge.net/projects/jfact/`
- Chainsaw `http://sourceforge.net/projects/chainsaw/`
- WSClassifier `https://code.google.com/p/wsclassifier/`

## Acknowledgements

We thank all members of the program committee, competition organisers, additional reviewers, authors of the submitted papers, developers of the submitted reasoners and ontologies, and local organizers for their invaluable effort.

We also thank Konstantin Korovin (supported by the Royal Society grant RG080491) at the University of Manchester who kindly provided us with the PC cluster for the competition.

We also gratefully acknowledge the support of our sponsors. In particular, we thank the main workshop sponsor: B2i Healthcare (`https://www.b2international.com/`).

We would also like to acknowledge that the work of the ORE organisers was greatly simplified by using the EasyChair conference management system (`http://www.easychair.org`) and the CEUR Workshop Proceedings publication service (`http://ceur-ws.org/`).

## Organisers, PC Chairs

| | |
|---|---|
| Samantha Bail | University of Manchester |
| Ernesto Jiménez-Ruiz | University of Oxford |

## Competition Organisers

| | |
|---|---|
| Rafael Gonçalves | University of Manchester |
| Nicolas Matentzoglu | University of Manchester |
| Bijan Parsia | University of Manchester |

## Local Organisers

| | |
|---|---|
| Birte Glimm | University of Ulm |
| Yevgeny Kazakov | University of Ulm |

## Program Committee

| | |
|---|---|
| Ana Armas | University of Oxford |
| Franz Baader | TU Dresden |
| Christine Golbreich | LIRMM, CNRS, Montpellier & Univ. Versailles Saint-Quentin |
| Janna Hastings | European Bioinformatics Institute |
| Pavel Klinov | University of Ulm |
| Despoina Magka | University of Oxford |
| Francisco Martin-Recuerda | Universidad Politécnica de Madrid |
| Christian Meilicke | University of Mannheim |
| Julian Mendez | TU Dresden |
| Maria Del Mar Roldán García | Universidad de Malaga |
| Stefan Schlobach | Vrije Universiteit Amsterdam |
| Kavitha Srinivas | IBM Research |
| Dmitry Tsarkov | University of Manchester |
| Zhe Wang | Griffith University |

## Additional Reviewers

| | |
|---|---|
| Andrew Bate | University of Oxford |
| David Carral | Wright State University |
| Maria Copeland | University of Manchester |
| Jared Leo | University of Manchester |
| Jose Mora | Universidad Politécnica de Madrid |
| Weihong Song | University of New Brunswick |
| Yujiao Zhou | University of Oxford |

# Table of Contents

## Evaluation results

OWL Reasoner Evaluation (ORE) Workshop 2013 Results: Short Report ..... 1
*Rafael Gonçalves, Samantha Bail, Ernesto Jiménez-Ruiz, Nicolas Matentzoglu, Bijan Parsia, Birte Glimm and Yevgeny Kazakov*

## System papers

YARR!: Yet Another Rewriting Reasoner ..... 19
*Joerg Schoenfisch and Jens Ortmann*

TReasoner: System Description ..... 26
*Andrew Grigorev and Alexander Ivashko*

Snorocket 2.0: Concrete Domains and Concurrent Classification ..... 32
*Alejandro Metke Jimenez and Michael Lawley*

A Transformation Approach for Classifying $\mathcal{ALCHI}(\mathcal{D})$ Ontologies with a Consequence-based $\mathcal{ALCH}$ Reasoner ..... 39
*Weihong Song, Bruce Spencer and Weichang Du*

Android goes Semantic: DL Reasoners on Smartphones ..... 46
*Roberto Yus, Carlos Bobed, Guillermo Esteban, Fernando Bobillo and Eduardo Mena*

FRaQuE: A Framework for Rapid Query Processing Evaluation ..... 53
*Jean-Rémi Bourguet and Luca Pulina*

MORe: a Modular OWL Reasoner for Ontology Classification ..... 61
*Ana Armas, Bernardo Cuenca Grau, Ian Horrocks and Ernesto Jiménez-Ruiz*

Experimenting with ELK Reasoner on Android ..... 68
*Yevgeny Kazakov and Pavel Klinov*

Extending Datatype Support for Tractable Reasoning with OWL 2 EL Ontologies ..... 75
*Oleksandr Pospishnyi*

DRAOn: A Distributed Reasoner for Aligned Ontologies ..... 81
*Chan Le Duc, Myriam Lamolle, Antoine Zimmermann and Olivier Curé*

The ELepHant Reasoner System Description ..... 87
*Bariş Sertkaya*

Evaluating SPARQL-to-SQL translation in ontop ..... 84
*Mariano Rodriguez-Muro, Martin Rezk, Josef Hardi, Mindaugas Slusnys, Timea Bagosi and Diego Calvanese*

OBDA with Ontop ..... 101
*Mariano Rodriguez-Muro, Roman Kontchakov and Michael Zakharyaschev*

Reasoning the FMA Ontologies with TrOWL ..... 107
*Jeff Z. Pan, Yuan Ren, Nophadol Jekjantuk and Jhonatan Garcia*

## Ontology and benchmark papers

*KB_Bio_*101 : A Challenge for OWL Reasoners ..... 114
*Michael Wessel, Vinay Chaudhri and Stijn Heymans*

Evaluating OWL 2 Reasoners in the context of Clinical Decision Support in Lung Cancer Treatment Selection ..... 121
*Berkan Sesen, Ernesto Jimenez-Ruiz, Rene Banares-Alcantara and Michael Brady*

Genomic CDS: an example of a complex ontology for pharmacogenetics and clinical decision support ..... 128
*Matthias Samwald*

A large-scale gene-centric semantic web knowledge base for molecular biology ..... 134
*Jose Cruz-Toledo, Alison Callahan and Michel Dumontier*

# OWL Reasoner Evaluation (ORE) Workshop 2013 Results: Short Report

Rafael S. Gonçalves[1], Samantha Bail[1], Ernesto Jimenez-Ruiz[2], Nicolas Matentzoglu[1], Bijan Parsia[1], Birte Glimm[3], and Yevgeny Kazakov[3]

[1] School of Computer Science, The University of Manchester, UK
[2] Department of Computer Science, University of Oxford, UK
[3] Institut für Künstliche Intelligenz, Ulm University, Germany

**Abstract.** The OWL reasoner evaluation (ORE) workshop brings together reasoner developers and ontology engineers in order to discuss and evaluate the performance and robustness of modern reasoners on OWL ontologies. In addition to paper submissions, the workshop featured a live and offline reasoner competition where standard reasoning tasks were tested: classification, consistency, and concept satisfiability. The reasoner competition is performed on several large corpora of real-life OWL ontologies obtained from the web, as well as user-submitted ontologies which were found to be challenging for reasoners. Overall there were 14 reasoner submissions for the competition, some of which dedicated to certain subsets or profiles of OWL 2, and implementing different algorithms and optimisations. In this report, we give an overview of the competition methodology and present a summary of its results, divided into the respective categories based on OWL 2 profiles and test corpora.

## 1 Introduction

The OWL Reasoner Evaluation Workshop (ORE) aims at being an international venue for the annual systematic evaluation of reasoners for (subsets of) the Web Ontology Language OWL [9,3] and bringing together both users and developers of such reasoners. The first ORE workshop was organized in 2012 as a satellite event[4] of the IJCAR conference [10], and started as an initiative in the context of the SEALS (Semantic Evaluation At Large Scale) project [29]. In 2013 the ORE workshop was organized together with the Description Logic (DL) workshop.

This report summarizes the results of the ORE 2013 reasoner competition. All test data, results, and further information about the competition are available online: `http://ore2013.cs.manchester.ac.uk`.

The remainder of the report is organized as follows. In Section 2, we present the methodology of the competition. Section 3 provides a brief description of each participating OWL reasoner. The results of the offline and live competitions are shown in Sections 4 and 5, respectively. In Section 6 we present the results for the user-submitted ontologies. Finally, Section 7 provides a summary of the competition results.

[4] `http://www.cs.ox.ac.uk/isg/conferences/ORE2012/`

## 2 Methodology

We start by describing the reasoning tasks considered in the competition, followed by the presentation of the benchmark framework created for ORE 2013. Subsequently, we report on the hardware and ontology corpora used.

### 2.1 Reasoning tasks

The competition was based on three standard reasoning tasks, namely classification, consistency checking, and concept satisfiability checking. The call for submissions also included query answering, but there were no reasoners submitted for this task.

**2.1.1 Ontology classification** The classification task was chosen as the most complex of the three tasks. Given an ontology, the reasoners were asked to return an ontology file (parseable by the OWL API) in OWL functional syntax, containing a set of `SubClassOf` axioms of the form $\alpha := A \sqsubseteq B$, for named concepts $A$, $B \in sig(\mathcal{O})$, where $\mathcal{O} \models \alpha$ according to the following specifications:

1. Non-tautology:
   - $A \in sig(\mathcal{O}) \cup \top$
   - $B \in sig(\mathcal{O}) \cup \bot$
   - $A \neq B$
2. Directness: there exists no named concept $C \in sig(\mathcal{O})$ s.t. $\mathcal{O} \models A \sqsubseteq C$ and $C \sqsubseteq B$, where $C$ is not equivalent to $A$, $B$, or $\bot$.
3. Conciseness: if $\mathcal{O} \models A \equiv \bot$, the only axiom with $A$ on the left-hand side is $A \sqsubseteq \bot$.
4. Consistency: if the given ontology is inconsistent, the only output is the axiom $\top \sqsubseteq \bot$.
5. Non-strictness: if $\mathcal{O} \models A \equiv B$, output $A \sqsubseteq B$ and $B \sqsubseteq A$.

These specifications were selected in order to obtain a set of `SubClassOf` axioms that would represent all subsumptions between named classes, while omitting irrelevant information.

**2.1.2 Ontology consistency** For this task, the reasoner was asked to test the consistency of the ontology (i.e. whether $\mathcal{O} \models \top \sqsubseteq \bot$), and return 'true' or 'false', respectively.

**2.1.3 Concept satisfiability** This task was performed by randomly selecting ten concepts from each ontology in the respective corpus, giving precedence to unsatisfiable concepts where possible. The reasoner was then asked to test the satisfiability of the concept, i.e. whether $\mathcal{O} \models A \equiv \bot$ for a named concept $A$, and return 'true' or 'false', respectively.

### 2.2 Benchmark framework

**2.2.1 Implementation** The aim of the benchmarking framework is to work with as many different reasoner configurations as possible, without the need to interfere with reasoner internals. We therefore asked the system developers to write a simple executable wrapper for their reasoner which would accept input arguments (ontology file name, reasoning task, output directory, concept name) and output results according to our specification (a valid OWL file with the class hierarchy, 'true'/'false' for the consistency and satisfiability tasks, as well as the time taken for the task, or a separate file with an error trace).

The time measured is the wall-clock time (in milliseconds) elapsed from the moment preceding reasoner creation (e.g. before the call to `ReasonerFactory.createReasoner(ontology)` in the OWL API [8] where the ontology has already been parsed into an OWL object) to the completion of the given task, i.e. it includes the loading and possibly pre-processing time required by the *reasoner*, but excludes time taken for file I/O. While measuring CPU time would be more accurate, it comes with added complexity for concurrent implementations – for instance, in Java, one would have to aggregate the run times of each thread. The reasoners are also asked to enforce a five minute timeout, that is, if the measured time exceeds 5 minutes then the reasoner should stop the ongoing operation, and terminate itself. Failure to do so will trigger a kill command sent to the running process after another minute in order to give enough time for the process to terminate; i.e. the hard timeout is six minutes.

While one might argue that leaving the reporting of operation times to the reasoners may be error-prone, we believe that letting reasoner developers themselves handle the input and output of their system, as well as the time measurement, is the most straightforward way to include as many systems as possible; regardless of their implementation programming language, whether they use the OWL API, employ concurrent implementations, and so on. The large number of reasoners that was submitted to the competition shows that writing this simple wrapper script lowered the barrier for participation, and despite some difficulties with non-standard output, most reasoners adhered to the specifications closely enough for us to analyse their outputs.

Additionally, it is clear that reasoners which do *not* implement the five minute timeout, but rather rely on the kill signal after the six minute timeout sent by the benchmark framework, *could* potentially gain a slight advantage through this additional minute. However, not only is the number of successfully completed tasks between the five and six minute marks negligible, but also we have automatically induced a timeout for those reasoners that exceeded a runtime of five minutes for some input.

**2.2.2 Correctness check** The reasoner output was checked for correctness by a majority vote, i.e. the result returned by the most reasoners was considered to be correct.[5] Since the ontologies in the test corpus were 'real-life' ontologies,

[5] Unless most reasoners return an empty OWL file, in which case the majority vote is taken based on those reasoners which output a non-empty file.

this was the most straightforward way to automatically determine correctness without manual inspection or artificially generating test cases.

In the case of the consistency and satisfiability challenges the output was a simple unambiguous 'true' or 'false', so any incorrect results were unlikely to be caused by erroneous output from a sound reasoner; however, for ontology classification, the reasoners output an ontology file containing OWL `SubClassOf` axioms, which may lead to errors if the systems did not exactly follow the above specifications on which types of axioms to include or exclude. For the purpose of verifying correctness of the output we rely on an ontology *diff* to determine whether two given results are logically equivalent [6]. The diff is tuned to ignore (1) tautological axioms of the type $A \sqsubseteq \top$ for any named concept $A$, (2) axioms of the form $\bot \sqsubseteq B$ or $A \sqsubseteq B$, where $A, B$ are named concepts and $A$ is unsatisfiable, and (3) if two result files are not equivalent due to OWL `EquivalentClassOf` axioms, these axioms are ignored.[6]

**2.2.3 Success and failure** In the end, the outcome of a reasoning task on an ontology was either 'success' or 'fail'. A reasoner would pass the test ('solve the problem') successfully if it met the following three criteria:

- Process the ontology without throwing an error (e.g. parsing error, out of memory, unsupported OWL feature, etc.).
- Return a result within the allocated timeout.
- Return the *correct* result (based on the majority vote).

Likewise, a reasoner would *fail* a task if it did one of the following:

- Throw an error and abort the reasoning task.
- Return no result within the allocated time.
- Return an incorrect result (based on the majority vote).

Note that these criteria mean that a reasoner could successfully solve a task while being unsound or incomplete, or without completing the reasoning task within the allocated time. For example, for the classification task, if the reasoner has already found all required entailed atomic subsumptions without performing all possible entailment checks within the five minute time frame, it can simply output this 'intermediate' result before terminating the process. Since the correctness check is performed on whatever the reasoner returns within the timeout, the resulting output would be considered to be correct, despite the fact that the reasoner has not fully completed the task.

Likewise, a reasoner which does not support certain OWL 2 features, such as datatypes, might find (if there are *any to find*) the required atomic subsumptions via some other 'route' if there are several reasons why the entailment holds. In other words, if there exist multiple *justifications* (minimal entailing subsets) for a subsumption of which at least one only contains *supported* features, then the reasoner will still be able to find the subsumption without having to process the

[6] While the presence of equivalences in some result should not a problem when compared to a result with these equivalences in subsumption form, reasoners tend not to produce the latter because they are non-strict subsumptions, so we allowed equivalences and tuned our diff to ignore them where applicable.

unsupported feature. This is an issue we are planning to address with the next iteration of the benchmark framework by modifying ontologies (i.e. 'breaking' their justifications) in order to specifically test certain OWL 2 features.

### 2.3 Hardware

The experiments were run on a cluster of identical computers (one reasoner per computer) that were made available to us by Konstantin Korovin of the iProver project[7] at The University of Manchester, supported by the Royal Society grant RG080491. Each computer had the following configuration:

- Intel Xeon QuadCore CPU @2.33GHz
- 12GB RAM (8GB assigned to the process)
- Running the Fedora 12 operating system
- Java version 1.6

### 2.4 Test corpora

**2.4.1 Main test corpus** For each of the OWL 2 profiles [16] used in the competition (OWL 2 EL, RL, and DL ontologies which were not in any of the sub-profiles) we gathered a test set of up to 200 ontologies. The pool of ontologies we sampled from was composed of three corpora: *(i)* the NCBO BioPortal[8] corpus [17], *(ii)* the Oxford Ontology Library[9], and *(iii)* the corpus of ontologies from the Manchester Ontology Repository[10] [13]. The corpora were filtered to only include OWL 2 ontologies which had at least 100 axioms and 10 named concepts. Note that the sample ontology pool, composed of 2499 ontologies, does contain some degree of duplication due to the intersection of BioPortal, the Manchester OWL Repository, and the Oxford Ontology Library.

The ontologies were then binned by profile, i.e. one bin for each of the following: OWL 2 EL ontologies, OWL 2 RL, and OWL 2 DL. Regarding the latter, we chose to include here those ontologies that do not fall into any of the sub-profiles (i.e. OWL 2 EL, RL, or QL) in order to ensure that features outside the sub-profiles were tested. For each of these profile bins, a stratified random sample was drawn to obtain a set of 200 ontologies:

- 50 small ontologies (between 100 and 499 logical axioms)
- 100 medium sized ontologies (between 500 and 4,999 logical axioms)
- 50 large ontologies (5,000 and more logical axioms)

Note that these thresholds and weightings were chosen based on the distribution of ontology sizes we have found in several ontology corpora which follow (roughly) a normal distribution, with a large number of medium-sized ontologies and fewer small and large ontologies. While it would have been possible to select

[7] http://www.cs.man.ac.uk/~korovink/iprover/
[8] http://bioportal.bioontology.org/
[9] http://www.cs.ox.ac.uk/isg/ontologies/
[10] http://owl.cs.manchester.ac.uk/owlcorpus

exclusively medium-sized and large ontologies, we also expected some small ontologies to be fairly complex for the reasoners, which is why they were included in the test corpus.

In addition to the ontologies from BioPortal, the Oxford Library, the Manchester Repository, and user-submitted ontologies, the May 2013 version of the National Cancer Institute (NCI) Thesaurus (NCIt) [5], and the January 2011 version of the Systematized Nomenclature of Medicine (SNOMED) Clinical Terms (SNOMED CT) [21] were also added to the corpus, respectively to the DL and EL profile bins.

The experiments were run on the OWL functional syntax serialisations of the selected ontologies, except for one reasoner (Konclude) which currently only supports OWL/XML syntax. A number of ontologies serialised into functional syntax (55 across all the sets) turned out to be broken (they were correctly loaded and serialised, but the serialisation could not be parsed back by the OWL API), possibly due to problems with the respective serialiser in the OWL API (version 3.4.4). These were replaced by random selections for their respective bin. The same occurred for 12 ontologies serialised into OWL/XML.

The entire sampling process was performed twice in order to create two complete test sets: Set A for the offline competition, and Set B for the live competition. Note that some ontologies occurred in both Set A and B: 40 ontologies occurred in both Set A and B for the DL category, Set A and B were fully identical for the EL category, and 29 ontologies were shared between Set A and B in the RL category.

**2.4.2 User-submitted ontologies** In the call for submissions to the ORE 2013 workshop, we also included a call for 'hard' ontologies and potential reasoner benchmark suites. Several groups of ontology and reasoner developers submitted their ontologies, which were either newly developed OWL ontologies or modifications of existing ones. These included:

- C. M. Keet, A. Lawrynowicz, C. d'Amato, M. Hilario: the Data Mining OPtimization Ontology (DMOP) [12], a complex ontology with around 3,000 logical axioms in the $\mathcal{SROIQ}(D)$ description logic which makes uses of all OWL 2 DL features.
- M. Samwald: Genomic CDS [20], an $\mathcal{ALCQ}$ ontology containing around 4,000 logical axioms, which involves a high number of qualified number restrictions of the type 'exactly 2'.
- V. Chaudhri, M. Wessel: Bio KB 101 [2], a set of OWL approximations of the first-order logic representation of a biology textbook, which consists of 432 different approximations containing various OWL 2 features. Only 72 of these files were in the OWL 2 DL profile and thus used for the reasoner evaluation.
- W. Song, B. Spencer, W. Du: three ontology variants:
  - FMA-FNL, a variant of the FMA (Foundational Model of Anatomy) ontology [19], a large and highly cyclic $\mathcal{ALCOI}(D)$ ontology with over 120,000 locial axioms.

  - GALEN-FNL, a highly cyclic $\mathcal{ALCHOI}(D)$ variant of the well-known Galen ontology [18], which contains around 37,000 logical axioms and 951 object properties.
  - GALEN-Heart: a highly cyclic $\mathcal{ALCHOI}(D)$ ontology containing a module extracted from the Galen ontology with over 10,000 logical axioms.
- S. Croset: Functional Therapeutic Chemical Classification System (FTC)[11], a large ontology with nearly 300,000 logical axioms in the OWL 2 EL profile.

As mentioned above, some of the user-submitted ontologies (all except Bio KB and DMOP) were added to the set used in the competition. Additionally, we also performed a separate benchmark on *all* of the user-submitted ontologies.

## 3 Participating reasoners

### 3.1 OWL 2 DL reasoners

**Chainsaw** [28] is a 'metareasoner' which first computes modules for an ontology, then delegates the processing of those modules to an existing OWL 2 DL reasoner, e.g. FaCT++ in the current implementation.

**FaCT++** [27] is a tableaux reasoner written in C++ which supports the full OWL 2 DL profile.

**HermiT** [4] is a Java-based OWL 2 DL reasoner implementing a hypertableau calculus.

**JFact** is a Java implementation of the FaCT++ reasoner with extended datatype support.[12]

**Konclude** is a C++ reasoner supporting the full OWL 2 DL profile except datatypes. It uses an optimised tableau algorithm which also supports parallelised processing of non-deterministic branches and the parallelisation of higher-level reasoning tasks, e.g. satisfiability and subsumption tests.[13]

**MORe** [1] is Java-based *modular* reasoner which integrates a fully-fledged (and slower) reasoner with a profile specific (and more efficient) reasoner. In the competition, MORe has integrated both HermiT and Pellet [24] as OWL 2 DL reasoners and ELK as the OWL 2 EL profile specific reasoner.

**Treasoner** [7] is a Java reasoner which implements a standard tableau algorithm for $\mathcal{SHIQ}$.

**TrOWL** [26] is an approximative OWL 2 DL reasoner. In particular, TrOWL utilises a semantic approximation to transform OWL 2 DL ontologies into OWL 2 QL for conjunctive query answering and a syntactic approximation from OWL 2 DL to OWL 2 EL for TBox and ABox reasoning.

**WSClassifier** [25] is a Java reasoner for the $\mathcal{ALCHOI}(D)$ fragment of OWL 2 DL, using a hybrid of the consequence based reasoner ConDOR [23] and hypertableau reasoner HermiT.

[11] `https://www.ebi.ac.uk/chembl/ftc/`
[12] `http://sourceforge.net/projects/jfact/`
[13] `http://www.derivo.de/en/produkte/konclude/`

### 3.2 OWL 2 EL reasoners

**ELepHant** [22] is a highly optimised consequence-based $\mathcal{EL}+$ reasoner written in C, which is aimed at platforms with limited memory and computing capabilities (e.g. embedded systems).

**ELK** [11] is a consequence-based Java reasoner which utilises multiple cores/processors by parallelising multiple threads.

**jcel** [14] uses a completion-based algorithm, which is a generalization of CEL's algorithm. It is a Java reasoner which supports ontologies in $\mathcal{EL}+$.

**SnoRocket** [15] is a Java reasoner developed for the efficient classification of the SNOMED CT ontology. It implements a multi-threaded saturation algorithm similar to that of ELK, thus support concurrent classification.

### 3.3 OWL 2 RL reasoners

**BaseVISor** is a Java-based forward-chaining inference engine which supports OWL 2 RL and XML Schema Datatypes.[14]

## 4 Results – Offline competition

### 4.1 OWL 2 DL results

Nine reasoners entered the OWL 2 DL category, although not all of them competed in the three reasoning tasks. MORe participated with both HermiT and Pellet as the internal DL reasoner. The results for the classification, consistency, and satisfiability tasks are shown in Figure 1.

In the classification task, HermiT performed best in terms of robustness with 147 out of 204 ontologies that were correctly processed within the timeout (at 12.3s per ontology), whereas MORe-Pellet achieved the smallest mean time (2.8s per ontology) for the 141 ontologies it processed correctly.

In the consistency task, Konclude processed the highest number of ontologies correctly (186 out of 204), while also performing fastest on average with 1.7s per ontology; Konclude was also twice as fast as the second faster reasoner (HermiT).

Finally, for the DL satisfiability task, Konclude also processed the highest number of concepts correctly (1,929 out of 2,040) within the given timeout, while coming second after Chainsaw (1.3s) in terms of speed, with a mean time of 1.8s per ontology.

### 4.2 OWL 2 EL results

In addition to the EL-specific reasoners, all OWL 2 DL reasoners also participated in the EL category; the results for all participating reasoners on the three reasoning tasks in the EL profile are shown in Figure 2. In both the classification and consistency categories, ELK performed extremely well both in terms

[14] `http://vistology.com/basevisor/basevisor.html`

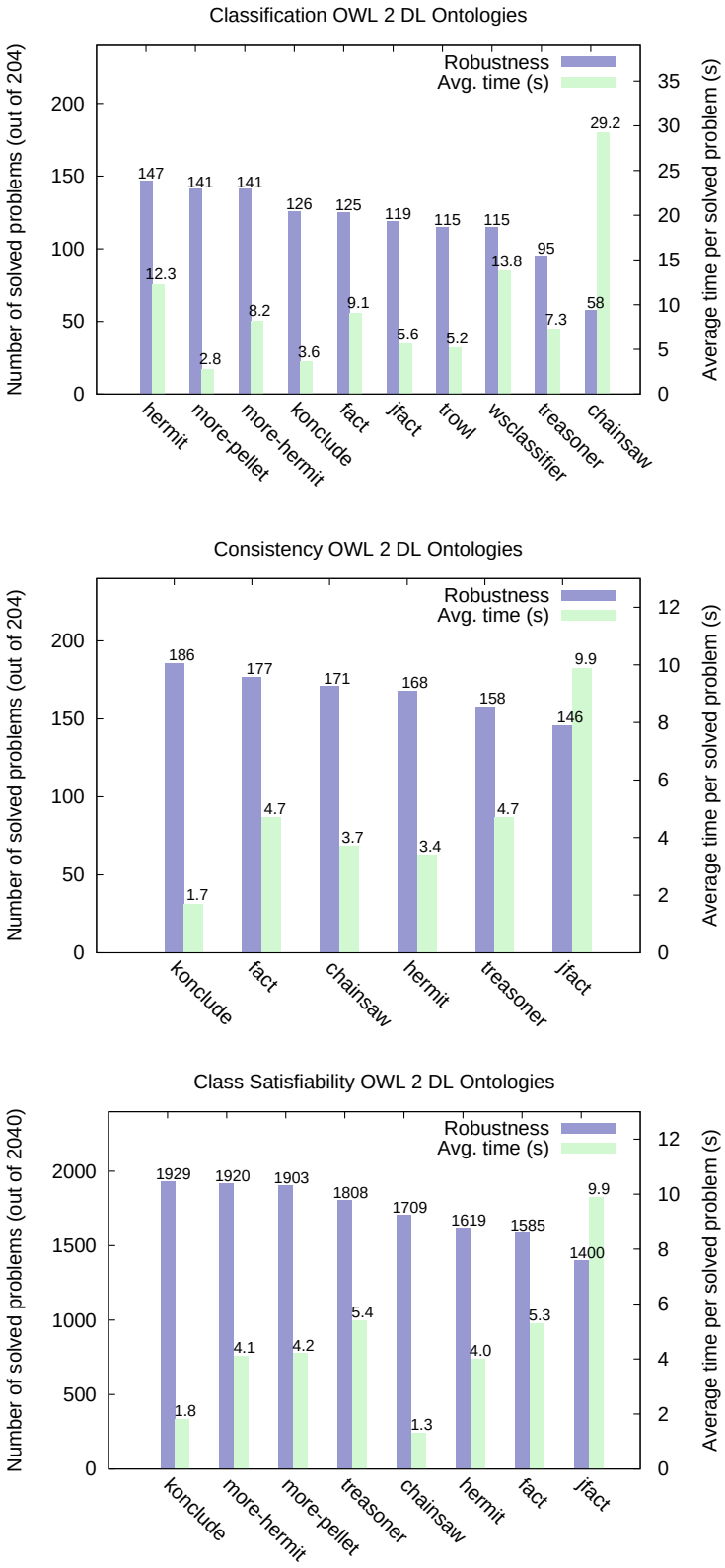



Fig. 1: Results (robustness/average time) for the OWL 2 **DL** category.

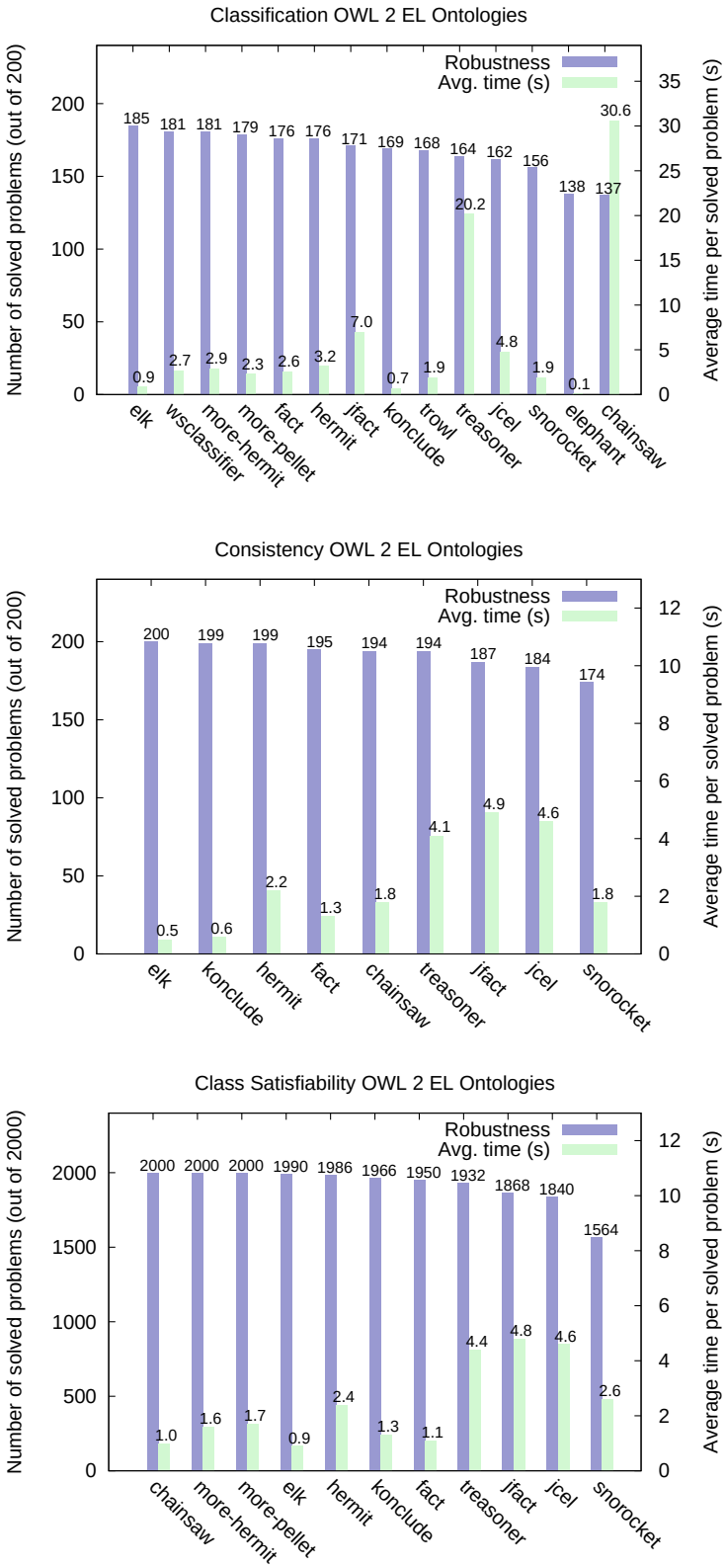



Fig. 2: Results (robustness/average time) for the OWL 2 **EL** category.

of robustness (185 and 200 out of 200 correctly processed ontologies) as well as average speed (0.9s for classification, 0.5s for consistency checking). Perhaps surprisingly, MORe with both HermiT and Pellet performed worse than ELK on robustness, as we expected its combination of ELK with a DL reasoner to handle *more* ontologies than the stand-alone version of ELK which does not support the full OWL 2 EL profile. However, it is possible that the DL reasoners in MORe got in fact 'held up' by those parts of the ontologies that the stand-alone ELK simply ignored, which may have caused a this slightly worse result.

Two of the EL-specific reasoners SnoRocket and ELepHant both performed comparatively fast on those ontologies they did successfully process, but failed to process a large number of ontologies (44 and 62, respectively). The remaining EL reasoner, jcel, was slower than most other reasoners, while also failing to process 38 of the 200 ontologies in the given time.

Finally, for the satisfiability checking task in the EL category, Chainsaw processed the highest number of concepts (all 2,000) correctly while also being second fastest with an average of 1s per concept. MORe with both Pellet and HermiT also completed all 2,000 concepts within the given timeouts, while ELK performed fastest on those 1,990 concepts it did process.

### 4.3 OWL 2 RL results

Only one profile-specific reasoner (BaseVISor) competed in the OWL 2 RL category. Figure 3 shows the results for the three challenges in the RL profile category. Out of the eleven competing reasoners, BaseVISor failed on a significantly large number of ontologies in the classification challenge and processed only 34 of the 197 ontologies correctly. 17 of these failures were due to parsing errors, ten were caused by timeouts that did not return any results, and the remaining failures were due to incorrect results (according to our correctness check). The winning reasoner here was TReasoner, which—despite being the second-slowest reasoner in the group—correctly classified 181 of the 197 ontologies, while most other reasoners correctly processed between 151 and 157 ontologies.

In the consistency checking task, Konclude correctly processed all 197 ontologies, while also performing significantly faster than the other reasoners. Finally, the RL satisfiability category was won by both MORe versions, which correctly processed all 1,970 concepts at an average speed of 0.7s per concept.

## 5 Results – Live competition

The live competition was performed using only the **classification** task in the OWL 2 DL and EL categories, since this is the task supported by most reasoners. The setup was slightly modified from that of the live competition: rather than running the reasoners until they had processed all ontologies in the corpus, we set a strict timeout of one hour for the EL classification task and two hours for the DL classification task, and measured how many ontologies the reasoners would successfully classify in the given time (applying the same five/six minute

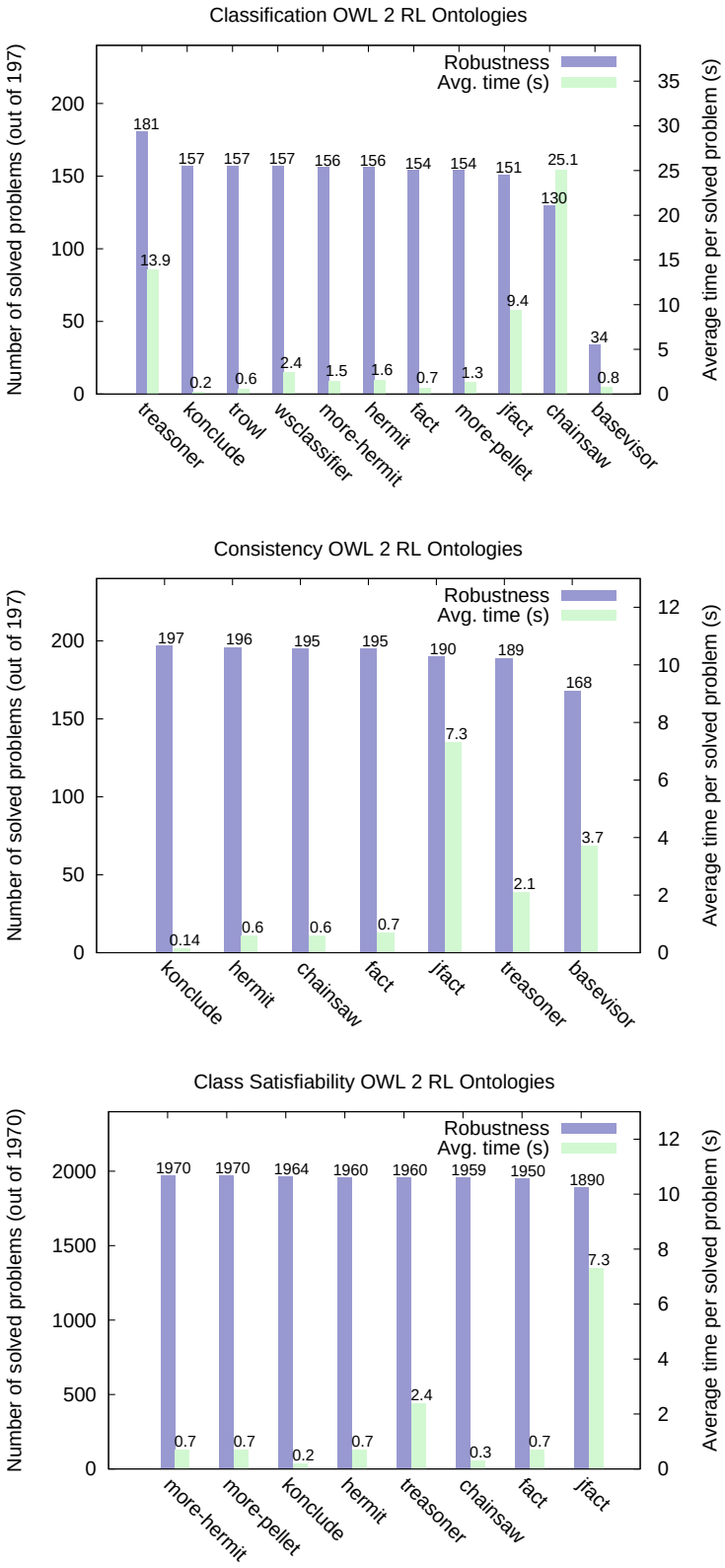



Fig. 3: Results (robustness/average time) for the OWL 2 **RL** category.

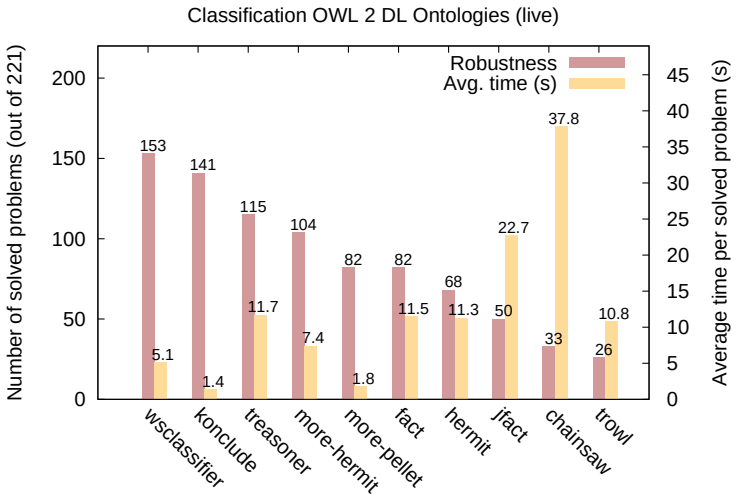



Fig. 4: Results (robustness/average time) for the live competition **DL** category.

timeout per ontology as in the offline competition). As mentioned above, the live competition was performed on Set B, which was entirely different for the DL category, but nearly identical (due to the small number of available EL ontologies) to Set A in the EL category. That is, we expected the results for the DL category to differ from the offline competition, while the results for the EL competition would be largely identical.

The live competition was held on the second day of the Description Logic 2013 workshop, allowing workshop participants to place bets on the reasoner performance, while the current status for each reasoner (number of attempted and number of successfully classified ontologies) was shown and continuously updated on a screen.

### 5.1 OWL 2 DL classification results

Due to the use of the different test corpus (Set B) in the live competition, we expected a slightly different outcome from the offline competition. And indeed, the winning reasoner (in terms of number of correctly processed ontologies) was WSClassifier, which had shown an average performance in the offline competition. WSClassifier processed 153 out of the 221 ontologies in the test corpus, with an average time of 5.1s per ontology, while the reasoner in second place was Konclude, with 141 ontologies and an average time of 1.4s per ontology. Figure 4 shows an overview of the number of processed ontologies and classification times in the DL category.

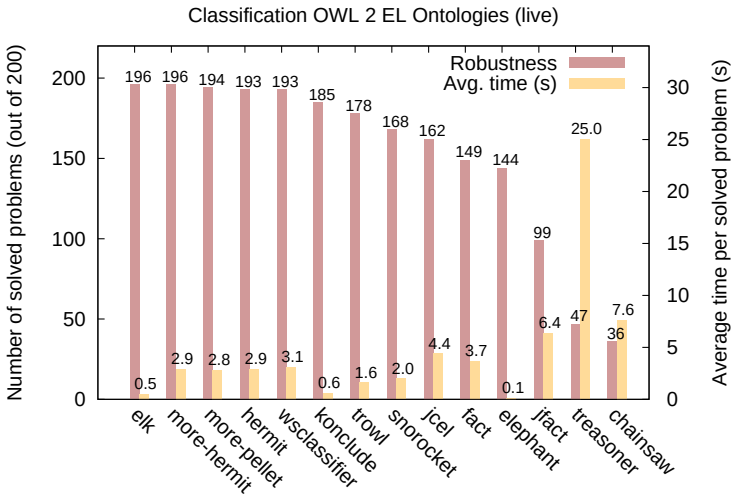



Fig. 5: Results (robustness/average time) for the live competition **EL** category.

### 5.2 OWL 2 EL classification results

The number of processed ontologies and mean classifications for all reasoners participating in the EL live competition can be found in Figure 5. Perhaps unsurprisingly, in the EL classification challenge the results were very similar to the offline challenge, with ELK classifying 196 out of the 200 ontologies at an average speed of 0.5s per ontology. Again, ELepHant was clearly the fastest reasoner with less than 0.1 seconds per ontology, but it also failed on 56 of the 200 ontologies.

## 6 Results – User-submitted ontologies

As with the live competition, the results for the user-submitted ontologies presented here are limited to the classification task, as we consider this to be the most relevant (TBox) reasoning task which is supported by all reasoners in the competition. Note that due to the high number of timeouts and errors on some of these ontologies, the correctness of the successful reasoners could not be determined.

### 6.1 OWL 2 DL classification results

In total, 66 user-submitted ontologies fell into the OWL 2 DL profile, which included the three modified versions of FMA and GALEN discussed above, two versions of the Genomic CDS knowledge base (CDS and CDS-demo), 58 different variants of the Bio KB 101 ontology (of which four were considered to be the most challenging by the ontology developers), and three of the DMOP ontologies.

Except for Chainsaw and TReasoner, all reasoners could successfully classify 54 of the Bio KB ontologies within the five minute timeout, while none of the reasoners processed any of the four 'hard' Bio KB ontologies within the timeout.

The only reasoners that could process both Genomic CDS ontologies were TrOWL and WSClassifier (at an average time of approximately 3 and 8 seconds), while FaCT++ also managed to classify the complete Genomic CDS in 100 seconds. Interestingly, HermiT was the only reasoner to report that the Genomic CDS ontology was inconsistent.

TrOWL and WSClassifier were also the only reasoners to classify the FMA and GALEN modifications within the timeout (perhaps unsurprisingly, since WSClassifier was tuned to work with these ontologies), while both Chainsaw and FaCT++ successfully processed the two GALEN versions, and MORe-Pellet processed the GALEN-FNL version in 14 seconds. For the remaining ontologies, all reasoners except FaCT++ and TrOWL reported datatype errors.

At an average of 0.17 seconds per processed ontology, Konclude was clearly fastest, while most other reasoners also managed average times of less than five seconds for the ontologies they processed correctly.

### 6.2 OWL 2 EL classification results

There were 19 user-submitted OWL 2 EL ontologies, 18 of which were variants of the Bio KB 101 ontology, and the FTC knowledge base. Neither Chainsaw nor ELepHant could process any of the Bio KB ontologies within the five minute timeout, while ELK reported a parsing error. The remaining reasoners, except Snorocket, processed all 18 Bio KB ontologies correctly within the timeout, with Konclude being fastest at 0.1 seconds per ontology.

ELK, Konclude, and WSClassifier all successfully processed the FTC KB, with ELK clearly being fastest at five seconds (it did, however, ignore three ObjectPropertyRange axioms which are outside the OWL 2 EL fragment supported by ELK), and the other two reasoners taking between 20 and 30 seconds. The remaining reasoners either timed out or reported an error for this ontology.

### 6.3 OWL 2 RL classification results

All 18 ontologies in the OWL 2 RL profile were variants of Bio KB 101. BaseVISor failed to parse the input on all files, while Chainsaw timed out on 15 of the ontologies. The remaining reasoners all classified the ontologies correctly within the five minute timeout, with Konclude processing the ontologies at an average of 0.15 seconds.

## 7 Summary

In this report we presented an overview of the methodology and results of the ORE reasoner competition for the different categories, OWL 2 profiles, and test corpora. There were a total of 14 OWL reasoners submitted for participation

in the competition, which made it all the more successful. Out of these, 5 were profile specific reasoners (4 OWL 2 EL and 1 OWL 2 RL) while 9 were OWL 2 DL reasoners or supported a large fragment of $\mathcal{SROIQ}(D)$ not included within the OWL 2 EL, RL or QL profiles. The reasoners were evaluated with a random sample of ontologies from known repositories, on three standard reasoning tasks: classification, consistency checking, and concept satisfiability. In the competition we gave preference to how robust the systems were, that is, the number of tests correctly passed within the given timeout, rather than reasoning times alone. The top 3 reasoners for each category are listed below:

**OWL 2 DL Ontologies**
- *Classification:* (1) HermiT (2) MORe-HermiT/MORe-Pellet (3) Konclude
- *Consistency:* (1) Konclude (2) FaCT++ (3) Chainsaw
- *Satisfiability:* (1) Konclude (2) MORe-Pellet/MORe-HermiT (3) TReasoner
- *Classification (live):* (1) WSClassifier (2) Konclude (3) TReasoner

**OWL 2 EL Ontologies**
- *Classification:* (1) ELK (2) WSClassifier (3) MORe-HermiT
- *Consistency:* (1) ELK (2) HermiT (3) Konclude
- *Satisfiability:* (1) Chainsaw (2) MORe-Pellet (3) TrOWL
- *Classification (live):* (1) ELK (2) MORe-HermiT/MORe-Pellet (3) HermiT

**OWL 2 DL Ontologies**
- *Classification:* (1) TReasoner (2) Konclude (3) TrOWL
- *Consistency:* (1) Konclude (2) HermiT (3) Chainsaw
- *Satisfiability:* (1) MORe-HermiT/MORe-Pellet (2) Konclude (3) HermiT

Additionally, the MORe and ELepHant reasoners were also given a special recognition prize. MORe was selected as the *best newcomer reasoner* since it consistently performed well in terms of time and robustness. The ELepHant reasoner, although it struggled with a high number of errors, was incredibly fast for the ontologies that it was able to classify correctly, and so was awarded a *special mention*. We look forward to seeing the evolution of these novel reasoners.

Regarding the user-submitted ontologies, it is interesting to see that most reasoners could either process *all* or *none* of the Bio KB ontologies. When they did process them, the classification times were fairly uniform. The results for the GALEN and FMA modifications, which were specifically developed for testing with WSClassifier, confirmed the robustness of the reasoner on these ontologies; however, the other two reasoners which *could* process the GALEN modifications (Chainsaw and FaCT++) were significantly faster within the timeout. Our experiments on the Genomic CDS ontologies confirmed the reports of the ontology developer [20] who found that out of the now 'mainstream' reasoners, only TrOWL could process the ontology in reasonable time, while HermiT (falsely) reported an inconsistency error. While we have seen that WSClassifier could also process the ontology, the correctness of the classification result is unclear, since WSClassifier does not support qualified number restrictions which are heavily used in Genomic CDS.

Finally, we have only carried out our benchmark with a fixed timeout of five minutes in the main offline and live competitions, which may have been too short for some of these ontologies, e.g. the four 'challenging' Bio KB ontologies could not be processed by any of the reasoners Thus, we are planning to re-run these tests with longer timeouts in the near future.

## Acknowledgements

We thank Konstantin Korovin (supported by the Royal Society grant RG080491) at the University of Manchester who kindly provided us with the PC cluster for the competition. We also thank the developers of the submitted reasoners and ontologies for their invaluable effort. We also gratefully acknowledge the support of the ORE workshop sponsor: B2i Healthcare (`https://www.b2international.com/`). Ernesto Jimenez-Ruiz was supported by the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, 'Optique', and the EPSRC projects Score!, ExODA and MaSI$^3$.

## References

1. Armas Romero, A., Cuenca Grau, B., Horrocks, I., Jiménez-Ruiz, E.: MORe: a Modular OWL Reasoner for Ontology Classification. In: OWL Reasoning Evaluation Workshop (ORE) (2013)
2. Chaudhri, V.K., Wessel, M.A., Heymans, S.: KB_Bio_101: A Challenge for OWL Reasoners. In: 2nd OWL Reasoner Evaluation Workshop (ORE) (2013)
3. Cuenca Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P.F., Sattler, U.: OWL 2: The next step for OWL. J. Web Sem. 6(4), 309–322 (2008)
4. Glimm, B., Horrocks, I., Motik, B., Shearer, R., Stoilos, G.: A novel approach to ontology classification. J. of Web Semantics 10(1) (2011)
5. Golbeck, J., Fragoso, G., Hartel, F.W., Hendler, J.A., Oberthaler, J., Parsia, B.: The National Cancer Institute's Thésaurus and Ontology. J. Web Sem. 1(1), 75–80 (2003)
6. Gonçalves, R.S., Parsia, B., Sattler, U.: Categorising logical differences between OWL ontologies. In: ACM Conference on Information and Knowledge Management (CIKM) (2011)
7. Grigoryev, A., Ivashko, A.: TReasoner: System Description. In: 2nd OWL Reasoner Evaluation Workshop (ORE) (2013)
8. Horridge, M., Bechhofer, S.: The OWL API: A java api for owl ontologies. Semantic Web 2(1), 11–21 (2011)
9. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: the making of a web ontology language. J. Web Sem. 1(1), 7–26 (2003)
10. Horrocks, I., Yatskevich, M., Jiménez-Ruiz, E. (eds.): Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE), CEUR Workshop Proceedings, vol. 858. CEUR-WS.org (2012)
11. Kazakov, Y., Krötzsch, M., Simancik, F.: Concurrent classification of EL ontologies. In: International Semantic Web Conference (ISWC). pp. 305–320 (2011)

12. Keet, C.M., Lawrynowicz, A., d'Amato, C., Hilario, M.: Modeling issues and choices in the Data Mining OPtimization Ontology. In: OWL: Experiences and Directions (2013)
13. Matentzoglu, N., Bail, S., Parsia, B.: A corpus of owl dl ontologies. In: 26th International Workshop on Description Logics (DL). pp. 829–841 (2013)
14. Mendez, J.: jcel: A Modular Rule-based Reasoner. In: 1st OWL Reasoner Evaluation Workshop (ORE) (2012)
15. Metke Jimenez, A., John Lawley, M.: Snorocket 2.0: Concrete Domains and Concurrent Classification. In: 2nd OWL Reasoner Evaluation Workshop (ORE) (2013)
16. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 web ontology language profiles. W3C Recommendation (2009)
17. Noy, N.F., Shah, N.H., Whetzel, P.L., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D.L., Storey, M.A.D., Chute, C.G., Musen, M.A.: Bioportal: ontologies and integrated data resources at the click of a mouse. Nucleic Acids Research 37(Web-Server-Issue), 170–173 (2009)
18. Rector, A.L., Rogers, J.E., Zanstra, P.E., Van Der Haring, E., Openg: OpenGALEN: open source medical terminology and tools. AMIA Annu Symp Proc (2003)
19. Rosse, C., Mejino Jr., J.: A reference ontology for biomedical informatics: the Foundational Model of Anatomy. J. Biomed. Informatics 36(6), 478–500 (2003)
20. Samwald, M.: Genomic CDS: an example of a complex ontology for pharmacogenetics and clinical decision support. In: 2nd OWL Reasoner Evaluation Workshop (ORE) (2013)
21. Schulz, S., Cornet, R., Spackman, K.A.: Consolidating SNOMED CT's ontological commitment. Applied Ontology 6(1), 1–11 (2011)
22. Sertkaya, B.: The ELepHant Reasoner System Description. In: 2nd OWL Reasoner Evaluation Workshop (ORE) (2013)
23. Simancik, F., Kazakov, Y., Horrocks, I.: Consequence-Based Reasoning beyond Horn Ontologies. In: 22nd International Joint Conference on Artificial Intelligence (IJCAI). pp. 1093–1098 (2011)
24. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL DL reasoner. J. of Web Semantics 5(2), 51–53 (2007)
25. Song, W., Spencer, B., Du, W.: A Transformation Approach for Classifying ALCHI(D) Ontologies with a Consequence-based ALCH Reasoner. In: 2nd OWL Reasoner Evaluation Workshop (ORE) (2013)
26. Thomas, E., Pan, J.Z., Ren, Y.: TrOWL: Tractable OWL 2 Reasoning Infrastructure. In: 7th Extended Semantic Web Conference (ESWC). pp. 431–435 (2010)
27. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: Third International Joint Conference on Automated Reasoning (IJCAR). pp. 292–297 (2006)
28. Tsarkov, D., Palmisano, I.: Chainsaw: a Metareasoner for Large Ontologies. In: 1st OWL Reasoner Evaluation Workshop (ORE) (2012)
29. Wrigley, S.N., Garcia-Castro, R., Nixon, L.J.B.: Semantic Evaluation At Large Scale (SEALS). In: The 21st World Wide Web Conf., WWW (Companion Volume). pp. 299–302 (2012), `http://www.seals-project.eu`

# YARR!: Yet Another Rewriting Reasoner

Joerg Schoenfisch and Jens Ortmann

Softplant GmbH, Agnes-Pockels-Bogen 1, 80992 Munich, Germany
{joerg.schoenfisch, jens.ortmann}@softplant.de

**Abstract.** In this paper we present our implementation of an OWL 2 QL reasoner using query rewriting to answer SPARQL queries in a relational database. To answer queries in the database through rewriting, ontologies are limited to the OWL 2 QL profile. The rewriting algorithm internally produces a non-recursive Datalog program from the given SPARQL query. This program is then translated to SQL and executed by the database.

**Keywords:** reasoning, query rewriting, Presto, OWL 2 QL

## 1 Introduction

In this paper we present YARR, our implementation of a reasoner using query rewriting which is part of the Living Semantic Platform [10]. The platform consists of a relational database as storage for ontologies, an importer for OWL 2 QL[1] and RDF[2] data, a web-based GUI to edit the knowledge base, and our reasoner. YARR exposes a SPARQL[3] endpoint and answers queries through the database.

To enable processing of semantic queries by a relational database some restrictions apply and several steps are performed. The ontologies stored in the database are limited to the expressiveness of the OWL 2 QL profile which is specifically designed to facilitate query answering through query rewriting and processing in a relational database.

Query rewriting is employed to retrieve complete and sound answers from the database. The rewriting which incorporates knowledge from the TBox into the query to also enable extraction of implicit knowledge from the ABox takes place in three steps. First, the SPARQL query is parsed and translated to Datalog. Second, this query is then rewritten into a non-recursive Datalog program to include the knowledge from the TBox. Third, the program is translated to SQL and executed in the database system.

YARR fully supports ontologies formulated in OWL 2 QL and most of the SPARQL 1.1 SELECT syntax. Some built-ins and complex mathematical expressions in FILTER and ORDER BY clauses still have to be implemented. Furthermore, there is no reasoning on datatypes.

[1] http://www.w3.org/TR/owl2-profiles/
[2] http://www.w3.org/TR/rdf-primer/
[3] http://www.w3.org/TR/sparql11-overview/

## 2 Related Work

The OWL 2 QL profile is based on the description logic DL-Lite [1]. Calvanese et al. [1] proposed PerfectRef as one of the first rewriting algorithms. A second, popular rewriting algorithm is REQUIEM [6], of which an optimized version called Blackout [7] is used in Stardog[4]. Pérez-Urbina et al. also provide an overview over other rewriting algorithms, for instance Nyaya [5], Clipper [4], Rapid [3], Presto [9] and Prexto [8].

YARR is based on the Presto algorithm, which produces a non-recursive Datalog program.

## 3 Query Rewriting

Query rewriting is the process of transforming a query over a knowledge base in such a way that it produces sound and complete answers without the need for any reasoning in the knowledge base itself. In the case of OWL 2 QL and similar description logics this means that the query is expanded with knowledge from the TBox so that implicit knowledge can be extracted from the ABox without the need for any information about the ABox itself. This is achieved by limiting the expressiveness of the description logic. A thorough overview is given in the description of the DL-Lite family by Calvanese et al. [2].

An opposing approach to this is materialization. Here, all knowledge that can be inferred from known facts is explicitly stored when the data is loaded. Thus, no reasoning is needed later on, as long as the data is not modified. If the data is modified the materialization has to be computed anew which can be quite expensive, e.g. when removing a subclass-of relation, the class assertion with the super class has to be removed from every single instance of the subclass.

There are three reasons why we chose query rewriting in our implementation. First, rewriting allows the query to be processed by regular RDBMS, which are readily available in enterprise environments and require well-known effort concerning administration, maintenance, backup, etc.

The omission of a reasoning step in the ABox during query answering is the second point in favor of query rewriting. This way, changes in the ABox, which might happen quite frequently due to the collaborative setting in which we want to deploy YARR, can directly be reflected in the answers to a query.

Third, in an ontology-based data access scenario it is often not feasible or possible to modify or expand the ABox due to its size or missing write permissions.

Presto rewrites a Datalog query in three major steps. The first step splits each Datalog rule into its independent join components. This produces more, smaller Datalog rules, resulting in smaller rewritings in the end. This is beneficial as every Datalog rule is later translated to SQL and must be processed by the RDBMS. Thus, smaller rewritings positively affect the speed of query answering.

[4] http://stardog.com/

The second step removes redundant rules. These rules would produce answers other rules already did, so there is no need to rewrite, translate and execute them. An example would be two rules, one asking for all individuals, and the other asking for individuals of a specific type. The individuals of a specific type are already included in all individuals, and thus the specific rule is superfluous.

The last step defines TBox views for each concept and role, e.g. a view for a concept would be the union of all individuals directly of this type or of the type of one of its subclasses.

As an example, the simple SPARQL query that selects all triples is translated to a single Datalog rule with one atom, and results in an SQL query consisting of a union over 3 sub-selects with 3 joins each. If the rewriting step is left out, the size of the SQL grows linearly with the number of triple patterns in the SPARQL query.

## 4 Architecture

The two major parts of YARR's architecture are on the one hand the steps and libraries involved to transform and rewrite a query from SPARQL to the corresponding SQL, and the database architecture on the other hand.

Other parts include the import and export functionality for which we utilize the OWL API[5] to parse and write OWL documents, and consistency checking which is currently transferred to Jena[6]. Jena is the only reasoner freely available for commercial use. However, it does not officially support OWL 2 as of yet and its performance is behind that of other reasoners like Pellet or HermiT.

### 4.1 Rewriting Architecture

The rewriting takes place in several steps, along which our architecture is split. We try to use existing libraries as much as possible for the steps not directly related to the rewriting.

The first step is the parsing of the SPARQL query and its translation to our internal Datalog model. We are using the parser implementation of Sesame[7]. Their parser produces an abstract model of the query which we translate to Datalog rules. The rules incorporate some additional information, e.g. aggregate functions or mathematical expressions for FILTER clauses, which are not directly represented in Datalog.

These rules are then passed on to the Presto algorithm, which is a straightforward implementation of the rewriting procedure described by Rosati et al. [9]. The resulting Datalog program includes all the knowledge needed to produce sound and complete answers to the query.

The process is finished after the final translation from Datalog to SQL. To gain some syntax and type checking, and to be as agnostic to the underlying

[5] http://owlapi.sourceforge.net/
[6] http://jena.apache.org/
[7] http://www.openrdf.org/

database systems as possible, we us jOOQ[8], a domain specific language for SQL in Java. Each rule of the Datalog program is translated to an SQL query. The queries are then aggregated into a single query as subqueries. This SQL query can then be passed to the database to retrieve the answers.

### 4.2 Database Design

Our database design is loosely based on the way in which facts are stated in OWL. Overall, it consists of 22 tables. There are individual tables for each type of assertions, for subclass and subproperty axioms, disjointness and equivalency axioms, range and domain axioms, literals, and one table for all types of entities (classes, properties, and individuals).

Another quite obvious choice would have been a design which follows the structure of RDF. There would have been one (or to optimize for performance several partitioned) table(s) storing only triples. In fact, this is the layout several triple stores, like Sesame, Jena, or OWLIM, chose. However, as our reasoner is part of a platform that focuses on OWL semantics and also offers an editor, historization, and versioning, we opted for the initially more complex layout to ease these other tasks.

## 5 Expected Performance

We conducted a benchmark to compare YARR to state-of-the-art triple stores. As benchmark we chose SP2Bench[9] and various sizes of the data. The other stores we used for comparison are OWLIM-Lite 5.2[10] and Stardog 1.0.7 Community Edition. OWLIM uses a materialization approach, which means all inferences are computed and stored before a query is processed. Stardog uses a query rewriting algorithm similar to our approach.

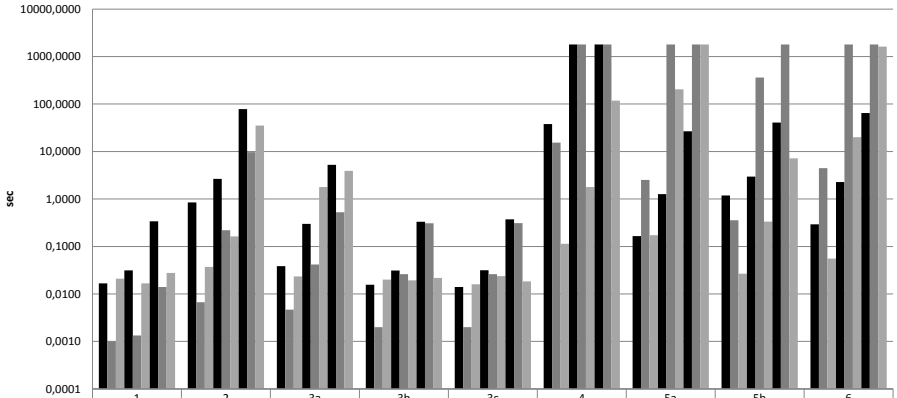
Figures 1 and 2 show a comparison to OWLIM and Stardog for different sizes of SP2Bench (10k, 250k, and 5M triples). This benchmark was run on a desktop machine with a Intel Core 2 Duo at 3 GHz and 8GB RAM. As database backend for YARR we used Oracle 11g Express Edition[11]. Note that to make for a fair comparison the times include the time needed to send the results to the client.

Most of the time that YARR needs for query answering is spent by the RDBMS for query planning, processing and result serialization. The overhead of the rewriting step, and the translation from SPARQL to Datalog and SQL is negligible for larger datasets and complex queries (well below 50ms).

The diagram clearly shows that our implementation behaves similarly in terms of scalability as OWLIM and Stardog. We have some disadvantages for very fast queries due to the rewriting step and the translation to SQL, and the round-trip to the database (Queries 1, 12c). Although the database is hosted on

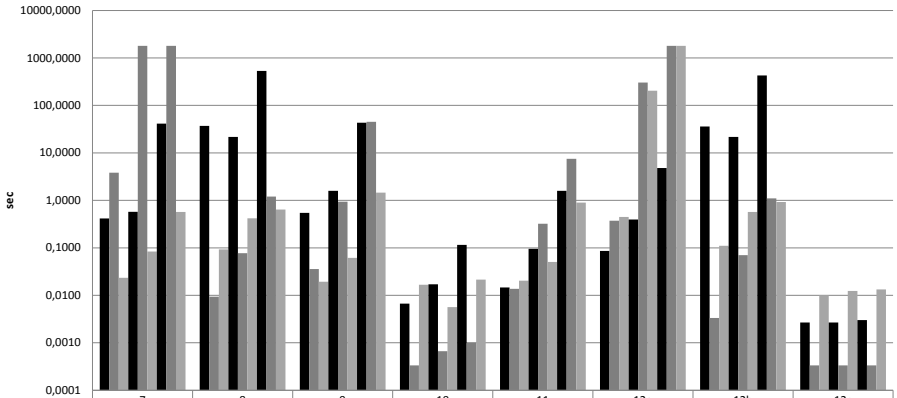
[8] http://www.jooq.org/
[9] http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B
[10] http://www.ontotext.com/owlim
[11] http://www.oracle.com/technetwork/products/express-edition/overview/

V




| | 1 | 2 | 3a | 3b | 3c | 4 | 5a | 5b | 6 |
|---|---|---|---|---|---|---|---|---|---|
| 10k LSP | 0,0167 | 0,8453 | 0,0383 | 0,0157 | 0,0140 | 37,8023 | 0,1647 | 1,1867 | 0,2933 |
| 10k OWLIM | 0,0010 | 0,0067 | 0,0047 | 0,0020 | 0,0020 | 15,3383 | 2,5103 | 0,3570 | 4,4750 |
| 10k Stardog | 0,0210 | 0,0370 | 0,0233 | 0,0200 | 0,0160 | 0,1137 | 0,1723 | 0,0267 | 0,0557 |
| 250K LSP | 0,0313 | 2,6503 | 0,3000 | 0,0310 | 0,0317 | 1800,0000 | 1,2663 | 2,9617 | 2,2693 |
| 250k OWLIM | 0,0013 | 0,2200 | 0,0417 | 0,0260 | 0,0260 | 1800,0000 | 1800,0000 | 360,8620 | 1800,0000 |
| 250k Stardog | 0,0167 | 0,1630 | 1,7893 | 0,0193 | 0,0237 | 1,7873 | 203,2773 | 0,3327 | 20,0597 |
| 5M LSP | 0,3380 | 78,3443 | 5,2533 | 0,3317 | 0,3733 | 1800,0000 | 26,6530 | 40,6263 | 64,5100 |
| 5M OWLIM | 0,0140 | 9,6743 | 0,5257 | 0,3077 | 0,3097 | 1800,0000 | 1800,0000 | 1800,0000 | 1800,0000 |
| 5m Stardog | 0,0277 | 35,1750 | 3,9043 | 0,0217 | 0,0183 | 117,5650 | 1800,0000 | 7,2007 | 1617,1033 |

**Fig. 1.** Benchmark SP2Bench Queries 1 - 6




| | 7 | 8 | 9 | 10 | 11 | 12a | 12b | 12c |
|---|---|---|---|---|---|---|---|---|
| 10k LSP | 0,4143 | 36,9597 | 0,5450 | 0,0067 | 0,0147 | 0,0863 | 35,9443 | 0,0027 |
| 10k OWLIM | 3,8273 | 0,0093 | 0,0357 | 0,0003 | 0,0137 | 0,3727 | 0,0033 | 0,0003 |
| 10k Stardog | 0,0233 | 0,0927 | 0,0193 | 0,0167 | 0,0203 | 0,4457 | 0,1103 | 0,0100 |
| 250K LSP | 0,5723 | 21,6380 | 1,5933 | 0,0170 | 0,0957 | 0,3947 | 21,7037 | 0,0027 |
| 250k OWLIM | 1800,0000 | 0,0773 | 0,9390 | 0,0007 | 0,3207 | 302,2110 | 0,0703 | 0,0003 |
| 250k Stardog | 0,0833 | 0,4183 | 0,0613 | 0,0057 | 0,0503 | 204,4030 | 0,5690 | 0,0123 |
| 5M LSP | 41,4237 | 533,6507 | 43,1820 | 0,1153 | 1,5877 | 4,7917 | 427,6407 | 0,0030 |
| 5M OWLIM | 1800,0000 | 1,1983 | 45,1623 | 0,0010 | 7,5273 | 1800,0000 | 1,1003 | 0,0003 |
| 5m Stardog | 0,5690 | 0,6397 | 1,4493 | 0,0213 | 0,8983 | 1800,0000 | 0,9240 | 0,0133 |

**Fig. 2.** Benchmark SP2Bench Queries 7 - 12

the same machine it is running in a different process outside the JVM and has to be accessed over the network which is considerably slower than to access a database in the same process.

However, more importantly, we are on par with OWLIM's performance for some queries on the larger datasets (Queries 3b, 3c, 9) and sometimes considerably faster (Queries 5a, 5b, 6, 7, 11, 12a). For six of these OWLIM is not able to complete the request within the 30 minutes timeout that SP2Bench imposes on reasoners, whereas this only happens for one query in YARR (Query 4).

The comparison to Stardog shows similar results. The overall impression is that Stardog performs slightly better than the other two. It is especially fast on query 4, where it is the only reasoner without a timeout, and queries 7 and 9, but also seems to have some penalty if the result of the query can be computed very fast (Queries 1, 10, 12c).

Further investigation is needed to determine the source for YARRs slow performance on the remaining queries (Queries 2, 3a, 8, 10, 12b). Possible reasons are bad query plans, missing indexes, or queries inherently hard for relational database systems.

We also did benchmarks for different database backends, i.e. Oracle, Postgres[12], HSQLDB[13] and H2[14]. Oracle and Postgres showed comparable performance for all benchmarks. The tests on HSQLDB and H2 were only done in-memory and for small datasets, so the performance values we have for those are not conclusive, yet. However, it was quite surprising that for some queries, H2 was not able to find a query plan for the SQL and did not produce any results.

## 6 Future Work and Conclusion

We presented our implementation of an OWL 2 QL reasoner using the Presto algorithm to answer SPARQL queries in a relational database. The first benchmarks we conducted to compare it to state-of-the-art triple stores are promising. We still have planned further optimization but we already expect our reasoner to compete well with other implementations.

Our future work is focused on a more thorough support of SPARQL 1.1, mainly for SELECT, ASK and CONSTRUCT queries. Built-ins defined by the recommendation will also be implemented as the need for them arises.

Furthermore, we have planned several optimizations, e.g. caching of TBox statistics to reduce the size of the SQL queries or to improve the join order.

In a wider perspective there is ongoing work to implement adapters for ontology-based data access (OBDA) to support arbitrary database schemes. One possibility herein is the use of R2RML[15], which provides a mapping language from relational schemas to RDF.

[12] http://www.postgresql.org/
[13] http://hsqldb.org/
[14] http://www.h2database.com/html/main.html
[15] http://www.w3.org/TR/r2rml/

## References


1. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Dl-lite: Tractable description logics for ontologies. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 602. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.
2. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *Journal of Automated reasoning*, 39(3):385–429, 2007.
3. Alexandros Chortaras, Despoina Trivela, and Giorgos Stamou. Optimized query rewriting for owl 2 ql. In *Automated Deduction–CADE-23*, pages 192–206. Springer, 2011.
4. Thomas Eiter, Magdalena Ortiz, M Simkus, Trung-Kien Tran, and Guohui Xiao. Towards practical query answering for horn-shiq. *Description Logics*, 846, 2012.
5. Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Ontological queries: Rewriting and optimization. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 2–13. IEEE, 2011.
6. Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, 8(2):186–209, 2010.
7. Héctor Pérez-Urbina, Edgar Rodrıguez-Dıaz, Michael Grove, George Konstantinidis, and Evren Sirin. Evaluation of query rewriting approaches for owl 2. In *Joint Workshop on Scalable and High-Performance Semantic Web Systems (SSWS+ HPCSW 2012)*, page 32, 2012.
8. Riccardo Rosati. Query rewriting under extensional constraints in dl-lite. In *Proceedings of the international workshop on description logics, DL-2012*, 2012.
9. Riccardo Rosati and Alessandro Almatelli. Improving query answering over dl-lite ontologies. *Proc. of KR*, 2010, 2010.
10. Joerg Schoenfisch, Florian Lautenbacher, Julian Lambertz, and Willy Chen. *Living Semantic Platform.* 10th International Semantic Web Conference - Industry Track, 25 October 2011. Available at http://www.softplant.de/innovation/konferenzteilnahmen.html.

# TReasoner: System Description

Andrey V. Grigorev and Alexander G. Ivashko

Tyumen State University,
Semakova. 18, 625003 Tyumen, Russian Federation
{ivashco,107th}@mail.ru

**Abstract.** TReasoner is a reasoning system supporting the $SHOIQ(D)$ logic expressiveness, which forms the basis of the OWL DL language. The TReasoner was developed for using in the enterprise architecture verification expert systems, but the OWL API package allows to use the system for performing ontology operations. The reasoner implements a tableau algorithm and optimization techniques, some of them were developed and were used for the first time. This description also contains an assessment of the developed system efficiency.

**Keywords:** Description Logic, OWL, Tableau Algorithm, Reasoner, Classification

## 1 Introduction

Ontologies are a powerful tool of knowledge representation, which became very popular for using by expert systems [8]. First of all because of the fact that they are based on the description logic formalism, which has a formally defined semantics allowing to develop tableau algorithm for a logic inference. OWL [13] is the basic ontology representation language recommended by the W3C consortium. Nowadays the OWL 2 standard is valid. The OWL DL language uses the $SHOIN$ [1] description logic with support of data values.

To date many OWL reasoning systems such as FaCT++ [15], HermiT [7] (for OWL DL), jcel [11], ELK [10] (for OWL 2 EL) were developed, they implement different algorithms for a logic inference.

The article introduces a new OWL Reasoner. The TReasoner is $SHOIQ(D)$ reasoner implementing tableau algorithm with some novel optimization techniques. TReasoner is free distributed by GNU General Public License v2. Source code of the TReasoner, compiled class library and wrapper for system usage are available at `http://treasoner.googlecode.com`.

This system description has the following structure. Section 2 provides a supported language and an implemented algorithm. Section 3 contains information about architecture, implementation and optimization techniques that are used by the TReasoner. Results of the system testing are described in section 4. Section 5 concludes this work.

## 2 Supported Language Subset and Implemented Reasoning Algorithm

The TReasoner allows to perform a concept satisfiability checking, a consistency checking and a classification on OWL ontologies that use the $SHOIQ(D)$ description logic. It means that the system works correctly with concepts described by the disjunction, the conjunction, the existential quantifier and the universal quantifier. Besides, the $SHOIQ$ allows roles to be transitive and inverse to other roles. There may be concepts consisting of one individual (nominal), at the same time the $SHOIQ$ is extended by number restrictions ($n \leq R.C$ or $n \geq R.C$). D letter at $SHOIQ(D)$ logics allows to describe knowledge with support of datatypes (strings, numbers, dates and etc.).

The TReasoner implements the tableau algorithm [6]. The concept satisfiability checking is carried out through graph-model existence checking.

The tableau algorithm for $SHOIQ$ has NExpTime complexity, but the developed system implements different new and old optimization techniques, which allow to significantly reduce worktime in practice.

## 3 Architecture and Implementation

The TReasoner was developed using the Java language, because of the cross-platform portability. The system consists of 6 packages. The RuleGraph package implements data structures for the inner representation of concepts. Also this package implements algorithms for the concepts simplification and the axiom simplification. TBox, ABox and RBox axioms are contained in KnowledgeBase package classes. The OWL API package is used for loading the OWL ontologies and transforming them to inner system representation. Main package is Checker. It contains classes that implement tableau algorithm and optimization techniques for it. Checker package classes use Interpretation package classes, which implement data structures for the interpretation building. All packages use classes of the Help package, which implements different supporting algorithms and data structures such as binary heap, hash-table, etc. The UML package diagram is presented on the Fig. 1.

The TReasoner implements optimization techniques, which can be divided into 3 groups:

1. Preprocessing optimizations;
2. Tableau algorithm optimizations;
3. Classification optimizations.

The system uses both time-tested and newly developed optimizations.

### 3.1 Optimization Techniques

*Preprocessing optimizations* are used by the ontologies transformation to inner structures, which are understandable by the TReasoner. Also they are used for

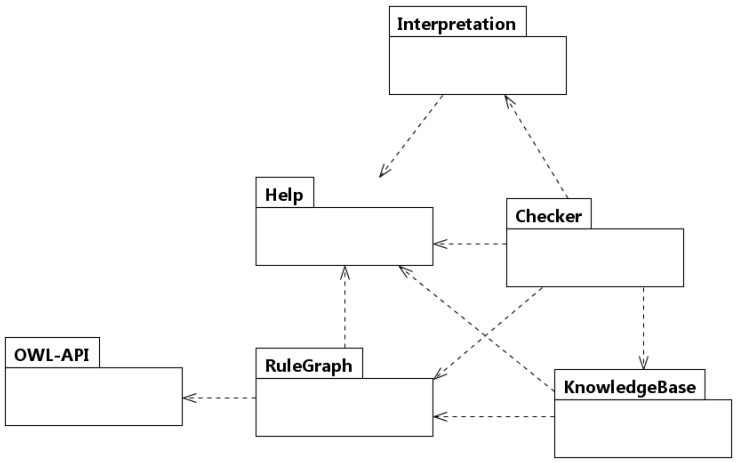



**Fig. 1.** The structure of packages of the TReasoner system

the transformation of GCIs and equivalence axioms. For the concept representation the system uses direct acyclic graph (DAG), each vertex of the graph corresponds to some operation or quantifier, and neighbours of this vertex are operands of the operation, in addition each vertex in the DAG has a number that uniquely identifies it. To reduce memory usage, same concepts are represented by only one subgraph. For each vertex (in order of height increasing), a hash-function value is calculated, this function consider unique numbers of all neighbours, operation type of the vertex, unique number of a role and number restriction (for existential and universal quantifiers, and for number restriction operations). If this function value doesn't exists in hash-table, the vertex with its hash-function value will be added to hash-table. If function value is found then all edges which enter to this vertex will change its direction to vertex with corresponding value of hash-function that contained in hash-table.

To reduce memory usage, removal of brackets technique was developed. The algorithm is performed in two runs. In first run, for each vertex $v$ (in order of height increasing) that represent a concept, set of concepts $H(v)$ is calculated. Concepts of this set defined as follows:

1. If current graph vertex $v$ represents atomic cocnept $C$, then $H(v)$ set contains two elements: $C$ and $\top$;
2. If current graph vertex $v$ represents $\sqcap$-cocnept then $H(v)$ set contains elements of $H(u_1) \sqcup H(u_2) \sqcup ... \sqcup H(u_k)$ for all $u_i$ which are neighbours of $v$;
3. If current graph vertex $v$ represents $\sqcup$-cocnept then $H(v)$ set contains elements of $H(u_1) \sqcap H(u_2) \sqcap ... \sqcap H(u_k)$ for all $u_i$ which are neighbours of $v$;

In second run, vertexes are considered in order of height decreasing, each vertex is transformed to $\sqcap$-vertex with neighbours of all concepts from $H(v)$ and itself

vertex, so each concept of $H(v)$ will be deleted from $H(u_i)$ for all $u_i$ which are neighbours of $v$. For example, concept $(((C \sqcap B) \sqcup (D \sqcap B)) \sqcap A) \sqcup (B \sqcap ((C \sqcap A) \sqcup (E \sqcap C)))$ will be transformed to $B \sqcap ((C \sqcap (A \sqcup E)) \sqcup (A \sqcap (C \sqcup D)))$

After loading and preprocessing of concepts, a processing of axioms will be performed. Absorption technique [5] is used for this task.

*Tableau algorithm optimizations* that are implemented in the TReasoner, include such optimizations as backJumping [14], caching [3, 5] and global caching [12]. The system implements novel optimization techniques. The main of such techniques is the SS-branching [9], which determine disjointness of concepts without using of tableau algorithm. It is applicable not in every cases, but in wide range of concepts. The SS-branching procedure determine disjointness of two concepts by analyzing of structures of DAGs that represent its. For example, if concepts $C$ and $D$ are conjunctions of other concepts ($C \equiv C_1 \sqcap C_2 \sqcap ... \sqcap C_n, D \equiv D_1 \sqcap D_2 \sqcap ... \sqcap D_n$) and some of the concepts $C_i$ and $D_j$ are disjoint, then $C$ and $D$ are disjoint. Conditions of disjointness for cases when $C$ and $D$ are disjunctions, disjunction and conjunction, existential and unversal quantifiers were identified. However, SS-branching can not to determine disjointness of concepts. To cover wider class of concepts Bron-Kerbosch algorithm was used. For disjointness checking of the concepts like $E_1 \sqcap E_2 \sqcap ... \sqcap E_n$, where every concept $E_i$ is a disjunction ($E_I \equiv F_1 \sqcup F_2 \sqcup ... \sqcup F_k$). Such concepts will be presented of n-partite form, where every vertex of the part presents $F_j$ concept, so vertexes form different parts will be connected, if corresponding concepts are disjointness. Model existing checking of such concepts performed by using of Bron-Kerbosch algorithm, which used for n-clique finding in n-partite graph.

*Classification optimizations* allow to reduce system worktime to perform the classification operation. Enhanced traversal method [2] is used for the classification, information about disjointness is extracted not only from subsumption test, but during the concept satisfiability testing. During the construction of model by tableau algorithm, labels of all individuals are checked in the presence of concepts $A$ and $\neg B$, though $A$ and $B$ are concept names. If those individuals exist, then $A \not\sqsubseteq B$, without performing $A \sqsubseteq B$ subsumption test.

## 4 System Performance Evaluation

The TReasoner system performance testing uses ontologies classification tests that were used on the OWL Reasoner Evaluation Workshop 2012 and compares the results received by HermiT (ver. 1.3.6) and FaCT++ (1.6.2) reasoners. They implement hypertableau and tableau algorithms and support the $SROIQ(D)$ logic. Information about used ontologies is shown in table 1.

System testing results in comparison with other reasoners are shown in table 2. First column of the table contains name of used ontology, and every subsequent column shows time spent for ontology classification by the relevant system. Testing was carried out on ASUS Notebook VX7SX Series Intel Core i7-2630QM CPU@2.00 GHz 2.00 GHz; 6.00 GB RAM running under Windows 7.

**Table 1.** Used ontologies

| Ontology | Logic | Axioms | Concepts | Roles | Individuals |
|---|---|---|---|---|---|
| obi | SHOIN(D) | 8530 | 1161 | 60 | 140 |
| plant_trait | ALC | 4317 | 976 | 1 | 3177 |
| po_temporal | ALC | 2839 | 284 | 1 | 2559 |
| DLPOnts_Information_397 | SHOIN | 1037 | 120 | 198 | 12 |
| DLPOnts_DOLCE-Lite_397 | SHIF | 351 | 37 | 70 | 0 |
| DLPOnts_Plans_397 | SHOIN | 1281 | 117 | 264 | 27 |
| pathway | ALC | 1927 | 646 | 1 | 1160 |
| protein | ALCS | 5821 | 1055 | 2 | 4768 |
| quality | ALCSH | 4815 | 1980 | 13 | 2653 |
| rex | ALC | 1725 | 555 | 2 | 991 |

**Table 2.** Testing results

| Ontology | TReasoner | FaCT++ | HermiT |
|---|---|---|---|
| obi | 17,065 | 1,313 | 130,359 |
| plant_trait | 1,035 | 0,099 | 0,228 |
| po_temporal | 0,098 | 0,071 | 0,064 |
| DLPOnts_Information_397.owl.txt | 5,177 | 0,94 | 11,443 |
| DLPOnts_DOLCE-Lite_397.owl.txt | 0,045 | 0,13 | 0,4 |
| DLPOnts_Plans_397.owl.txt | 5,515 | 0,167 | 217,667 |
| pathway.owl | 0,458 | 0,094 | 0,519 |
| protein.owl | 0,851 | 0,179 | 0,376 |
| quality.owl | 3,337 | 0,101 | 0,411 |
| rex.owl | 0,504 | 0,53 | 0,124 |

The resulting classification coincides to the reference classification provided together with chosen ontologies.

## 5 Conclusion

This system description presents the new reasoning system, implemented algorithms and implemented optimization techniques, which contribute to reduce worktime of different ontologies operations (classification, concept satisfiability checking, consistency checking). The developed system allows to perform logical analysis for expressive $SHOIN(D)$ logic that is used in OWL DL. This fact allows to use TReasoner to perform operations on a wide class of ontologies.

The presented testing results show that TReasoner may not compete yet with most popular systems such as HermiT and FaCT++ reasoners, but in future implementation of tableau algorithm will be improved and reduce of system worktime is expected.

In further researches improving of the TReasoner is expected in order to use it not only as module of the enterprise architecture verification expert system, but as self-dependent OWL reasoning system.

## References


1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications.* CUP, 2003.
2. F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, E. Franconi. An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or Making KRIS get a move on* *KR-92*, pages 270-281, 1992
3. Y. Ding and V. Haarslev. Tableau caching for description logics with inverse and transitive roles. In *Proc. DL-2006: International Workshop on Description Logics*, pages 143-149, 2006.
4. Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. Semantic Web 2(1): 11-21, 2011.
5. I. Horrocks. *Optimising Tableaux Decision Procedures for DescriptionLogics.* PhD thesis, University of Manchester, 1997.
6. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic SHIQ. In D. MacAllester, editor, *Proc. of the CADE 2000*, number 1831, pages 482496. Springer-Verlag, 2000.
7. Ian Horrocks, Boris Motik, and Zhe Wang. The HermiT OWL Reasoner. OWL Reasoner Evaluation Workshop. 2012.
8. A. Ivashko, E. Ivanova, E. Ovsyannikova, S. Kolomiyets. Applying DL for information system architecture description. *Vestnik of Tyumen State University*, 98(4): 137-142, 2012.
9. A. Ivashko, A. Grigorjev, M. Grigorjev. Modification of tableau algorithm based on checking disjointness of complex concepts. *Vestnik of Tyumen State University*, 98(4): 143-150, 2012.
10. Yevgeny Kazakov, Markus Krotzsch and Frantisek Simancik. ELK Reasoner: Architecture and Evaluation. OWL Reasoner Evaluation Workshop. 2012.
11. Julian Mendez. jcel: A Modular Rule-based Reasooner. OWL Reasoner Evaluation Workshop. 2012.
12. Linh Anh Nguyen. An Efficient Tableau Prover using Global Caching for the Description Logic ALC. *Artificial Intelligence*, 93(1):273-288, 2009.
13. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL WebOntology Language: Semantics and Abstract Syntax, W3C Recommendation, February 10 2004. http://www.w3.org/TR/owl-semantics/.
14. P. Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*. 9(3): 268-299, 1993.
15. D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. IJCAR 2006*, pages 292-297, 2006.

# Snorocket 2.0: Concrete Domains and Concurrent Classification

Alejandro Metke-Jimenez and Michael Lawley

The Australian e-Health Research Centre
ICT Centre, CSIRO
Brisbane, Queensland, Australia
{alejandro.metke,michael.lawley}@csiro.au
http://aehrc.com

**Abstract.** Snorocket is a high-performance ontology reasoner that supports a subset of the OWL EL profile. In the newest version, additional expressive power has been added to support concrete domains, enabling the classification of ontologies that use these constructs. Also, the reasoning algorithm has been modified to support concurrent classification. This feature is important because it enables the use of the full processing power available in modern multi-processor hardware.

**Keywords:** ontology, classification, concrete domains, concurrent

## 1 Introduction

This paper presents Snorocket 2.0, a high-performance ontology reasoner based on the CEL algorithm [6]. Snorocket was the first reasoner to provide ultra-fast classification of SNOMED CT and support incremental classification [1]. The initial version is used in the IHTSDO workbench to support SNOMED CT authoring and it is designed to work with a small heap footprint*. Snorocket 2.0 is now an open source project available at GitHub†.

Some biomedical ontologies, such as SNOMED CT, have been built using a subset of the OWL EL profile. Even though most of their content can be correctly modelled using this subset, some concepts cannot be fully modelled without concrete domains. For example, it is not possible to fully represent a "Hydrochlorothiazide 50mg tablet" without using a data literal to represent the quantity of the active ingredient. To overcome this limitation AMT v3, an extension of SNOMED CT used in Australia to model medicines, has recently introduced concrete domains. This has motivated the inclusion of concrete domains into the subset of constructs supported by Snorocket.

The development of extensions to SNOMED CT also means that ontology reasoners should be able to support classification of larger ontologies. This has motivated the implementation of a concurrent classification algorithm that allows using the extra processing power available in multi-processor machines.

*This was required to support 32-bit JVMs running on Windows machines.
†https://github.com/aehrc/snorocket

## 2 Background

The initial version of Snorocket was developed to support the fast classification of SNOMED CT and therefore only included support for a limited number of constructs. A table comparing the OWL EL constructs supported by Snorocket and other EL reasoners is available on the Snorocket website[‡].

Concrete domains are supported by several general tableaux-based reasoners such as FaCT++ [8] and HermiT [9]. The only specialised EL reasoner that currently supports concrete domains is ELK [4].

Concrete domains are used in AMT mainly to model quantities in the definition of medicines. An OWL version of AMT v3 can be obtained by using an updated version of the Perl script originally included in the SNOMED CT distribution. An example of a typical axiom found in AMT is available on the Snorocket website[§].

Most of the commonly used reasoners, including FACT++ [8], HermiT [9], CEL [6], and jCEL [7] are only capable of using a single processor. To our knowledge, the only reasoner that has successfully implemented a concurrent classification algorithm is ELK [4]. Because most modern hardware achieves better performance by providing more than one processor or core, it is important to be able to make use of this extra processing power.

## 3 Architecture

Figure 1 shows a high-level architecture diagram of Snorocket 2.0. The public Snorocket API, shown inside the **snorocket-core** module, enables third party applications to use the reasoner. The API uses a simple model to represent ontologies. This model is vastly simpler than other publicly available ontology models, such as OWL API, and excludes all the constructs not currently supported. This simplifies the usage of the public API. A more detailed description of this model is available on the Snorocket website[¶]. All external formats, such as OWL, RF1, and RF2, are transformed to and from this model. Additional ontology formats can be supported by adding new importer-exporter modules.

The Snorocket API defines an interface, IReasoner, to perform several reasoning functions. The reasoner interface defined by OWL API, OWLReasoner, is also implemented in the **snorocket-owlapi** module. Applications that want to use the reasoner can use either one. SNOMED CT-specific applications can use the RF1 and RF2 importer-exporter components to generate the axioms in our simple ontology format from the distribution files. It is also possible to create these axioms programatically. OWL ontologies are imported using OWL API and transformed into our simple model using the OWL importer-exporter component. A plugin for Protégé is also available in the **snorocket-protege** module.

[‡] http://aehrc.com/software/snorocket/index.html#constructs

[§] http://aehrc.com/software/snorocket/index.html#amtv3

[¶] http://aehrc.com/software/snorocket/index.html#model

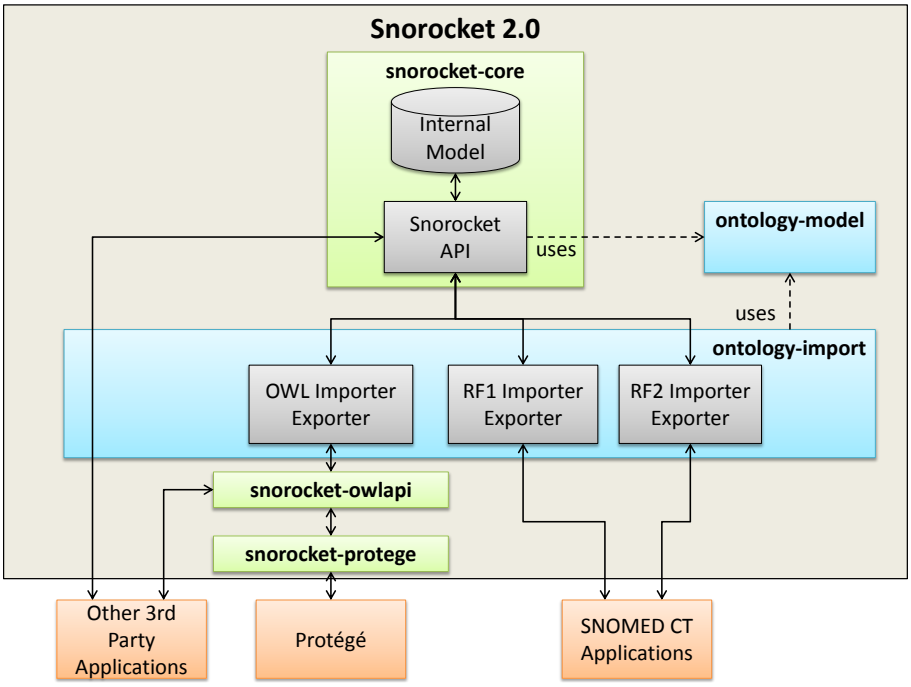



**Fig. 1.** Snorocket 2.0 architecture. The labels in bold refer to Maven modules.

## 4 Implementation

The current implementation of Snorocket is targeted at supporting the OWL EL profile. It is implemented using Java and built using Maven. The following sections describe the implementation details of the new features.

### 4.1 Concrete domains

In description logics a concrete domain is a construct that can be used to define new classes by specifying restrictions on attributes that have literal values (as opposed to relationships to other concepts). For example, children of age six can be defined by using the concrete domain expression $\exists hasAge.(=, 6)$. The class of individuals, in this case children of age six, is expressed as a restriction on the age attribute, which has a numeric value. The binary operators $<, <=, >, >=$ can also be used in a concrete domain expression, and attributes can have other types of literal values such as floating point numbers, string literals, and dates.

**Support for equality** An ontology can contain many complex axioms that include nested sub-expressions. The CEL algorithm works with normalised axioms and therefore creates a conservative extension of the original ontology containing

**Table 1.** Normal forms and completion rules.

| Normal form | Completion rules |
|---|---|
| $A_1 \sqcap A_2 \sqsubseteq B$ | **R1** If $A_1, A_2 \in S(X)$, $A_1 \sqcap A_2 \sqsubseteq B \in O$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$ |
| $A \sqsubseteq \exists r.B$ | **R2** If $A \in S(X)$, $A \sqsubseteq \exists r.B \in O$, and $(X, B) \notin R(r)$ then $R(r) := R(r) \cup \{(X, B)\}$ |
| $\exists r.A \sqsubseteq B$ | **R3** If $(X, Y) \in R(r)$, $A \in S(Y)$, $\exists r.A \sqsubseteq B \in O$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$ |
| $r \sqsubseteq s$ | **R4** If $(X, Y) \in R(r)$, $r \sqsubseteq s \in O$, and $(X, Y) \notin R(s)$ then $R(s) := R(s) \cup \{(X, Y)\}$ |
| $r \circ s \sqsubseteq t$ | **R5** If $(X, Y) \in R(r)$, $(Y, Z) \in R(s)$, $r \circ s \sqsubseteq t \in O$, and $(X, Z) \notin R(t)$ then $R(t) := R(t) \cup \{(X, Z)\}$ |
| $A \sqsubseteq \exists f.(o, v)$<br>$\exists f.(o, v) \sqsubseteq B$ | **R6** If $A \in S(X)$, $A \sqsubseteq \exists f.(o_1, v_1) \in O$, $\exists f.(o_2, v_2) \sqsubseteq B \in O$, $eval(o_1, v_1, o_2, v_2) = true$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$ |

only axioms in normal form [2]. The normal forms and the corresponding completion rules R1 to R5 from the original CEL algorithm are shown in Table 1. The last two normal forms and completion rule R6 have been added to support concrete domains.

The normalised forms of concrete domain expressions are $A \sqsubseteq \exists f.(o, v)$ and $\exists f.(o, v) \sqsubseteq A$, where $f$ represents a feature, $o$ an operator, and $v$ a value. The original normalisation algorithm requires only minor changes to deal with these new constructs.

The classification algorithm does require significant changes to deal with the new concrete domain axioms. A new type of queue is introduced to deal with the queue entries of the form $A \sqsubseteq \exists f.(o, v)$ and it is initialised with these axioms. The entries are then processed in the following way:

1. The axioms of the form $\exists f.(o, v) \sqsubseteq B$ that match the feature $f$ of the data type in the queue entry are retrieved.
2. The data types are then compared using the $eval()$ function.
3. If only the equality operator needs to be supported then the two data types are considered to be matching if their literal value is equal.

**Support for other operators** It is known that supporting arbitrary combinations of different operators leads to intractability [3]. In this implementation no checks are made to ensure that the ontology being classified complies with the restrictions that guarantee tractability. If non-compliant axioms are found then the reasoning procedure will be sound but possibly incomplete.

Adding support for other operators requires a modification to the $eval()$ function that compares the data types when evaluating feature queue entries. The different combinations of operators and values have to be evaluated to determine if there is a match or not.

For example, consider the following axioms:

$$toddler \equiv person \sqcap \exists hasAge.(\leq, 3)$$
$$child \equiv person \sqcap \exists hasAge.(\leq, 17)$$

After the normalisation process these axioms are transformed into the following:

| | |
|---|---|
| $\exists hasAge.(\leq, 17) \sqsubseteq A$ | $person \sqcap A \sqsubseteq child$ |
| $child \sqsubseteq person$ | $child \sqsubseteq \exists hasAge.(\leq, 17)$ |
| $\exists hasAge.(\leq, 3) \sqsubseteq B$ | $person \sqcap B \sqsubseteq toddler$ |
| $toddler \sqsubseteq person$ | $toddler \sqsubseteq \exists hasAge.(\leq, 3)$ |

These axioms allow us to infer that a toddler is also a child (but a child is not necessarily a toddler). This conclusion is derived when evaluating the expressions $toddler \sqsubseteq \exists hasAge.(\leq, 3)$ and $\exists hasAge.(\leq, 17) \sqsubseteq A$. The $eval()$ function in this case takes the arguments $(\leq, 3, \leq, 17)$ and returns a positive match because all the possible values of the first operator-value pair are covered by the possible values of the second operator-value pair. Whenever this is not the case the function returns false. Notice that this happens in some cases regardless of the literal values. For example, assuming we are dealing with integer values, $eval(x, <, y >)$ and $eval(x, >, y, <)$ will always return false because no matter what values are assigned to $x$ and $y$, the second operator-value pair will never be able to cover all the possible values expressed by the first pair.

### 4.2 Concurrent classification

This new version of Snorocket implements a multi-threaded saturation algorithm inspired by the algorithm used by ELK. The main idea of the algorithm is to split the computation into contexts that can be processed by workers independently while generating minimal locking overhead. Details of the original algorithm can be found in [5]. The main techniques in the algorithm can be applied in a straightforward manner to the CEL algorithm implemented by Snorocket.

## 5 Experimental results

Protégé was used to compare the performance of Snorocket against four other ontology reasoners: FaCT++, HermiT, jCel, and ELK. The previous version of Snorocket was also included. Two OWL ontologies were used in the tests: SNOMED CT and AMT v3. Both of these were derived from the RF2 distribution files using the corresponding Perl scripts.

The experiments were run in a computer equipped with a 3.3 GHz Intel i5 processor with 4 cores, 8 GB of physical memory, and running Windows 7. Protégé was run with Java 7 and a heap size of 4 GB. All the experiments use elapsed time as an indicator and use the external timing reported by Protégé. The multi-threaded reasoners (ELK 0.32 and Snorocket 2.0.1) were run using 4 threads.

Table 2 shows the profiles of the selected ontologies and Table 3 shows the classification times, in seconds, achieved by the reasoners, averaged over 5 runs.

**Table 2.** Profiles of the test ontologies.

| Ontology | #Classes | #Object Properties | #Data Properties | #Axioms | |
|---|---|---|---|---|---|
| | | | | Original | Normalised |
| SNOMED CT | 296518 | 62 | 0 | 660610 | 1169913 |
| AMT | 61059 | 78 | 4 | 150750 | 561331 |

**Table 3.** Average classification times in seconds using various reasoners in Protégé on Windows averaged over 5 runs. *mem* indicates an OutOfMemory error.

| | SNOMED CT | AMT |
|---|---|---|
| FACT++ 1.6.2 | 330 | 4220 |
| HermiT 1.3.7 | 1567.3 | mem |
| jCel 0.15[‡] | 761 | - |
| ELK 0.32 | 9.1 | 10.5 |
| Snorocket 1.3.4 | 33.8 | - |
| Snorocket 2.0.1 | 26 | 26.2 |

The results show that the performance of the tableaux-based reasoners was very poor when classifying AMT. On the other hand, the specialised EL reasoners were able to classify it in a fraction of the time. ELK currently provides the best performance, which is expected since Snorocket's multi-threaded implementation is based on the same techniques but has not been optimised. Also, Snorocket only runs the saturation phase concurrently, while the rest of the steps are still run sequentially.

## 6 Conclusions and future work

This paper presented Snorocket 2.0 and compared it against its previous version and four other reasoners using two large medical ontologies. Even though ELK obtained the fastest results, Snorocket 2.0 achieved competitive performance. Snorocket's built-in support for SNOMED CT distribution formats makes it an interesting alternative to ELK for SNOMED CT-centric applications.

Future work will include adding multi-threading to the whole classification process and incorporating the restrictions necessary to ensure tractability when dealing with concrete domains, either as a hard restriction or as a warning to the user.

[‡]The current version of the jCel plugin is 0.18.2 but version 0.15 was the most recent one that was compatible with our testing environment.

## References


1. Lawley, M. J., Bousquet, C.: Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In: Proc. 6th Australasian Ontology Workshop (IAOA10). Conferences in Research and Practice in Information Technology, pp. 45–49. (2010)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the EL envelope. In: International Joint Conference on Artificial Intelligence, p. 364 (2005)
3. Magka, D., Kazakov, Y., Horrocks, I.: Tractable Extensions of the Description Logic EL with Numerical Datatypes. In: Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2010). LNAI, vol. 6173, pp. 61–75. Springer (2010)
4. Kazakov, Y., Krötzsch, M., Simančk, F.: ELK Reasoner: Architecture and Evaluation. In: Proceedings of the 1st International Workshop on OWL Reasoner Evaluation, CEUR Workshop Proceedings, (2012)
5. Kazakov, Y., Kötzsch, M., Simančk, F.: Concurrent Classification of EL+ Ontologies. In: The Semantic Web ISWC 2011, pp. 305–320 (2011)
6. Baader, F., Lutz, C., Suntisrivaraporn, B.: Efficient reasoning in EL+. In: Proceedings of DL 2006, p.189 (2006)
7. Mendez, J. jcel: A Modular Rule-based Reasoner. In: Proc. of the 1st Int. Workshop on OWL Reasoner Evaluation (ORE12), pp. 130-135 (2012)
8. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Proc. 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006). LNCS, vol. 4130, pp. 292-297. Springer (2006)
9. Motik, B., Shearer, R., Horrocks, I.: HermiT: Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research 36, pp. 165–228 (2009)
10. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. of Web Semantics 5(2), pp. 51-53 (2007)

# A Transformation Approach for Classifying $\mathcal{ALCHI}(\mathcal{D})$ Ontologies with a Consequence-based $\mathcal{ALCH}$ Reasoner

Weihong Song, Bruce Spencer, and Weichang Du

Faculty of Computer Science, University of New Brunswick, Fredericton, Canada

**Abstract.** Consequence-based techniques have been developed to provide efficient classification for less expressive languages. Ontology transformation techniques are often used to approximate axioms in a more expressive language by axioms in a less expressive language. In this paper, we present an approach to use a fast consequence-based $\mathcal{ALCH}$ reasoner to classify an $\mathcal{ALCHI}(\mathcal{D})$ ontology with a subset of OWL 2 datatypes and facets. We transform datatype and inverse role axioms into $\mathcal{ALCH}$ axioms. The transformed ontology preserves sound and complete classification w.r.t the original ontology. The proposed approach has been implemented in the prototype WSClassifier which exhibits the high performance of consequence reasoning. The experiments show that for classifying large and highly cyclic $\mathcal{ALCHI}(\mathcal{D})$ ontologies, WSClassifier's performance is significantly faster than tableau-based reasoners.

## 1 Introduction

Ontology classification is the foundation of many ontology reasoning tasks. Recently, consequence-based techniques have been developed to provide efficient classification for sublanguages of OWL 2 DL profile, e.g. $\mathcal{EL}^{++}$ [2,3,8], Horn-$\mathcal{SHIQ}$ [7], $\mathcal{EL}^{\bot}(\mathcal{D})$ [9], $\mathcal{ALCH}$ [13]. There have been some approaches to use existing consequence-based reasoners to classify more expressive ontologies, like MORe [1]. In this paper, we propose an approach to use a consequence-based $\mathcal{ALCH}$ reasoner to classify an $\mathcal{ALCHI}(\mathcal{D})$ ontology by transforming it into an $\mathcal{ALCH}$ ontology with soundness and completeness preserved. The purpose of the approach is to extend the expressiveness of the existing consequence-based reasoner without changing its complex inference rules and implementation. All proofs and further technical details can be found in our technical report [15].

Ontology transformation is often accomplished by approximating non-Horn ontologies/theories by Horn replacements [12,11,16]. These approximations can be used to optimize reasoning by exploiting more efficient inference for Horn ontologies/theories. The approximation $\mathcal{O}'$ in Ren *et al.* [11] is a lower bound of the original ontology $\mathcal{O}$, i.e. $\mathcal{O}'$ entails no more subsumptions than $\mathcal{O}$ does. In contrast, approximation in Zhou *et al.* [16] provides an upper bound. Kautz *et al.* [12] computes both upper and lower bounds of propositional logic theories. Another approach preserves both soundness and completeness of classification results such as the elimination of transitive roles in Kazakov [7]. Our work of this paper is of the second kind. We classify an $\mathcal{ALCHI}(\mathcal{D})$ ontology $\mathcal{O}$ in two stages: (1) transform $\mathcal{O}$ into an $\mathcal{ALCHI}$ ontology $\mathcal{O}_{\mathcal{D}}^{-}$ s.t. $\mathcal{O} \models A \sqsubseteq B$ iff $\mathcal{O}_{\mathcal{D}}^{-} \models A \sqsubseteq B$;

(2) transform $\mathcal{O}_{\mathcal{D}}^{-}$ into an $\mathcal{ALCH}$ ontology $\mathcal{O}_{\mathcal{ID}}^{-}$ s.t. $\mathcal{O}_{\mathcal{ID}}^{-} \models A \sqsubseteq B$ iff $\mathcal{O}_{\mathcal{D}}^{-} \models A \sqsubseteq B$. We use these approaches to implement a reasoner called WSClassifier which transforms an $\mathcal{ALCHI}(\mathcal{D})$ ontology into an $\mathcal{ALCH}$ ontology and classifies it with a fast consequence $\mathcal{ALCH}$ reasoner ConDOR [13]. WSClassifier is significantly faster than tableau-based reasoners on large and highly cyclic ontologies.

In our previous work [14] we approximated an $\mathcal{ALCHOI}$ ontology by an $\mathcal{ALCH}$ ontology which was then classified by a hybrid of consequence- and tableau-based reasoners. Unlike [14], in this paper we claim completeness for $\mathcal{I}$'s transformation. Calvanese *et al.* [4] introduces a general approach to eliminate inverse roles and functional restrictions from $\mathcal{ALCFI}$ to $\mathcal{ALC}$. For eliminating $\mathcal{I}$, the approach needs to add one axiom for each inverse role and each concept. So the number of axioms added can be very large. Ding *et al.* [6] introduces a new mapping from $\mathcal{ALCI}$ to $\mathcal{ALC}$ and further extends it to a mapping from $\mathcal{SHI}$ to $\mathcal{SH}$ in [5]. The approach allows tableau-based decision procedures to use some caching techniques and improve the reasoning performance in practice. Both approaches in [4,5] preserve the soundness and completeness of inference after elimination of $\mathcal{I}$. Our approach is similar to the one in [6,5]. However, the NNF normalized form in [6,5] in which $\top$ appears in the left side of all axioms will dramatically degrade the performance of our consequence-based $\mathcal{ALCH}$ reasoner. Thus we eliminate the inverse role based on our own normalized form and our approach is more suitable for consequence-based reasoners.

## 2 Preliminary

Due to space limitation, we only list the most necessary syntax and semantics of $\mathcal{ALCHI}(\mathcal{D})$ in the paper, the complete illustration can be found in our Technical Report [15]. The syntax of $\mathcal{ALCHI}(\mathcal{D})$ uses atomic concepts $N_C$, atomic roles $N_R$ and features $N_F$. We use $A, B$ for atomic concepts, $C, D$ for concepts, $r, s$ for atomic roles, $R, S$ for roles, $F, G$ for features. The parameter $\mathcal{D}$ defines a *datatype map* $\mathcal{D} = (N_{DT}, N_{LS}, N_{FS}, \cdot^{\mathcal{D}})$, where: (1) $N_{DT}$ is a set of datatype names; (2) $N_{LS}$ is a function assigning to each $d \in N_{DT}$ a set of constants $N_{LS}(d)$; (3) $N_{FS}$ is a function assigning to each $d \in N_{DT}$ a set of facets $N_{FS}(d)$, each $f \in N_{FS}(d)$ has the form $(p_f, v)$; (4) $\cdot^{\mathcal{D}}$ is a function assigning a datatype interpretation $d^{\mathcal{D}}$ to each $d \in N_{DT}$ called the *value space* of $d$, a data value $v^{\mathcal{D}} \in d^{\mathcal{D}}$ for each $v \in N_{LS}(d)$, and a facet interpretation $f^{\mathcal{D}}$ for each $f \in \bigcup_{d \in N_{DT}} N_{FS}(d)$. Since one facet may be shared by multiple datatypes, we define its interpretation as containing subsets of all the relevant datatypes. $\top_{\mathcal{D}}$, $d$, $d[f]$ or $\{v\}$ are basic forms of data ranges, which we call *atomic data range*s. A data range $dr$ is defined recursively using $\sqcap$, $\sqcup$, and $\neg$. A role $R$ is either an atomic role $r$ or *inverse role* $r^{-}$. Semantics of $\mathcal{ALCHI}(\mathcal{D})$ is defined via an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \Delta^{\mathcal{D}}, \cdot^{\mathcal{I}}, \cdot^{\mathcal{D}})$. $\Delta^{\mathcal{I}}$ and $\Delta^{\mathcal{D}}$ are disjoint non-empty sets called *object domain* and *data domain*. $d^{\mathcal{D}} \subseteq \Delta^{\mathcal{D}}$ for each $d \in N_{DT}$. $\cdot^{\mathcal{I}}$ assigns a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to each $A \in N_C$, a relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each $r \in N_R$ and a relation $F^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{D}}$ to each $F \in N_F$. $F^{\mathcal{I}}(x) = \{v \mid (x, v) \in F^{\mathcal{I}}\}$. $\cdot^{\mathcal{D}}$ interprets data ranges and concepts, as shown in Table 1. We write $N_{DT}(\mathcal{O})$ and $ADR(\mathcal{O})$ for all datatypes and atomic data ranges in $\mathcal{O}$. And $ADR_d(\mathcal{O})$ denotes the subset of $ADR(\mathcal{O})$ in datatype $d$, i.e. of the form $d$, $d[f]$ or $\{v\}$ where $v \in N_{LS}(d)$.

**Table 1.** Part of Model Theoretic Semantics of $\mathcal{ALCHI}(\mathcal{D})$

| Semantics of Data Ranges | | |
|---|---|---|
| $(\top_{\mathcal{D}})^{\mathcal{D}} = \Delta^{\mathcal{D}}$ | $\{v\}^{\mathcal{D}} = \{v^{\mathcal{D}}\}$ | $(dr_1 \sqcap dr_2)^{\mathcal{D}} = dr_1^{\mathcal{D}} \cap dr_2^{\mathcal{D}}$ |
| $(d[f])^{\mathcal{D}} = d^{\mathcal{D}} \cap f^{\mathcal{D}}$ | $(\neg dr)^{\mathcal{D}} = \Delta^{\mathcal{D}} \setminus dr^{\mathcal{D}}$ | $(dr_1 \sqcup dr_2)^{\mathcal{D}} = dr_1^{\mathcal{D}} \cup dr_2^{\mathcal{D}}$ |
| Semantics of Concepts, Roles and Axioms | | |
| $F \sqsubseteq G \Rightarrow F^{\mathcal{I}} \subseteq G^{\mathcal{I}}$ | $(\exists F.dr)^{\mathcal{I}} = \{x \mid F^{\mathcal{I}}(x) \cap dr^{\mathcal{D}} \neq \emptyset\}$ | $(\forall F.dr)^{\mathcal{I}} = \{x \mid F^{\mathcal{I}}(x) \subseteq dr^{\mathcal{D}}\}$ |

## 3 Transformation for Datatypes

In this section we introduce how we transform an $\mathcal{ALCHI}(\mathcal{D})$ ontology $\mathcal{O}$ into an $\mathcal{ALCHI}$ ontology $\mathcal{O}_{\mathcal{D}}^{-}$ such that $\mathcal{O} \models A \sqsubseteq B$ iff $\mathcal{O}_{\mathcal{D}}^{-} \models A \sqsubseteq B$. We assume all the datatypes in $\mathcal{D}$ are disjoint, as do Motik et al [10]. We apply our approach to some commonly used datatypes: (1) real with facets rational, decimal, integer, $>_a$, $\geq_a$, $<_a$ and $\leq_a$; (2) strings with equal value; (3) boolean values.

Our basic idea to produce $\mathcal{O}_{\mathcal{D}}^{-}$ from $\mathcal{O}$ is to encode features into roles and data ranges into concepts, and then add extra axioms to preserve the subsumptions between atomic concepts in $N_C(\mathcal{O})$. Table 2 gives the definition of encoding function $\varphi$ over atomic elements in $\mathcal{O}$, where $A_d, A_f, A_v$ are fresh concepts and $R_F$ is a fresh role. $\varphi$ over complex data ranges, roles, concepts and axioms are defined recursively using corresponding constructors. It is easy to prove that classification of $\varphi(\mathcal{O}) = \{\varphi(\alpha) \mid \alpha \in \mathcal{O}\}$ is sound w.r.t. $\mathcal{O}$ (proof see [15]). In order to preserve classification completeness, extra axioms need to be added to $\varphi(\mathcal{O})$ to get $\mathcal{O}_{\mathcal{D}}^{-}$. Algorithm 1 shows how $\mathcal{O}_{\mathcal{D}}^{-}$ is computed. In the procedure

**Table 2.** Encoding $\varphi$ for atomic concepts/roles/features/data ranges

| | | | |
|---|---|---|---|
| $\varphi(\top_{\mathcal{D}}) = \top$ | $\varphi(d[f]) = A_d \sqcap A_f$ | $\varphi(\top) = \top$ | $\varphi(A) = A$ |
| $\varphi(d) = A_d$ | $\varphi(\{v\}) = A_v$ | $\varphi(R) = R$ | $\varphi(F) = R_F$ |

we use two functions $\mathsf{normalize}_d$ and $\mathsf{getAxioms}_d$ for each datatype $d \in N_{DT}(\mathcal{O})$. $\mathsf{normalize}_d$ rewrites data ranges $d[f]$ into normalized forms to reduce the kinds of facets used. $\mathsf{getAxioms}_d$ produces a set of $\mathcal{ALCHI}$ axioms $\mathcal{O}_d^{+}$ to be included into $\mathcal{O}_{\mathcal{D}}^{-}$. Details will be explained later for the datatypes and facets supported. In order to preserve classification completeness w.r.t. $\mathcal{O}$, $\mathsf{getAxioms}_d$ must generate axioms explicitly showing the relationships implicit among data ranges before encoding, i.e., the **data-range-relationship-preserving property**: for any $ar_1, \ldots, ar_n, ar'_1, \ldots, ar'_m \in ADR_d(\mathcal{O})$, if $\left((\bigsqcap_{i=1}^{n} ar_i) \sqcap (\bigsqcap_{j=1}^{m} \neg ar'_j)\right)^{\mathcal{D}} = \emptyset$, then $(\bigsqcap_{i=1}^{n} \varphi(ar_i)) \sqcap (\bigsqcap_{j=1}^{m} \neg \varphi(ar'_j))$ is unsatisfiable in $\mathcal{O}_d^{+} = \mathsf{getAxioms}_d(ADR_d(\mathcal{O}), \varphi)$. We prove this condition is sufficient for completeness in [15].

For boolean type, we do not have any facets, so $\mathsf{normalize}_d$ does nothing. Since the only atomic data ranges are $xsd:boolean$, $\{true\}$ and $\{false\}$, $\mathsf{getAxioms}_d$ only needs to return two axioms $\varphi(xsd:boolean) \equiv \varphi(\{true\}) \sqcup \varphi(\{false\})$ and $\varphi(\{true\}) \sqcap \varphi(\{false\}) \sqsubseteq \bot$. For string type, currently we do not support any facets, so $\mathsf{normalize}_d$ does nothing either. Atomic data ranges are ei-

**Algorithm 1:** Datatype Transformation

**Input**: An $\mathcal{ALCHI}(\mathcal{D})$ ontology $\mathcal{O}$
**Output**: An $\mathcal{ALCHI}$ ontology $\mathcal{O}^-_{\mathcal{D}}$ with the same classification result as $\mathcal{O}$

```
1 foreach d ∈ N_DT(O) do
2 |   foreach adr ∈ ADR_d(O) do
3 |   |   Replace adr with normalize_d(adr) in O;
4 Create an encoding φ for O and initialize O^-_D with φ(O);
5 foreach d_1, d_2 ∈ N_DT(O), d_1 ≠ d_2 do O^-_D ← O^-_D ∪ {φ(d_1) ⊓ φ(d_2) ⊑ ⊥};
6 foreach d ∈ N_DT(O) do O^-_D ← O^-_D ∪ getAxioms_d(ADR_d(O), φ);
7 return O^-_D;
```

ther $xsd : string$ or of the form $\{c\}$, where $c$ is a constant. We need to add $\varphi(\{c\}) \sqsubseteq \varphi(xsd : string)$ for each $\{c\} \in ADR_{\mathbb{R}}(\mathcal{O})$, as well as pairwise disjoint axioms for all such $\varphi(\{c\})$. Numeric datatypes are the most commonly used datatypes in ontologies. Here we discuss the implementation for $owl : real$, which we denote by $\mathbb{R}$. $owl : rational$, $xsd : decimal$ and $xsd : integer$ are treated as facets $rat$, $dec$ and $int$ of $\mathbb{R}$, respectively. Comparison facets of the forms $>_a$, $<_a$, $\geq_a$, $\leq_a$ are supported. For $\mathsf{normalize}_d$ with input $ar$, we need: (1) if $adr = \mathbb{R}[f]$, transform it to equivalent data ranges using only facets of the form $>_a$, e.g. $\mathbb{R}[<_a] = \mathbb{R} \sqcap \neg(\mathbb{R}[>_a] \sqcup \{a\})$; (2) replace any constant $a$ used in $ar$ with a normal form, so that any constants having the same interpretation becomes the same after normalization, e.g. integer constants $+3$ and 3 are both interpreted as real number 3, so they are normalized into the same form $\texttt{"3"}\hat{}\hat{}xsd : integer$. Algorithm 2 gives the details of $\mathsf{getAxioms}_{\mathbb{R}}$ for real numbers. For boolean and string, it is obvious that the corresponding $\mathsf{getAxioms}_d$ has data-range-relationship-preserving property. For $\mathsf{getAxioms}_{\mathbb{R}}$, we prove this property in [15]. So if $\mathcal{O} \models A \sqsubseteq B$, then $\mathcal{O}^-_{\mathcal{D}} \models A \sqsubseteq B$.

## 4 Transformation for Inverse Roles

In this section, we discuss how we transform an $\mathcal{ALCHI}$ ontology $\mathcal{O}^-_{\mathcal{D}}$ into an $\mathcal{ALCH}$ ontology $\mathcal{O}^-_{\mathcal{ID}}$, such that $\mathcal{O}^-_{\mathcal{D}} \models A \sqsubseteq B$ iff $\mathcal{O}^-_{\mathcal{ID}} \models A \sqsubseteq B$(proof see [15]). Algorithm 3 shows the details of transformation for inverse roles. In the procedure $\mathsf{Inv}_r$ contains the set of atomic roles which are inverses of $r$. Line 1 initializes $\mathcal{O}^-_{\mathcal{ID}}$ with $\mathcal{ALCH}$ axioms in $\mathcal{O}^-_{\mathcal{D}}$. Lines 2 to 6 initializes $\mathsf{Inv}_r$ and put all $r$ where $\mathsf{Inv}_r \neq \emptyset$ into $\mathsf{RolesToBeProcessed}$. Lines 7 to 16 processes each role in $\mathsf{RolesToBeProcessed}$ and adds axioms into $\mathcal{O}^-_{\mathcal{ID}}$ to address the effect of inverse role axioms. Detail explanations of Algorithm 3 are in our technical report [15].

## 5 Experiment and Conclusion

In experiments we compare the runtime of our WSClassifier with all other available $\mathcal{ALCHI}(\mathcal{D})$ reasoners HermiT, Fact++ and Pellet, which all happen to be tableau-based reasoners. We use all large and highly cyclic ontologies we can

**Algorithm 2:** $\mathsf{getAxioms}_{\mathbb{R}}$ for $\mathbb{R}$

**Input**: A set of atomic data ranges $ADR_{\mathbb{R}}(\mathcal{O})$ of type $\mathbb{R}$, encoding function $\varphi$
**Output**: A set of axioms $\mathcal{O}^+_{\mathbb{R}}$

$\mathcal{O}^+_{\mathbb{R}} \leftarrow \emptyset$;
**foreach** $\{v\} \in ADR_{\mathbb{R}}(\mathcal{O})$ **do**
  **if** $\mathbb{R}[int] \in ADR_{\mathbb{R}}(\mathcal{O})$ *and* $v^{\mathcal{D}} \in (\mathbb{R}[int])^{\mathcal{D}}$ **then** add $\varphi(\{v\}) \sqsubseteq \varphi(int)$ to $\mathcal{O}^+_{\mathbb{R}}$;
  **if** $\mathbb{R}[dec] \in ADR_{\mathbb{R}}(\mathcal{O})$ *and* $v^{\mathcal{D}} \in (\mathbb{R}[dec])^{\mathcal{D}}$ **then** add $\varphi(\{v\}) \sqsubseteq \varphi(dec)$ to $\mathcal{O}^+_{\mathbb{R}}$;
  **if** $\mathbb{R}[rat] \in ADR_{\mathbb{R}}(\mathcal{O})$ *and* $v^{\mathcal{D}} \in (\mathbb{R}[rat])^{\mathcal{D}}$ **then** add $\varphi(\{v\}) \sqsubseteq \varphi(rat)$ to $\mathcal{O}^+_{\mathbb{R}}$;
**if** $\mathbb{R}[int], \mathbb{R}[dec] \in ADR_{\mathbb{R}}(\mathcal{O})$ **then** add $\varphi(int) \sqsubseteq \varphi(dec)$ to $\mathcal{O}^+_{\mathbb{R}}$;
**if** $\mathbb{R}[int], \mathbb{R}[rat] \in ADR_{\mathbb{R}}(\mathcal{O})$ **then** add $\varphi(int) \sqsubseteq \varphi(rat)$ to $\mathcal{O}^+_{\mathbb{R}}$;
**if** $\mathbb{R}[dec], \mathbb{R}[rat] \in ADR_{\mathbb{R}}(\mathcal{O})$ **then** add $\varphi(dec) \sqsubseteq \varphi(rat)$ to $\mathcal{O}^+_{\mathbb{R}}$;
Put all $\mathbb{R}[>_a] \in ADR_{\mathbb{R}}(\mathcal{O})$ in *fArray* with ascending order of $a$;
**foreach** *pair of adjacent elements* $\mathbb{R}[>_a]$ *and* $\mathbb{R}[>_b]$ *(*$a < b$*) in fArray* **do**
  add $\varphi(>_b) \sqsubseteq \varphi(>_a)$ to $\mathcal{O}^+_{\mathbb{R}}$;
  **if** $\mathbb{R}[int] \in ADR_{\mathbb{R}}(\mathcal{O})$ **then**
    $M \leftarrow \{\{a_i\}\}_{i=1}^n$, where $a_1, \ldots, a_n$ are all integer constants in $(a, b]$;
    **if** $M \subseteq ADR_{\mathbb{R}}(\mathcal{O})$ **then**
      add $\varphi(int) \sqcap \varphi(>_a) \sqcap \neg\varphi(>_b) \sqcap (\prod_{i=1}^n \neg\varphi(\{a_i\})) \sqsubseteq \bot$ to $\mathcal{O}^+_{\mathbb{R}}$;
  Let $N$ be all $v$ such that $\{v\} \in ADR_{\mathbb{R}}(\mathcal{O})$ and $v^{\mathcal{D}} \in (a, b]$;
  **foreach** $v \in N$ **do** add $\varphi(\{v\}) \sqsubseteq \varphi(>_a)$, $\varphi(\{v\}) \sqcap \varphi(>_b) \sqsubseteq \bot$ to $\mathcal{O}^+_{\mathbb{R}}$;
  **foreach** $v_1, v_2 \in N, v_1 \neq v_2$ **do** add $\varphi(\{v_1\}) \sqcap \varphi(\{v_2\}) \sqsubseteq \bot$ to $\mathcal{O}^+_{\mathbb{R}}$;
**return** $\mathcal{O}^+_{\mathbb{R}}$;

access to. FMA-constitutionalPartForNS(FMA-C) is the only large and highly cyclic ontology that contains $\mathcal{ALCHI}(\mathcal{D})$ constructors. We remove seven axioms using $xsd : float$. For Full-Galen which language is $\mathcal{ALEHIF}+$ without "D", we introduce some new data type axioms by converting some axioms using roles hasNumber and hasMagnitude into axioms with new features hasNumberDT and hasMagnitudeDT. Some concepts which should be modeled as data ranges are also converted to data ranges. Wine is a small but cyclic ontology. We also include two commonly used ontologies ACGT and OBI which are not highly cyclic. For Wine, ACGT and OBI, we change xsd:int, xsd:positiveInteger, xsd:nonNegativeInteger to xsd:integer, xsd:float to owl:rational, and remove xsd:dateTime if applicable. For all the ontologies, we reduce their language to $\mathcal{ALCHI}(\mathcal{D})$. The ontologies are available from our website.[1].The experiments were conducted on a laptop with Intel Core i7-2670QM 2.20GHz quad core CPU and 16GB RAM. We set the Java heap space to 12GB and the time limit to 24 hours.

Table 3 summarizes the result. HermiT is set to configuration with simple core blocking and individual reuse. WSClassifier is significantly faster than the tableau-based reasoners on the three highly cyclic large ontologies Galen-Heart, Full-Galen and FMA-C. ACGT is not highly cyclic, but WSClassifier is still faster. For the other two ontologies where WSClassifier is not the fastest, Wine is cyclic but small, OBI is not highly cyclic. The classification time for them on

[1] `http://isel.cs.unb.ca/~wsong/ORE2013WSClassifierOntologies.zip`

**Algorithm 3:** Transformation for inverse roles

**Input**: Normalized ontology $\mathcal{ALCHI}$ ontology $\mathcal{O}_{\mathcal{D}}^{-}$
**Output**: An $\mathcal{ALCH}$ ontology $\mathcal{O}_{\mathcal{ID}}^{-}$ having the same classification result as $\mathcal{O}_{\mathcal{D}}^{-}$

```
Initialize O⁻_ID with all ALCH axioms in O⁻_D, excluding inverse role axioms;
foreach r ∈ N_R(O) do Inv_r ← ∅ ;
RolesToBeProcessed ← ∅;
foreach r' = r⁻ ∈ O⁻_D do
    Inv_r ← Inv_r ∪ {r'}; Inv_r' ← Inv_r' ∪ {r};
    RolesToBeProcessed ← RolesToBeProcessed ∪ {r, r'};
while RolesToBeProcessed ≠ ∅ do
    remove a role r from RolesToBeProcessed and pick a role r' from Inv_r;
    foreach r* ∈ Inv_r where r* is not r' do add r' ≡ r* to O⁻_ID;
    foreach r ⊑ s ∈ O⁻_D do
        if Inv_s = ∅ then
            add a fresh atomic role s' to Inv_s;
            RolesToBeProcessed ← RolesToBeProcessed ∪ {s};
        pick a role s' from Inv_s and add r' ⊑ s' to O⁻_ID;
    foreach ∃r.A ⊑ B ∈ O⁻_D do add A ⊑ ∀r'.B to O⁻_ID;
    foreach A ⊑ ∀r.B ∈ O⁻_D do add ∃r'.A ⊑ B to O⁻_ID;
return O⁻_ID
```

all reasoners are significantly shorter comparing with the time on large highly cyclic ontologies. Then WSClassifier took a larger percentage of time on the overhead to transmit the ontology to and from ConDOR.

**Table 3.** Comparison of classification performance of $\mathcal{ALCHI}(\mathcal{D})$ ontologies

| | **HermiT** | **Pellet** | **FaCT++** | **WSClassifier** |
|---|---|---|---|---|
| Wine | 1.160 sec | 0.430 sec | 0.005 sec | 0.400 sec |
| ACGT | 9.603 sec | 2.955 sec | * | 1.945 sec |
| OBI | 3.166 sec | 45.261 sec | * | 8.835 sec |
| Galen-Heart | 123.628 sec | – | – | 2.779 sec |
| Full-Galen | – | – | – | 16.774 sec |
| FMA-C | – | – | – | 32.74 sec |

**Note**: "–": out of time or memory "*": some datatypes are not supported

We have transformed some commonly used OWL 2 datatypes and facets and inverse role axioms in an $\mathcal{ALCHI}(\mathcal{D})$ ontology to $\mathcal{ALCH}$ and classified it on an $\mathcal{ALCH}$ reasoner with soundness and completeness of classification preserved. WSClassifier greatly outperforms tableau-based reasoners when the ontologies are large and highly cyclic. Future work includes extension to other data types and facets, and further optimization, e.g. adapting the idea of Magka *et al.* [9] to WSClassifier to distinguish positive and negative occurrences of data ranges, in order to reduce the number of axioms to be added.

## References


1. Armas Romero, A., Cuenca Grau, B., Horrocks, I.: MORe: Modular combination of OWL reasoners for ontology classification. In: Proc. of ISWC. pp. 1–16 (2012)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: IJCAI. pp. 364–369 (2005)
3. Baader, F., Lutz, C.: Pushing the $\mathcal{EL}$ envelope further. In: Proc. of OWLED (2008)
4. Calvanese, D., De Giacomo, G., Rosati, R.: A note on encoding inverse roles and functional restrictions in alc knowledge bases. In: Proc. of the 5th Int. Description Logic Workshop. DL. vol. 98, pp. 11–20 (1998)
5. Ding, Y.: Tableau-based reasoning for description logics with inverse roles and number restrictions. `http://users.encs.concordia.ca/~haarslev/students/Yu_Ding.pdf` (2008)
6. Ding, Y., Haarslev, V., Wu, J.: A new mapping from alci to alc. In: Proc. DL-2007. CEUR Workshop Proceedings. vol. 250. Citeseer (2007)
7. Kazakov, Y.: Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In: Proc. of IJCAI. pp. 2040–2045 (2009)
8. Kazakov, Y., Krötzsch, M., Simančík, F.: Concurrent classification of $\mathcal{EL}$ ontologies. In: Proc. of ISWC. pp. 305–320 (2011)
9. Magka, D., Kazakov, Y., Horrocks, I.: Tractable extensions of the description logic $\mathcal{EL}$ with numerical datatypes. J. Automated Reasoning 47(4), 427–450 (2011)
10. Motik, B., Horrocks, I.: Owl datatypes: Design and implementation. In: International Semantic Web Conference. pp. 307–322 (2008)
11. Ren, Y., Pan, J.Z., Zhao, Y.: Soundness preserving approximation for TBox reasoning. In: Proc. of AAAI (2010)
12. Selman, B., Kautz, H.: Knowledge compilation and theory approximation. J. ACM 43(2), 193–224 (1996)
13. Simančík, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond Horn ontologies. In: Proc. of IJCAI. pp. 1093–1098 (2011)
14. Song, W., Spencer, B., Du, W.: WSReasoner: A prototype hybrid reasoner for $\mathcal{ALCHOI}$ ontology classification using a weakening and strengthening approach. In: Proc. of the 1st Int. OWL Reasoner Evaluation Workshop (2012)
15. Song, W., Spencer, B., Du, W.: Technical report of a transformation approach for classifying $\mathcal{ALCHI}(\mathcal{D})$ ontologies with a consequence-based $\mathcal{ALCH}$ reasoner. Tech. rep. (2013), `http://www.cs.unb.ca/tech-reports/documents/TR13-225.pdf`
16. Zhou, Y., Cuenca Grau, B., Horrocks, I.: Efficient upper bound computation of query answers in expressive description logics. In: Proc. of DL (2012)

# Android goes Semantic: DL Reasoners on Smartphones

Roberto Yus, Carlos Bobed, Guillermo Esteban, Fernando Bobillo, and Eduardo Mena

University of Zaragoza
Maria de Luna 1, Zaragoza, Spain
{ryus,cbobed,gesteban,fbobillo,emena}@unizar.es

**Abstract.** The massive spread of mobile computing in our daily lives has attracted a huge community of mobile apps developers. These developers can take advantage of the benefits of semantic technologies (such as knowledge sharing and reusing, knowledge decoupling, etc.) to enhance their applications. Moreover, the use of semantic reasoners would enable them to create more intelligent applications capable of inferring logical consequences from the knowledge considered. However, using semantic APIs and reasoners on current Android-based devices is not problem-free and, currently, there are no remarkable efforts to enable mobile devices with semantic reasoning capabilities.
In this paper, we analyze whether the most popular current available DL reasoners can be used on Android-based devices. We evaluate the efforts needed to port them to the Android platform, taking into account its limitations, and present some tests to show the performance of these reasoners on current smartphones/tablets.

## 1 Introduction

In the last few years, we have witnessed a massive spread of mobile computing in our daily lives. The progress and popularity of the different mobile devices (smartphones, tablets, etc.) has attracted a huge community of developers that are continually releasing new applications via the different app stores. For example, 136 millions of Android-based devices were sold in third quarter of 2012 (which represented 75% of the market)[1] and the Google Play[2] market contained almost 700,000 available applications in April 2013.

Due to the current device capabilities, we think that we could also start to enhance all these applications with semantic technologies [16]. Using these techniques, developers can enhance their applications based on the principles of knowledge sharing and reusing [3], making explicit domain assumptions and thus decoupling the knowledge from the application, etc. For example, the use

[1] IDC studies, http://www.idc.com/getdoc.jsp?containerId=prUS23771812, last accessed 18th April 2013
[2] http://play.google.com

of ontologies and semantic reasoners would enable them to create more intelligent applications capable of inferring logical consequences from the knowledge considered [6]. However, the use of semantic technologies on mobile apps has not (yet) spread due, in part, to the fact that there are currently no remarkable efforts to enable mobile devices with semantic reasoning capabilities. Moreover, having local reasoners on the devices of the users would enable developers and mobile apps to manage knowledge even when network disconnections make impossible to rely on third-party devices/computers to carry out the reasoning. While authors in [11] have implemented a mobile reasoner from scratch which supports the Description Logic (DL) $\mathcal{ALCN}$ (they state that "... current Semantic Web reasoners cannot be ported without a significant re-write effort."); we have adopted another approach and we have started a survey to analyze if this re-writing effort is worth enough and how we can handle semantics in mobile applications by adapting existing DL reasoners.

In this paper, we evaluate whether the most popular among the current DL reasoners can be used directly in mobile devices, the efforts needed if they do not, and their performance with five well-know ontologies once we were able to make them work. We have focused on Android-based devices due to their spread, their openness, and the fact that it has a native virtual machine that is really close to Java (Dalvik). The existence of this Java-like virtual machine is really appealing in order to have important APIs working on the mobile devices, and thus allows reusing a lot of already developed code in new applications.

## 2 Reasoning on Android

Most of current popular semantic reasoners are implemented using Java (e.g., Pellet and HermiT) and are usually used along with semantic APIs (e.g., OWL API and Jena). Android, which is a Linux-based operating system whose middleware, libraries, and APIs are written in C, supports Java code as it uses a Java-like virtual machine called Dalvik [9]. In fact, Dalvik runs "dex-code" (Dalvik Executable), and Java bytecodes can be converted to Dalvik-compatible .dex files to be executed on Android. However, Dalvik does not align to Java SE and so it does not support Java ME classes, AWT or Swing. Thus, running semantic APIs and reasoners on Android could require some rewriting efforts.

### 2.1 Running Semantic APIs on Android

In the following we explain how to use two useful semantic APIs on Android.

**OWL API** [5] is an ontology API to manage OWL 2 ontologies in Java applications and provides a high-level way to interact with DL reasoners. It can be considered as a de facto standard, as the most recent versions of the DL reasoners use OWL API to load and process the ontologies.

We considered the last available version of the OWL API 3.4.3[3] which could be converted to Dalvik without any modifications and so, imported into an

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

Android project directly. However, we tried the OWL API 2.2.0 (automatically imported by Pellet along with the OWL API 3.2.4) but it uses Java classes that are not supported by Dalvik. Nevertheless, as new developers are expected to use the OWL API 3 they would not find any problems when importing the library.

**Jena** is an ontology API to manage OWL ontologies and handle RDF data in Java applications, but support for OWL 2 is not available yet. Jena reasoners are based on answering queries over RDF graphs.

Although Jena cannot be directly imported into an Android project, there exist a project called Androjena[4] to port it to the Android platform. The last version of Androjena 0.5, which was used in our tests, contains all the original code of Jena 2.6.2.

### 2.2 Running Reasoners on Android

In the following we present how to run some popular reasoners on Android (ordered incrementally according to the effort needed).

**JFact** is a port of FaCT++ to Java. FaCT++ reasoner [15], successor of Fact reasoner, is implemented in C++ and supports full OWL 2 with reasoning based on a tableaux algorithm.

We used JFact 0.9.1[5], which does not import any external libraries (except for the OWLAPI), and its code can be converted directly to Dalvik. In this way, we can develop an Android app that uses this reasoner by simply importing the JFact 0.9.1 and OWLAPI 3.4.3 .jar files in our Android project.

**CB** [7] reasoner is implemented in OCaml and supports a fragment of OWL 2 (Horn-$\mathcal{SHIF}$). Reasoning is based on a consequence-based procedure.

Android does not support the OCaml language natively but there exist some projects to develop OCaml interpreters for the platform. However, to run the reasoner on Android, we chose another approach: compiling CB build 6[6] to native Android code. The resulting native code can be executed on Android using the command line tool Android Debug Bridge (adb). To import this native code into an Android project we could use the Java Native Interface (JNI) and Android NDK (however, for the purpose of this paper we tested the native code directly).

**HermiT** [12] reasoner is implemented in Java and supports full OWL 2 and DL safe rules with reasoning based on a hypertableaux algorithm. It was the first DL reasoner that could classify some large ontologies thanks to a novel and efficient classification algorithm.

HermiT 1.3.6[7] cannot be converted directly to Dalvik as it references unsupported Java classes (both in its source code and in the external library JAutomata). Specifically, all the references to *java.awt.point* from both HermiT and

[4] https://code.google.com/p/androjena
[5] http://jfact.sourceforge.net
[6] https://code.google.com/p/cb-reasoner
[7] http://www.hermit-reasoner.com

JAutomata need to be changed or eliminated (they are used mainly for debugging, and Android does not support Java AWT as it has its own graphical libraries). In this way, we firstly eliminated the debug package of HermiT and its references, and reimplemented the methods of JAutomata that used these classes. However, in runtime, that version threw an error when loading ontologies with data properties due to a failure of Dalvik when unmarshalling the objects that the external library *dk.brics.automaton* serializes. To solve this problem, we reimplemented the marshalling/unmarshalling methods of these objects.

**Pellet** [13] reasoner is implemented in Java and supports full OWL 2 and DL safe rules with reasoning based on a tableaux algorithm. It was the first DL reasoner fully supporting OWL DL.

Pellet 2.3.0[8] (the last version available at the moment) cannot be converted directly to Dalvik. In this case, the reasoner uses three libraries that reference unsupported Java classes: Jena (which can be replaced by Androjena), OWL API 2.2.0 (which can be removed from the final compiled version), and JAXB (which uses the *javax.xml.bind* and the *Xerces* parser libraries not contained on Android). This problem can be solved by removing the JAXB .jar file and adding the source code of both *javax.xml.bind* and *Xerces* to our Android project. However, Dalvik has a limit of 65536 methods references per .dex file and it gets exceeded when applying this solution. To solve this, we removed the JAXB library and copied only the nine classes that Pellet needs from both the *java.xml.bind* package and the Xerces library to our Android project.

**Other Reasoners.** We also tried to load other reasoners on Android without success, namely RacerPro [4], KAON2 [8], QUEST [10], TrOWL [14], and fuzzyDL [2]. None of them can be directly converted to Dalvik because of their references to unsupported Java classes (similarly to what happened in HermiT and Pellet), and their source code was not publicly available. We identified that these reasoners use some problematic libraries that cannot be run on Android: Jena (QUEST, TrOWL), Java RMI (KAON2), Xerces (QUEST), Gurobi (fuzzyDL), etc. Both Jena and RMI could be replaced by projects that port these libraries to the Android platform (such as LipeRMI[9] or the aforementioned Androjena), and Xerces could be addressed as explained for Pellet. Finally, we have not found a replacement for Gurobi.

## 3 Experimental Evaluation

The main goal of this paper is to analyze whether current available DL reasoners can be used on Android. Hence, we considered also interesting to test their behavior on current devices. In this way, we tested the analyzed reasoners with five well-known ontologies (see Table 1): *Pizza*[10] and *Wine*[11], which are two

[8] http://clarkparsia.com/pellet
[9] http://lipermi.sourceforge.net
[10] http://www.co-ode.org/ontologies/pizza/pizza.owl
[11] http://www.w3.org/TR/owl-guide/wine.rdf

expressive ontologies; *DBpedia* 3.8[12] (T-BOX), which can be useful for mobile apps developers to access the structured content of DBPedia (a semantic entry point to Wikipedia) [1]; and the Gene Ontology (*GO*) and the US National Cancer Institute (*NCI*), which contain a high number of concepts.

**Table 1.** Selected ontologies for the tests.

| | DL Expressivity | Horn | $\|N_{LA}\|$ | $\|N_R\|$ | $\|N_C\|$ | $\|N_I\|$ |
|---|---|---|---|---|---|---|
| Pizza | $\mathcal{SHOIN}$ | × | 714 | 8 | 100 | 5 |
| Wine | $\mathcal{SHOIN}(\mathcal{D})$ | × | 950 | 17 | 138 | 206 |
| DBpedia | $\mathcal{ALF}(\mathcal{D})$ | ✓ | 3542 | 1894 | 436 | 0 |
| GO | $\mathcal{ALE}+$ | ✓ | 28897 | 1 | 20465 | 0 |
| NCI | $\mathcal{ALE}$ | ✓ | 46940 | 70 | 27652 | 0 |

Horn (✓): the ontology does not contain "non-deterministic" constructors.
$|N_{LA}|$: number of logical axioms; $|N_R|$: number of roles; $|N_C|$: number of concepts; $|N_I|$: number of individuals

For each ontology we tested the classification performance (i.e., time needed to compute the class subsumption hierarchy) of each reasoner on two devices with different Android versions. We also performed the tests on a PC to determine how slow is reasoning on Android compared to this baseline (taking into account that the PC hardware overperforms Android devices and their virtual machines are optimized for different purposes). The results obtained after performing 10 tests for each reasoner and device are shown in Table 2.

First, we want to highlight that the Galaxy Nexus with Android 4.2.1 (Android2) overperformed the Samsung Galaxy Tab with Android 2.3.3 (Android1) by 30% in all the tests. All the reasoners running on the Android 4.2.1 device were able to classify all the ontologies except *DBpedia* (which contains unsupported temporal data type properties for JFact –$gYear$–), and *NCI* (where JFact and Pellet ran out of memory). Notice that most of the reasoners running on the Android 2.3.3 device (JFact, HermiT, and Pellet) were unable to classify *GO* and *NCI* because of a memory constraint. Android 2.3 and earlier versions usually provide a maximum heap size limit per application of 64MB while later versions usually provide a maximum heap size of 256MB by using the *android:largeHeap="true"* attribute for the manifest of the application (the actual maximum size limit depends on the specific device). CB was able to classify all the ontologies but the reasoning for *Pizza* and *Wine* was incomplete because they are not Horn. The reasoning of CB on *DBpedia* was complete even when the profile of the ontology is not fully supported by the reasoner. In addition, CB does not face the same memory restriction than other reasoners as it was compiled from C code and runs outside Dalvik.

In summary, the major issue that reasoning on Android faces currently is the limited memory of smartphones/tablets (especially for apps running on Dalvik). This limitation affects especially when using large ontologies. Finally, as ex-

[12] http://dbpedia.org/Ontology

**Table 2.** Comparison of classification time for PC and Android in seconds.

| | | JFact | CB | HermiT | Pellet |
|---|---|---|---|---|---|
| Pizza | PC | 0.37 | $0^{\diamond}$ | 0.57 | 0.97 |
| | Android1 | 4.90 | $0^{\diamond}$ | 14.88 | 33.22 |
| | Android2 | 3.42 | $0^{\diamond}$ | 10.43 | 20.77 |
| Wine | PC | 10.39 | $0^{\diamond}$ | 6.54 | 2.22 |
| | Android1 | 2196.05 | $0^{\diamond}$ | 511.97 | 194.12 |
| | Android2 | 1609.32 | $0^{\diamond}$ | 361.38 | 131.80 |
| DBpedia | PC | UDT! | 0 | 0.10 | 1.39 |
| | Android1 | UDT! | 0 | 8.87 | 115.30 |
| | Android2 | UDT! | 0 | 5.13 | 63.15 |
| GO | PC | 7.77 | 0.11 | 1.56 | 1.96 |
| | Android1 | OOM! | 1.95 | OOM! | OOM! |
| | Android2 | 435.60 | 1.47 | 487.98 | 83.97 |
| NCI | PC | 2.61 | 0.24 | 2.23 | 4.24 |
| | Android1 | OOM! | 3.31 | OOM! | OOM! |
| | Android2 | OOM! | 2.69 | 2020.48 | OOM! |

*PC*: Windows 64-bits, i5-2320 3.00GHz, 16GB RAM; *Android1*: Samsung Galaxy Tab, 1.0GHz, 512MB RAM, Android 2.3.3; *Android2*: Galaxy Nexus, 1.2GHz dual-core, 1GB RAM, Android 4.2.1
0: time below 0.005s; $^{\diamond}$: incomplete reasoning; *OOM!*: Out of Memory; *UDT!*: Unsupported Data Type

pected, we observe that reasoning on Android is slower than reasoning on a PC but nevertheless the results show that this is feasible.

## 4 Conclusions and Future Work

The emergence of mobile computing in our daily lives allows to consider new applications where semantic technologies could be useful. However, some efforts are needed to enable developers to use ontologies and ontology reasoning in their mobile apps. In this paper, we shown that current Android devices could be able to use most of the semantic reasoners although they need some manual work due to unsupported Java libraries and classes for the virtual machine of Android (Dalvik). Once this issue has been addressed, the main limitation that reasoners will face on current smartphones/tablets concerns memory usage and processing requirements. However, we noticed an increment of 30% on the performance of the reasoners between two Android devices (a Samsung Galaxy Tab and a Google Galaxy Nexus –introduced on 2010 and 2011, respectively–) which could show the future trend. As future work we plan to further test current semantic reasoners on Android measuring other important aspects for mobile computing such as memory and battery usage.

*Acknowledgments*: This research work has been supported by the CICYT project TIN2010-21387-C02 and DGA-FSE.

## References


1. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.
2. F. Bobillo and U. Straccia. fuzzyDL: An expressive fuzzy description logic reasoner. In *Proceedings of the International Conference on Fuzzy Systems (FUZZ-IEEE 2008)*, 2008.
3. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
4. V. Haarslev and R. Möller. RACER system description. In *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR 2001)*, 2001.
5. M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web Journal*, 2(1):11–21, 2011.
6. S. Ilarri, A. lllarramendi, E. Mena, and A. Sheth. Semantics in location-based services – guest editors' introduction for special issue. *IEEE Internet Computing*, 15(6):10–14, 2011.
7. Y. Kazakov. Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In *Proceedings of the 21st International Joint Conference on Artificial intelligence (IJCAI 2009)*, 2009.
8. B. Motik and R. Studer. KAON2–a scalable reasoning tool for the semantic web. In *Proceedings of the 2nd European Semantic Web Conference (ESWC 2005)*, 2005.
9. H.-S. Oh, B.-J. Kim, H.-K. Choi, and S.-M. Moon. Evaluation of Android Dalvik virtual machine. In *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2012)*, 2012.
10. M. Rodriguez-Muro and D. Calvanese. Quest, an OWL 2 QL reasoner for ontology-based data access. In *Proceedings of the 9th International Workshop on OWL: Experiences and Directions (OWLED 2012)*, 2012.
11. M. Ruta, F. Scioscia, G. Loseto, F. Gramegna, and E. D. Sciascio. A mobile reasoner for semantic-based matchmaking. In *Proceedings of the 6th International Conference on Web Reasoning and Rule Systems (RR 2012)*, 2012.
12. R. Shearer, B. Motik, and I. Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*, 2008.
13. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.
14. E. Thomas, J. Z. Pan, and Y. Ren. TrOWL: Tractable OWL 2 reasoning infrastructure. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*, 2010.
15. D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: system description. In *Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR 2006)*, 2006.
16. R. Yus, E. Mena, S. Ilarri, and A. Illarramendi. SHERLOCK: A system for location-based services in wireless environments using semantics. In *22nd International World Wide Web Conference (WWW 2013)*, 2013.

# FRaQuE: A Framework for Rapid Query Processing Evaluation

Jean-Rémi Bourguet and Luca Pulina

POLCOMING, Università di Sassari, Italy
Viale Mancini 5 – 07100, Sassari – Italy
boremi@uniss.it - lpulina@uniss.it

**Abstract.** In this paper we present FRaQuE (**F**ramework for **Ra**pid **Qu**ery Processing **E**valuation). The main purpose of FRaQuE is to offer to a non-expert user a "push-button" solution aimed to help her to evaluate query processors for Ontology Based Data Access, focusing only on input and output data, without take into account both theoretical and technical issues.

## 1 Introduction

The choice of W3C to make ontologies the main tool to attach semantic information to web contents, and, consequently, the potential applications in the Semantic Web [2], has attracted a lot of interest inside the Automated Reasoning community, particularly in the past decade. It is well-established that reasoning with ontologies is one of the core task of research in description logics – see, e.g., [1] – and it is also witnessed by the large amount of reasoners currently available[1].

One of the reasoning tasks that can be accomplished by reasoning tools is query answering. In particular, in order to match the competing requirements of KR&R-style data handling with DB-style data size, research on ontology-based data access (OBDA) emerged at the crossroads of the two fields. According to [7], the keyword OBDA characterizes scenarios where access to data is mediated by an ontology, and data is either physically stored in a traditional DB, or comes in sizes which are typical of DB applications anyway. To this purpose, there are actually several OWL reasoners with the ability to support the SPARQL query language [18] – the W3C standard for querying semantic-enabled data stores.

Given the wide range of possible practical applications in which OBDA can be used, e.g., Decision Support Systems [8, 5, 3], practitioners aimed to leverage OBDA in their applications have to answer the question "Which query processor should I use?". In order to answer to this question, recently several events and projects have been implemented, e.g., the Joint Workshop on Scalable and High-Performance Semantic Web Systems [11] and the SEALS project `http://www.seals-project.eu`. More, the OWL Reasoner Evaluation

[1] See, e.g., `http://www.w3.org/2007/OWL/wiki/Implementations` or `http://www.cs.man.ac.uk/~sattler/reasoners.html` for a list

workshops organizes since 2012 a competitive event, in the same spirit of other Automated Reasoning communities, e.g., the CADE ATP System Competition (CASC) [24] for theorem provers in first order logic, the QBF Evaluation [17] for quantified Boolean formulas solvers, and the ASP Competition [6] for Answer Set Programming solvers. Also if in such kind of events reasoners are evaluated using transparent and fair methods, in a practical application context a practitioner could be interested to understand the current state of the art related to a particular problem or another for which data could not be available to the research community. This can lead a non-expert user to deal with several issues, both technical and theoretical.

In this paper we present FRaQuE (**F**ramework for **Ra**pid **Qu**ery Processing **E**valuation), a framework aimed to offer to a non-expert user the possibility to evaluate query processors for OBDA.The main purpose of FRaQuE is to offer to the user a "push-button" solution aimed to help the user to answer to the question above, focusing only on input data and queries at the user execution stage, and showing data in order to evaluate both correctness and performance at the user validation stage. Currently we include in FRaQuE research prototypes that can be considered active OBDA projects as soon as systems like Pellet [23], a full-fledged commercial description logic reasoner and ARQ [21], the built-in query processor of the Jena library. An important element in the selection is the ability to support the SPARQL query language [18]. However, the FRaQuE architecture is modular, allowing the extension to other reasoners in an easy way, as we will describe in Section 2, where we will detail both design and implementation of FRaQuE. About the rest of the paper, in Section 3 we discuss about some open points coming from the usage of the query processors, and we conclude in Section 4 with some final remarks.

## 2 Design and implementation of FRaQuE

Figure 1 presents the architecture of FRaQuE[2]. Because of, given a knowledge base $\mathcal{K}$ and a query $\alpha$, the goal of FRaQuE is to run different systems on the task of query answering, we also refer to the OBDA systems as *query processors*. Looking at the figure, we can see that FRaQuE is composed of the four modules described in the following.

`INTERFACE` manages both the input received by the user and the output of the whole system. It also dispatches the input data to both `QUERY MANAGER` and `ONTOLOGY MANAGER`, as denoted by the outgoing arrows. In particular, `INTERFACE` collects (*i*) the name of the query processor to fire; (*ii*) the TBox file name in RDF/XML or OWL/XML format; (*iii*) the ABox file name in RDF/XML or OWL/XML format; and (*iv*) a text file containing the query in SPARQL 1.0. Finally, `INTERFACE` manages the output received from `REASONER MANAGER`, in order to present it to the user. Actually, FRaQuE

[2] FRaQuE is available for download at `http://sites.google.com/site/ore2013fraque`.

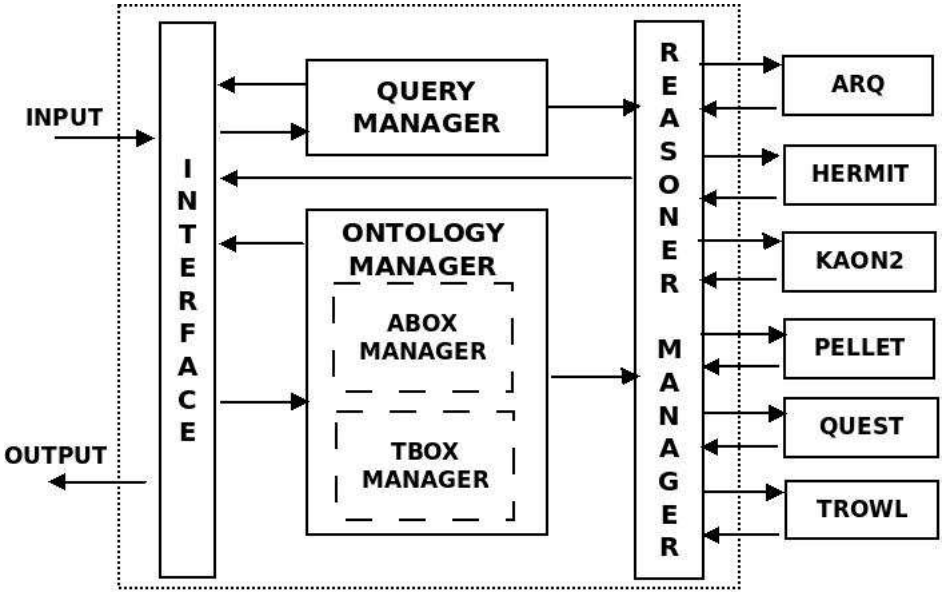



Fig. 1: The architecture of FRaQuE. The dotted box denotes the whole system and, inside it, each solid box represents its modules, while each dashed box represents a sub-module. Arrows denote functional connections between modules.

outputs the ontology loading CPU time, the query answering CPU time, and a text file containing the query result.

`QUERY MANAGER` is devoted to process the query input file received by `INTERFACE`. It checks the compliance of the query with the SPARQL 1.0 syntax, and, considering the query processor passed by `INTERFACE`, it applies syntactic modification to the input query file or returns to `INTERFACE` an error message if the input query is not supported by the selected query processor (see Section 3 for details).

`ONTOLOGY MANAGER` is devoted to manage both TBox and ABox input file, by means of sub-modules `TBOX MANAGER` and `ABOX MANAGER`, respectively.

`REASONER MANAGER` manages the interaction with the reasoners. It receives from `INTERFACE` information about the engine to fire, while it receives from `ONTOLOGY MANAGER` and `QUERY MANAGER` information about the ontology file and the query to process, respectively. At the end of the query processing, `REASONER MANAGER` returns to `INTERFACE` the result.

Concerning the modules described above, we can consider `REASONER MANAGER` as the core of FRaQuE, because it interacts with different query processors in a transparent way to the user. In particular, SPARQL expression semantics can change according to different *entailment regimes*. In our case, there are two entailment regimes which are relevant, namely the *RDFS entailment* and

the *OWL-DL entailment*. Under the second regime, we focus on the notion of entailment for the semantics of OWL 2 QL. In particular, the OWL 2 QL profile is described in the official W3C's recommendation as "*[the sub-language of OWL 2] aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task.*". Given our intended applications, we consider knowledge bases encoded using OWL 2 QL.

In details, the query processors actually included in FRaQuE are the following.

- ARQ [21] (version 2.9.4) is the built-in query processor of the Jena library. It processes queries according to the RDFS regime.
- HermiT [22] (version 1.3.6) is a DL reasoner based on hypertableau calculus [15]. It can be used to answer sparql1.0 queries by means of the sparql1.0 wrapper owl-bgp [10, 13]. For the sake of simplicity, in the following we will mention the composition between the reasoner and the wrapper simply as "HermiT".
- kaon2 [12, 14] (version 2008-06-29) implements reasoning algorithms aimed to reduce a knowledge base to a disjunctive datalog program, allowing the usage of deductive database techniques. So, with respect to other DL reasoners like HermiT, Pellet and TrOWL, it does not implement a tableau-like calculus. Queries can be formulated using a specific subset of sparql1.0 syntax – see `http://kaon2.semanticweb.org/` for details.
- Pellet [23] (version 2.3.0) is a description logic reasoner accepting sparql1.0 queries. As such, it supports OWL-DL regime and it could be used to perform logically-aware queries also on full-fledged OWL 2 knowledge bases.
- Quest [20] (version 1.7) is a reasoner that supports the OWL-DL entailment regime restricted to OWL 2 QL. Quest converts queries over the knowledge base into equivalent SQL queries over an internal relational database. In particular, Quest uses h2 (version 1.3)[3] and while the schema used internally to store triples is similar to the standard $\langle S, P, O\rangle$ schema, it is optimized to generate very small SQL queries even if there are big hierarchies in the ontology, as described in [19].
- TrOWL [25] (version 1.1) is an infrastructure aimed to reasoning, and querying OWL 2 ontologies by means of several techniques, e.g., quality guaranteed approximations and forgetting. In general, considering OWL 2 ontologies, TrOWL could not give complete answers, but it should not be the case considering OWL 2 QL ontologies, as in our case (see [25] for details). In FRaQuE we include TrOWL with the Jena library.

Finally, in Figure 2, we present the class diagram of FRaQuE. In the diagram, we denote Java classes with boxes, "part of" relationship with hollow diamond shape arrows, while the inheritance relationship is denoted using ordinary arrows. The upper part of boxes holds the name of the class, the middle part contains the attributes and the bottom part contains methods, "+", "-" and "#" are respectively public, private and protected attributes or methods.

[3] H2 Database Engine `http://www.h2database.com/html/main.html`.

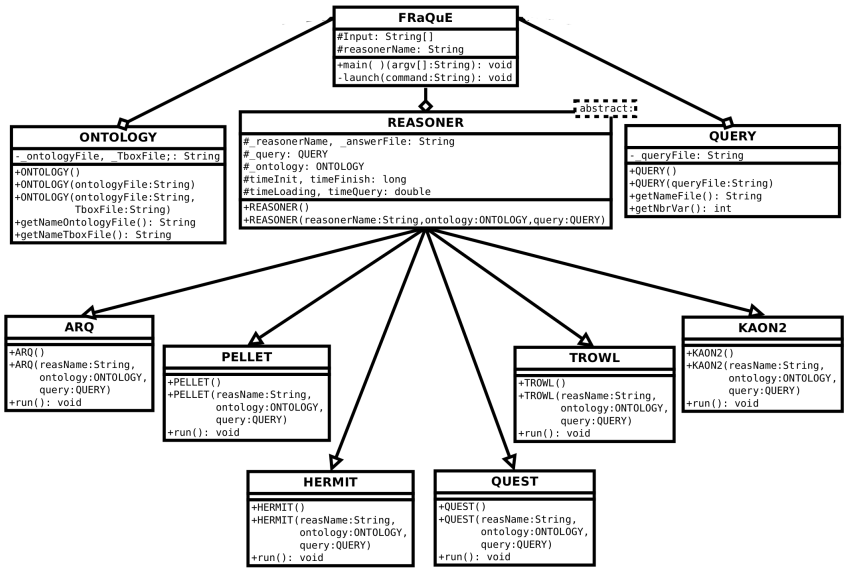



Fig. 2: Class diagram of FRaQuE.

## 3 Discussion

The main purpose of FRaQuE is to offer to a non-expert user a simple platform to evaluate both correctness and reasoning times of OBDA query processors. With this aim, about the evaluation of correctness of the answer, a text file containing the answer to the input query is produced by FRaQuE at the end of query processing – as mentioned in the description of the `INTERFACE` module. Concerning the reasoning time, we consider values that could be easily evaluated by a non-expert user aimed to roughly compare the CPU time needed to answer a query, avoiding technical details concerning different strategies implemented in a reasoner.

Considering query processors currently included in FRaQuE, we list in the following some technical issues. We report that queries containing some keywords, e.g., `FILTER` and `OPTIONAL`, are actually not supported by both KAON2 and Quest. About the query processors mentioned above, we also report that they do not allow the usage of variables after a `rdf:type` predicate. Some other features in the SPARQL syntax can lead to a failure during the query loading. In Quest, comment lines (starting with #) have to be removed, while in KAON2

absence of a white space between an object and the final dot is not allowed, while it is allowed between object and semi-colon[4].

The modular architecture of FRaQuE allows the user to easily extend the pool of query processors. Looking at Figure 2, we can see that all query processors currently available in FRaQuE are wrapped in a Java class derived from the abstract class REASONER. Such abstract class provides a common interface to manage input (ontology and query) and output (the query answer) files. Once implemented the derived class related to a new query processor – see the source code available at `http://sites.google.com/site/ore2013fraque` for an example – it is sufficient to update the code of FRaQuE* files.

Finally, we report that a preliminary version of FRaQuE has been used for the experimental evaluation in [4].

## 4 Conclusions and Future Works

In this paper, we presented the design and the implementation of FRaQuE. Our modular framework can allow to a non-expert user a rapid evaluation of the state of the art concerning query processors for OBDA.

Currently, we are working to extend FRaQuE in several directions. Firstly, we are extending the architecture in order to integrate query processors using rewriting-based techniques, e.g., CLIPPER [9] and REQUIEM [16]. Secondly, we are considering the usage of SPARQL 1.1 as input query format. This is mainly motivated by the fact that our tool aims to simplify practitioner's work, and the usage of operators like COUNT, MIN, MAX, or SUM could simplify the query formulation. Actually, ARQ is the only system supporting SPARQL 1.1 natively, so we are working on `QUERY MANAGER` in order to add a translator able to convert the input query to SPARQL 1.0 and use some JAVA code to replicate the behaviour of the operators above. Finally, we are designing a GUI version of `INTERFACE`.

*Acknowledgments* The authors are grateful to the reviewers for their valuable comments and suggestions for improving the paper. The authors would like to thank Giuseppe Cicala and Armando Tacchella for fruitful discussion, and Mariano Rodriguez-Muro for his help in using Quest.

This work is supported by Regione Autonoma della Sardegna e Autorità Portuale di Cagliari con L.R. 7/2007, Tender 16 2011, CRP-49656 "Metodi innovativi per il supporto alle decisioni riguardanti lottimizzazione delle attività in un terminal container"

[4] Notice that the usage of white spaces aiming to separate two terminals is a W3C recommendation [18].

## References


1. F. Baader. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
2. Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.
3. Eva Blomqvist. The use of semantic web technologies for decision support – a survey. *Semantic Web*, 2012.
4. Jean-Remi Bourguet, Giuseppe Cicala, Luca Pulina, and Armando Tacchella. An experimental evaluation of tools for ontology-based data access. In *Proceedings of the 20th RCRA International Workshop on Experimental Evaluation of Algorithms for solving problems with combinatorial explosion*, 2013. Avalaible on-line from `https://docs.google.com/file/d/0B8dEUbPKR1laYjFnMUJLcks0ZnM/edit?usp=sharing`.
5. Jean-Remi Bourguet, Giuseppe Cicala, Luca Pulina, and Armando Tacchella. Obda and intermodal logistics: Active projects and applications. In *Web Reasoning and Rule Systems (RR) 2013*, volume 7994 of *LNCS*, pages 210–215. Springer Verlag, 2013.
6. Francesco Calimeri, Giovambattista Ianni, Francesco Ricca, Mario Alviano, Annamaria Bria, Gelsomina Catalano, Susanna Cozza, Wolfgang Faber, Onofrio Febbraro, Nicola Leone, et al. The third answer set programming competition: Preliminary report of the system competition track. In *Logic Programming and Nonmonotonic Reasoning*, pages 388–403. Springer, 2011.
7. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati. Ontologies and Databases: The *DL-Lite* approach. *Reasoning Web. Semantic Technologies for Information Systems*, pages 255–356, 2009.
8. Matteo Casu, Giuseppe Cicala, and Armando Tacchella. Ontology-based data access: An application to intermodal logistics. *Information Systems Frontiers*, 2012.
9. Thomas Eiter, Magdalena Ortiz, M Simkus, Trung-Kien Tran, and Guohui Xiao. Towards practical query answering for horn-shiq. *Description Logics*, 846, 2012.
10. Birte Glimm et al. OWL-BGP – A framework for parsing SPARQL basic graph patterns (BGPs) into an OWL object representation. `http://code.google.com/p/owl-bgp`.
11. Achille Fokoue, Thorsten Liebig, Eric Goodman, Jesse Weaver, Jacopo Urbani, and David Mizell. Joint workshop on scalable and high-performance semantic web systems (ssws+ hpcsw 2012). 2012.
12. Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing shiq- description logic to disjunctive datalog programs. *Proc. KR*, 4:152–162, 2004.
13. Ilianna Kollia, Birte Glimm, and Ian Horrocks. Sparql query answering over owl ontologies. In *The Semantic Web: Research and Applications*, pages 382–396. Springer, 2011.
14. Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for owl-dl with rules. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, 2005.
15. Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. *Automated Deduction–CADE-21*, pages 67–83, 2007.
16. Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Efficient query answering for owl 2. In *The Semantic Web-ISWC 2009*, pages 489–504. Springer, 2009.

17. Claudia Peschiera, Luca Pulina, Armando Tacchella, Uwe Bubeck, Oliver Kullmann, and Inês Lynce. The seventh qbf solvers evaluation (qbfeval10). In *Theory and Applications of Satisfiability Testing–SAT 2010*, pages 237–250. Springer, 2010.
18. E. Prud'Hommeaux and A. Seaborne. SPARQL Query Language for RDF. *W3C working draft*, 4(January), 2008.
19. M. Rodriguez-Muro and D. Calvanese. High Performance Query Answering over DL-Lite Ontologies. In *Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012)*, pages 308–318, 2012.
20. M. Rodrıguez-Muro and D. Calvanese. Quest, an OWL 2 QL Reasoner for Ontology-based Data Access. *OWLED 2012*, 2012.
21. A. Seaborne. ARQ – A SPARQL Processor for Jena, 2010. `http://jena.sourceforge.net/ARQ/` – [accessed 1/5/2010].
22. Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*, pages 26–27, 2008.
23. E. Sirin, B. Parsia, B. Cuenca-Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007. Available on-line from `http://pellet.owldl.com/`.
24. Geoff Sutcliffe. The cade-23 automated theorem proving system competition–casc-23. *AI Communications*, 25(1):49–63, 2012.
25. Edward Thomas, Jeff Z. Pan, and Yuan Ren. TrOWL: Tractable OWL 2 Reasoning Infrastructure. In *the Proc. of the Extended Semantic Web Conference (ESWC2010)*, 2010.

# MORe: a Modular OWL Reasoner for Ontology Classification

Ana Armas Romero, Bernardo Cuenca Grau,
Ian Horrocks, Ernesto Jiménez-Ruiz

Department of Computer Science, University of Oxford

**Abstract.** MORe exploits module extraction techniques to divide the workload of ontology classification between two reasoners: a reasoner for the lightweight profile EL of OWL 2, and a fully fledged OWL 2 reasoner. This division is carried out in such a way that the bulk of the workload is assigned, as much as possible, to the OWL 2 EL reasoner, in order to exploit the more efficient classification techniques specific to this profile.

## 1 Introduction

MORe [1] is an OWL 2 reasoning system dedicated to ontology classification that integrates a general purpose OWL 2 reasoner (OWL reasoner for short) and a reasoner specific for the OWL 2 EL[1] profile (EL reasoner for short). The current implementation of MORe uses ELK [8] as its EL reasoner, and offers the possibility to choose between two OWL reasoners: HermiT 1.3.7 [4] and Pellet 2.3.0 [12]. The EL and OWL reasoners are, however, integrated in a "black-box" way: our implementation of MORe provides the required infrastructure to bundle any other OWL and/or EL reasoner.

Given an input ontology $\mathcal{O}$, MORe identifies a part of the classification of $\mathcal{O}$ that can be computed using the EL reasoner and limits the use of the OWL reasoner to a fragment of $\mathcal{O}$ as restricted as possible. The main advantage of MORe lies in its "pay-as-you-go" behaviour when an OWL 2 EL ontology is extended with axioms in a more expressive logic: the use of an efficient EL reasoner is not necessarily precluded by the extension; in fact, it is to be expected that the EL reasoner will still perform most of the computational work.

MORe performs only terminological reasoning and ignores any assertional axioms that the input ontology might contain. Therefore, completeness is only guaranteed for ontologies that contain no ABox assertions.

MORe[2] is implemented in Java using the OWL API[3] [6]. It can therefore process ontologies in any format handled by the OWL API, such as RDF/XML, OWL Functional Syntax, or OBO. It is available both as a Java library and a Protégé[4] plugin, and it can also be used via a commmand line interface.

[1] http://www.w3.org/TR/owl2-profiles/#OWL_2_EL
[2] https://code.google.com/p/more-reasoner/
[3] http://owlapi.sourceforge.net/
[4] http://protege.stanford.edu/

## 2 The Technique

The main idea behind the technique implemented in MORe is to identify, given an ontology $\mathcal{O}$ with signature $\mathsf{Sig}(\mathcal{O})$, two subsets $\mathcal{M}_1$, $\mathcal{M}_2$ of $\mathcal{O}$ such that

- $\mathcal{M}_1$ is as small as possible;
- the output of classifying $\mathcal{M}_2$ with the EL reasoner is *complete* for $\mathsf{Sig}(\mathcal{M}_2)$ w.r.t. $\mathcal{O}$ (i.e. it contains all subsumption relations $\mathsf{A} \sqsubseteq \mathsf{B}$ entailed by $\mathcal{O}$ such that $\mathsf{A} \in \mathsf{Sig}(\mathcal{M}_2)$);
- the output of classifying $\mathcal{M}_1$ with the OWL reasoner is *complete* for $\mathsf{Sig}(\mathcal{M}_1)$ w.r.t. $\mathcal{O}$; and
- $\mathsf{Sig}(\mathcal{M}_1) \cup \mathsf{Sig}(\mathcal{M}_2) = \mathsf{Sig}(\mathcal{O})$.

Our implementation of MORe relies on ELK, which does not yet implement the whole of OWL 2 EL. The unsupported constructs are documented and hence we can identify the fragment $\mathcal{L}_{\text{ELK}}$ of OWL 2 EL implemented by ELK.

They key to identifying $\mathcal{M}_1$ and $\mathcal{M}_2$ is in computing an $\mathcal{L}_{ELK}$-*signature*: a signature $\Sigma^{\text{ELK}} \subseteq \mathsf{Sig}(\mathcal{O})$ such that the $\bot$-*module* for $\Sigma^{\text{ELK}}$ in $\mathcal{O}$ is an ontology in the language $\mathcal{L}_{\text{ELK}}$ for which ELK is complete.

The $\bot$-module for $\mathcal{O}$ and $\Sigma$, $\mathcal{M}_{[\mathcal{O},\Sigma]}$, is the smallest subset of $\mathcal{O}$ such that all axioms in $\mathcal{O} \setminus \mathcal{M}_{[\mathcal{O},\Sigma]}$ are $\bot$-local w.r.t. $\Sigma \cup \mathsf{Sig}(\mathcal{M}_{[\mathcal{O},\Sigma]})$. Intuitively, an axiom $\alpha$ is $\bot$-local w.r.t. $\Sigma$ if replacing by $\bot$ all occurrences in $\alpha$ of symbols not in $\Sigma$ would turn $\alpha$ into a syntactically recognisable tautology; e.g., the axiom $\mathsf{A} \sqsubseteq \mathsf{B}$ is $\bot$-local w.r.t. $\Sigma = \{\mathsf{B}\}$. Cuenca Grau et al. [2] offer a deeper insight into the notions of $\bot$-module, $\bot$-locality, and modularity in a more general sense. For the scope of this system description, we only remark the following properties:

1. For any class $A$ in $\mathsf{Sig}(\mathcal{M}_{[\mathcal{O},\Sigma]})$:
   (a) if $A$ is unsatisfiable in $\mathcal{O}$ then it is also unsatisfiable in $\mathcal{M}_{[\mathcal{O},\Sigma]}$
   (b) if another class $B$ in $\mathsf{Sig}(\mathcal{O})$ is a superclass of $A$ in $\mathcal{O}$, then it is so in $\mathcal{M}_{[\mathcal{O},\Sigma]}$ as well —and so $B$ is in $\mathsf{Sig}(\mathcal{M}_{[\mathcal{O},\Sigma]})$ too.
2. If $\Sigma_1 \subseteq \Sigma_2$, then if some axiom $\alpha$ is $\bot$-local w.r.t. $\Sigma_2$, it is also $\bot$-local w.r.t. $\Sigma_1$, and therefore $\mathcal{M}_{[\mathcal{O},\Sigma_1]} \subseteq \mathcal{M}_{[\mathcal{O},\Sigma_2]}$.
3. both checking $\bot$-locality and extracting a $\bot$-module can be done in polynomial time.

Property 1(b), in particular, is not shared by other kinds of modules, and makes $\bot$-modules especially well suited for classification purposes.

### 2.1 Modular Combination of Reasoners

The integration of the two reasoners is performed as follows. Given an OWL 2 ontology $\mathcal{O}$, MORe first tries to compute a nonempty $\mathcal{L}_{\text{ELK}}$-signature $\Sigma^{\text{ELK}}$ for $\mathcal{O}$ (details of how this is done are given in Section 2.2). If it suceeds, then ELK is used to classify $\mathcal{M}_{[\mathcal{O},\Sigma^{\text{ELK}}]}$, and HermiT or Pellet to classify $\mathcal{M}_{[\mathcal{O},\mathsf{Sig}(\mathcal{O})\setminus\Sigma^{\text{ELK}}]}$; finally, both partial hierarchies are unified into a single one. If MORe fails to find a nonempty $\mathcal{L}_{\text{ELK}}$-signature, then it delegates the whole classification to either HermiT or Pellet. Details about the correctness (soundness and completeness) of this technique can be found in Armas Romero et al. [1].

### 2.2 Computing an $\mathcal{L}_{\text{ELK}}$-signature

To find a suitable $\mathcal{L}_{\text{ELK}}$-signature $\Sigma^{\text{ELK}}$ for a given ontology $\mathcal{O}$, MORe first identifies the set $\mathcal{S}$ of axioms that ELK cannot process, and —if possible— a subset $\Sigma$ of $\mathsf{Sig}(\mathcal{O})$ such that all the axioms in $\mathcal{S}$ are $\bot$-local w.r.t. $\Sigma$. This alone, however, does not guarantee that $\mathcal{M}_{[\mathcal{O},\Sigma]} \cap \mathcal{S} = \emptyset$.

*Example 1.* Consider the ontology $\mathcal{O}_{\mathsf{ex}}$ consisting of the following axioms:

$$\mathsf{A} \equiv \mathsf{B} \sqcup \mathsf{C} \qquad \mathsf{B} \equiv \mathsf{D} \sqcap \exists \mathsf{R}.\mathsf{E} \qquad \mathsf{F} \sqsubseteq \exists \mathsf{R}.\mathsf{G}$$

All the axioms in $\mathcal{O}_{\mathsf{ex}}$ are in $\mathcal{L}_{\text{ELK}}$ except for $\alpha = \mathsf{A} \equiv \mathsf{B} \sqcup \mathsf{C}$. Now, we have that $\alpha$ is $\bot$-local w.r.t. a signature $\Sigma$ iff $\Sigma \cap \{\mathsf{A}, \mathsf{B}, \mathsf{C}\} = \emptyset$, therefore, $\alpha$ is $\bot$-local w.r.t. $\Sigma = \mathsf{Sig}(\mathcal{O}_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$. However, $\beta = \mathsf{B} \equiv \mathsf{D} \sqcap \exists \mathsf{R}.\mathsf{E}$ is not $\bot$-local w.r.t. $\Sigma$, so $\beta \in \mathcal{M}_{[\mathcal{O},\Sigma]}$ and $\mathsf{B} \in \mathsf{Sig}(\mathcal{M}_{[\mathcal{O},\Sigma]})$, and therefore $\alpha$ is not $\bot$-local w.r.t. $\Sigma \cup \mathsf{Sig}(\mathcal{M}_{[\mathcal{O},\Sigma]})$ and needs to be in $\mathcal{M}_{[\mathcal{O},\Sigma]}$. ♢

All we need to do is progressively reduce $\Sigma$ until $\mathsf{Sig}(\mathcal{M}_{[\mathcal{O},\Sigma]}) \subseteq \Sigma$. This can be done as follows:

1. Let $\mathcal{S}_0$ be the set of axioms in $\mathcal{O}$ that are not in $\mathcal{L}_{\text{ELK}}$ and let $\Sigma_0 = \mathsf{Sig}(\mathcal{O})$.
2. Reduce $\Sigma_0$ to some $\Sigma_1 \subset \Sigma_0$ such that $\mathcal{S}_0$ is $\bot$-local w.r.t. $\Sigma_1$. If this is not possible, then make $\Sigma_1 = \emptyset$.
3. Compute the set $\mathcal{S}_1$ of axioms in $\mathcal{M}_{[\mathcal{O},\Sigma_1]}$ containing symbols outside $\Sigma_1$.
4. Repeat Steps 2–3 until $\mathcal{S}_i = \emptyset$ (i.e. until $\mathsf{Sig}(\mathcal{M}_{[\mathcal{O},\Sigma_i]}) \subseteq \Sigma_i$) or $\Sigma_i = \emptyset$.

It is important to note that, in some cases, there may be several different ways of obtaining $\Sigma_{i+1}$ from $\Sigma_i$.

*Example 2.* As shown in the previous example, taking $\Sigma = \mathsf{Sig}(\mathcal{O}_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$ is not enough to keep $\mathsf{A} \equiv \mathsf{B} \sqcup \mathsf{C}$ outside $\mathcal{M}_{[\mathcal{O},\Sigma]}$. We need to remove more symbols from $\Sigma$ to keep $\mathsf{B} \equiv \mathsf{D} \sqcap \exists \mathsf{R}.\mathsf{E}$ outside $\mathcal{M}_{[\mathcal{O},\Sigma]}$ too. One possibility would be to remove $\mathsf{D}$ from $\Sigma$, but we could also choose to remove $\mathsf{R}$ or $\mathsf{E}$ instead. It turns out that choosing one option over another can change things substantially.

If we chose to take $\Sigma_1 = \mathsf{Sig}(\mathcal{O}_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{R}\}$ then, because $\mathsf{F} \sqsubseteq \exists \mathsf{R}.\mathsf{G}$ is not $\bot$-local w.r.t. $\Sigma_1$ and contains the symbol $\mathsf{R}$, we would need to further reduce $\Sigma_1$ to some $\Sigma_2 \subset \Sigma_1$.

However, if we took $\Sigma_1 = \mathsf{Sig}(\mathcal{O}_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}\}$ or $\Sigma_1 = \mathsf{Sig}(\mathcal{O}_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{E}\}$, then we would already have $\mathsf{Sig}(\mathcal{M}_{[\mathcal{O}_{\mathsf{ex}},\Sigma_1]}) = \Sigma_1$, and we would be done. ♢

The $\bot$-module $\mathcal{M}_{[\mathcal{O},\mathsf{Sig}(\mathcal{O}) \setminus \Sigma^{\text{ELK}}]}$ that the OWL reasoner needs to classify is likely to be smaller the larger $\Sigma^{\text{ELK}}$ is. Therefore, it is desirable to find heuristics to choose each $\Sigma_i$ in a way that leads to an $\mathcal{L}_{\text{ELK}}$-signature as large as possible. Below we describe the main heuristics that we have implemented in MORe.

**Keeping Properties** As far as possible, we try not to remove properties from $\Sigma_i$. The reason for this is that most ontologies contain fewer properties than classes, and each property usually appears in more axioms than any class. Thus, removing a property from $\Sigma_i$ is more likely to bring more axioms into the next $\mathcal{S}_{i+1}$ and lead to a smaller $\mathcal{L}_{\text{ELK}}$-signature.

**Global Symbols** We perform a preprocessing stage to identify a (possibly empty) set $\Gamma$ of *global symbols* in $\mathsf{Sig}(\mathcal{O})$, such that either $\Gamma \subseteq \Sigma^{\text{ELK}}$ or $\Sigma^{\text{ELK}} = \emptyset$. For this, we first find the set $\mathcal{G}$ of all *global axioms* in $\mathcal{O}$, i.e. those that cannot possibly be made $\bot$-local, (e.g. axioms of the form $\top \sqsubseteq \mathsf{C}$) and take $\Gamma = \mathsf{Sig}(\mathcal{G})$. We then keep adding to $\Gamma$ the signatures of all the axioms in $\mathcal{O}$ that would only be $\bot$-local w.r.t. $\Sigma^{\text{ELK}}$ if some symbol in $\Gamma$ was left outside $\Sigma^{\text{ELK}}$, until no more symbols need to be added to $\Gamma$.

Then, we will only ever consider sets $\Sigma_i$ such that $\Gamma \subseteq \Sigma_i$. As the following example shows, this can sometimes mark the difference between finding a non-empty $\mathcal{L}_{\text{ELK}}$-signature or not.

*Example 3.* Consider the ontology $\mathcal{O}'_{\mathsf{ex}} = \mathcal{O}_{\mathsf{ex}} \cup \{\top \sqsubseteq \exists \mathsf{R}.\mathsf{E}\}$. In the previous example we saw how both $\mathsf{Sig}(\mathcal{O}_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}\}$ and $\mathsf{Sig}(\mathcal{O}_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{E}\}$ were equally good choices when choosing a suitable $\Sigma_1$ for $\mathcal{O}_{\mathsf{ex}}$ . This is not the case any more with $\mathcal{O}'_{\mathsf{ex}}$, as after choosing $\mathsf{Sig}(\mathcal{O}'_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{E}\}$ we would need to try to keep $\top \sqsubseteq \exists \mathsf{R}.\mathsf{E}$ outside $\mathcal{M}_{[\mathcal{O}'_{\mathsf{ex}}, \Sigma^{\text{ELK}}]}$ too; but this is not possible, so in the end we would have $\Sigma^{\text{ELK}} = \emptyset$. ♢

**Reducing Nondeterminism** In each iteration of the algorithm, instead of considering the set $\mathcal{S}_i$ as a whole, we split it into two subsets: $\mathcal{S}_i^{nondet}$, containing those axioms in $\mathcal{S}$ for which there are several ways in which $\Sigma_i$ can be reduced to make them $\bot$-local, and $\mathcal{S}_i^{det}$, those for which there is only one way.

Whenever $\mathcal{S}_i^{det} \neq \emptyset$, we obtain $\Sigma_{i+1}$ by removing from $\Sigma_i$ the symbols required by each axiom in $\mathcal{S}_i^{det}$, and ignore $\mathcal{S}_i^{nondet}$. When $\mathcal{S}_i^{det} = \emptyset$, we deal with the axioms in $\mathcal{S}_i^{nondet}$ taking a greedy approach —finding the optimal solution is often too expensive.

The intuition behind this heuristic is that, by postponing making any nondeterministic decisions as much as possible, we might eliminate the need to make them altogether.

Note that, using this heuristic, we are not guaranteed to handle all the non $\mathcal{L}_{\text{ELK}}$-axioms in the first iteration any more, therefore we also have to consider in each $\mathcal{S}_i$ those non-$\mathcal{L}_{\text{ELK}}$ axioms that are still not $\bot$-local w.r.t. $\Sigma_i$.

*Example 4.* Consider the ontology $\mathcal{O}''_{\mathsf{ex}}$ consisting of all the axioms in $\mathcal{O}_{\mathsf{ex}}$, plus the following additional axioms:

$$\mathsf{E} \sqsubseteq \mathsf{C} \qquad \mathsf{H} \sqsubseteq \exists \mathsf{R}.\mathsf{E} \qquad \mathsf{I} \equiv (\mathsf{E} \sqcap \mathsf{F}) \sqcup (\mathsf{G} \sqcap \mathsf{H})$$

We first get $\mathcal{S}_0^{nondet} = \{\mathsf{I} \equiv (\mathsf{E} \sqcap \mathsf{F}) \sqcup (\mathsf{G} \sqcap \mathsf{H})\}$ and $\mathcal{S}_0^{det} = \{\mathsf{A} \equiv \mathsf{B} \sqcup \mathsf{C}\}$. The new non-$\mathcal{L}_{\text{ELK}}$ axiom, $\mathsf{I} \equiv (\mathsf{E} \sqcap \mathsf{F}) \sqcup (\mathsf{G} \sqcap \mathsf{H})$, goes into $\mathcal{S}_0^{nondet}$ because it could be handled by removing any of the following sets of symbols: $\{\mathsf{I}, \mathsf{E}, \mathsf{G}\}$, $\{\mathsf{I}, \mathsf{F}, \mathsf{G}\}$, $\{\mathsf{I}, \mathsf{E}, \mathsf{H}\}$ or $\{\mathsf{I}, \mathsf{F}, \mathsf{H}\}$. For now we only deal with $\mathcal{S}_0^{det} = \{\mathsf{A} \equiv \mathsf{B} \sqcup \mathsf{C}\}$, and we do so by taking $\Sigma_0 = \mathsf{Sig}(\mathcal{O}''_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$.

Then we obtain the sets $\mathcal{S}_1^{nondet} = \{\mathsf{B} \equiv \mathsf{D} \sqcap \exists \mathsf{R}.\mathsf{E}, \mathsf{I} \equiv (\mathsf{E} \sqcap \mathsf{F}) \sqcup (\mathsf{G} \sqcap \mathsf{H})\}$ and $\mathcal{S}_1^{det} = \{\mathsf{E} \sqsubseteq \mathsf{C}\}$ and we deal with $\mathsf{E} \sqsubseteq \mathsf{C}$ by taking $\Sigma_1 = \mathsf{Sig}(\mathcal{O}''_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{E}\}$.

**Table 1.** Ontology metrics

| Metrics / Ontology | Expressivity | $\lvert\mathsf{Sig}(\mathcal{O})\rvert$ | $\lvert\mathcal{O}\rvert$ | $\lvert\mathcal{O} \setminus \mathcal{O}_{\mathcal{L}_{\text{ELK}}}\rvert$ | $\lvert\mathcal{M}_{\text{OWL2}}\rvert$ |
|---|---|---|---|---|---|
| Gazetteer | $\mathcal{ALE}+$ | 517,039 | 652,361 | 0 | 0% |
| Cardiac Electrophys. | $\mathcal{SHF}(\mathcal{D})$ | 81,020 | 124,248 | 22 | 1% |
| Protein | $\mathcal{S}$ | 35,205 | 46,114 | 15 | 22% |
| Biomodels | $\mathcal{SRIF}$ | 187,577 | 439,248 | 22,104 | 45% |
| Cell Cycle v0.95 | $\mathcal{SRI}$ | 144,619 | 511,354 | 1 | <0.1% |
| Cell Cycle v2.01 | $\mathcal{SRI}$ | 106,517 | 624,702 | 9 | 98% |
| NCI v09.12d | $\mathcal{SH}(\mathcal{D})$ | 77,571 | 109,013 | 4,682 | 58% |
| NCI v13.03d | $\mathcal{SH}(\mathcal{D})$ | 97,652 | 136,902 | 158 | 57% |
| $\text{SNOMED}_{15\sqcup}$ | $\mathcal{ALCR}$ | 291,216 | 291,185 | 15 | 3% |
| SNOMED+LUCADA | $\mathcal{ALCRIQ}(\mathcal{D})$ | 309,405 | 550,453 | 122 | 0.1% |

In the next iteration, we get $\mathcal{S}_2^{nondet} = \{\mathsf{I} \equiv (\mathsf{E} \sqcap \mathsf{F}) \sqcup (\mathsf{G} \sqcap \mathsf{H})\}$ —note that axiom $\mathsf{B} \equiv \mathsf{D} \sqcap \exists \mathsf{R}.\mathsf{E}$ has been taken care of indirectly— and $\mathcal{S}_2^{det} = \{\mathsf{H} \sqsubseteq \exists \mathsf{R}.\mathsf{E}\}$, and we handle $\mathsf{H} \sqsubseteq \exists \mathsf{R}.\mathsf{E}$ by taking $\Sigma_2 = \mathsf{Sig}(\mathcal{O}''_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{E}, \mathsf{H}\}$.

After that, we find $\mathcal{S}_2^{nondet} = \emptyset$ and $\mathcal{S}_2^{det} = \{\mathsf{I} \equiv (\mathsf{E} \sqcap \mathsf{F}) \sqcup (\mathsf{G} \sqcap \mathsf{H})\}$, and take $\Sigma_3 = \mathsf{Sig}(\mathcal{O}''_{\mathsf{ex}}) \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{E}, \mathsf{H}, \mathsf{I}\}$, which finally gives $\mathcal{S}_3 = \emptyset$. Thus, we have computed $\Sigma^{\text{ELK}} = \Sigma_3$ without making any nondeterministic decisions.

## 3 Evaluation

We have tested MORe using an Ubuntu 12.04 64-bit machine with 7.8 GiB of RAM (fully assigned to the JVM) and an Intel Core i7-3770 CPU @ 3.40GHz x 8 processor. Our test ontology suite includes six BioPortal ontologies[5] [3] —for Biomodels we consider only its TBox—, two different versions of NCI[6] [5], and two extensions of SNOMED[7] [9]: $\text{SNOMED}_{15\sqcup}$ was built from a 2012 version of SNOMED, following the suggestions of domain experts, by adding 15 axioms containing disjunctions; SNOMED+LUCADA was obtained by mapping a 2011 version of SNOMED to the terminological part of the LUCADA ontology[8] [10, 11] using the ontology matching system LogMap [7]. Table 1 gives an overview of the general features of these ontologies, including the number of non-$\mathcal{L}_{\text{ELK}}$ axioms they contain and the size of the module extacted by MORe for the OWL reasoner, $\mathcal{M}_{[\mathcal{O}, \mathsf{Sig}(\mathcal{O}) \setminus \Sigma^{\text{ELK}}]}$, referred to as $\mathcal{M}_{\text{OWL2}}$ in Table 1.

[5] http://bioportal.bioontology.org/ontologies/1397
[6] http://evs.nci.nih.gov/ftp1/NCI_Thesaurus/archive
[7] http://www.ihtsdo.org/snomed-ct/
[8] The LUCADA ontology ($\mathcal{ALCHI}(\mathcal{D})$) contains 476 entities and can be classified by both HermiT and Pellet in less than 2 seconds

**Table 2.** Classification times in seconds

| Ontology \ Reasoner | MORe$_{\text{HermiT}}$ | | HermiT | MORe$_{\text{Pellet}}$ | | Pellet |
|---|---|---|---|---|---|---|
| | HermiT | total | | Pellet | total | |
| Gazetteer | 0 | 20.6 | 651 | 0 | 20.3 | 1,414 |
| Cardiac Electrophys. | 0.3 | 6.3 | 22.7 | 0.3 | 5.5 | 11.0 |
| Protein | 2.0 | 4.8 | 10.0 | 2.0 | 4.7 | 2,920 |
| Biomodels | 377 | 487 | 582 | 373 | 483 | 1,915 |
| Cell Cycle v0.95 | <0.1 | 9.9 | mem | <0.1 | 10.4 | 3,433 |
| Cell Cycle v2.01 | mem | mem | mem | mem | mem | 3,435 |
| NCI v09.12d | 244 | 252 | 261 | 256 | 266 | 93.6 |
| NCI v13.03d | 45.1 | 62.7 | 68.4 | 45.7 | 62.9 | 191 |
| SNOMED$_{15\sqcup}$ | 4.5 | 25.4 | 1,395 | 4.4 | 22.9 | 4,314 |
| SNOMED+LUCADA | 1.1 | 28.8 | 1,302 | 1.2 | 29.2 | mem |

We analyse our results by comparing the performance of MORe using HermiT vs. HermiT alone, and of MORe using Pellet vs. Pellet alone. A summary of all results can be found in Table 2. mem indicates an out of memory error.

When integrating HermiT, MORe is always able to improve, or at least maintain its performance. We remark the case of Cell Cycle v0.95, where the performance is improved from an out of memory error to termination in under 10s.

Integrating Pellet, however, sometimes has an unexpected effect. In the cases of NCI v09.12d and Cell Cycle v2.01, Pellet takes longer to classify $\mathcal{M}_{\text{OWL2}}$ when integrated in MORe than to classify the whole ontology on its own. This however, does not happen when Pellet is used to classify $\mathcal{M}_{\text{OWL2}}$ independently of MORe. We are still unsure about the causes of this phenomenon. Apart from these two cases, MORe is still often able to improve on the performance of Pellet.

It is worth mentioning that the reason why, in the case of Cell Cycle v2.01, the portion of the ontology that the OWL reasoner has to process is so close to the whole ontology (98%) is because the 9 non $\mathcal{L}_{\text{ELK}}$ axioms are symmetric property axioms, which force their 9 respective properties out of $\Sigma^{\text{ELK}}$, reducing it to a very small set.

## 4 Conclusions and Future Directions

We are continuing to develop MORe, exploring new ways of further reducing the workload assigned to a general purpose OWL 2 classification algorithm. We are looking into the possibility of alternative modularity notions specific for this application, and also into exploiting computational properties of other lightweight ontology languages to combine our modular approach with one based on finding lower and upper bounds for the classification.

## Acknowledgements

This work was supported by the Royal Society, the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, "Optique", and the EPSRC projects Score!, ExODA and MaSI$^3$.

## References

1. Armas Romero, A., Cuenca Grau, B., Horrocks, I.: MORe: Modular Combination of OWL Reasoners for Ontology Classification. In: International Semantic Web Conference (ISWC). pp. 1–16 (2012)
2. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: Theory and practice. J. Artificial Intelligence Research 31, 273–318 (2008)
3. Fridman Noy, N., Shah, N.H., Whetzel, P.L., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D.L., Storey, M.A.D., Chute, C.G., Musen, M.A.: BioPortal: ontologies and integrated data resources at the click of a mouse. Nucleic Acids Research 37(Web-Server-Issue), 170–173 (2009)
4. Glimm, B., Horrocks, I., Motik, B., Shearer, R., Stoilos, G.: A novel approach to ontology classification. J. of Web Semantics 10(1) (2011)
5. Golbeck, J., Fragoso, G., Hartel, F.W., Hendler, J.A., Oberthaler, J., Parsia, B.: The National Cancer Institute's Thésaurus and Ontology. J. Web Semantics 1(1), 75–80 (2003)
6. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semantic Web 2(1), 11–21 (2011)
7. Jiménez-Ruiz, E., Cuenca Grau, B., Zhou, Y., Horrocks, I.: Large-scale interactive ontology matching: Algorithms and implementation. In: European Conference on Artificial Intelligence (ECAI). pp. 444–449 (2012)
8. Kazakov, Y., Krötzsch, M., Simancik, F.: Concurrent classification of EL ontologies. In: International Semantic Web Conference (ISWC). pp. 305–320 (2011)
9. Schulz, S., Cornet, R., Spackman, K.A.: Consolidating SNOMED CT's ontological commitment. Applied Ontology 6(1), 1–11 (2011)
10. Sesen, M.B., Bañares-Alcántara, R., Fox, J., Kadir, T., Brady, J.M.: Lung Cancer Assistant: An Ontology-Driven, Online Decision Support Prototype for Lung Cancer Treatment Selection. In: OWL: Experiences and Directions Workshop (2012)
11. Sesen, M.B., Jiménez-Ruiz, E., Bañares-Alcántara, R., Brady, J.M.: Evaluating OWL 2 Reasoners in the context of Clinical Decision Support in Lung Cancer Treatment Selection. In: OWL Reasoner Evaluation (ORE) Workshop (2013)
12. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL DL reasoner. J. of Web Semantics 5(2), 51–53 (2007)

# Experimenting with ELK Reasoner on Android

Yevgeny Kazakov and Pavel Klinov

The University of Ulm, Germany
{yevgeny.kazakov, pavel.klinov}@uni-ulm.de

**Abstract.** This paper presents results of a preliminary evaluation of the OWL EL reasoner ELK running on a Google Nexus 4 cell phone under Android 4.2 OS. The results show that economic and well-engineered ontology reasoners can demonstrate acceptable performance when classifying ontologies with thousands of axioms and take advantage of multi-core CPUs of modern mobile devices. The paper emphasizes the engineering aspects of ELK's design and implementation which make this performance possible.

## 1 Introduction and Motivation

Mobile computing has been on the rise for the last decade and the Semantic Web applications are no exception. Increasingly many mobile applications can benefit from semantic technologies, especially when it comes to context-aware information processing [1]. Specifically, it is desirable to be able to combine data obtained by various mobile IO devices (sensors), such as GPS devices, Wifi or cellular networks, etc., with background information supplied by ontologies. For example, an intelligent application can use an ontology representing various kinds of businesses, e.g., restaurants, grocery stores, etc., with facts determining the user's location to suggest places to go. Or a medical application can use a medical ontology in conjunction with private user's medical data to provide counselling or other services. Such applications require reasoning to make use of implicit knowledge and sometimes *may* require reasoning to happen on the device itself (rather than on a remote server or in the cloud) for reasons such as privacy [2].

Recently there has been interest in ontology reasoners designed specifically for mobile platforms. Some researchers claim that mobile devices, being resource-constrained, require reasoner developers design their reasoning engines *specifically* for mobile computing environments [3]. Few such reasoner implementation and evaluation reports are available, for example, Delta reasoner [3] and Pocket KR Hyper [2]. At the same time we are not aware of any experience of porting existing reasoners to mobile platforms. This is a little surprising since modern devices boast substantial computational power. Having the same reasoning core working for both desktop/server and mobile devices with minimal changes would be attractive from the maintainability point of view.

This paper is a step in that direction. It presents an evaluation of ELK, a concurrent reasoner for OWL EL profile [4] implemented in Java, on Google's Nexus 4 phone under Android 4.2 operating system. The results demonstrate that ELK is able to provide acceptable classification performance on mid-to-large sized ontologies (up to tens of thousands of axioms) and is even able to classify SNOMED CT, one of the largest medical ontologies, which remains a challenge for many OWL reasoners even on desktops.

$$\mathbf{R_0}\ \frac{}{C \sqsubseteq C}: C \text{ occurs in } \mathcal{O} \qquad \mathbf{R}^+_\sqcap\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2}: D_1 \sqcap D_2 \text{ occurs in } \mathcal{O}$$

$$\mathbf{R}_\top\ \frac{}{C \sqsubseteq \top}: C \text{ and } \top \text{ occur in } \mathcal{O} \qquad \mathbf{R}_\exists\ \frac{E \sqsubseteq \exists R.C \quad C \sqsubseteq D}{E \sqsubseteq \exists S.D}: \begin{array}{l}\exists S.D \text{ occurs in } \mathcal{O}\\ R \sqsubseteq^*_\mathcal{O} S\end{array}$$

$$\mathbf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E}: D \sqsubseteq E \in \mathcal{O} \qquad \mathbf{R}_\circ\ \frac{E \sqsubseteq \exists R_1.C \quad C \sqsubseteq \exists R_2.D}{E \sqsubseteq \exists S.D}: \begin{array}{l}S_1 \circ S_2 \sqsubseteq S \in \mathcal{O}\\ R_1 \sqsubseteq^*_\mathcal{O} S_1\\ R_2 \sqsubseteq^*_\mathcal{O} S_2\end{array}$$

$$\mathbf{R}^-_\sqcap\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}$$

**Fig. 1.** The inference rules for reasoning in $\mathcal{EL}^+$

Importantly, the changes between the standard and mobile versions of ELK are negligible. This work is preliminary, in particular, it does not aim at comparing performance of ELK on a mobile device to that of other existing OWL reasoners (or across different mobile devices).

## 2 Preliminaries

In this paper, we will focus on the DL $\mathcal{EL}^+$ [5], which can be seen as $\mathcal{EL}^{++}$ [6] without nominals, datatypes, and the bottom concept $\bot$. $\mathcal{EL}^+$ concepts are defined using the grammar $\mathbf{C} ::= A \mid \top \mid C_1 \sqcap C_2 \mid \exists R.C$, where $A$ is an *atomic concept*, $R$ an *atomic role*, and $C$, $C_1$, $C_2 \in \mathbf{C}$. $\mathcal{EL}^+$ *axiom* is either a *concept inclusion* $C_1 \sqsubseteq C_2$ for $C_1$, $C_2 \in \mathbf{C}$, a *role inclusion* $R \sqsubseteq S$, or a *role composition* $R_1 \circ R_2 \sqsubseteq S$, where $R$, $R_1$, $R_2$, $S$ are role names. $\mathcal{EL}^+$ *ontology* $\mathcal{O}$ is a finite set of $\mathcal{EL}^+$ axioms. Given an ontology $\mathcal{O}$, we write $\sqsubseteq^*_\mathcal{O}$ for the smallest reflexive transitive binary relation over roles such that $R \sqsubseteq^*_\mathcal{O} S$ holds for all $R \sqsubseteq S \in \mathcal{O}$.

Entailment of axioms by an ontology is defined in a usual way; a formal definition can be found, e.g., in [5]. A concept $C$ is *subsumed* by $D$ w.r.t. $\mathcal{O}$ if $\mathcal{O} \models C \sqsubseteq D$. In this case, we call $C \sqsubseteq D$ an *entailed subsumption*. The *ontology classification task* requires to compute all entailed subsumptions between atomic concepts occurring in $\mathcal{O}$.

The $\mathcal{EL}^+$ reasoning procedure implemented in ELK works by applying inference rules to derive subsumptions between concepts. Figure 1 shows the rules from $\mathcal{EL}^{++}$ [6] restricted to $\mathcal{EL}^+$, but presents them in a way that does not require the normalization stage [4]. Some rules have side conditions given after the colon that restrict the expressions to which the rules are applicable. For example, rule $\mathbf{R}^+_\sqcap$ applies to each $C, D_1, D_2$, such that $D_1 \sqcap D_2$ occurs in $\mathcal{O}$ with premises $\{C \sqsubseteq D_1,\ C \sqsubseteq D_2\}$, and the conclusion $C \sqsubseteq D_1 \sqcap D_2$. Note that the axioms in the ontology $\mathcal{O}$ are only used in side conditions of the rules and never used as premises of the rules.

The rules in Figure 1 are complete for deriving subsumptions between the concepts occurring in the ontology. That is, if $\mathcal{O} \models C \sqsubseteq D$ for $C$ and $D$ occurring in $\mathcal{O}$, then $C \sqsubseteq D$ can be derived using the rules in Figure 1 [6]. Therefore, in order to classify the ontology, it is sufficient to compute the closure under the rules and take the derived subsumptions between atomic concepts.

Computing the closure under inference rules, such as in Figure 1, can be performed using a well-known *forward chaining* procedure presented in Algorithm 1 in an abstract

**Algorithm 1:** Abstract rule-based classification procedure

**input** : **C**: the set of named concepts from $\mathcal{O}$, R: a set of inference rules
**output** : Closure: a set of inferences closed under R

```
1 Closure, Todo ← ∅;
2 for A ∈ C do                                          /* initialize */
3     Todo ← Todo ∪ apply(R_0[A]) ∪ apply(R_⊤[A]);
4 while (exp ← Todo.poll()≠ null) do                    /* compute closure */
5     if exp ∉ Closure then
6         Closure ← Closure ∪ exp;
7         for r ∈ R[exp, Closure] do
8             Todo ← Todo ∪ apply(r);
9 return Closure;
```

way. The algorithm works with *expressions* of the form $C \sqsubseteq D$ or $C \sqsubseteq \exists R.D$, where $C$ and $D$ are concepts and $R$ is a role. It derives expressions by applying rules R in Figure 1. It collects those expressions to which all rules have been applied in a set Closure and the remaining ones in a queue Todo. The algorithm first initializes Todo with conclusions of the initialization rule $\mathbf{R_0}$, see lines 2–3. Then it repeatedly takes the next expression *exp* $\in$ Todo, inserts it into Closure if it does not occur there, and applies all applicable rules to it (lines 4–8). Informally, we use R[. . .] to denote selection of rules for specific premises and/or side conditions. The conclusions derived by the applied rules are then inserted in Todo.

ELK implements a concurrent version of Algorithm 1 which maintains a *context* for each concept that occurs on the left hand-side of an axiom in $\mathcal{O}$. Contexts maintain their own Todo queues and are processed in parallel threads of execution (referred to as *workers*). Details can be found in [4].

## 3 Evaluation on Google Nexus 4

This section present the results of a preliminary evaluation of ELK's classification performance on a Google Nexus 4 cell phone. The device runs under Android 4 OS and features a Qualcomm Snapdragon™ S4 Pro CPU (4 cores, 1.7 GHz) and 2 GB RAM, of which 500 MB was allocated to JVM. To put the results into a perspective, we also ran ELK on a PC with Intel Core i5-2520M 2.50GHz CPU with 8 GB RAM (JVM was allocated the same 500 MB).

Five $\mathcal{EL}$ ontologies often used for benchmarking $\mathcal{EL}$ reasoners have been selected for the experiments (the number of logical axioms given in parenthesis): Chemical Entities of Biological Interest (ChEBI, 67,182), the e-Mouse Atlas Project (EMAP, 13,730), and the Fly Anatomy (19,137) are some of large OBO Foundry[1] and Ontobee[2] ontologies that also include some non-atomic concepts. GO (28,896) is the older version of the

[1] http://www.obofoundry.org/
[2] http://www.ontobee.org/

Gene Ontology published in 2006. $\mathcal{EL}$-GALEN (36,547) is an $\mathcal{EL}^+$-restricted version of the GALEN ontology. All ontologies are freely available from the ELK website.[3]

Each ontology was classified with different, from 1 to 6, number of workers and the results are presented in Table 1. The most obvious experimental outcome is that classification on the cell phone is about two orders of magnitude slower than on a PC, i.e., the difference appears larger than in the mere computational power of the two systems (at least, if the latter is compared in terms of just CPU rate and the amount of RAM). Comparison of the "LI Ratio" columns reveals that the relative difference during the classification stage (CPU-bound processing) is larger than during the loading and indexing stage (mostly IO-bound). One possible explanation is that CPU caches, for which ELK's data structures are optimized (see the next section), are more effective on PC than on this cell phone. Difference in the RAM speed may have also played a role.

It can be noted that ELK's concurrent classification algorithm brings benefits on the cell phone just as well as on PC. The difference is especially visible between 1 worker and 2 (or more) workers. It is only visible for the classification stage because during indexing most time is spent on loading axioms from external memory and parsing.

Finally, we attempted to classify the official January 2013 release of SNOMED CT, one of the largest medical ontologies (296,529 axioms).[4] The intent was to push ELK (and the phone) to its limits. Tad surprisingly, ELK still managed to complete classification in 1h and 20m, out of which nearly 10m was spent on loading/indexing and the rest on reasoning. It has used nearly all (475 MB) memory available to JVM. For reference, it takes about 10s to classify SNOMED CT on a laptop with Intel Core i5-2520M 2.50GHz CPU and 4GB of RAM available to JVM.

## 4 Implementation Notes

This section provides some engineering details on implementation of ELK. The methods listed below are not specific to a particular computational platform. However, they are particularly relevant to mobile devices since they seek to reduce the memory footprint of the reasoner.

**Entity filtering:** In large ontologies it is often the case that some OWL entities (concept (sub)expressions or roles) appear in many axioms. ELK's internal entity filter guarantees that each entity is represented by precisely one Java object. This has two advantages: First, it reduces memory consumption and thus reduces the number of GC cycles. Second, it allows for fast equality checking by comparing references (basically, memory pointers). The latter is especially important for searching for an object in collections, e.g., sets or arrays.

**Economic data structures:** The ELK's classification algorithm operates with many collections of objects representing OWL entities, such as subsumers for a given concept, conjuncts in a given conjunctive concept expression, etc. The important thing is that most of those collections are small, i.e. usually up to hundred elements. ELK provides a custom array-based, cache-friendly hashtable implementation with linear prob-

[3] `https://code.google.com/p/elk-reasoner/wiki/TestOntologies`

[4] We did not include it in the main experiment since it would take too much time to vary the number of workers for it.

**Table 1.** Time (in ms) and memory usage (in MB) results for loading/indexing and classification on a Google Nexus 4 and a PC. The LI Ratio column shows the proportion of total time (in %) spent for loading and indexing the ontology.

| | | Google Nexus 4 | | | | PC | | |
|---|---|---|---|---|---|---|---|---|
| Ontology | Workers | Load./Index. | Classif. | LI Ratio | Memory | Load./Index. | Classif. | LI Ratio |
| ChEBI | 1 | 31,370 | 207,020 | 13 | 67 | 351 | 1,055 | 25 |
| ChEBI | 2 | 29,423 | 160,334 | 16 | 72 | 323 | 715 | 31 |
| ChEBI | 3 | 32,213 | 148,369 | 18 | 72 | 337 | 611 | 36 |
| ChEBI | 4 | 32,443 | 147,868 | 18 | 68 | 324 | 646 | 33 |
| ChEBI | 5 | 32,900 | 114,054 | 22 | 65 | 362 | 570 | 39 |
| ChEBI | 6 | 29,997 | 107,033 | 22 | 72 | 341 | 597 | 36 |
| EMAP | 1 | 20,667 | 6,970 | 75 | 23 | 366 | 93 | 80 |
| EMAP | 2 | 19,580 | 4,337 | 82 | 24 | 389 | 83 | 82 |
| EMAP | 3 | 20,311 | 3,750 | 84 | 25 | 413 | 72 | 85 |
| EMAP | 4 | 19,081 | 3,508 | 84 | 24 | 396 | 68 | 85 |
| EMAP | 5 | 19,921 | 3,467 | 85 | 23 | 383 | 73 | 84 |
| EMAP | 6 | 19,949 | 3,390 | 95 | 25 | 360 | 86 | 81 |
| Fly Anatomy | 1 | 7,882 | 31,478 | 20 | 22 | 195 | 276 | 39 |
| Fly Anatomy | 2 | 8,231 | 18,953 | 30 | 23 | 248 | 252 | 52 |
| Fly Anatomy | 3 | 9,143 | 16,951 | 35 | 24 | 256 | 223 | 51 |
| Fly Anatomy | 4 | 8,483 | 16,041 | 35 | 24 | 225 | 275 | 45 |
| Fly Anatomy | 5 | 7,743 | 15,439 | 33 | 26 | 278 | 253 | 52 |
| Fly Anatomy | 6 | 8,462 | 15,992 | 34 | 25 | 283 | 250 | 53 |
| GO | 1 | 30,745 | 33,441 | 48 | 38 | 518 | 214 | 71 |
| GO | 2 | 33,856 | 20,503 | 62 | 38 | 651 | 217 | 75 |
| GO | 3 | 31,395 | 15,752 | 67 | 38 | 639 | 236 | 73 |
| GO | 4 | 31,348 | 15,516 | 67 | 38 | 581 | 217 | 73 |
| GO | 5 | 31,419 | 18,721 | 63 | 39 | 713 | 222 | 76 |
| GO | 6 | 30,464 | 17,055 | 64 | 38 | 714 | 232 | 75 |
| $\mathcal{EL}$-GALEN | 1 | 21,319 | 211,839 | 9 | 76 | 403 | 1,582 | 20 |
| $\mathcal{EL}$-GALEN | 2 | 21,053 | 145,657 | 12 | 76 | 389 | 979 | 28 |
| $\mathcal{EL}$-GALEN | 3 | 21,230 | 129,322 | 14 | 76 | 394 | 922 | 30 |
| $\mathcal{EL}$-GALEN | 4 | 21,702 | 176,283 | 11 | 76 | 444 | 841 | 35 |
| $\mathcal{EL}$-GALEN | 5 | 21,996 | 157,872 | 12 | 80 | 385 | 867 | 31 |
| $\mathcal{EL}$-GALEN | 6 | 22,259 | 114,014 | 16 | 85 | 409 | 897 | 31 |

ing which is fine-tuned for small sets and supports very fast lookup and iteration (as, consequently, intersection) operations.

**Optimized class taxonomy:** ELK's implementation of taxonomy is specifically optimized for ontology class hierarchies, which are mostly shallow trees (or DAGs) with a possible large branching factor. For example, the bottom node which represents unsatisfiable concepts ($\bot$ and others) does not store references to its parent nodes (satisfiable concepts with no subsumees) and neither do its parents store a reference to $\bot$. Also, the taxonomy supports concurrent updates and can be built incrementally, i.e., new nodes can be added as soon as all subsumers for a given concept have been inferred.

**Indexing:** ELK does not explicitly store axioms after the ontology has been loaded. Instead, it creates instances of the rules in Figure 1 as it loads the axioms and stores them in the objects which represent entities occurring in the axiom. This is done to, first, avoid the cost of storing potentially a large number of complex axioms, second, enable a fast implementation of the $\mathsf{R}[\ldots]$ operator for finding applicable rules, and finally, group together different rules applicable to the same premises. For example, if $\mathcal{O}$ contains axioms $A \sqsubseteq B$, $A \sqsubseteq C$, and $A \sqsubseteq D$, they can be grouped into a threefold instance of $\mathbf{R}_{\sqsubseteq} : A \mapsto \{B, C, D\}$, which derives $X \sqsubseteq B$, $X \sqsubseteq C$, and $X \sqsubseteq D$ in one go when applying to $X \sqsubseteq A$ (for some concept $X$). In addition, such indexing ensures that if some construct, e.g., role composition axioms, never occurs in the ontology, then the corresponding rule, e.g., $\mathbf{R}_{\mathbf{o}}$, will never even be considered for selection. Finally, the rule instances can be quickly updated if some axioms are added or deleted without the need to reload the ontology from external memory.

## 5 Conclusion

ELK has not been designed for mobile platforms. However, it has been heavily engineered to minimize memory consumption and take advantage of multi-core CPUs. This paper provides some insight into what happens when such a reasoner is run on a mobile device. Our experimental results are preliminary but they suggest that well-engineered reasoners can provide acceptable performance on modern cell phones, and can even classify some of the largest available ontologies. It is worth mentioning that ELK does not depend on external libraries (other than for logging) and thus runs nearly out-of-the-box under the Android's JVM. Therefore, mobile users can immediately benefit from all improvements being made in the main ELK's development branch.

In the future it would make sense to perform a more detailed evaluation on multiple devices, including a comparison between existing reasoners (at least those which can work on multiple platforms out-of-the-box). Also, more fine-grained experiments and thorough profiling are required to understand the reasons why reasoning is that much slower on a cell phone. This may lead not only to faster mobile reasoning but also improve performance on desktops and servers.

## References

1. Specht, G., Weithöner, T.: Context-aware processing of ontologies in mobile environments. In: Mobile Data Management Conference. (2006) 86–89

2. Kleemann, T.: Towards mobile reasoning. In Parsia, B., Sattler, U., Toman, D., eds.: Proc. 19th Int. Workshop on Description Logics (DL'06). Volume 189 of CEUR Workshop Proceedings., CEUR-WS.org (2006)
3. Motik, B., Horrocks, I., Kim, S.M.: Delta-reasoner: a semantic web reasoner for an intelligent mobile platform. In: World Wide Web Conference (Companion Volume). (2012) 63–72
4. Kazakov, Y., Krötzsch, M., Simančík, F.: Concurrent classification of $\mathcal{EL}$ ontologies. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E., eds.: Proc. 10th Int. Semantic Web Conf. (ISWC'11). Volume 7032 of LNCS., Springer (2011) 305–320
5. Baader, F., Lutz, C., Suntisrivaraporn, B.: Efficient reasoning in $\mathcal{EL}^+$. In Parsia, B., Sattler, U., Toman, D., eds.: Proc. 19th Int. Workshop on Description Logics (DL'06). Volume 189 of CEUR Workshop Proceedings., CEUR-WS.org (2006)
6. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In Kaelbling, L., Saffiotti, A., eds.: Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05). (2005) 364–369

# Extending Datatype Support for Tractable Reasoning with OWL 2 EL Ontologies

Pospishnyi Oleksandr[1],

[1] National Technical University of Ukraine "Kyiv Polytechnic Institute", Kiev, Ukraine
pospishniy@kpi.in.ua

**Abstract.** It was mentioned on multiple occasions that datatype expressions are a necessary component of any production quality knowledge base and will surely play a major role in the upcoming Semantic Web. This paper briefly summarizes the work on improving the support for tractable reasoning with datatype expressions in ELK - a highly efficient $\mathcal{EL}$ reasoner. Tests have shown an exceptional speed of ontology classification of great size which opens up new perspectives for applying ontologies with datatype expressions in practice.

**Keywords:** ontology, dataypes, reasoning, EL++, ELK

## 1 Background

In Description Logics, *datatypes* (also called concrete domains) can be used to define new concepts by referring to particular values, such as strings or integers. For example the following axioms from computer hardware ontology provide definitions for the notions of dual-, quad-, and many-core processors:

$$\mathbf{DualCoreCPU} \equiv \mathbf{CPU} \sqcap \exists \mathbf{hasCores}.(=, \mathbf{2})$$
$$\mathbf{QuadCoreCPU} \equiv \mathbf{CPU} \sqcap \exists \mathbf{hasCores}.(=, \mathbf{4})$$
$$\mathbf{MultiCoreCPU} \equiv \mathbf{CPU} \sqcap \exists \mathbf{hasCores}.(>, \mathbf{1})$$

In mentioned example, $(>, \mathbf{1})$ refers to the domain of natural numbers and the relation > is used to constrain possible values to those larger than 1. Restriction $(=, \mathbf{2})$ uses the relation = to constrain the value to element 2, and similarly for $(=, \mathbf{4})$.

Any ontology reasoner, with a support of datatype expressions, as presented above, should be able to derive new axioms such as:

$$\mathbf{DualCoreCPU} \sqsubseteq \mathbf{MultiCoreCPU}$$
$$\mathbf{QuadCoreCPU} \sqsubseteq \mathbf{MultiCoreCPU}$$

In order to ensure that reasoning remains polynomial, logic $\mathcal{EL}^{++}$ allows only for datatype restrictions that *cannot* implicitly express concept disjunction, which is a well-known cause of intractability. Consider, for example, the axioms:

$$\mathbf{CPU} \sqsubseteq \exists \mathbf{hasCores}.(\geq, \mathbf{1})$$
$$\mathbf{SingleCoreCPU} \equiv \mathbf{CPU} \sqcap \exists \mathbf{hasCores}.(=, \mathbf{1})$$
$$\mathbf{MultiCoreCPU} \equiv \mathbf{CPU} \sqcap \exists \mathbf{hasCores}.(\geq, \mathbf{2})$$

It could be seen, that these axioms imply the disjunction:

$$\mathbf{CPU} \sqsubseteq \mathbf{SingleCoreCPU} \sqcup \mathbf{MulticoreCPU}$$

To prevent such situations, the EL Profile of OWL 2, which is based on $\mathcal{EL}^{++}$, admits only equality $(=)$ in datatype restrictions. Unfortunately, it would be almost impossible to adequately model knowledge about real world domains using ontologies with such severe limitations on allowed datatype expressions.

It was recently demonstrated by D. Magka, Y. Kazakov and I. Horrocks [1] that the mentioned restrictions could be significantly relaxed without losing tractability. This paper introduced a notion of "safety" for datatype restrictions and classified all safe combinations of datatype restrictions for the domain of natural numbers, integers, rational and real numbers. Conducted theoretical work opened up an opportunity to implement datatype support for $\mathcal{EL}$ reasoners that would be both, safe and practically useful.

But more work still needs to be done. Namely, a large amount of ontologies relies on other common datatypes, such as date/time, strings, binary data, URIs, etc:

$$\mathbf{E30\text{-}1280} \sqsubseteq \mathbf{Xeon} \sqcap \exists \mathbf{releaseDate}.(=, \mathbf{2011\text{-}06\text{-}03})$$
$$\mathbf{E30\text{-}1280} \equiv \exists \mathbf{hasPartNo}.(=, \text{"}\mathbf{BX80623E31280}\text{"})$$
$$\mathbf{E30\text{-}1280} \equiv \exists \mathbf{hasCPUID}.(=, \mathbf{0206D7h}),$$

restrictions on them:

$$\exists \mathbf{hasPartNo}.(\mathbf{pattern}, \mathbf{'CM806230*'}) \sqsubseteq \mathbf{Xeon}$$
$$\mathbf{IntelCPU} \sqsubseteq \exists \mathbf{hasPartNo}.(\boldsymbol{length}_{=}, \mathbf{15}),$$

and interval relations, such as:

$$\exists \mathbf{releaseDate}.(\geq \mathbf{2011\text{-}01\text{-}01}, \leq \mathbf{2011\text{-}12\text{-}31})$$
$$\exists \mathbf{hasCores}.(\geq \mathbf{2}, < \mathbf{8}).$$

Thus, a goal was set to use the approach, presented in [1], and extend it to include all datatypes from the OWL 2 EL profile, as well as complex datatype restrictions, all without compromising the tractability.

Recent studies [2] showed that the abovementioned datatype expressions are already widely used in different ontologies all over the Internet, despite their poor support by most reasoners. It also has been often mentioned that datatype assertions would be a major part of the Semantic Web and lack of their support would greatly hamper its development.

Newly developed reasoning procedures were implemented and tested in ELK – a state of the art tractable $\mathcal{EL}$ reasoner [3].

## 2 Technical approach

According to Web Ontology Language specification, OWL 2 EL profile provides for 19 various datatypes, most of which are defined in XML Schema Definition Language (XSD) specification. Figure 1 provides a summary of all datatypes, allowed by the OWL 2 EL profile and displays their inheritance and a set of allowed facet restrictions.

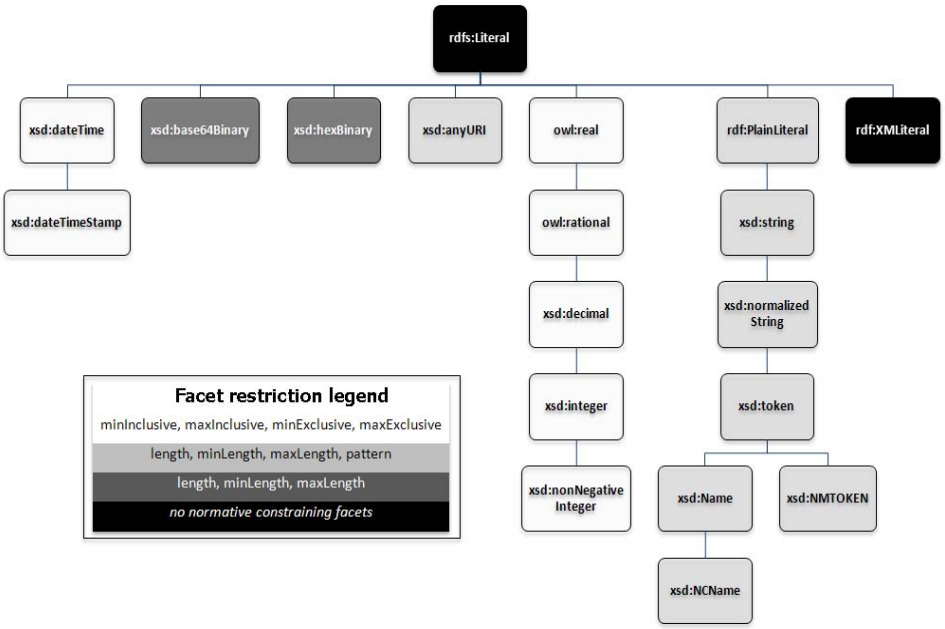



**Fig. 1.** Datatypes allowed by the OWL 2 EL profile

It is worth mentioning, that OWL 2 EL profile does not allow the following datatypes: *xsd:double, xsd:float, xsd:nonPositiveInteger, xsd:positiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte, xsd:language* and *xsd:boolean*. The set of supported datatypes has been designed such that the intersection of the value spaces of any set of these datatypes is either empty or infinite, which is necessary to obtain the desired computational properties.

In order to support reasoning with abovementioned allowed datatypes, a set of new inference rules was implemented for the ELK reasoner:

$$\frac{C \sqsubseteq \exists R.\, r_+}{C \sqsubseteq D} \qquad \exists R.\, r_- \sqsubseteq D \in \mathcal{O},\ \ r_+ \subseteq r_-$$

$$\frac{}{C \sqsubseteq \bot} \qquad C \sqsubseteq \exists R.\, r_+ \in \mathcal{O},\ \ r_+ = \emptyset$$

where $\mathcal{O}$ is an ontology, C and D are the concepts, $R$ is a datatype property and $\exists R.\, r_{\pm}$ denotes an existential datatype expression created with constraining facets. Symbols $_+$ and $_-$ indicate that an expression is occurring right or left of $\sqsubseteq$ respectively.

With $r_+ \subseteq r_-$ we represent a fact that a value space constrained by the datatype restriction $r_+$ is a subset of a value space constrained by the $r_-$ datatype restriction, or in other words $r_-$ includes $r_+$. With $r_+ = \emptyset$ we represent an empty value space produced by the datatype restriction $r_+$.

In the proposed implementation, all datatype expressions are parsed with respect to their lexical form and then transformed to internal representation that reflects the nature of their respective value space. All possible value spaces, created by the datatype restrictions, could be conventionally divided into 3 categories:

- *Values*: binary value, date/time value, literal value and numeric value.
- *Intervals*: numeric interval, date/time interval and length restriction
- *Other*: empty value space, entire value space and pattern

During the computation of all conclusions under the inference rules, for each encountered $\exists R.\, r_+$ expression a search is conducted for all $\exists R.\, r_-$ expressions in the ontology where $r_+ \subseteq r_-$. To increase the efficiency of reasoning, a special datatype index is used to optimize the search for all such $\exists R.\, r_-$ expressions.

Table 1 summarizes all possible $r_+ \subseteq r_-$ scenarios where *A* represents $r_-$ datatype restriction, and *B* represents $r_+$ datatype restriction.

My means of $B_D \subseteq A_D$ expression we state that the datatype of a restriction *B* is equal to or inherited from the datatype of a restriction *A*, for example $\mathrm{xsd{:}\,integer} \subseteq \mathrm{owl{:}\,real}$. Expression $A_{low}$ and $A_{high}$ denote a minimum and maximum value implied by the restriction *A*, while $B_{len}$ denotes a length of a corresponding value. Finally, by $B \xrightarrow{A} \notin \emptyset$ expression we represent a fact that literal value *A* matches a corresponding pattern restriction *B*.

## 3 Evaluation

The evaluation of proposed modifications was conducted using a large OWL 2 EL ontology that was generated by the Grid-DL Semantic Grid information service [4].

Two test ontologies were considered. The full ontology consisted of 1,087,124 axioms, 65 classes, 33 object properties, 109 datatype properties and 131,637 individual assertions. Truncated version of the ontology connsisted of only 230,670 axioms and 36,191 individual assertions.

Three classes were added to act as a "query" to the knowledge base:

```
UK_Site ≡ Site and hasLocation some
    (Location and hasName some string[pattern ".*, UK"])
Idle_CE ≡ ComputingElement and hasState some
    (CEState and hasRunningJobs value 0 and hasWaitingJobs value 0
     and hasFreeJobSlots some integer[>0])
     and hasState some (CEState and hasStatus value Production)
x64_Cluster ≡ SubCluster and (describedBy some
     (hasPlatformType value "x86_64"^^string) and
     (describedBy some (hasRAMSize some integer[>= 4096, <= 8192])))
```

Table 2 presents the test results. For comparison the Pellet [5] and HermiT [6] reasoners were included, both capable of reasoning on OWL 2 EL ontologies with datatype expressions. The following test setup was used: Intel Core 2 Duo T9300 @ 2.50GHz, 4 Gb RAM, OpenSUSE 12.1 (Linux 3.1.10), Java 1.7.0_05 (-Xmx3200M -Xms3200M), Pellet 2.2.0, HermiT 1.3.6. All tests were conducted three times and then averaged.

**Table 2.** Datatype restrictions subsumption matrix

| A \ B | Empty ValueSp. | Entire ValueSp. | Binary Value | DateTime Value | Literal Value | Numeric Value | DateTime Interval | Length Restriction | Numeric Interval | Pattern |
|---|---|---|---|---|---|---|---|---|---|---|
| Empty ValueSp | - | - | - | - | - | - | - | - | - | - |
| Entire ValueSp | $B_D \subseteq A_D$ | $B_D \subseteq A_D$ | $B_D \subseteq A_D$ | $B_D \subseteq A_D$ | $B_D \subseteq A_D$ | $B_D \subseteq A_D$ | $B_D \subseteq A_D$ | $B_D \subseteq A_D$ | $B_D \subseteq A_D$ | $B_D \subseteq A_D$ |
| Binary Value | - | - | $B_D = A_D$<br>$A = B$ | - | - | - | - | - | - | - |
| DateTime Value | - | - | - | $A =^1 B$ | - | - | - | - | - | - |
| Literal Value | - | - | - | - | $A =^2 B$ | - | - | - | - | - |
| Numeric Value | - | - | - | - | - | $A =^3 B$ | - | - | - | - |
| DateTime Interval | - | - | - | $B_D \subseteq A_D$<br>$A_{low} \leq B$<br>$A_{high} \geq B$ | - | - | $B_D \subseteq A_D$<br>$A_{low} \leq B_{low}$<br>$A_{high} \geq B_{high}$ | - | - | - |
| Length Restrict. | - | - | $B_D \subseteq A_D$<br>$A_{low} \leq B_{len}$<br>$A_{high} \geq B_{len}$ | - | $B_D \subseteq A_D$<br>$A_{low} \leq B_{len}$<br>$A_{high} \geq B_{len}$ | - | - | $B_D \subseteq A_D$<br>$A_{low} \leq B_{low}$<br>$A_{high} \geq B_{high}$ | - | $B_D \subseteq A_D$<br>$B \subseteq^4 A$ |
| Numeric Interval | - | - | - | - | - | $B_D \subseteq A_D$<br>$A_{low} \leq^3 B$<br>$A_{high} \geq^3 B$ | - | - | $B_D \subseteq A_D$<br>$A_{low} \leq^3 B_{low}$<br>$A_{high} \geq^3 B_{high}$ | - |
| Pattern | - | - | - | - | $B_D \subseteq A_D$<br>$B \xrightarrow{A} \notin \emptyset$ | - | - | $B_D \subseteq A_D$<br>$B \subseteq^4 A$ | - | $B_D \subseteq A_D$<br>$B \subseteq^4 A$ |

[1] Equality evaluation takes into account a position of A and B on the timeline with respect to specified time zones. If time zone is specified only for one parameter, a ± 14:00 offset is used in evaluation.

[2] Literals are considered to be equal when all characters of their lexical form are equal to each other, both missing a language tag and/or datatype tag or (if present) they are equal for both literals.

[3] During comparison all numbers are cast to common, most specific datatype.

[4] Verify that every interpretation of regular expression (restriction) B will satisfy a regular expression (restriction) A. All restrictions are viewed as regular expressions.

**Table 2.** Evaluation results

| Reasoner | Truncated ontology classification time, ms | Full ontology classification time, ms |
|---|---|---|
| ELK | 7366 | 54912 |
| ELK[5] | 5598 | 12567 |
| Pellet | 165419 | out of mem |
| HermiT | timeout | timeout |

Unfortunately, HermiT reasoner could not classify the ontology within a 1 hour timeout constraint and Pellet ran out of memory while processing the full ontology. Evidently, ELK greatly outperformed abovementioned reasoners due to its efficient reasoning procedures.

The profiling analysis showed that it is possible to considerably speed up the reasoning procedures by using only one type of literals in the ontology, for example *xsd:string*. If such requirements are met, simplified literal handling algorithm could yield a considerable performance boost.

Currently the work is focused on designing and implementing an ontology analysis tool that would be capable of detecting unsafe datatype expressions in the processed ontology. It would ensure that ontology being reasoned upon does not contain conflicting datatype expressions that might cause incomplete results. In case if implicit disjunction is detected, a warning would be issued to the user, informing him about the source of the problem.

The source code of modified version of ELK with the support of tractable datatype reasoning can be found in `elk-parent-datatypes` branch in the official ELK repository: https://code.google.com/p/elk-reasoner.

## 4 References


1. *D. Magka, Y. Kazakov, I. Horrocks.* Tractable Extensions of the Description Logic $\mathcal{EL}$ with Numerical Datatypes. Journal of Automated Reasoning 47(4), Pp. 427–450, Springer, 2011.
2. *B. Glimm, A. Hogan, M. Krötzsch, A. Polleres.* OWL: Yet to Arrive on the Web of Data?. arXiv preprint, 2012.
3. *Y. Kazakov, M. Krötzsch, F. Simančík.* Concurrent Classification of EL Ontologies. In: Proc. of the 10th International Semantic Web Conference (ISWC-11). LNCS 7032, Springer, 2011.
4. *O. Pospishniy, S. Stirenko.* GRID-DL: Semantic GRID Information Service. In Proc. 9th OWL Experiences and Directions Workshop (OWLED). Heraklion, Crete, 2012.
5. *B. Parsia and E. Sirin.* Pellet: An OWL-DL Reasoner. In Proc. ISWC 2004, Hiroshima, Japan. 2004.
6. *R. Shearer, B. Motik, and I. Horrocks*. HermiT: a Highly-Efficient OWL Reasoner. In Proc. 5th OWL Experiences and Directions Workshop (OWLED). Karlsruhe, Germany, 2008. Pp. 26-27


[5] Simplified literal reasoning algorithm

# DRAOn: A Distributed Reasoner for Aligned Ontologies

Chan Le Duc[1], Myriam Lamolle[1], Antoine Zimmermann[2], and Olivier Curé[3]

[1] LIASD Université Paris 8 - IUT de Montreuil, France
{chan.leduc, myriam.lamolle}@iut.univ-paris8.fr
[2] École Nationale Supérieure des Mines, FAYOL-ENSMSE, LSTI, F-42023 Saint-Étienne, France antoine.zimmermann@emse.fr
[3] LIGM Université Paris-Est, France
ocure@univ-mlv.fr

**Abstract.** DRAOn is a distributed reasoner which offers inference services for a network of OWL ontologies correlated by alignments. Reasoning with such networks of ontologies depends on the semantics we define for alignments with respect to ontologies. DRAOn supports two semantics for a network of ontologies: the standard Description Logics (DL) semantics for non-distributed reasoning, and the Integrated Distributed Description Logics (IDDL) semantics for distributed reasoning. Unlike the DL semantics where alignments are considered as inter-ontology axioms, the IDDL semantics interprets alignments as correspondences enabling to propagate non-emptiness (always satisfiable) and unsatisfiability of atomic concepts from an ontology to another one. Consequently, this makes distributed reasoning for a network of ontologies possible since consistency of the whole network can be decided from consistency of each ontology with axioms built from alignments such that these axioms ensure just necessary propagations of knowledge.

## 1 Introduction

We present DRAOn, a reasoner for a distributed network of aligned ontologies. The goal of DRAOn is to be able to reason on a set of independently developed ontologies that may overlap in concepts but are following different modelling perspectives, granularity, coverage, etc. To be able to do that, we assume that there exists explicit correspondences between different ontologies, that have been built by automatic ontology matchers, by humans, or partly by both. Therefore, the structure that DRAOn is reasoning with is not a monolithical theory, but a network of aligned ontologies. Moreover, DRAOn is able to make use of existing reasoners as black boxes for some of its tasks.

The issues faced when reasoning with heterogeneous, distributed ontologies, even when they have been already matched, are twofold: (a) the semantics of cross-ontology correspondences can be tricky to define when ontologies have different modelling perspectives because a correspondence of terms from different ontologies does not necessarily translates directly to an axiom in the ontology language (or languages); and (b) the distribution of ontologies, and possibly of reasoning services associated with them may influence the design of the global reasoner.

## 2 Background and related work

In this paper, we assume that ontologies are all Description Logic theories and we assume some familiarity with DLs. We simply repeat that DL ontologies are interpreted according to an interpretation function $\cdot^{\mathcal{I}}$ over a domain $\Delta^{\mathcal{I}}$, and interpretations that satisfy all the axioms of an ontology are called *models* of the ontology. There are a number of existing OWL DL reasoners such as HermiT [1], Pellet [2] or FacT++ [3].

In order to use several ontologies for reasoning, we assume the existence of *correspondences* between ontologies of the form $(e_i, e_j, r)$, where $e_i$ is a term from one ontology, $e_j$ a term from another ontology, and $r$ denotes a fixed binary relation such as equality, subsumption, disjointness, etc. In this paper, only correspondences where $r$ is $=$ (equality of individuals), $\in$ (class membership) or $\sqsubseteq$ (subsumption) will be considered. A set of correspondences between a pair of ontologies is an *ontology alignment* (or alignment for short). A set of ontologies together with their pairwise alignments is called a *network of aligned ontologies* (NAO). We will use bold face to denote sets, and therefore, the notation for an NAO will generally be $\langle \mathbf{O}, \mathbf{A} \rangle$.

Reasoning with NAOs strongly depends on the semantics of alignments. We say that a certain semantics of alignments defines a *distributed logic* (sometimes called "contextual logic" or even "modular ontology language"). In a distributed logic $L$, correspondences map to formula of the logic in question, so we will use a function $\tau_L$ to denote the logical formula associated to the correspondence. For instance, the simplest form of distributed logic is normal DL. An NAO can be interpreted as a DL ontology by mapping $(e_i, e_j, \sqsubseteq)$ (resp. $(e_i, e_j, =)$, $(e_i, e_j, \in)$) to $\tau_{\mathrm{DL}}(e_i, e_j, \sqsubseteq) = e_i \sqsubseteq e_j$ (resp. $e_i = e_j, e_j(e_i)$). Reasoning over an NAO is then the same as reasoning with an ontology formed by the union of all the ontologies in the NAO, and the axioms corresponding to the correspondences.

A number of authors, including ourselves, have argued that this is not satisfying when dealing with heterogeneous ontologies. Therefore, several other distributed logics were proposed. Here we focus on the ones that are adopting a Local Model Semantics (LMS [4]), among which we find Distributed Description Logics (DDL [5]), $\mathcal{E}$-connections [6], and IDDL [7]. LMS states that ontologies in an NAO have to be interpreted separately (*i.e.,* an interpretation of an NAO has a set of DL interpretations) and the NAO is satisfied if each ontology is locally satisfied, and some extra knowledge, such as alignments, can constrain further that satisfying models of the NAO. In DDL, a correspondence is interpreted as a *bridge rule* $\tau_{\mathrm{DDL}}(e_i, e_j, \sqsubseteq) = i{:}e_i \xrightarrow{\sqsubseteq} j{:}e_j$ which intuitively states that, from the view point of $e_j$'s ontology, the term $e_i$ is a subclass of the term $e_j$. In $\mathcal{E}$-connections, correspondences can take many more forms but can be translated into axioms using *links*, that are terms similar to DL roles, but which denote binary relations between elements of different interpretations domains. More precisely, $\tau_{\mathrm{Econn}}(e_i, e_j, \sqsubseteq) = e_i \sqsubseteq \exists \langle L \rangle e_j$, where $L$ is a link, says that the elements of $e_i$ in the first ontology are in relationship with the elements of $e_j$ in the second ontology, via link $L$. Such a link axiom would be part of ontology $e_i$, which means that it represents a fact from $e_i$'s ontology point of view.

As opposed to $\mathcal{E}$-connections and DDL, which treats correspondences as subjective views of an ontology wrt another ontology's terms, we proposed Integrated Distributed Description Logics (IDDL) as a distributed logic that assumes local ontologies

are agnostic to each others, and correspondences express knowledge of a mediation view that ties together the ontologies. In IDDL, $\tau_{\mathrm{DDL}}(e_i, e_j, \sqsubseteq) = i{:}e_i \xleftrightarrow{\sqsubseteq} j{:}e_j$ is simply called a correspondence. An IDDL interpretation of an NAO $\langle \mathbf{O}, \mathbf{A} \rangle$ is a structure $\langle \mathbf{I}, \boldsymbol{\epsilon} \rangle$ such that $\mathbf{I} = \{\mathcal{I}_o\}_{o \in \mathbf{O}}$ is a set of DL interpretations of the ontologies, and $\boldsymbol{\epsilon} = \{\epsilon_o : \Delta^{\mathcal{I}_o} \to \Delta\}_{o \in \mathbf{O}}$ is a set of functions (so called *equalising functions*) from local domains of interpretations to a set called the global domain. The composition of a local interpretation function $\cdot^{\mathcal{I}_i}$ with its corresponding equalising function $\epsilon_i$ defines the global interpretations of local terms. A correspondence $i{:}e_i \xleftrightarrow{\sqsubseteq} j{:}e_j$ is satisfied iff $\epsilon_i(e_i^{\mathcal{I}_i}) \subseteq \epsilon_j(e_j^{\mathcal{I}_j})$.[4]

There are implementations of DDL (Drago, a tableau-based peer-to-peer reasoner [8]) and $\mathcal{E}$-connections (embedded in an earlier version of Pellet. DRAOn, the reasoner described here, implements both the classical DL semantics of NAOs, and the IDDL semantics, according to our algorithm described in [9].

## 3 Implementation and Architecture

DRAOn implements the algorithm in [9] in Java, and provides a Java API. We first present an overview of the algorithm before giving more implementation details.

### 3.1 Algorithm

Let $\langle \mathbf{O}, \mathbf{A} \rangle$ be an NAO where $\mathbf{O}$ is a set of DL decidable ontologies, and $\mathbf{A}$ is a set of alignments between these ontologies. We assume that each ontology $o \in \mathbf{O}$ is attached to a reasoner that can be queried with a set $X$ of DL axioms and answer whether $o \cup X$ is consistent. Then, checking the consistency of an NAO amounts to querying the local reasoners multiple times with a well chosen set of axioms. The main idea is that, when correspondences are restricted to cross-ontology subsumption, equality or membership, alignments can only influence the vacuity (unsatisfiability) and non-vacuity (non-emptiness) of concepts or roles appearing in the alignments.

The main steps of the algorithm are the following:

1. build an *alignment ontology*, noted $A$, which corresponds to the translation of correspondences into OWL axioms (using $\tau_{\mathrm{DL}}$ mentioned in Sect. 2);
2. if $A$ is inconsistent then the NAO is inconsistent;
3. choose a subset $S$ (resp. $P$) of the concepts (resp. roles) appearing in $A$ to build a *global configuration* $\Omega$ consisting of axioms $C_i(x)$ (non-emptiness) with $x$ a fresh individual name for each $C_i \in S$ and $C_j \sqsubseteq \bot$ (unsatisfiability) for each $C_j \notin S$ (resp., $R_i(x, y)$ for $R \in P$ and $\top \sqsubseteq \forall R_j.\bot$ instead);
4. for a given ontology $o$, define a *local configuration* wrt $o$ as the subset of $\Omega$ which involves only terms from $o$, noted $\omega_o$;
5. if $A \cup \Omega$ is inconsistent, then go back to step 3;
6. for each $o \in \mathbf{O}$, query the local reasoner attached to $o$ using the axioms in $\omega_o$; if all local reasoners answer positively, then the network is consistent. Otherwise, go back to step 3 until there are no more configuration available.

[4] For a set $S$, $\epsilon(S)$ has to be understood as the set $\{\epsilon(x) \mid x \in S\}$.

### 3.2 Architecture

In terms of system architecture, DRAOn consists of a *global OWL reasoner* and several *local OWL reasoners*. The global reasoner is in charge of (i) computing global configurations $\Omega$, (ii) checking consistency of $A \cup \Omega$ for each global configuration $\Omega$ and (iii) sending a local configuration to each local reasoner. Each local reasoner checks consistency of $o \cup \Omega$.

The distributed behavior of the algorithm results from a feature of IDDL which does not enforce a strong relationship between alignments and ontologies. Instead of merging ontologies and alignments into a unique ontology, DRAOn distributes reasoning tasks over ontologies to different local reasoners. In addition to early improvements, we implement optimizations for reducing the number of configurations to be considered and for improving communication protocol between global and local reasoners. The main ideas are:

- Global configurations are built in an incremental way, and results obtained from checking previous configurations are reused. For each configuration, we check entailment rather than consistency. This allows for putting forward eventual backtracking points. For example, if $O_i \models C_i \sqsubseteq \bot$ (resp. $O_i \models C_i(x)$) then we do not need to check the configurations that contain $O_i \models C_i(x)$ (resp. $C_i \sqsubseteq \bot$).
- If there is a set of concepts (or roles) which are equivalent, we need to check only configurations which contain one representative concept (or role) from this set.
- Configurations consisting of non-emptiness axioms $C_i(x)$ are checked prior to those consisting of unsatisfiability axioms $C_j \sqsubseteq \bot$. This idea is based on the fact that concepts are commonly satisfiable in ontologies.
- The communication protocol between global and local reasoners is parallelized, that means, configurations are sent to local reasoners in a broadcasting way rather than a sequential one. The global reasoner uses Java threads to manage communication with the local reasoners. We have to use sockets to establish the communication between global and local reasoners instead of OWLLink[5]. This is due to efficiency question and an intrinsic characteristic of IDDL.

Apart from checking consistency of an NAO, the current version of DRAOn implements entailment services under the IDDL semantics for some kinds of entailment concept axioms. Therefore, DRAOn offers the following inference services:

- Checking consistency of an NAO under the DL semantics in a non-distributed way. In this setting, DRAOn creates a unique OWL ontology, namely global ontology, that is obtained by merging the axioms of all ontologies and the correspondences of alignments. Reasoning on the global ontology is performed in a non-distributed way, that means, a OWL reasoner is used for checking consistency of the global ontology. The network is consistent if and only if the global ontology is consistent
- Checking entailment of an OWL axiom by an NAO under the DL semantics in a non-distributed way;

[5] http://www.owllink.org/

– Checking consistency of an NAO based on the IDDL semantics in a non-distributed way, that is, all reasoning over ontologies and alignments is sequentially performed at one site on a computer.
– Checking consistency of an NAO based on the IDDL semantics in a distributed way.
– Checking entailment of some kinds of OWL axioms by a network of aligned ontologies based on the IDDL semantics in a distributed way.

## 4 Experiments

DRAOn uses the OWL API[6] to manipulate ontologies, the Alignment API [10] to deal with alignments, and the HermiT (version 1.3.6) [7] reasoner to check locally consistency of ontologies and alignments. We have performed some experiments with well-known ontologies and alignment methods. The following table presents empirical results obtained from the current version of DRAOn[8]. These experiments have run on a MAC with 4Gb RAM, 2.4 GHz Intel Core i5. The result for the distributed IDDL reasoning was obtained with a computer network consisting of the MAC and a DELL with 2Gb RAM, 1.06GHz Intel i2.

| Ontology 1 | Ontology 2 | Alignment | DL | non-distr. IDDL | distr. IDDL |
|---|---|---|---|---|---|
| Small NCI (10,000 axioms, 6,500 entities) | Small FMA (3,800 axioms, 3,700 entities) | Alcomo Map. (2,800 corr.) | 7,5s | 46s | 30s |
| Human (5,500 axioms, 3,300 entities) | Mouse ( 4,500 axioms, 2,750 entities) | Ref. Map. (1516 corr.) | 6s | 4.5s | 4s |

IDDL reasoning performances depend on the two services `OWLReasoner.getTypes(.)` and `OWLReasoner.getUnsatisfiableClasses()` which are called for checking unsatisfiability and non-emptiness of concepts involved in alignments. These services check for unsatisfiability and non-emptiness every concept name occurring in an ontology while the algorithm for IDDL consistency requires to check for unsatisfiability and non-emptiness only concepts involved in alignments. Consequently, HermiT raises an "OutOfMemoryError " when it is called for `getTypes(.)` on the ontology "Small SNOMED".

Theoretically, response time of inference services under the DL and IDDL semantics increases, respectively, double exponentially and exponentially in the number of ontologies (and the size of alignments). To make this difference occur, we might need a very large network of aligned ontologies. In addition, it might be required to perform further experiments to establish relationships between kinds of aligned ontologies and IDDL reasoning performance.

[6] http://owlapi.sourceforge.net/
[7] http://www.hermit-reasoner.com/
[8] http://iddl.gforge.inria.fr/

## 5 Conclusion

We have presented an OWL-based distributed reasoner, called DRAOn, which offers inference services for a network of aligned OWL ontologies. The distributed behavior of DRAOn is based on a feature of the IDDL semantics which allows a reasoner to distribute reasoning tasks over ontologies to different reasoners. In terms of performance, DRAOn is considerably penalized by unavailability of specific services of an OWL reasoner, for instance, returning all unsatisfiable concepts which belong to a given set of concepts; or returning all non-empty concepts which belong to a given set of concepts. Availability of such services in a future release of OWL reasoners may improve dramatically the performance of DRAOn.

We intend to extend DRAOn to reasoning on a network with disjointness correspondences in alignments. This may require us to develop new optimizations since the current algorithm for such a network is highly intractable.

## References

1. Shearer, R., Motik, B., Horrocks, I.: HermiT: A Highly-Efficient OWL Reasoner. In: Proc. of the OWLED 2008. (2008)
2. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a pratical OWL-DL reasoner. Journal of Web Semantics **5**(2) (2007) 51–53
3. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Proc. of IJCAR 2006. Volume 4130., Springer (2006) 292–297
4. Ghidini, Giunchiglia: Local Models Semantics, or contextual reasoning=Locality+Compatibility. ai **127**(2) (2001) 221–259
5. Borgida, A., Serafini, L.: Distributed description logics : Assimilating information from peer sources. Journal Of Data Semantics (1) (2003) 153–184
6. Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M.: E-connections of abstract description systems. Artif. Intell. **156**(1) (2004) 1–73
7. Zimmermann, A.: Integrated distributed description logics. In: Proceedings of the International Workshop on Description Logics. (2007)
8. Serafini, L., Tamilin, A.: Drago: Distributed reasoning architecture for the semantic web. In: Proceedings of the Europeen Semantic Web Conference. (2005) 361–376
9. Zimmermann, A., Le Duc, C.: Reasoning with a network of aligned ontologies. In: RR. (2008) 43–57
10. David, J., Euzenat, J., Scharffe, F., dos Santos, C.T.: The Alignment API 4.0. In: Semantic web journal. Volume 2(1). (2011) 3–10

# The $\mathcal{ELL}$ep$\mathcal{H}$ant Reasoner System Description

Barış Sertkaya

sertkaya.baris@googlemail.com

**Abstract.** We intoduce the $\mathcal{ELL}$ep$\mathcal{H}$ant reasoner, a consequence-based reasoner for the $\mathcal{EL}^+$ fragment of DLs. We present optimizations, implementation details and experimental results for classification of several large bio-medical knowledge bases.

## 1 Introduction

In [6, 5] Brandt has shown that the tractability result in [1] for subsumption w.r.t. cyclic $\mathcal{EL}$ TBoxes can be extended to the DL $\mathcal{ELH}$, which in addition to $\mathcal{EL}$ allows for general concept inclusion axioms and role hierarchies. Later in [2], Baader et. al. have shown that the tractability result can even be further extended to the DL $\mathcal{EL}^{++}$ which in addition to $\mathcal{ELH}$ allows for the bottom concept, nominals, role inclusion axioms, and a restricted form of concrete domains. In addition to these promising theoretical results, it turned out that despite their relatively low expressivity, these fragments are still expressive enough for the well-known bio-medical knowledge bases SNOMED [8] and (large parts of) Galen [19], and the Gene Ontology GO [7]. In [3, 4, 21] the practical usability of these fragments on large knowledge bases has been investigated. The CEL Reasoner [18] was as a result of these studies the first reasoner that could classify the mentioned knowledge bases from the life sciences domain in reasonable times.

Successful applications of the $\mathcal{EL}$ family increased investment in further work in this direction. The $\mathcal{EL}$ family now provides the basis for the profile OWL2 EL[1]. Moreover, there are now several other reasoners specifically tailored for the $\mathcal{EL}$ family, like Snorocket [16], TrOWL [22], CB [11] (which extends the $\mathcal{EL}^{++}$ algorithm to Horn $\mathcal{SHIQ}$), JCEL [17] (which is a Java implementation of CEL) and ELK [12, 14, 13, 15] (which is currently the only reasoner that can classify large ontologies from real-life applications within only a few seconds). A comprehensive study comparing the performace of several reasoners on large bio-medical knowledge bases has been presented in [9]. A more recent comparison can be found in the experimental results section of [15].

In the present paper we introduce the $\mathcal{ELL}$ep$\mathcal{H}$ant reasoner,[2] a consequence-based reasoner for the $\mathcal{EL}^+$ fragment of DLs. It is the successor of our prototype reasoner CHEETAH [20]. Our motivation to develop $\mathcal{ELL}$ep$\mathcal{H}$ant is on the one hand

[1] http://www.w3.org/TR/owl2-profiles/#OWL_2_EL

[2] http://code.google.com/p/elephant-reasoner

push the performance of OWL EL reasoning further by investigating different optimizations, and on the other hand provide a reasoner with a small footprint that can be used on platforms with limited memory and computing capabilities for applications like embedded reasoning [10].

The paper is organized as follows: After describing the implementation details, we present the results of our experiments for classification of large ontologies from the biomedical domain. Although there is still room for improvements like multithreading, the experimental results show that the performance is still promising.

## 2 Implementation Details

The $\mathcal{EL}$ep$\mathcal{H}$ant reasoner is the successor of the CHEETAH prototype [20]. The motivation for the CHEETAH prototype was to improve the worst-case complexity of the $\mathcal{EL}^+$ classification algorithm by using the linear-closure algorithm from databases. There we used a modified version of the linear-closure algorithm for computing the closure of atomic concepts under the axioms of the knowledge base, i.e., for saturating the knowledge base. For each concept name, we kept a counter that is used to check whether it already satisfies the left-hand side of an axiom. The experimental results there showed that the overhead of this method was too big compared to the performance gain.

The $\mathcal{EL}$ep$\mathcal{H}$ant reasoner does not use this method. Instead, it implements the consequence-based algorithm used in ELK [12, 15] with some small modifications. It differs from ELK in the implementation of the inference rules, and in scheduling of the input and derived axioms.

As also pointed out in [15], the most time consuming phase of consequence-based classificiation is the phase where the inference rules are applied for saturating the knowledge base. Therefore it is important to optimize this phase for getting a good performance. The original saturation algorithm uses a queue for keeping the scheduled axioms. Our experiments showed that the queue operations take a considerable amount of time since these operations are executed millions of times for classifying large knowledge bases like SNOMED CT. Removal from the front and addition at the back are indeed costly operations compared to adding and removing on only one side since in the former case the links to the next queue element have to be maintained properly. In order to avoid this overhead, in $\mathcal{EL}$ep$\mathcal{H}$ant we keep the scheduled axioms in a stack. Our experiments show that for some of the ontologies this results in a larger number of derivations, but the overall performance becomes better. The performance difference between queue and stack processing is shown in Table 2.

One other optimization that $\mathcal{EL}$ep$\mathcal{H}$ant implements is that it uses the told subsumer information as input axioms for initializing the stack. For each concept name $A$, instead of using axioms of type $A \sqsubseteq A$ as input, it uses $A \sqsubseteq B$, where $B$ is a told subsumer of $A$.

Apart from these basic optimizations, $\mathcal{EL}$ep$\mathcal{H}$ant also implements some of the optimization techniques introduced in [15]. It implements the optimization

of rules for decomposing conjunctions and existential restrictions. More precisely, it does not decompose a conjunction if it has been previously derived by conjunction introduction. Similarly, if an existential restriction has previously been derived via the existential introduction rule, it does not decompose this existential restriction. Unlike ELK, when an existential restriction is decomposed, $\mathcal{EL}$ep$\mathcal{H}$ant does not schedule a so-called *init* axiom, it directly schedules an axiom with the filler of the existential on both sides. $\mathcal{EL}$ep$\mathcal{H}$ant does not yet support concurrent reasoning, but it is planned for future versions.

During saturation, we often need to do a lookup to check if a concept is subsumed by another, or if an axiom has already been processed before, etc. For such operations, we need an efficient data structure. Besides doing a lookup, we also often need to insert new elements to these data structures, like adding a new subsumer to the subsumers list, marking an axiom as processed, etc. But we never delete elements from these data structures. We also sometimes need to iterate over the elements of these data structures. For these operations, the most appropriate data structure is a an associative array. As associative array implementation, we used the Judy array library[3] for `C`. The library is optimized to avoid CPU cache misses as often as possible. Its memory consumption scales smoothly with number of entries, even when the keys are sparsely distributed.

For some of the data structures that are traversed often during saturation, $\mathcal{EL}$ep$\mathcal{H}$ant keeps double indexes. For instance, the list of subsumers of a concept is stored once as a conventional array and once as Judy array. If during saturation we need to check whether a concept is subsumed by another, we do a lookup in the Judy array. But if we need to traverse the subsumer list, for instance in existential introduction rule, we use the conventional array.

Just like its predecessor, the $\mathcal{EL}$ep$\mathcal{H}$ant reasoner is implemented in the `C` programming language. The reason why CHEETAH was implemented in `C` is the large amount of memory required by the algorithm that it implements. Due to the large number of concept names and axioms in real-life ontologies, this algorithm requires a large amount of memory and an efficient memory management. This is why we chose `C` as the implementation language. Although the $\mathcal{EL}$ep$\mathcal{H}$ant reasoner does not use this algorithm, large part of its code is based on the code of the CHEETAH prototype.

## 3 Experimental Results

In order to test the performance of $\mathcal{EL}$ep$\mathcal{H}$ant, we performed a series of experiments on large biomedical knowledge bases from real-life applications. We used the January 2013 international release of SNOMED CT by converting it to OWL functional syntax by the converter provided. Additionally, we used 6 ontologies, namely GO1, FMA, ChEBI, EMAP, Molecule Role and Galen 7 provided in the test ontology suite on the ELK web page.[4] We did not use the Galen8, GO2 and Fly Anatomy ontologies provided there since they contain disjointness axioms,

[3] http://judy.sourceforge.net
[4] http://code.google.com/p/elk-reasoner

| | $A$ | $r$ | $C \sqsubseteq D$ | $C \equiv D$ | $r \sqsubseteq s$ | $r_1 \circ r_2 \sqsubseteq s$ | Trans(r) |
|---|---|---|---|---|---|---|---|
| GO1 | 19468 | 1 | 28869 | 0 | 0 | 0 | 1 |
| FMA | 80469 | 14 | 126544 | 0 | 3 | 0 | 1 |
| SNOMED CT | 296518 | 57 | 228954 | 67563 | 12 | 1 | 0 |
| ChEBI | 31160 | 9 | 67182 | 0 | 0 | 0 | 2 |
| EMAP | 13731 | 1 | 13730 | 0 | 0 | 0 | 0 |
| Molecule Role | 9217 | 2 | 9627 | 0 | 0 | 0 | 2 |
| Galen7 | 28482 | 964 | 27820 | 19326 | 1357 | 385 | 0 |

**Table 1.** Number of concepts, roles and different types of axioms.

which is not yet supported by $\mathcal{EL}$ep$\mathcal{H}$ant. The metrics of the used ontologies are shown in Table 1.

In order to measure the effects of optimizations described in Section 2, we have run a series of experiments. The experiments were run on a computer with Intel Core i3 processor with 2.1 GHz clock speed, 8 GB of main memory and Linux operating system with 3.2.0 kernel. The results were obtained as average of 5 runs per setting per ontology. We tested the performance gain obtained by using a stack instead of a queue and performance gain obtained by initializing the stack with told subsumer information. Runtimes in miliseconds, and also total and unique number of derivations obtained from these experiments are presented in Table 2. Test results for the setting where a queue is used are marked with 'queue', results for the setting where a stack is used with 'stack' and the results for setting where a stack is used and the stack is initialized with told subsumer information is marked with 'stack+told'. The results show that except for the EMAP and Molecule Role ontologies, using a stack improves the runtime performance even if the number of total or unique derivations does not change. This is due to the overhead of enqueue and dequeue operations compared to the push and pop operations. It is also seen in the table that using the told subsumer information for preparing the input axioms always improves the runtime performance and reduces both the number of total and unique derivations.

We have also run a series of tests for comparing the loading and classification performances of $\mathcal{EL}$ep$\mathcal{H}$ant to that of ELK. We have run ELK 5 times for each ontology with the `-XX:+AggressiveHeap` parameter and taken the average of these runtimes. The results presented in Table 3 show that ELK classifies the SNOMED CT ontology faster, but needs more time to load it compared to $\mathcal{EL}$ep$\mathcal{H}$ant. For all smaller ontologies, both classification and loading times of $\mathcal{EL}$ep$\mathcal{H}$ant are slightly shorter. We conjecture that this is due to the overhead of starting the Java virtual machine. In terms of memory usage the performance of $\mathcal{EL}$ep$\mathcal{H}$ant is quite good as well. For classifying SNOMED CT, the maximum memory usage is around 420MB as the Linux `top` command shows. For ELK, it is around 1.6GB with the aggressive heap option and around 1GB without this option.

| | classification time | total derivations | unique derivations |
|---|---|---|---|
| GO1 | | | |
| queue | 94 | 261302 | 206205 |
| stack | 65 | 261302 | 206205 |
| stack+told | 61 | 241834 | 186737 |
| FMA | | | |
| queue | 637 | 1314973 | 1312745 |
| stack | 373 | 1314973 | 1312745 |
| stack+told | 359 | 1234504 | 1232276 |
| SNOMED CT | | | |
| queue | 24013 | 23299190 | 12576268 |
| stack | 16396 | 21373957 | 12576268 |
| stack+told | 16170 | 20956134 | 12346881 |
| ChEBI | | | |
| queue | 661 | 1250852 | 1022277 |
| stack | 482 | 1250852 | 1022277 |
| stack+told | 471 | 1219692 | 991117 |
| EMAP | | | |
| queue | 8 | 27461 | 27461 |
| stack | 40 | 27461 | 27461 |
| stack+told | 6 | 13730 | 13730 |
| Molecule Role | | | |
| queue | 10 | 32857 | 32391 |
| stack | 23 | 32857 | 32391 |
| stack+told | 11 | 23640 | 23174 |
| Galen7 | | | |
| queue | 1598 | 1658475 | 1068115 |
| stack | 1246 | 1715446 | 1068115 |
| stack+told | 1197 | 1658677 | 1055492 |

**Table 2.** Performance gain obtained by optimizations. Runtimes are in miliseconds.

| | GO1 | FMA | Molecule Role | ChEBI | EMAP | SNOMED CT | Galen7 |
|---|---|---|---|---|---|---|---|
| | *classification* | | | | | | |
| ELK | 989 | 1564 | 657 | 1275 | 693 | 10674 | 2676 |
| $\mathcal{EL}$ep$\mathcal{H}$ant | 61 | 359 | 11 | 471 | 6 | 16170 | 1197 |
| | *loading + classification* | | | | | | |
| ELK | 3338 | 7522 | 2695 | 3215 | 2368 | 23169 | 4883 |
| $\mathcal{EL}$ep$\mathcal{H}$ant | 284 | 1144 | 222 | 746 | 106 | 20032 | 1554 |

**Table 3.** Loading and classification times in miliseconds.

## 4 Concluding Remarks and Future Work

We have introduced the consequence-based $\mathcal{EL}^+$ reasoner $\mathcal{EL}$ep$\mathcal{H}$ant, described implementation details and presented experimental results.

Currently the $\mathcal{EL}$ep$\mathcal{H}$ant reasoner is still under heavy construction, and there is a number of improvements that we plan to do as future work. First of all, we are going to investigate the role of ordering derived axioms for reducing the number of derivations. We are going to check whether this can be implemented with a feasible overhead. We are going to implement concurrent reasoning in order to further improve the performance. We are going to extend the supported expressivity by allowing disjointness axioms. Last but not least, we are going to implement an OWL API wrapper using the Java native interface in order to make it compatible with ontology editors and other practical applications.

## References


1. F. Baader. Terminological cycles in a description logic with existential restrictions. In G. Gottlob and T. Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 325–330. Morgan Kaufmann, 2003.
2. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In L. P. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, (IJCAI 05)*, pages 364–369. Professional Book Center, 2005.
3. F. Baader, C. Lutz, and B. Suntisrivaraporn. Is tractable reasoning in extensions of the description logic $\mathcal{EL}$ useful in practice? In *Proceedings of the Methods for Modalities Workshop (M4M-05)*, 2005.
4. F. Baader, C. Lutz, and B. Suntisrivaraporn. Is tractable reasoning in extensions of the description logic $\mathcal{EL}$ useful in practice? In *Journal of Logic, Language and Information, Special Issue on Method for Modality (M4M)*, 2007. To appear.
5. S. Brandt. On subsumption and instance problem in $\mathcal{ELH}$ w.r.t. general tboxes. In V. Haarslev and R. Möller, editors, *Proceedings of the 2004 International Workshop on Description Logics, (DL2004)*, volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.
6. S. Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and - what else? In R. L. de Mántaras and L. Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, (ECAI 2004)*, pages 298–302. IOS Press, 2004.
7. T. G. O. Consortium. Gene ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.
8. R. Cote, D. Rothwell, J. Palotay, R. Beckett, and L. Brochu. The systematized nomenclature of human and veterinary medicine. Technical report, International, Northfield, IL: College of American Pathologists, 1993.
9. K. Dentler, R. Cornet, A. ten Teije, and N. de Keizer. Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semantic Web Journal*, 2011. To appear.
10. S. Grimm, M. Watzke, T. Hubauer, and F. Cescolini. Embedded $\mathcal{EL}+$ reasoning on programmable logic controllers. In *Proceedings of the 11th International Semantic*

*Web Conference (ISWC 2012)*, volume 7650 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 2012.

11. Y. Kazakov. Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In C. Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence, (IJCAI 2009)*, pages 2040–2045, 2009.
12. Y. Kazakov, M. Krötzsch, and F. Simančík. Concurrent classification of $\mathcal{EL}$ ontologies. In L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, and E. Blomqvist, editors, *Proceedings of the 10th International Semantic Web Conference (ISWC'11)*, volume 7032 of *Lecture Notes in Computer Science*. Springer-Verlag, 2011.
13. Y. Kazakov, M. Krötzsch, and F. Simančík. ELK: a reasoner for OWL EL ontologies. System description, University of Oxford, 2012. available from http://code.google.com/p/elk-reasoner/wiki/Publications.
14. Y. Kazakov, M. Krötzsch, and F. Simančík. ELK reasoner: Architecture and evaluation. In *Proceedings of the OWL Reasoner Evaluation Workshop 2012 (ORE'12)*, volume 858 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.
15. Y. Kazakov, M. Krötzsch, and F. Simančík. The incredible ELK. 2013. available from http://code.google.com/p/elk-reasoner/wiki/Publications.
16. M. Lawley and C. Bousque. Fast classification in Protege: Snorocket as an OWL2 EL reasoner. In *Proceedings of Australasian Ontology Workshop*, 2010.
17. J. Mendex. jcel: A modular rule-based reasoner. In *Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE 2012)*, volume 858 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.
18. J. Mendez and B. Suntisrivaraporn. Reintroducing CEL as an OWL 2 EL reasoner. In B. C. Grau, I. Horrocks, B. Motik, and U. Sattler, editors, *Proceedings of the 22nd International Workshop on Description Logics (DL 2009)*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.
19. A. Rector and I. Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*. AAAI Press, 1997.
20. B. Sertkaya. In the search of improvements to the $\mathcal{EL}+$ classification algorithm. In *Proceedings of the 24th International Workshop on Description Logics (DL 2011)*, volume 745 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
21. B. Suntisrivaraporn. *Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies*. Ph.D. dissertation, Institute for Theoretical Computer Science, TU Dresden, Germany, 2009.
22. E. Thomas, J. Z. Pan, and Y. Ren. TrOWL: Tractable owl 2 reasoning infrastructure. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, (ESWC 2010)*, volume 6089 of *Lecture Notes in Computer Science*, pages 431–435. Springer-Verlag, 2010.

# Evaluating SPARQL-to-SQL translation in ontop

Mariano Rodriguez-Muro, Martin Rezk, Josef Hardi, Mindaugas Slusnys
Timea Bagosi and Diego Calvanese

KRDB Research Centre, Free University of Bozen-Bolzano
{rodriguez,mrezk, josef.hardi, mindaugas.slusnys,
timea.bagosi, calvanese}@inf.unibz.it

**Abstract.** In this paper we evaluate the performance of the SQL queries generated by ontop, a system that uses a formal approach to translate and optimize SPARQL queries and R2RML mappings. We show that the performance of ontop's SQL queries is superior to that of the performance of well known systems that rely on SQL to execute SPARQL, and superior to to that of well-known triple stores. We highlight some of the techniques and factors that allow for this.

## 1 Introduction

The integration of SPARQL and RDF with RDBMs is crucial for the adoption Semantic Technologies in industry. This importance is reflected in the creation of the R2RML [3] standard for mapping RDBMs into RDF, and in all the research focused towards translating SPARQL queries into efficient SQL over RDBs by means of mappings. This last topic is specially relevant since such techniques allow to use SPARQL and the RDF data model without costly ETL (i.e., of Extract Transform and Load from RDBs to RDF) processes, and may also allow to profit from the features of industrial strength RDBMS that are not available in triples stores, e.g., redundancy, robust transaction support, security, etc.

Execution of SPARQL with SQL is common. However, previous techniques often suffered from problems that limited their use in practice. Some techniques assume a fixed relational schema and do not support general mapping languages like R2RML, others generate complex SQL queries that do not perform well, last, some generate efficient SQL but use techniques that are not grounded formally and have limited scope. In this paper we evaluate ontop[1] [5,4], a system that tackles these issues.

ontop allows to query virtual RDF graphs defined by a relational DB and an R2RML mapping. The core of the query answering technique implemented ontop is depicted in Figure 1. In a first step (R2RML) mappings and SPARQL query are translated into a set of Datalog rules that capture the semantics of the execution of the SPARQL query over the original database. Second, the program is optimized using query containment based techniques and Semantic Query Optimization, in particular we:

[1] available at ontop.inf.unibz.it for Protege 4, OWLAPI, Sesame and stand-alone endpoint

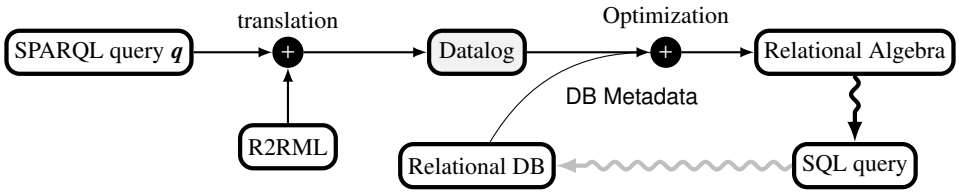



**Fig. 1.** Query answering with mappings in ontop

- use *SLD-resolution* to compute a *partial evaluation* of the program in which FILTER expressions and JOIN conditions are expressed in terms of the original database columns and not over the RDF terms constructed from these (e.g., consider that URI templates in R2RML mappings generate URI's on the fly). This will allows the DBMS to exploit any indexes defined on the original tables;
- optimize the query(ies) using Semantic Query Optimization (SQO) with respect to w.r.t. Primary Keys to avoid redundant self-joins.

Last, the optimized program is translated into an equivalent relational algebra expression, the SQL query is generated and executed by the DBMS. This rule based technique for SPARQL, R2RML and SQL provides a formal framework that gives clear guarantees w.r.t. to the semantics of the technique and which is extendible with more advanced optimizations (e.g., to other forms of constraints like Foreign Keys or Check constraints, etc.) and functionality, including OWL 2 entailment regimes (OWL 2 QL is already available in ontop and not discussed in here).

In this paper we show that the performance of the SQL queries generated by ontop using these techniques is superior to that of other systems that also perform SPARQL through SQL, and superior to that of well known triple stores.

## 2 Evaluation

This evaluation provides an overview of the performance of DB2 and MySQL while executing SQL queries generate by ontop. We use two benchmarks scenarios with a total of 36 queries and over 350 million triples. We considered two systems that offer similar functionality to ontop (i.e., SPARQL trough SQL and mappings): Virtuoso RDF Views 6.1 (open source edition) and D2RQ 0.8.1 Server over MySQL. We also compare performance with 3 well known triple stores, i.e., OWLIM 5.3, Stardog 1.2 and Virtuoso RDF 6.1 (Open Source). For ontop we used two different DB engines as backend, i.e., MySQL and DB2. The benchmarks used are:

**BSBM** The Berlin SPARQL Benchmark (BSBM) [2] evaluates the performance of query engines using use cases from e-commerce domain. The benchmark comes with a suite of tools for data generation and query execution. The benchmark also includes a relational version of the data, for which mappings can be created (D2RQ mappings are included).

**FishMark** The FishMark benchmark [1] is a benchmark for RDB-to-RDF systems that is based on an extract of the FishBase DB, a publicly available database about fish species. The benchmark comes with an extract of the database (approx. 16 M triples in RDF and SQL version), and 22 SPARQL queries obtained from the logs of FishBase. The queries are substantially larger (max 25 atoms, mean 10) than those in BSBM. Also, they make extensive use of OPTIONAL graph patterns.

The basic setup for the experiment is as follows: In the case of BSBM, out of the 12 query templates of BSBM (i.e., queries with place holders for constant values) a predefined sequence of 25 of these templates constitutes a *Query Mix*; then a BSBM run is the instantiation of a query mix with random constants and execution of the resulting queries. Performance is then measured in Query Mixes per Hour (QMpH), to compute QMpH we ran 150 query mixes, out of which 50 are considered warm up runs and their statistics are discarded. The collected statistics for QMpH over BSBM instances with 25, 100 and 200 million triples (or the equivalent in relational form). In the case of FishMark, the 22 queries are already instantiated and they constitute the query mix. We ran 150 query mixes, discarding the initial 50. In both cases, we tested with 1, 4, 8, 16 and 64 simultaneous clients.

In order to only get the performance of *SQL* queries generated by ontop we exploited ontop's simple SQL caching mechanism which stores SQL queries generated for any SPARQL query that has been rewritten previously. This allows to avoid the rewritten process completely and hence, the cost of query execution of a cached query is only the cost of evaluating the SQL query over the DBMS. To force the use of this cache, we re-ran the BSBM benchmark 5 more times (and averaged the results). For FishMark, this was not necessary since the queries are always the same. All experiments were conducted on a HP Proliant server with 24 Intel Xeon CPUs (144 cores @3.47GHz), 106GB of RAM and a 1TB 15K RPM HD. The OS is Ubuntu 12.04 64-bit edition. All the systems run as SPARQL end-points. All configuration files are available online[2]. The results are summarized in Figure 2.

**Discussion.** First we note that the D2RQ server always ran out of memory, timed out in some queries or crashed. This is why it doesn't appear in our summary table. D2RQ's SPARQL-to-SQL technique is not well documented, however, by monitoring the queries being sent by D2RQ to MySQL, it appears that D2RQ doesn't translate the SPARQL query into a single SQL query, instead it computes multiple queries and retrieves part of the data from the database. We conjecture that D2RQ then uses this data to compute the results. Such approach is, in general, limited in scalability and prone to large memory consumption, being the last point the reason for the observed behavior. Also, Virtuoso Views is not included in the FishMark benchmark because it provided wrong results, we reported this to the developers which confirmed the issue. Also, we did not run ontop with DB2 for FishMark due to errors during data loading.

Next, we can see is that for BSBM in almost every case, the performance obtained with ontop's SQL queries executed by MySQL or DB2 outperforms all other systems by a large margin. The only cases in which this doesn't hold are when the number of

[2] `https://babbage.inf.unibz.it/trac/obdapublic/wiki/BSBMFISH13aBench`

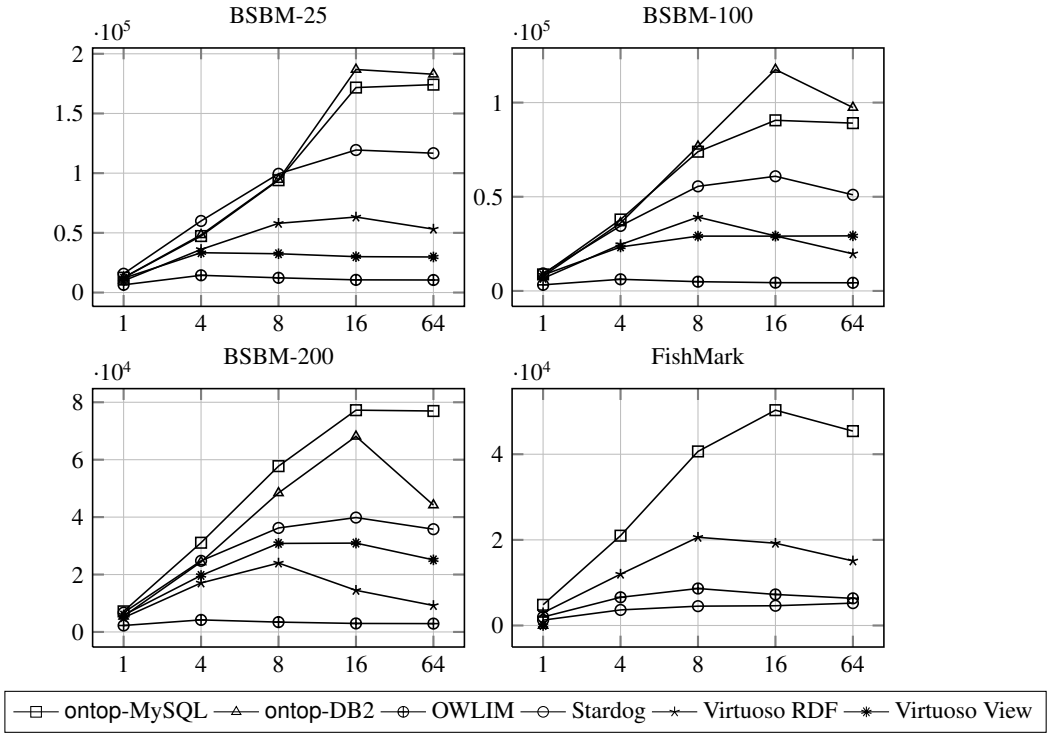



**Fig. 2.** Query performance comparison summary. X axis = parallel clients, Y axis = Query Mixes per Hour (QMpH, higher is better)

clients is less than 16 and the dataset is small (BSBM 25). This can be explained as follows.

Note that in ontop performance can be divided in three parts, *(i)* the cost of generating the SQL query , *(ii)* the cost of execution over the RDBMs and *(iii)* cost of fetching and transforming the SQL results into RDF terms. When the queries are cached, *(i)* is absent, and if the scenario includes little data (i.e., BSBM 25), the cost of *(ii)*, both for MySQL and DB2, is very low and hence *(iii)* dominates. We attribute the performance difference to a poor implementation of *(iii)* in ontop, and the fact triple stores do not need to perform this step.

From 16 parallel clients however, executing ontop's SQL queries with MySQL or DB2 outperforms other systems by a large margin. We attribute this to DB2's and MySQL's better handling of parallel execution (i.e., better transaction handling, table locking, I/O, caching, etc.).

When the datasets are larger, e.g., BSBM 100 and 200, for ontop *(i)* stays the same. From *(ii)* and *(iii)* we have that the former dominates since in both benchmarks queries return few results. Then, again, we have that MySQL's an DB2's advantage over triple stores can be attributed to similar reasons as before, better I/O, planning, caching, etc.

This is witnessed by the fact that DB2 and MySQL with ontop's SQL outperform the rest already at 1 single client for BSBM 100 and BSBM 200.

These experiments do not allow to fully see the benefit of the optimizations on SQL that is performed by ontop; this would require to be able to enable and disable them, and in ontop this is not possible at the moment. However, some observations are possible, in particular we can see the strong effect of SELF JOIN elimination by Primary Keys. Consider the FishMark benchmark that has little data, only 16M triples, but in which in almost all queries ontop's SQL executed over MySQL (we didn't run DB2 in this case) outperforms the rest almost in every case even from 1 single client. In this setting, 1 client and little data, the cost of *(ii)* falls in the cost of planning and executing JOINs and LEFT JOINs by the DBMS or triple store. At the same time, in FishMark, the original tables are structured in such a way that many of the SPARQL JOINs can be simplified dramatically when expressed as optimized SQL. For example, consider the FishMark query:

```
SELECT ?order ?family ?genus ?species ?occ ?name ?gameref ?game
WHERE {
 ?ID fd:cComName ?name; fd:coC_Code ?ccode; fd:cSpecCode ?x.
 ?x  fd:sGenus ?genus; fd:sSpecies ?species; fd:sGameFish ?game;
     fd:sGameRef ?gameref; fd:sFamCode ?f .
 ?f  fd:fFamily ?family; fd:fOrder ?order .
 ?c  fd:cSpecCode ?x; fd:cStatus ?occ; fd:cC_Code ?cf;
     fd:cGame 1 . ?cf fd:cPAESE "Indonesia" . }
```

This query expresses a total of 16 Join operations. When translated into SQL, ontop is able to generate the following query:

```
SELECT V3.FamilyOrder AS order, V3.Family AS family,
 V1.Genus AS genus, V1.Species AS species, V4.Status AS occ,
 V1.ComName AS name, V1.GameRef AS gameref, V1.GameFish AS game
FROM species V1, comnames V2, families V3, country V4, countref V5
WHERE V1.SpecCode = V2.SpecCode AND V4.Game = 1 AND V5.PAESE =
 'Indonesia' AND V4.C_Code = V5.C_Code AND V1.Genus = V8.Genus
```

A simple and flat SQL query (easy to execute) with a total of 3 Joins. Note that the use of a large number of JOIN operations is intrinsic to SPARQL since the RDF data model is ternary. However, if the data is stored in a n-ary schema (as usual in RDBMs), ontop can use semantic query optimization w.r.t. primary keys to construct the optimal query over the n-ary tables. Triple stores has no means to do this since data is de-normalized once it is transformed into RDF.

In BSBM this optimization is weaker since queries are smaller and have fewer joins, however a trend pointing to this same observation can be seen. Consider the results for Q2, Q3 and Q4 in Table 2 Q2, Q3 and Q4.

**Conclusions.** In this evaluation we confirmed that in BSBM and FishMark, the SQL queries generated by ontop (executed by 2 representative RDBMS) can provide much better performance than that of executing SPARQL queries executed over similar SQL-based systems (D2RQ, Virtuoso Views) and well known triple stores. This is a strong

| | ontop-MySQL | ontop-DB2 | OWLIM | Stardog | V. RDF | V. Views |
|---|---|---|---|---|---|---|
| Q1 | 3,22k | 1,28 k | 1,43k | 1,42 | 23 | 45 |
| Q2 | 1,40k | 928 | 276 | 790 | 123 | 315 |
| Q3 | 1,92k | 1,12k | 111 | 543 | 155 | 341 |
| Q4 | 1,53k | 955 | 295 | 669 | 140 | 265 |
| Q5 | 27 | 72 | 2 | 29 | 14 | 48 |
| Q6 | - | - | - | - | - | - |
| Q7 | 9,78k | 1,59k | 541 | 400 | 101 | 316 |
| Q8 | 13,99k | 1,68k | 3,22k | 648 | 96 | 180 |
| Q9 | 13,62k | 1,01k | 3,7k | 1,99k | 59 | 188 |
| Q10 | 20,22k | 2,04k | 3,9k | 610 | 228 | 305 |
| Q11 | 20,75k | 2,04k | 3,52k | 1,89k | 1,66k | 1,13k |
| Q12 | 11,34k | 1,64k | 5,99k | 1,4k | 1,25k | 1,19k |
| QueryMix | 44,19k | 76,96k | 2,91k | 35,79k | 9,21k | 25,11k |

**Fig. 3.** Summary of results (per query) for BSBM-200 with 64 clients. Individual queries are in *queries per second*, totals are in *query mixes per hour*

pointer that on-the-fly SPARQL-to-SQL translation might be a much better option than the ETL approach in some use cases, e.g., when the data is already stored in a relational schema and the SPARQL queries do not use advanced SPARQL features (i.e., regular paths).

Several points were neither discussed nor evaluated in this paper. For example, the cost of generating the SQL queries is at the moment very high in ontop; in our experiments we observed a dramatic performance drop when considering the SQL generation process, often putting ontop at the same level of performance than the worst of the other systems. This observation calls for several points of action. First, we need to understand how much of this cost can be attributed to poor implementation (ontop is an academic system after all). Second, the template based queries involved in BSBM and FishMark can be considered a common use case, e.g., applications often repeat the same queries over and over, only variating some constants. With this in mind, a *prepared-statement like approach* or a more intelligent caching mechanism for query rewritings could be devised so that the cost of SQL generation is removed.

Last, we note the choice of the optimizations and techniques implemented in ontop is not arbitrary, they are guided by empirical experiences on what constitute efficient SQL on a wide range of modern RDBMs engines. The evaluation presented here shows that those choices, as a whole, seem to be *on the right track*; however, it doesn't provide a deep understanding of the individual benefit of each of these choices and optimizations and further work in this direction is required.

## References

1. Samantha Bail, Sandra Alkiviadous, Bijan Parsia, David Workman, Mark van Harmelen, Rafael S. GonÃ*S*alves, and Cristina Garilao. FishMark: A linked data application benchmark. In *Proc. of the Joint Workshop on Scalable and High-Performance Semantic Web Systems*

*(SSWS+HPCSW 2012)*, volume 943, pages 1–15. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2012.

2. Christian Bizer and Andreas Schultz. The Berlin SPARQL benchmark. *Int. Journal On Semantic Web and Information Systems*, 5(2):1–24, 2009.
3. Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF mapping language. http://www.w3.org/TR/r2rml/, September 2012.
4. Mariano Rodriguez-Muro and Diego Calvanese. Quest, an owl 2 ql reasoner for ontology-based data access. In *Proc. of the 9th Int. Workshop on OWL: Experiences and Directions (OWLED 2012)*, volume 849 of *CEUR Electronic Workshop Proceedings, http://ceur-ws.org/*, 2012.
5. Mariano Rodriguez-Muro, Josef Hardi, and Diego Calvanese. Quest: Effcient sparql-to-sql for rdf and owl. In *Proc. of the ISWC 2012 Posters Demonstrations Track (ISWC-PD 2012)*, volume 914 of *CEUR Electronic Workshop Proceedings, http://ceur-ws.org/*, 2012.

# OBDA with *Ontop*

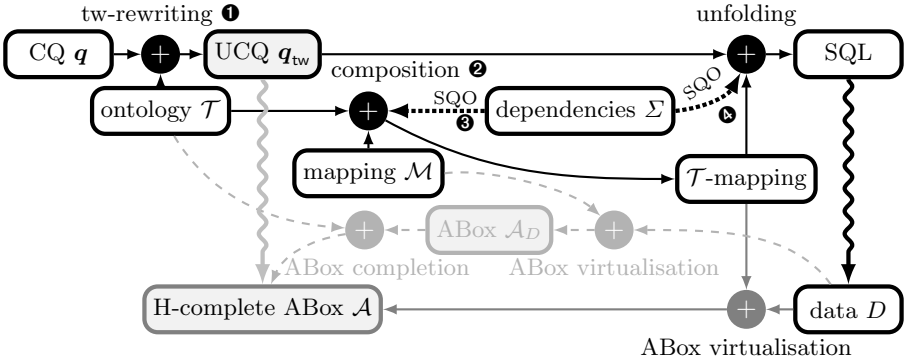
Mariano Rodríguez-Muro[1], Roman Kontchakov[2] and Michael Zakharyaschev[2]

[1] Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
[2] Dept. of Computer Science and Inf. Syst., Birkbeck, University of London, U.K.

**Abstract.** We evaluate the performance of the OBDA system *Ontop* and compare it with other systems. Our experiments show that (*i*) the *Ontop* tree-witness query rewriter is fast and outperforms the competitors, and (*ii*) query evaluation in *Ontop* using the semantic index is as efficient as in materialisation-based systems (but without the materialisation overhead).

## 1 Introduction

*Ontop* (`ontop.inf.unibz.it`) is an ontology-based data access (OBDA) system implemented at the Free University of Bozen-Bolzano and available as a plugin for Protégé 4, SPARQL end-point and OWLAPI and Sesame libraries. The architecture of *Ontop* is as follows:




A user formulates a conjunctive query (CQ) $\boldsymbol{q}$ in the signature of an OWL 2 QL ontology $\mathcal{T}$. The ontology $\mathcal{T}$ is related to the data $D$ by means of a GAV mapping $\mathcal{M}$, which is a collection of database queries defining the vocabulary of $\mathcal{T}$. The mapping $\mathcal{M}$ could be applied to the data $D$ to obtain the so-called *virtual ABox* $\mathcal{A}_D$ [9]. *Ontop* does not materialise it. Instead, it constructs a $\mathcal{T}$-mapping [9] by taking the composition (❷) of $\mathcal{M}$ with the taxonomy in $\mathcal{T}$ and simplifying it using SQL features (disjunction and interval expressions) and Semantic Query Optimisation (SQO ❸) with database integrity constraints (dependencies $\Sigma$). By applying the resulting $\mathcal{T}$-mapping to the data $D$, we could obtain another virtual ABox, $\mathcal{A}$, which is *H-complete with respect to* $\mathcal{T}$ in the sense that:

$$A(a) \in \mathcal{A} \quad \text{if} \quad A'(a) \in \mathcal{A}, \mathcal{T} \models A' \sqsubseteq A \quad \text{or} \quad R(a,b) \in \mathcal{A}, \mathcal{T} \models \exists R \sqsubseteq A,$$
$$P(a,b) \in \mathcal{A} \quad \text{if} \quad R(a,b) \in \mathcal{A} \text{ and } \mathcal{T} \models R \sqsubseteq P.$$

*Ontop* constructs the tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}$ of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes [5], and then uses the $\mathcal{T}$-mapping to unfold $\boldsymbol{q}_{\mathsf{tw}}$ to an SQL query, which is then simplified with SQO (❹) and passed to a database system for evaluation.

In the remainder of the paper, we evaluate the performance of *Ontop* and compare it with other OBDA systems (more details can be found at `tinyurl.com/ontop-benchmark` and `www.dcs.bbk.ac.uk/~roman/tw-rewriting`). We begin by analysing the structure of tree-witness rewritings over H-complete ABoxes and show that dealing with concept/role hierarchies (taxonomies) is the most critical component of any OBDA system. We then concentrate on a particular shape of $\mathcal{T}$-mappings, called the semantic index [9], which is most suitable for triple stores or data in the form of universal tables in a database and which allows *Ontop* to deal with taxonomies efficiently.

## 2 Tree Witnesses: The Topology of *Ontop* Rewritings

We have run the *Ontop* tree-witness rewriter on the usual set of CQs and ontologies: Adolena, StockExchange and the extension $\text{LUBM}^{\exists}_{20}$ [6] of LUBM [7] tailored to stress query answering techniques for OWL 2 QL. Our aim was to understand the size of the *topological* part of the rewritings reflecting matches (tree witnesses) in the anonymous part of the canonical models (as opposed to the taxonomical one). The tables below provide the statistics on the tree-witness rewritings over H-complete ABoxes: the number of tree witnesses, the number of CQs in the rewriting (which is a UCQ), the number of atoms in the original query, and the number of atoms in each of the CQs in the rewriting.

| | Adolena | | | | | StockExchange | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s'_1$ | $s'_2$ | $s'_3$ | $s'_4$ | $s'_5$ |
| tree witnesses | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| CQs in $\boldsymbol{q}_{\mathsf{tw}}$ | **2** | **2** | **1** | **2** | **1** | **1** | **1** | **1** | **1** | **1** |
| atoms in $\boldsymbol{q}$ | 2 | 3 | 5 | 3 | 5 | 1 | 3 | 5 | 5 | 7 |
| atoms in $\boldsymbol{q}_{\mathsf{tw}}$ | 2+2 | 1+3 | 5 | 2+3 | 5 | 1 | 1 | 3 | 2 | 4 |

For $\text{LUBM}^{\exists}_{20}$, $r_1$–$r_5$ are the queries from the Requiem evaluation [7], $q_1$–$q_6$ from the combined approach evaluation [6], and $q_7$–$q_9$ from the Clipper evaluation [11]:

| | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $q_1$ | $q_2$ | $q_3$ | $q_4$ | $q_5$ | $q_6$ | $q_7$ | $q_8$ | $q_9$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tree witnesses | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 3 | 1 |
| CQs in $\boldsymbol{q}_{\mathsf{tw}}$ | **1** | **1** | **1** | **1** | **1** | **2** | **2** | **1** | **2** | **1** | **1** | **1** | **1** | **1** |
| atoms in $\boldsymbol{q}$ | 2 | 3 | 6 | 3 | 4 | 8 | 4 | 6 | 8 | 5 | 8 | 13 | 13 | 34 |
| atoms in $\boldsymbol{q}_{\mathsf{tw}}$ | 2 | 1 | 4 | 1 | 2 | 4+6 | 3+4 | 5 | 5+8 | 4 | 6 | 12 | 6 | 33 |

Note first that these CQs and ontologies have very few tree witnesses. More precisely, in 67% of the cases there are *no tree witnesses* at all, and in 29% we have only one tree witness. Even for the specially designed $q_8$, the structure of tree witnesses is extremely simple (they do not even overlap). Note also that, although $q_8$ and $q_9$ do have tree witnesses, the resulting UCQs contain only one CQ because these tree witnesses are generated by other atoms of the queries.

The size of each of the CQs in the rewritings does not exceed the size of the input query; moreover, the domain/range optimisation simplifies the obtained CQs further for $r_2$–$r_5$, $q_1$, $q_3$, $q_5$–$q_8$ and most of the StockExchange queries. To illustrate, consider the following subquery of $q_8$:

$$\boldsymbol{q}_0(x_0) = \mathit{Publication}(x_0),\ \mathit{publicationAuthor}(x_0, x_{11}),\ \mathit{Subj1Professor}(x_{11}),\\ \mathit{worksFor}(x_{11}, x_{12}),\ \mathit{Department}(x_{12}),$$

where $x_{11}$, $x_{12}$ do not occur in the remainder of $q_8$. This CQ has a tree witness with the last two atoms because of the $\text{LUBM}^{\exists}_{20}$ axiom $\mathit{Faculty} \sqsubseteq \exists \mathit{worksFor}$. However, *Subj1Professor* is a subconcept of *Faculty*, and so any of its instances will anyway be connected to *Department* by *worksFor* (either in the ABox or in the anonymous part). It follows that the last two atoms of $\boldsymbol{q}_0$ do not change answers to the CQ and can be ignored. The first atom is also redundant because the ontology contains the domain axiom $\exists \mathit{publicationAuthor} \sqsubseteq \mathit{Publication}$. As $\boldsymbol{q}_0$ represents a natural and common pattern for expressing queries—select a *Publication* whose *publicationAuthor* is a *Subj1Professor*, etc.—any OBDA system should be able to detect such redundancies automatically. Finally, we note that all of the rewritings in our experiments contain at most two CQs.

For comparison, we computed the rewritings of the CQs over $\text{LUBM}^{\exists}_{20}$ using Requiem [7], Nyaya [4], IQAROS (v 0.2) [10], Clipper (v 0.1) [3] and Rapid (v 0.3) [2]. The first 3 return UCQ rewritings, while the last 2 nonrecursive datalog rewritings, with Rapid having an option to rewrite into UCQs, too. The rewritings are over arbitrary ABoxes; to make them comparable with *Ontop*, in the last 2 cases we omitted the taxonomical rules for completing the ABoxes.

| | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $q_1$ | $q_2$ | $q_3$ | $q_4$ | $q_5$ | $q_6$ | $q_7$ | $q_8$ | $q_9$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UCQ (number of CQs) | | | | | | | | | | | | | | |
| Requiem | 2 | 1 | 23 | 2 | 10 | DNF | 2 | DNF | 14,880 | 690 | DNF | DNF | DNF | DNF |
| Nyaya | 2 | 1 | 23 | 2 | 10 | DNF | 2 | DNF | DNF | 690 | DNF | DNF | 8 | DNF |
| IQAROS | 2 | 1 | 23 | 2 | 10 | DNF | **1** | 15,120 | 14,400 | 990 | 23,552 | DNF | DNF | DNF |
| Rapid | 2 | 1 | 23 | 2 | 10 | 3,887 | 2 | 15,120 | 14,880 | 690 | 23,552 | DNF | DNF | 16 |
| datalog (number of non-taxonomical rules) | | | | | | | | | | | | | | |
| Rapid | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 1 | 2 | 1 | 1 | 1 | 27 | 1 |
| Clipper | 1 | 1 | 1 | 1 | 1 | 8 | 7 | 1 | 5 | 1 | 1 | 1 | 512 | 16 |
| tw-rewriter | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 2 | 1 | 1 | 1 | 1 | 1 |

The UCQs returned by Requiem, Nyaya, IQAROS and Rapid correspond to our tree-witness rewritings with every concept/role replaced by all of its subconcept/subroles (IQAROS's rewritings of $q_2$ and possibly of $q_4$, $q_5$ are incorrect): for instance, $q_7$ produces 216,000 ($= 30^3 \times 2^3$) CQs, $q_3$ produces 15,120 ($=4\times5\times21\times36$) CQs and $q_1$ produces 3,887 ($= 23 + 2\times4\times21\times23$) CQs because *Student*, *Faculty* and *Professor* have 23, 36 and 30 subconcepts, respectively, *worksFor* has 2 subroles, etc. Evidently, these large UCQs are generated by the taxonomies and none of the query subsumption algorithms can reduce the number of CQs in them. In fact, all four systems require substantial resources (in particular, for query subsumption checks): e.g., Nyaya rewrites $q_8$ in 91s and runs out of memory on $q_1$, $q_3$, etc.; IQAROS rewrites $q_6$ in 546s and runs out

of memory on $q_1$, $q_7$–$q_9$; Rapid rewrites $q_6$ in 247s, $q_9$ in 90.2s and runs out of memory on $q_7$, $q_8$ (but is quite fast in all other cases); Requiem takes 232s on $r_3$, 236s on $q_4$, 64.7s on $q_5$ and does not terminate in 600s on $q_1$, $q_3$, $q_6$–$q_8$.

On the other hand, *Ontop* and the datalog-based systems produce rewritings very quickly: Rapid needs $< 0.1$s for all CQs except $q_8$ (0.41s) and $q_9$ (10.8s); it was impossible to extract the rewriting time in Clipper, but each query was processed in $< 2$s; *Ontop* computes the rewritings in $< 0.025$s (except the last two that take $< 0.055$s). Clearly, the huge UCQs could be produced in fractions of a second by simply unfolding a few CQs by means of the taxonomy. Interestingly, Clipper and Rapid return single-CQ rewritings in the cases without tree witnesses, but generate more CQs than *Ontop* (e.g., $q_8$ and $q_9$) otherwise.

## 3 $\mathcal{T}$-mappings: Concept and Role Hierarchies

We compare the query execution time in *Ontop*, Stardog 1.2 [8] and OWLIM [1]. Both Stardog and OWLIM use internal data structures to store data in the form of triples. Stardog is based on rewriting into UCQs (as we saw in Section 2, such systems can run out of memory during the rewriting stage, even before accessing data). OWLIM is based on materialising the inferences (forward chaining); but the implemented algorithm is incomplete for OWL 2 QL [1]. In the presented experiments, *Ontop* uses DB2 and MySQL to store the data as triples based on the *semantic index* technique, which chooses concept/role IDs in such a way that the resulting $\mathcal{T}$-mapping uses SQL interval expressions (e.g., (ID>=1) AND (ID<=10)) rather than UNIONs to obtain all instances of a concept including its subconcepts (and similarly for roles). It was impossible to compare *Ontop* with other systems: Rapid and IQAROS are just query rewriting algorithms; the publicly available Clipper v 0.1 supports only Datalog engines that read queries and triples at the same time, which would be a serious disadvantage for large datasets. All experiments were run on an HP Proliant with 144 cores 3.47GHz in 24 Intel Xeon CPUs, 106GB RAM and a 1TB@15000rpm HD under 64-bit Ubuntu 11.04 with Java 7, MySQL 5.6 and DB2 10.1.

We took $\text{LUBM}^{\exists}_{20}$ with the data created by the modified LUBM data generator [6] for 50, 100 and 200 universities (with 5% incompleteness) with 7m, 14m and 29m triples, respectively. OWLIM requires a considerable amount of time for loading and materialising the inferences—14min, 33min and 1h 23min, respectively—producing about 93% additional triples and resulting in 13m, 26m and 52m triples (neither Stardog nor *Ontop* need an expensive loading stage). The results of executing the queries from Section 2 are presented in Table 1. We first note that Stardog runs out of memory on 50% of queries, with a likely cause being the query rewriting algorithm, which is an improved version of Requiem (cf. the results in Section 2). On the remaining queries, however, Stardog is fast, which might be due to its optimised triple store. In contrast to Stardog, both OWLIM and *Ontop* return answers to all queries (although the former is incomplete) and their performance is comparable: in fact, in 76% of the cases *Ontop* with DB2 outperforms OWLIM (the lines 'DB2 *' give the execution times for

| | | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $q_1$ | $q_2$ | $q_3$ | $q_4$ | $q_5$ | $q_6$ | $q_7$ | $q_8$ | $q_9$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | 50 universities | | | | | | | | |
| *Ontop* | DB2 fetch | 0 | 0.02 | 0.76 | 0.04 | 0 | 1.38 | 4.16 | 0 | 1.50 | 1.77 | 1.24 | 1.83 | 1.47 | 0 |
| *Ontop* | DB2 * | **0** | **0.11** | **0.93** | **0.03** | **0** | 125.0 | 0.86 | **0** | 0.39 | **2.13** | **0.24** | **0.21** | 0.48 | **0** |
| *Ontop* | MySQL | **0** | 1.53 | 12.40 | 1.57 | **0** | 681.6 | 6.91 | **0** | 1.76 | 5.52 | 1.26 | 8.88 | 2.69 | **0** |
| | OWLIM | 0.00 | 1.85 | 2.81 | 0.58 | 0.16 | **20.64** | 2.83 | 0.24 | **0.34** | 6.16 | 0.87 | 0.26 | **0.27** | 0.04 |
| | Stardog | 0.01 | 0.79 | 1.56 | 0.34 | 0.10 | DNF | **0.10** | DNF | DNF | DNF | DNF | DNF | DNF | 0.04 |
| | result size | – | 102k | 12k | 34k | – | 1.1m | – | – | – | 302k | – | – | – | – |
| | | | | | | | 100 universities | | | | | | | | |
| *Ontop* | DB2 fetch | 0 | 0.02 | 0.81 | 0.05 | 0 | 1.78 | 5.06 | 0 | 2.15 | 3.41 | 1.36 | 1.85 | 2.11 | 0 |
| *Ontop* | DB2 * | **0** | **0.28** | **1.38** | **0.29** | **0** | 131.0 | 5.15 | **0** | 2.24 | **3.98** | **1.09** | 1.26 | 2.10 | **0** |
| *Ontop* | MySQL | **0** | 2.98 | 24.89 | 2.90 | **0** | 1445 | 12.63 | **0** | 3.37 | 11.08 | 2.52 | 17.71 | 5.19 | **0** |
| | OWLIM | 0.00 | 3.72 | 5.51 | 1.20 | 0.38 | **41.12** | 5.82 | 0.49 | **0.67** | 12.92 | 1.83 | **0.51** | **0.55** | 0.04 |
| | Stardog | 0.01 | 1.78 | 2.56 | 0.60 | 0.38 | DNF | **0.20** | DNF | DNF | DNF | DNF | DNF | DNF | 0.03 |
| | result size | – | 205k | 24k | 69k | – | 2.4m | – | – | – | 607k | – | – | – | – |
| | | | | | | | 200 universities | | | | | | | | |
| *Ontop* | DB2 fetch | 0 | 0.02 | 0.97 | 0.05 | 0 | 2.05 | 6.37 | 0 | 3.97 | 6.84 | 1.47 | 1.86 | 5.97 | 0 |
| *Ontop* | DB2 * | **0** | **0.34** | **1.79** | **0.37** | **0** | 157.0 | 6.26 | **0** | **3.79** | **7.94** | **1.20** | 1.26 | 5.83 | **0** |
| *Ontop* | MySQL | **0** | 5.73 | 58.24 | 7.78 | **0** | 2888 | 27.68 | **0** | 6.66 | 20.96 | 4.35 | 36.21 | 11.16 | **0** |
| | OWLIM | 0.00 | 8.49 | 11.85 | 2.73 | 0.64 | **95.87** | 11.37 | 1.03 | 17.64 | 29.61 | 3.60 | **1.01** | **1.00** | 0.04 |
| | Stardog | 0.01 | 3.27 | 2.92 | 1.12 | 0.27 | DNF | **0.33** | DNF | DNF | DNF | DNF | DNF | DNF | 0.06 |
| | result size | – | 410k | 48k | 137k | – | 4.7m | – | – | – | 1.2m | – | – | – | – |

**Table 1.** Query execution time (in seconds) and the result size over $\text{LUBM}^{\exists}_{20}$.

queries that return the size of the result). Moreover, some of the slowest queries return enormous results (e.g., 4.7m tuples for $q_1$ over 200 universities). Such queries are hardly typical for databases, and both DB2 and MySQL show a significant degradation in performance. However, as can be seen from the lines 'DB2 fetch' that give the time required to plan the query and fetch the first 500 answers (which does not depend much on the size of the result), DB2 is very fast. It is to be emphasised that *Ontop* can work with a variety of database engines and that, as these experiments demonstrate, *Ontop* with MySQL is considerably worse in executing queries than with DB2 (but is still competitive with OWLIM). Finally, observe that some queries do not need evaluation because *Ontop* simplifies them to empty queries: in fact, $r_1$, $r_5$, $q_3$, $q_6$ contain atoms that have no instances in the generated data and only 5 out of the 14 CQs return any answers (which probably reflects the artificial nature of the benchmark).

These experiments confirm once again that rewritings into UCQs over arbitrary ABoxes can be prohibitively large even for high-performance triple stores such as Stardog. The materialisation approach should obviously cope with large taxonomies. However, the semantic index used in *Ontop* is able to cope with this problem as well as (and often better than) inference materialisation, but does not incur its considerable extra costs.

We have also evaluated the performance of $\mathcal{T}$-mappings when answering queries over the Movie Ontology (MO, `www.movieontology.org`) and the data

from the SQL version of the Internet Movie Database (IMDb, `www.imdb.com/interfaces`). The reader can find all the results at `tinyurl.com/ontop-benchmark`. Those experiments demonstrate that on-the-fly inference over real databases by means of the tree-witness rewriting and $\mathcal{T}$-mappings is efficient enough to compete (and often outperform) materialisation-based techniques. Moreover, the usual problems associated with approaches based on query rewriting do not affect *Ontop*: $\mathcal{T}$-mappings efficiently deal with hierarchical reasoning, avoiding the exponential blowup, which is usually associated with query rewriting, and the SQO is able to improve performance of the produced SQL queries by taking account of the structure and integrity constraints of the database.

## 4 Conclusions

To conclude, this paper shows that—despite the negative theoretical results on the worst-case query rewriting and sometimes disappointing experiences of the first OBDA systems—high-performance OBDA is achievable in practice when applied to standard ontologies, queries and data stored in relational databases. In such cases, query rewriting together with SQO and SQL optimisations is fast, efficient and produces SQL queries of high quality whose performance makes materialisation of inferences unnecessary in the OWL 2 QL setting.

**Acknowledgements.** We thank Giorgio Orsi for providing Nyaya, Guohui Xiao for providing queries and the *Ontop* development team (Josef Hardi, Timea Bagosi and Mindaugas Slusnys) for their help with the experiments.

## References

1. B. Bishop and S. Bojanov. Implementing OWL 2 RL and OWL 2 QL rule-sets for OWLIM. In *Proc. of OWLED*, 2011.
2. A. Chortaras, D. Trivela, and G. Stamou. Optimized query rewriting for OWL 2 QL. In *Proc. of CADE-23*, volume 6803 of *LNCS*, pages 192–206. Springer, 2011.
3. T. Eiter, M. Ortiz, M. Šimkus, T.-K. Tran, and G. Xiao. Query rewriting for Horn-SHIQ plus rules. In *Proc. of AAAI 2012*. AAAI Press, 2012.
4. G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proc. of ICDE 2011*, pages 2–13. IEEE Computer Society, 2011.
5. S. Kikot, R. Kontchakov, and M. Zakharyaschev. Conjunctive query answering with OWL 2 QL. In *Proc. of KR 2012*. AAAI Press, 2012.
6. C. Lutz, İ. Seylan, D. Toman, and F. Wolter. The combined approach to OBDA: Taming role hierarchies using filters. In *Proc. of SSWS+HPCSW*, 2012.
7. H. Pérez-Urbina, B. Motik, and I. Horrocks. A comparison of query rewriting techniques for DL-lite. In *Proc. of DL 2009*, volume 477 of *CEUR-WS*, 2009.
8. H. Pérez-Urbina, E. Rodríguez-Díaz, M. Grove, G. Konstantinidis, and E. Sirin. Evaluation of query rewriting approaches for OWL 2. In *SSWS+HPCSW*, 2012.
9. M. Rodríguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work. In *Proc. of AMW 2011*, volume 749. CEUR-WS.org, 2011.
10. T. Venetis, G. Stoilos, and G. Stamou. Query extensions and incremental query rewriting for OWL 2 QL ontologies. *Journal on Data Semantics*, 2013.
11. G. Xiao. Personal communication, 2013.

# Reasoning the FMA Ontologies with TrOWL

Jeff Z. Pan, Yuan Ren, Nophadol Jekjantuk, and Jhonatan Garcia

Department of Computing Science, University of Aberdeen,
Aberdeen, AB23 3UE, United Kingdom

**Abstract.** In this paper, we briefly introduce the TrOWL ontology reasoning infrastructure and share our experience of using TrOWL to reason with various versions of the Foundational Model of Anatomy Ontology (FMA), which are among the most challenging ontologies for Description Logic reasoners.

## 1 TrOWL

TrOWL[1] is a tractable reasoning infrastructure for the second version of the Web Ontology Language, or simply OWL2[2], which comes with a family of ontology languages, including:

- OWL2-DL, the most expressive decidable language in the OWL2 family, and
- three tractable sub-languages of OWL2-DL, i.e. OWL2-EL, OWL2-QL and OWL2-RL.

There are at least three approaches to reasoning in OWL2:

1. Sound and complete reasoning in OWL2-DL. Until recently, no OWL2-DL reasoners could classify the FMA ontology, due to the high worst case complexity of OWL2-DL (2NEXP-TIME-complete). In 2010, Glimm et. al. [2] proposed the core blocking optimisation, enabling HermiT[3] to classify the TBox of the FMA-Constitutional ontology in about 30 minutes [4].
2. Sound and complete reasoning in OWL2-EL, OWL2-QL and OWL2-RL. Although these sub-languages are tractable, none of them are sufficiently expressive to cover FMA.
3. Approximate reasoning for OWL2-DL. The idea here is to approximate OWL2-DL ontologies to those in its tractable sub-languages, so as to exploit the efficient and scalable reasoner. In Sections 3 and 4 of the paper, we will provide more details on the performance of our approximate reasoner in TrOWL for the FMA ontologies.

TrOWL supports OWL2 by using the approaches 2 and 3 mentioned above. On the tractable language level, TrOWL contains an OWL2-EL reasoner (REL)

[1] http://trowl.eu/
[2] http://www.w3.org/TR/owl2-overview/
[3] http://www.hermit-reasoner.com/

and an OWL2-QL reasoner (Quill). The approach of TrOWL is to offer tractable support for all the expressive power of OWL2 by using quality guaranteed (in terms of soundness and/or completeness) approximate reasoning. TrOWL contains an OWL2 profile checker to detect which profile an ontology may fit into.

The semantic approximation from OWL2 to OWL-QL is based on the work described in Pan and Thomas [11]. Semantic approximation applies the knowledge compilation [18] to precompute the entailment of an arbitrary ontology into a DL-Lite ontology. In general, this approach is soundness preserving and could be incomplete. Furthermore, a conditional completeness condition is identified: if input queries are database style queries, i.e. the variables only bound to named individuals, the approach is also complete. In other words, database style queries are not expressive enough to tell the difference between the original OWL2-DL ontology and its approximation. A drawback for this approach is that reasoners are required to compute the semantic approximation; therefore, the construction of the approximation is usually done off-line.

The syntactic approximation from OWL2 to OWL2-EL is based on the soundness preserving approximate reasoning approach presented in Ren et al. [14]. The construction of the approximation is on the syntax and hence can be done efficiently just before applying approximate reasoning. The idea is not to throw away the axioms that are beyond OWL2-EL; otherwise, we might suffer from low recall — for example, if we naively remove from the Cyc ontology all the axioms that are beyond OWL2-EL, the recall is only 1% for classification. Therefore, in this approach, we introduce some fresh named classes to represent non-OWL2-EL class expressions. In order to recover the hidden semantics within these fresh named classes, some relation between such named classes and existing classes are maintained and some extra completion rules (beyond those in OWL2-EL) are introduced, with the extended set of completion rules still being tractable. In [14], we reported that the recall of such approximate reasoning is very high for TBox classification, 100% for most existing benchmark ontologies except the Wine Ontology (99.4%). A further investigation indicates that it is due to the syntax sensitivity nature of our approach. After adding a further normalisation step into TrOWL, the recall for the Wine Ontology is also 100%.

TrOWL supports both OWL and Jena APIs. It has a plug-in for the Protégé ontology editor v4.3.

## 2 FMA Ontologies

The Foundational Model of Anatomy Ontology (FMA) [16] is an evolving computer-based knowledge source for biomedical informatics, mainly developed by the University of Washington since 1994. The importance of this ontology resides in the fundamental underlining of anatomy in all fields of medicine. Proper interpretation of these data relies on an implicit understanding of anatomy. The inferences entailed in such reasoning call upon cognitive or computational processing of abstractions about physical entities of the body, making use of relationships that exist among anatomical concepts. Relevance and impact of the

| File | Source | File size | Cls | Props | Inds | Expressivity |
|---|---|---|---|---|---|---|
| FMA-DLR[4] | bioontology.org | 147.6 Mb | 78989 | 110 | 139374 | $\mathcal{ALUIN}(\mathcal{D})$ |
| FMA-FullR[4] | bioontology.org | 37.7 Mb | 23597 | 77 | 82935 | $\mathcal{ALCOF}(\mathcal{D})$ |
| FMA-Constitutional | [5] | 42.1 Mb | 41647 | 148 | 85 | $\mathcal{ALCOIF}(\mathcal{D})$ |
| FMA-OWL2G_noMTC | [4] | 261.7 Mb | 85005 | 140 | 74698 | $\mathcal{SROIQ}(\mathcal{D})$ |
| FMA-DLR_M1[5] | this paper | 140Kb | 26 | 133 | 26 | $\mathcal{ALCOIF}(\mathcal{D})$ |
| FMA-DLR_M2[6] | this paper | 225Kb | 56 | 133 | 56 | $\mathcal{ALCOIF}(\mathcal{D})$ |

**Table 1.** List of FMA ontologies

FMA ontology on its field can now be compared to other well-known medical ontologies such as SNOMED [12] or GALEN [6].

Nowadays, the FMA can be viewed as a complex, highly connected network in which nearly 70,000 anatomical concepts, from over 170,000 frames, are interrelated by over 570,000 relationship instances. There have been a number of approaches [5, 10, 4] to translating the knowledge encapsulated by the FMA ontology into the OWL ontology language. Golbreich et al. [4] developed the FMA-OWLizer tool, which can be applied to automatically obtain a translation of the FMA ontology into OWL 2.

Given the size and complexity of the FMA ontology, reasoning under OWL has proven to be a real challenge. Table 1 provides a list of FMA ontologies written in OWL that we used in our evaluation.

## 3 FMA with Metamodeling

The FMA features a complex structure of superclasses and subclasses that requires the support of metamodeling. For example, "Physical anatomical entity" is an instance of "Anatomical entity template", and a subclass of both "Anatomical entity template" and "Anatomical entity" [1]. In OWL, a class is interpreted as a set of objects. Similarly, a metaclass is interpreted as a set of sets in metamodeling extensions of OWL, such as OWL-FA [7]. For example, the metaclass Vertebra can be interpreted as a set of different types of vertebrae, such as cervical, thoracic, lumbar, which in turn can be interpreted as subsets of other sets, e.g., first, ..., fifth lumbar vertebra.

There have been several attempts in dealing with metamodeling in FMA. Dameron et al. [1] converted the frame-based FMA ontology into an OWL1-DL version and an OWL1-Full version, with metaclasses included in the latter one. Golbreich et al. [5] tried to capture (some of) the knowledge encoded at metaclasses differently in OWL1-DL directly (cf. the FMA-Constitutional ontology in Table 1). The idea is to replace instance-of links between a class and its metaclasses with subClassOf links. The structure of their instances, property

[4] http://www.bioontology.org/wiki/index.php/FMAInOwl
[5] http://homepages.abdn.ac.uk/jeff.z.pan/pages/onto/fma-dlr-m1.owl
[6] http://homepages.abdn.ac.uk/jeff.z.pan/pages/onto/fma-dlr-m2.owl

restrictions of metaclasses are interpreted as closure axioms and approximated by universal restrictions while restrictions of classes are translated into existential restriction. Later on, Golbreich et al. further encoded the FMA ontology in OWL2-DL, producing two ontologies, one with metaclasses and one without them. The idea is to use the OWL 2 metamodeling capability, i.e., punning, to represent metaclasses, using the same URI to refer to a class and an individual at the same time in FMA-OWL2G_MT. For example, the name `Heart` can be used both for the metaclass `Heart` and for the class `Heart`, instance of `Organ_with_cavitated_organ_parts`.

The drawback of the punning approach is that, although a class and an individual can share the same name, say C, they are treated as different entities. For example, even if the class C is entailed to be equivalent to a class D, the individuals C and D can still be different. This has been regarded as non-intuitive due to the lack of expected entailments (e.g., the individuals C and D should be the same). To deal with this problem, we apply the class-based approach from Glimm et al. [3] to enrich some small (but already challenging for DL reasoners) subsets of FMA-DLR ontology and accommodate metaclasses (cf. the last two ontologies in Table 1) with the `Typing` and `MatSubClass` functions proposed in [3].

| Ontology | RT | FaCT++ | HermiT | TrOWL | MORe | Recall |
|---|---|---|---|---|---|---|
| FMA-DLR | C | 32.425 s | 46.94 s | 32.596 s | 47.794 s | 100% |
| FMA-FullR | C | 121.041 s | 1064.947 s | 4.45 s | 4.571 s | 100% |
| FMA-Constitutional | C | t/o | 3043.61 s | 155.808 s | t/o | 100% |
| FMA-OWL2G_noMTC | C | o/m | t/o | 967.59 s | t/o | N/A |
| FMA-DLR_M1 | M | 932.5 s | 26.39 s | 0.819 s | N/A | 100% |
| FMA-DLR_M2 | M | t/o | 737.43 s | 2.863 s | N/A | 100% |

**Table 2.** Reasoning with FMA ontologies via OWL API ('RT' for Reasoning Task, 'C' for Classification, 'M' for Materialisation, 's' for second, 't/o' for time out after one hour, 'o/m' for out of memory)

Table 2 lists the classification (for the first four ontologies) and materialisation (for the last two ontologies) time (reasoning time + retrieving time) from some of the state of the art DL reasoners (including FaCT++ v1.6.2, HermiT v1.3.8, TrOWL v1.3 and MORe v0.1.3 [15]) over the FMA ontologies in Table 1. The machine used for the experiment is a MacBook, with CPU 2.26 core 2 duo, Ram 8 GB and 6 GB allocated to JVM. The last column of Table 2 reports the recall (on the number of subsumptions among named classes) of TrOWL with respect to results of HermiT. Note that FaCT++ had a datatype error for the FMA-DLR ontology, so we did not import the FMA-DLR ontology into the FMA-FullR ontology when testing FMA-FullR. Moreover, as MORe does not support ABox reasoning, its time with the FMA-DLR_M1 and FMA-DLR_M2 ontologies is not reported.

## 4 Dealing with Unsatisfiable Concepts in FMA Constitutional

An ontology is called *incoherent* [17] if it contains unsatisfiable concepts, which are equivalent to the bottom concept $\bot$ and can not have any instance. Unsatisfiable named concepts (except the bottom concept $\bot$) in a constructed ontology usually indicates possible design flaws. For example, 33,433 out of 41,648 concepts are unsatisfiable in the FMA-Constitutional ontology. This was apparently not intended since the current version of FMA has already eliminated all these unsatisfiabilities.

Understanding ontology incoherence is not trivial. Incoherence can usually be explained and resolved by computing justifications [8], i.e., minimal entailment-preserving sub-ontologies. However, computing justifications with a black-box algorithm requires a large number of entailment checking, which can be expensive given the complexity of reasoning and size of the ontology. Also, looking into 33,433 justifications to debug the ontology will be very time consuming.

We notice that some of the concepts are unsatisfiable due to other unsatisfiable concepts. For example in FMA-Constitutional we can infer $Neuron \sqsubseteq \bot$ and $Central\_neuron \sqsubseteq Neuron$, hence we also have $Central\_neuron \sqsubseteq \bot$. In this case, $Central\_neuron$ is unsatisfiable due to the unsatisfiability of $Neuron$. Such a phenomena has been formally characterised by Kalyanpur et al. [9] as *root* and *derived* unsatisfiable concepts. Particularly, $A$ is a derived unsatisfiable concept if there is a justification for $A \sqsubseteq \bot$ that contains a justification for $B \sqsubseteq \bot$, where $B$ is another unsatisfiable concept, and $B$ is called the *parent* of $A$. Otherwise, $A$ is a *root* unsatisfiable concept.

The REL reasoner in TrOWL is using a forward-chaining completion-based algorithm, in which each rule infers a set of consequence axioms from a set of antecedence axioms. Such an algorithm can be easily extended to compute justifications on the fly by incorporating a truth-maintenance system (TMS) [13]. However naively applying such a solution has the following limitations:

1. For big and complex ontologies, maintaining the entire in TMS is a big overhead on reasoning. In fact, we are only interested in justifications for unsatisfiability but not the others so a fully-fledged TMS is unnecessary.
2. Repairing all the *root* unsatisfiabilities cannot always repair all the *derived* unsatisfiabilities because a derived unsatisfiability may have another justification that depends on no other unsatisfiability. In this case, one need to iteratively reclassify, debug and repair. For difficult ontologies such as FMA Constitutional, we would like to minimise the number of such iterations.

In order to improve efficiency, we introduce Type I and Type II unsatisfiable concepts as approximations to the root and derived unsatisfiable concepts with the following procedure in REL:

1. When a rule infers $A \sqsubseteq \bot$, if the antecedences contain $B \sqsubseteq \bot$, where both $A$ and $B$ are named concepts in the original ontology, we label $A$ as a *Type II* unsatisfiable concept. Otherwise, we label $A$ as a *Type I* unsatisfiable concept.

2. We continue reasoning on Type II concepts regardless their unsatisfiability, and label them with Type I if possible. They will not be treated immediately as sub-concept of all concepts.

The above point 2 is important to explore alternative derivation of unsatisfiability for Type II concepts. If such a derivation does not depend on other unsatisfiability, the concept will be labeled Type I as well. It is possible for a derived unsatisfiable concept to be labeled earlier than its parent, making it mistakenly labeled as a Type I. For example, considering the following axiom:

$$A \sqsubseteq B, \tag{1}$$
$$B \sqsubseteq C, \tag{2}$$
$$B \sqsubseteq \neg C, \tag{3}$$

if we infer $A \sqsubseteq C$ from (1) and (2), then $A \sqsubseteq \neg C$ from (1) and (3), and then $A \sqsubseteq \bot$, then we have $A$ as a Type I unsatisfiable concept. To avoid such situations as much as possible, we apply a depth-first classification strategy, always classifying super-concepts before classifying sub-concepts.

Using the above mechanism we are able to distinguish the different types of unsatisfiable concepts in the FMA-Constitutional. Results show that only 145 concepts belong to Type I, which is only 0.43% of all the unsatisfiable concepts. By examining their justifications, we realise that they are due to similar reasons. Particularly, there is a boolean-valued functional datatype property *has_mass* in the ontology. With the axioms in FMA Constitutional, it is possible to infer both $A \sqsubseteq \exists has\_mass.\{true\}$ and $A \sqsubseteq \exists has\_mass.\{false\}$ for concept $A$, making $A$ unsatisfiable. Another boolean-valued functional datatype property *has_inherent_3-D_shape* has a similar problem. There are in total only 6 concept axioms with these two properties. Debugging these 6 axioms is apparently much easier than debugging all the 122,136 logical axioms, or the justifications of all the 33,433 unsatisfiable concepts.

## 5 Conclusion and Outlook

This paper briefly introduces the TrOWL ontology reasoning infrastructure. Our evaluations with the FMA ontologies indicate that approximate reasoners can be useful for reasoning and debugging complex ontologies. The tested FMA ontologies are only some of the existing ones. We will further test more FMA ontologies, in particular those related to metamodeling.

## References

1. O. Dameron, D. L. Rubin, and M. A. Musen. Challenges in converting frame-based ontology into OWL: the Foundational Model of Anatomy case-study. In *the Proc. of the AMIA Annual Symposium*, 2005.
2. Birte Glimm, Ian Horrocks, and Boris Motik. Optimized description logic reasoning via core blocking. In *IJCAR*, pages 457–471, 2010.

3. Birte Glimm, Sebastian Rudolph, and Johanna Völker. Integrated metamodeling and diagnosis in owl 2. In *Proceedings of the 9th International Semantic Web Conference*, volume 6496 of *LNCS*, pages 257–272. Springer, November 2010.
4. C. Golbreich, J. Grosjean, and S. J. Darmoni. The fma in owl 2. In *Proceedings of the 13th conference on Artificial intelligence in medicine*, AIME'11, pages 204–214, Berlin, Heidelberg, 2011. Springer-Verlag.
5. Christine Golbreich, Songmao Zhang, and Olivier Bodenreider. The foundational model of anatomy in owl: Experience and perspectives. *J. Web Sem.*, 4(3):181–195, 2006.
6. Rogers JE, Roberts A, Solomon WD, van der Haring E, Wroe CJ, Zanstra PE, and AL. Rector. Galen ten years on: Tasks and supporting tools. In *Proceedings of the MEDINFO2001*, pages 256–260. IOS Press, 2001.
7. Nophadol Jekjantuk, Jeff Z. Pan, and Gerd Grner. Verifying and Validating Multi-Layered Models with OWL FA Toolkit. In *the Proc. of the Extended Semantic Web Conference (ESWC2010)*, 2008.
8. Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of owl dl entailments. In *ISWC/ASWC 2007*, pages 267–280, 2007.
9. Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in owl ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(4):268–293, 2005.
10. Natalya Fridman Noy and Daniel L. Rubin. Translating the foundational model of anatomy into owl. *J. Web Sem.*, 6(2):133–136, 2008.
11. Jeff Z. Pan and Edward Thomas. Approximating OWL-DL Ontologies. In *the Proc. of the 22nd National Conference on Artificial Intelligence (AAAI-07)*, pages 1434–1439, 2007.
12. Stearns M. Q., Price C., Spackman K. A., and Wang A. Y. Snomed clinical terms: overview of the development process and project status. In *Proceedings of the AMIA Symposium, American Medical Informatics Association*, pages 662–666, 2001.
13. Yuan Ren and Jeff Z Pan. Optimising ontology stream reasoning with truth maintenance system. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 831–836. ACM, 2011.
14. Yuan Ren, Jeff Z. Pan, and Yuting Zhao. Soundness Preserving Approximation for TBox Reasoning. In *the Proc. of the 25th AAAI Conference Conference (AAAI2010)*, 2010.
15. Ana Armas Romero, Bernardo Cuenca Grau, and Ian Horrocks. More: Modular combination of owl reasoners for ontology classification. In *Proceedings of the 11th International Semantic Web Conference (ISWC 2012)*, LNCS. Springer, 2012.
16. Cornelius Rosse and José L. V. Mejino Jr. A reference ontology for biomedical informatics: the foundational model of anatomy. *Journal of Biomedical Informatics*, 36(6):478–500, 2003.
17. Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *IJCAI*, pages 355–362. Morgan Kaufmann, 2003.
18. Bart Selman and Henry Kautz. Knowledge compilation and theory approximation. *J. ACM*, 43(2):193–224, March 1996.

# $KB\_Bio\_101$ : A Challenge for OWL Reasoners

Vinay K. Chaudhri, Michael A. Wessel, Stijn Heymans

SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, USA
firstname.lastname@sri.com

**Abstract.** We describe the axiomatic content of a biology knowledge base that poses both theoretical and empirical challenges for OWL reasoning. The knowledge base is organized hierarchically as a set of classes with necessary and sufficient properties. The relations have domain and range restrictions, are organized into a hierarchy, can have cardinality constraints and composition axioms stated for them. The necessary and sufficient properties of classes induce general graphs for which there are no known decidable reasoners. The OWL version of the knowledge base presented in this paper is an approximation of the original knowledge base. The knowledge content is practically motivated by an education application and has been extensively tested for quality.

## 1 Introduction

The goal of Project Halo is to develop a "Digital Aristotle" - a reasoning system capable of answering novel questions and solving problems in a broad range of scientific disciplines and related human affairs [11]. As part of this effort, SRI has created a system called Automated User-Centered Reasoning and Acquisition System (AURA) [8], which enables educators to encode knowledge from science textbooks in a way that it can be used for answering questions by reasoning.

A team of biologists used AURA to encode a significant subset of a popular biology textbook that is used in advanced high school and introductory college courses in the United States [15]. The knowledge base called $KB\_Bio\_101$ (for short: KBB101) is an outcome of this effort. KBB101 is a central component of an electronic textbook application called Inquire Biology [13] aimed at students studying from it.

AURA uses a frame-based knowledge representation and reasoning system called Knowledge Machine (KM) [7]. We have translated the original KM-version of KBB101 into first-order logic with equality. By using this representation as a common basis, we have translated it into multiple different formats including SILK [9], OWL2 functional [17], answer set programming [5], and the TPTP FOF syntax [6]. In this paper, we describe the OWL2 translation. The translations are available for download [4].[1]

## 2 Modeling in the AURA Project – The Role of Skolem Functions

AURA provides a graphical knowledge authoring environment for biologists. For example, the knowledge *Every Cell has a Ribosome part and a Chromosome part* is ex-

[1] This work is owned by Vulcan Inc. and is licensed for use under the *Creative Commons Attribution-NonCommerical-ShareAlike 3.0 license* (http://creativecommons.org/licenses/by-nc-sa/3.0/).

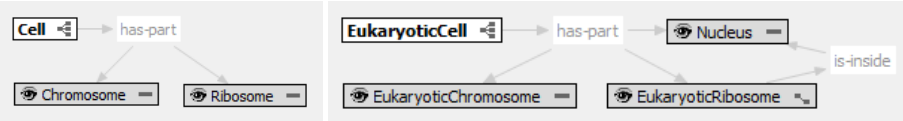



**Fig. 1.** (Simplified) Class Graphs for $Cell$ and $EukaryoticCell$ in AURA.

pressed graphically as shown in the left half of Fig. 1. Here, white nodes represent universally quantified variables, and grey nodes represent existentially quantified variables. $Cell$ hence corresponds to the following first-order formula:

$$\forall x : Cell(x) \Rightarrow$$
$$\exists y_1, y_2 : hasPart(x, y_1) \wedge hasPart(x, y_2) \wedge Ribosome(y_1) \wedge Chromosome(y_2)$$

Using the well-known technique of Skolemization, we can also write this as follows; the advantages of Skolem functions will become clear shortly:

$$\forall x : Cell(x) \Rightarrow$$
$$hasPart(x, f^1_{Cell}(x)) \wedge hasPart(x, f^2_{Cell}(x)) \wedge$$
$$Ribosome(f^1_{Cell}(x)) \wedge Chromosome(f^2_{Cell}(x))$$

The system supports inheritance. Consider the subclass $EukaryoticCell$, which inherits knowledge from $Cell$, see the right half of Fig. 1. The $Chromosome$ in $EukaryoticCell$ was inherited from $Cell$, and then specialized into a $EukaryoticChromosome$, and likewise for $Ribosome$. The $Nucleus$ was added locally in $EukaryoticCell$. The advantage of using Skolem functions is that the inheritance can be made explicit by means of equality atoms: by adding $f^3_{ECell}(x) = f^2_{Cell}(x), f^2_{ECell}(x) = f^1_{Cell}(x)$ to the Skolemized formula for $EukaryoticCell$ it is made explicit that the $EukaryoticChromosome$ in $EukaryoticCell$ is a specialization of the $Chromosome$ in $Cell$ and consequently, all knowledge which was modeled for that $Chromosome$ in the context of $Cell$ also applies to the $EukaryoticChromosome$ in the context of $EukaryoticCell$ (in addition to what was modeled for $Chromosome$ itself, of course):

$$\forall x : EukaryoticCell(x) \Rightarrow Cell(x) \wedge$$
$$hasPart(x, f^1_{ECell}(x)) \wedge hasPart(x, f^2_{ECell}(x)) \wedge hasPart(x, f^3_{ECell}(x)) \wedge$$
$$EukaryoticChromosome(f^3_{ECell}(x)) \wedge Nucleus(f^1_{ECell}(x)) \wedge$$
$$EukaryoticRibosome(f^2_{ECell}(x)) \wedge isInside(f^3_{ECell}(x), f^1_{ECell}(x)) \wedge$$
$$f^3_{ECell}(x) = f^2_{Cell}(x) \wedge f^2_{ECell}(x) = f^1_{Cell}(x)$$

The employed graphical modeling paradigm can be described as *inherit, specialize, and extend.* During the modeling process, the system keeps track of the specialized and extended Skolem functions and records the inheritance structures as demonstrated.

Since the above axiom defines a graph, it is not expressible in the known decidable description logics [12].

## 3 The Axiomatic Content of *KB_Bio_101*

We first describe the signature of an AURA KB in first order logic. KBB101 is an *AURA KB.* Let $CN$ be a set of class names (e.g., $Cell \in CN$), and $RN$ be a set of relation names (e.g., $hasPart \in RN$). Let $AN \subseteq RN$ be a set of attribute names (e.g., $color, temperature \in AN$). Let $C, C_1, C_2, \ldots, D, D_1, D_2, \ldots, E, E_1, E_2, \ldots, F, F_1, F_2, \ldots$ be class names, and $R, R_1, R_2, \ldots, S, S_1, S_2, \ldots, T, T_1, T_2, \ldots$ be relation names. Let $\{x, y, z, x_1, x_2, \ldots\}$ be a set of variables, and, for every $C \in CN$, let $\{fn^1_C, fn^2_C, \ldots\}$

be a set of function symbols. We have the following sets of constants: *scalar constant values* $SCs = \{small, big, \ldots\}$, *categorical constant values* $CCs = \{blue, green, \ldots\}$, *cardinal unit classes* $CUCs = \{meter, year, \ldots\}$, and $CN \cup RN$ are considered constants as well. There are three kinds of attributes; they are used in so-called *value atoms*, see below:

**Cardinal attribute values:** For example, *t is 43 years* would be represented as $age(t, t_1), theCardinalValue(t_1, 43), cardinalUnitClass(t_1, year)$.

**Categorial attribute values:** For example, *t has color green* would be represented as $color(t, t_1), theCategoricalValue(t_1, green)$.

**Scalar attribute values:** For example, *t is big w.r.t. a house* (where house is a class) would be represented as $size(t, t_1), theScalarValue(t_1, big), scalarUnitClass(t_1, house)$.

Next we describe the axiomatic content. An *AURA KB* is a tuple $(CTAs, CAs, RAs, EQAs)$, where $CTAs$ is a set of constant type assertions, $RAs$ is a set of relation axioms, $CAs$ is a set of class axioms, and $EQAs$ is a set of equality atoms. Those axioms are described in the following:

**CTAs** : The AURA KB contains, for every $c \in SCs \cup CCs \cup CUCs$, 1 to $n$ type assertions of the form $C(c)$, where $C \in CN$ (the types of the constant).

**EQAs** : A set of equality atoms for $C$, of the form $t = fn(t')$, where $t, t' \in \{x, fn^1_C(x), fn^2_C(x), \ldots\}$, and $fn \in \{fn^1_D, fn^2_D, \ldots\}$, with $C \neq D$, for some $D$ ($D$ is a class mentioned in $C$, or a direct or indirect superclass of $C$).

**CAs** : For every class name $C \in CN$, it may contain the following kinds of axioms: $DAs$ : disjointness axioms: $\forall x : C(x) \Rightarrow \neg D(x)$; $TAs$ : taxonomic axioms: $\forall x : C(x) \Rightarrow E(x)$; $NCAs$ : necessary conditions: $\forall x : C(x) \Rightarrow \Phi[x]$, where $\Phi[x]$ is a conjunction of unary (class) atoms and binary (relation) atoms over terms $\{x, fn^1_C(x), fn^2_C(x), \ldots\}$.

There are two special equality relations, namely $equal, notEqual$, which are user asserted equality atoms. The intended semantics is the semantics of first-order equality resp. in-equality. In order to distinguish them from the equalities in $EQAs$ we use different predicate names.

Moreover, $\Phi[x]$ can contain the following *value atoms:* for a term $t$, let *float* be a floating point number, $scalar \in SCs$, $categorical \in CCs$, $cardinalUnitClass \in CUCs$, and $scalarUnitClass \in CN$, then the following atoms are *value atoms:* $theCardinalValue(t, float)$, $theScalarValue(t, scalar)$, $theCategoricalValue(t, categorical)$, $cardinalUnitClass(t, cardinalUnitClass)$, and $scalarUnitClass(t, scalarUnitClass)$.

In addition, an AURA KB can contain *qualified number restrictions.* Due to a lack of counting quantifiers, we represent them by means of quadrary atoms $maxCardinality(t, R, n, C)$ (resp. $minCardinality$ and $exactCardinality$), where $n$ is a non-negative integer, $C$ is a class, and $R$ is a relation name.

**SCAs** : sufficient conditions: $\forall x : \Theta[x, \ldots] \Rightarrow C(x) \wedge EQs[x, \ldots]$, where $\Theta[x, \ldots]$ is a conjunction of unary, binary, value and qualified number restriction atoms over terms $\{x, x_1, x_2, \ldots\}$, the sufficient conditions, and $EQs[X, \ldots]$ is a conjunction of equality atoms of the form $t_1 = t_2$, where $t_1 \in \{x, x_1, x_2, \ldots\}$, and $t_2 \in \{x, fn^1_C(x), fn^2_C(x), \ldots\}$, linking the variables in the antecedent to the Skolem function values in

the consequent of the necessary conditions, $\Phi(x)$. Obviously, requiring the use of the Skolem functions in the antecedent of the sufficient condition would be a too strong requirement and render the sufficient condition inapplicable in many cases. Also note that $\Theta'[x] \subseteq \Phi[x]$, where $\Theta'[x]$ is the result of substituting the variables $\Theta[x]$ with their respective Skolem terms from $EQs[x,\ldots]$: $\Theta'[x] = \Theta[x]_{\{t_1 \mapsto t_2, t_1 = t_2 \in EQs[x,\ldots]\}}$. Hence, every sufficient condition is also necessary (a byproduct of the graphical modeling).

For a given class name $C$, we refer to the corresponding axioms as $DAs(C)$, $TAs(C)$, and $EQAs(C)$. We refer to the union of all axioms for $C$ as $CAs(C)$.

**RAs** : For every relation name $R \in RN$, $RAs$ may contain the following: $DRAs$ : relation domain restrictions $\forall x, y : R(x,y) \Rightarrow C_1(x) \vee \ldots \vee C_n(x)$; $RRAs$ : relation range restrictions $\forall x, y : R(x,y) \Rightarrow D_1(y) \vee \ldots \vee D_m(y)$; $RHAs$ : simple relation hierarchy $\forall x, y : R(x,y) \Rightarrow S(x,y)$; $QRHAs$ : qualified relation hierarchy $\forall x, y : R(x,y) \wedge C(x) \wedge D(y) \Rightarrow S(x,y)$; $IRAs$ : inverse relations $\forall x, y : R(x,y) \Rightarrow S(y,x)$; $12NAs$ : 1-to-N cardinality $\forall x, y, z : R(x,y) \wedge R(z,y) \Rightarrow x = z$; $N21As$ : N-to-1 cardinality $\forall x, y, z : R(x,y) \wedge R(x,z) \Rightarrow y = z$; $TRANSAs$ : simple transitive closure axioms $\forall x, y, z : R(x,y) \wedge Rstar(y,z) \wedge C(x) \wedge D(y) \wedge E(z) \Rightarrow Rstar(x,z)$, where $Rstar(x,z) = R^*(x,z)$; $GTRANSLAs$ : generalized transitive closure axioms (left composition) $\forall x, y, z : R(x,y) \wedge S(y,z) \wedge C(x) \wedge D(y) \wedge E(z) \Rightarrow Rstar(x,z)$; and $GTRANSRAs$ : generalized transitive closure axioms (right composition) $\forall x, y, z : R(x,y) \wedge S(y,z) \wedge C(x) \wedge D(y) \wedge E(z) \Rightarrow Sstar(x,z)$.

We refer to the axioms for a relation $R$ by $DRAs(R)$ etc. We refer to the union of all axioms for $R$ as $RAs(R)$.

## 4 The OWL Translations of *KB_Bio_101*

Our OWL translator produces OWL2 functional syntax [17], which has good human readability and is readily processed by most OWL2 reasoners. The generated KBs have been syntax-tested with Protégé 4.2 [14] (utilizing the OWLAPI parser) as well as RacerPro [16] (which has its own proprietary parser).

The following features of KBB101 might be challenging for OWL reasoners:

**Cycles:** KBB101 contains terminological cycles. It does not have the finite model property, nor the tree model property [1].

**Size:** the most complete export is 16 MBs big.

**Complexity:** the most complete export exploits $\mathcal{SHOIQ}(D_n)$ [3] (potentially we could use $\mathcal{SROIQ}(D_n)$ [10], but we currently do not include complex role inclusions, see below for a discussion).

**Graph structures:** we cannot represent the graph structures truthfully in OWL2. The original graph structures have to be approximated. We do this by rewriting and exporting the KBB101 in two flavors. **Flavor 1 - Unraveling:** We unravel the graph structures up to a certain maximal depth $n$. Unraveling is a standard technique from modal logics - let us give the following intuition: Given a class graph $C$ (see Fig. 1), an up to max. depth $n$ unraveled version of $C$ can be produced by an $n$-bounded depth-first graph traversal of $C$, starting from the root node $x$, that outputs the (inverse) edge label whenever an (inverse) edge is traversed to visit a successor node, together with the node label. This produces a tree. This tree with max-depth $n$ is then translated into OWL2 functional syntax as described. It results in an approximation of the original KBB101

which gets better the larger the value of $n$ is. Note that nodes reachable only over paths of length $> n$ are excluded. The filenames of KBB101s which were produced using unraveling start with `kb-owl-syntax-unraveled-depth-n`. We currently vary $n$ from 0 to 4 and produce the corresponding KBB101s. With $n = 0$, the axioms in $NCAs$ and $SCAs$ are basically ignored, as the unraveled tree consists of the root node only (hence, only the taxonomy is exported). **Flavor 2 - Node IDs:** We can represent the graph structure by introducing symbolic *node identifiers* in the OWL2 class expressions. Even though the OWL2 reasoner will be blind to the intended semantic meaning of these node IDs, modeling graph structure and co-references, the original graph structure is at least represented and could, in principle, be exploited for reasoning by some powerful extended future OWL2 reasoner. Note that node IDs are only introduced if required (in tree-shaped class descriptions they are not required). Moreover, those node IDs can either be rendered as atomic classes, or introduced as nominals. The filenames of the respective KBB101s start with `kb-owl-syntax-coreference-IDs`.

**Explicit inheritance and equality:** the inter-class co-references between Skolem function values and equality atoms cannot be represented in OWL2. We hence skip all the axioms in $EQNs$. We consider the OWL2 export underspecified. In principle, we could preserve some of those by using functional properties and encoding tricks, but even then, feature agreements or role value maps might be required, and already $\mathcal{ALCF}$ with general TBoxes is undecidable [2].

**Rendering of axioms** We can en- and disable the export of certain axiom types, e.g, there is a switch which determines whether $DAs$ are exported or not, and likewise for other axiom types. We produce all KBB101s for all possible combinations of those switches. Let us describe the rendering of class axioms and relation axioms. In the following, `C'` denotes the OWL2 version of class $C$, and `R'` the corresponding property of relation $R$.

The class axioms $CA(C)$ are exported as follows: The axioms $TAs(C)$ and $NCA(C)$ are combined into one axiom of the form $\forall x : C(x) \Rightarrow \Omega$, which is then rendered as a `SubClassOf(C` $\Omega'$`)` axiom. Here, $\Omega'$ is either an – up to depth $n$ – unraveled version of $\Omega$ as an OWL2 class expression, or the $\Omega'$ class expression uses node IDs for representing the graph structure as described. Note that the $DAs(C)$ and $EQAs(C)$ are excluded here. Moreover, if $C$ has a user-description or -comment, then this is rendered as `AnnotationAssertion(C' string)`. During rendering of $SCAs$, we are omitting the $EQs\,[x, \ldots]$ from $\forall x : \Theta\,[x, \ldots] \Rightarrow C(x) \wedge EQs\,[x, \ldots] \in SCAs$. KBB101s with $SCAs$ preserved have a `triggers` in their file names. We generate a `SubClassOf(`$\Theta'$ `C)` axiom, where $\Theta'$ is $\Theta\,[x, \ldots]$ as an OWL2 class expression, unraveled up to depth $n$. Disjointness axioms $DAs$ are represented by means of `DisjointClasses`. The rendering of $DAs$ can be suppressed; KBB101s with $DAs$ preserved have `-disjointness` in their file names. The rendering of cardinality constraints in necessary conditions $NCAs$ can be omitted. Also, we may choose to only export the cardinality constraints with cardinalities 0 and 1, as those are the only cardinality constraints used by the KM reasoning system [7] (the other cardinality constraints are ignored). KBB101s with cardinality constraints preserved have a `cardinalities` resp. `km-relevant-cardinalities` in their file names.

We also employ class and property annotation axioms to represent user descriptions and documentations.

The *inter-class equality axioms* $EQAs$ are ignored – as explained, there is no straightforward way to model our Skolem function inheritance in OWL2. However, the user asserted *intra-class equality and in-equality atoms* are retained, and we are using the `:same-as` and `:not-equal` object properties for that purpose.

Exporting the relation axioms $RAs$ is straightforward, too. KBB101s with relation axioms retained have a `relation-axioms` in their file names: The axioms $TRANSAs$, $GTRANSLAs$, $GTRANSRAs$ are analyzed. If an axiom can be truthfully encoded as an OWL2 complex role inclusion axiom obeying the regularity condition [10], then it is included in the file (unfortunately, none are, so the KBB101 ends up in $\mathcal{SHOIQ}(D_n)$ instead of $\mathcal{SROIQ}(D_n)$). If a relations $R$ turns out to be transitive, then this is declared by means of `TransitiveObjectProperty(R)` axiom. $RDAs(R)$ are rendered as `ObjectPropertyDomain(R, C)`, for every $\forall x, y : R(x, y) \Rightarrow C(x) \in RDAs(R)$. $RRAs(R)$ are rendered as `ObjectPropertyRange(R, C)`, for every $\forall x, y : R(x, y) \Rightarrow D(y) \in RRAs(R)$. $RHAs(R)$ are rendered as `SubObjectProperty(R,S)`, for every $\forall x, y : R(x, y) \Rightarrow S(x, y) \in RHAs(R)$. $IRAs(R)$ are rendered as `InverseObjectProperties(R, S)`, for every $\forall x, y : R(x, y) \Rightarrow S(y, x) \in IRAs(R)$. If $N21As(R) \neq \emptyset$, then we declare `FunctionalObjectProperty(R)`, and `ObjectProperty(R)` otherwise. If $R$ has a user-description string, then this is rendered as an `AnnotationAssertion(R string)`.

**Rendering of terms:** OWL2 is a term-free language. However, there is the analog of first-order constants, so-called nominals, and we may choose to use them for the representation of categorical property values (such as `green`) and scalar symbolic property values (such as `big`). A categorical property value such as `green` can either be represented as a type / instance assertion of the form `ClassAssertion(:ColorConstant :green)` and then used as a nominal object property filler in class subexpressions such as `ObjectHasValue(:color :green)`, or `:green` might be a special subclass of `:ColorConstant, SubClassOf(:green :ColorConstant)`, and then used in an `ObjectSomeValuesFrom(:color :green)` expression to represent the color of some object. However, for string- and float-based property values we need to use a datatype property-based representation, e.g. `DataHasValue(:theCardinalValue "43.0e0"^^xsd:float)`. KBB101s using the nominal representation have a `value-nominals` in their file names, and otherwise `value-classes`. The rendering of value classes and nominals can also be switched off completely.

## 5 Conclusion

An initial version of the $KB\_Bio\_101$ in OWL2 is now available [4] and we are very interested to actively engage with the research community to facilitate its use. We are also looking forward to seeing the reasoning runtimes of different systems participating in the ORE 2013 reasoner competition for the different OWL2 variants of KBB101. The reasoning problems we are currently interested in are consistency checking and classification.

**Acknowledgment: This work has been funded by Vulcan Inc.**

## References


1. F. Baader. Terminological Cycles in a Description Logic with Existential Restrictions. In *International Joint Conference on Atificial Intelligence*, 2003.
2. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
3. F. Baader and B. Hollunder. Qualifying Number Restrictions in Concept Languages. In *International Conference on Knowledge Representation and Reasoning*, 1991.
4. V. K. Chaudhri, S. Heymans, and M. A. Wessel. The $Bio\_KB\_101$ Download Page, 2012. See `http://www.ai.sri.com/halo/halobook2010/exported-kb/biokb.html`.
5. V. K. Chaudhri, M. W. Stijn Heymans, and S. C. Tran. Object-Oriented Knowledge Bases in Logic Programming. Technical report, SRI International, 2013. Available from `http://www.ai.sri.com/pub_list/1935`.
6. V. K. Chaudhri, M. A. Wessel, and S. Heymans. $KB\_Bio\_101$: A Challenge for TPTP First-Order Reasoners. In *CADE-24 Workshop on Knowledge Intensive Automated Reasoning*, 2013. Available from `http://www.ai.sri.com/pub_list/1937`.
7. P. Clark and B. Porter. Building Concept Representations from Reusable Components. In *AAAI*. AAAI Press, 1997.
8. D. Gunning and V. Chaudhri et al. Project Halo Update Progress Toward Digital Aristotle. *AI Magazine*, Fall 2010.
9. B. Grosof. The SILK Project: Semantic Inferencing on Large Knowledge, 2012. See `http://silk.semwebcentral.org/`.
10. I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible $\mathcal{SROIQ}$. In *International Conference on Knowledge Representation and Reasoning*, 2006.
11. V. Inc. Project Halo, 2012. See `http://www.projecthalo.com/`.
12. B. Motik, B. C. Grau, I. Horrocks, and U. Sattler. Representing Ontologies Using Description Logics, Description Graphs, and Rules. *Artificial Intelligence*, 173(14), Sept. 2009.
13. A. Overholtzer, A. Spaulding, V. K. Chaudhri, and D. Gunning. Inquire: An Intelligent Textbook. In *Proceedings of the Vidoe Track of AAAI Conference on Artificial Intelligence*. AAAI Press, 2012. See `http://www.aaaivideos.org/2012/inquire_intelligent_textbook/`.
14. Protégé Group. The Protégé Ontology Editor and Knowledge Acquisition System, 2012. See `http://protege.stanford.edu`.
15. J. B. Reece, L. A. Urry, M. L. Cain, S. A. Wasserman, P. V. Minorsky, and R. B. Jackson. *Campbell Biology, 9th ed.* Harlow: Pearson Education, 2011.
16. V. Haarslev and K. Hidde and R. Möller and M. Wessel. The RacerPro Knowledge Representation and Reasoning System. *Semantic Web Journal*, 3(3):267–277, 2012.
17. W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

# Evaluating OWL 2 Reasoners in the context of Clinical Decision Support in Lung Cancer Treatment Selection

M. Berkan Sesen[1], Ernesto Jiménez-Ruiz[2],
René Bañares-Alcántara[1], Sir Michael Brady[3]

[1] Department of Engineering Science, University of Oxford, UK
[2] Department of Computer Science, University of Oxford, UK
[3] Department of Oncology, University of Oxford, UK

**Abstract.** This paper evaluates the performances of the OWL 2 reasoners HermiT, FaCT++ and Pellet in the context of an ontological clinical decision support system in lung cancer care. In the first set of experiments, we compare how the classification and realisation times of the LUCADA and LUCADA-SNOMED CT ontologies vary as we expand their TBoxes with additional guideline rule knowledge. In the second set of experiments, we investigate the effect of increasing the ABox of the LUCADA ontology on the realisation times.

## 1 Introduction

Lung cancer is the most common and deadliest type of cancer, and is responsible for 21% of all cancer-related deaths globally. In England, care decisions for lung cancer patients are made by multidisciplinary teams (MDTs) that are comprised of clinical staff from diverse backgrounds. These teams meet weekly in cancer centres across the country in order to come to treatment decisions for each patient in their care. Usually, MDTs make use of their combined experience and knowledge of published clinical guidelines to decide upon the next stage of treatment for a patient [1]. The National Lung Cancer Audit (NLCA) data reveals that one of the major problems in the management of lung cancer care in England is the substantial level of unjustified variation in treatment decisions between different cancer centres [14, 13].

In order to reduce variability in clinical practice, clinical guidelines provide well defined sets of directions and evidence based standards to assist clinicians on decisions about appropriate clinical procedures [6]. However, as unstructured and free-text documents, clinical guidelines are usually not readily accessible at the point of decision making in the MDT meetings. Fortunately, clinical decision support (CDS) systems that computerise and automate the daily management of guidelines can facilitate access to guideline information in these meetings.

The computerisation of guideline rules can be achieved by structured logical languages which can express guideline rule eligibility and decision criteria. To date, many proprietary expression languages [4, 9, 11, 19, 20] have been proposed in order to encode and interpret guideline rules that are in a machine readable format. The interpretation of computerised guideline rules are carried out by execution engines that can match the encoded guideline rule criteria against existing patient records in order to infer rule applicability for different patient records.

In [16], we proposed OWL 2 [2] as a suitable candidate for encoding guideline rule criteria in the context of a CDS system for lung cancer care and we outlined a purely ontological guideline rule inference framework. In this paper, we focus on performance evaluations of off-the-shelf OWL 2 reasoners for inferring patient rule applicability based on the guideline rule inference framework presented in [16].

## 2 LUCADA ontology

Since 2004, the NLCA has collected all lung cancer patient data in England within the English Lung Cancer Dataset (LUCADA) [13] in order to gain a better understanding of the care delivered during referral, diagnosis and treatment of lung cancer patients. We have manually built a domain specific OWL 2 lung cancer ontology based on the LUCADA data model.[4] The LUCADA ontology provides the semantic layer of the Lung Cancer Assistant [16], an ontology-based system that is capable of providing guideline rule-based decision support during lung cancer MDT meetings.

SNOMED CT [15] is the reference ontology of choice across the information systems within the National Health Service (NHS). Thus, to facilitate interoperability with other NHS applications, we integrated LUCADA with a lung cancer-specific module of SNOMED CT. To this end, we have *(i)* identified the classes in SNOMED CT related to those in LUCADA and established correspondences (i.e. mappings) between them; and *(ii)* extracted a small fragment of SNOMED CT that captures the meaning of such relevant classes (i.e., a domain-specific module). SNOMED CT, however, is a complex ontology describing more than 300,000 classes; as a result, computing mappings with LUCADA is infeasible without suitable tool support. Thus, to perform task (i) we used the interactive-mode of the ontology matching system LogMap [7, 8]. Additionally, in order to perform task (ii), we used the ontology modularization technique described in [3]. Table 1 provides a side by side comparison of LUCADA and the integrated ontology LUCADA-SNOMED CT in terms of number of entities, axioms and expressivity.

In order to incorporate lung cancer guideline knowledge, we introduced the *patient scenario* class into both ontologies [16]. A guideline rule consists of an antecedent, i.e. rule body, which specifies the eligibility criteria for the rule and a consequent, i.e. rule head, which encapsulates the action(s) to take when the conditions in the antecedent are satisfied [5]. According to our guideline rule inference framework, we represent the guideline rule antecedents as defined *patient scenario* classes, whose equivalent class capture the semantics for rule eligibility criteria. As an example, the eligibility for the guideline rule[5] *"Consider radiotherapy for Stage I, II, III patients with good performance status"* is encoded as the following OWL 2 class equivalence axiom:

$$\begin{aligned} \mathsf{GR1} \equiv\ & \mathsf{GoodPerformancePatient} \sqcap \exists \mathsf{hasClinicalFinding}. \\ & (\mathsf{NeoplasticDisease} \sqcap \exists \mathsf{hasPreHistology}.\mathsf{NonsmallCellCarcinoma} \sqcap \\ & \exists \mathsf{hasPreTNMStaging}.string \sqcap \forall \mathsf{hasPreTNMStaging}.\{I, II, III\}) \end{aligned}$$

[4] Through a data sharing agreement between the University of Oxford and NLCA, we have been granted access to an anonymised version of LUCADA dataset.

[5] The guideline rules have been extracted from from National Institute for Clinical Excellence (NICE) document [12].

Table 1: Summary of the LUCADA and LUCADA-SNOMED CT ontology metrics

| Metric \ Ontology | LUCADA-SNOMED CT | LUCADA |
|---|---|---|
| **DL Expressivity** | $\mathcal{ALCHIF}(\mathcal{D})$ | $\mathcal{ALCHI}(\mathcal{D})$ |
| **# Classes** | 1553 | 376 |
| **# Object properties** | 63 | 37 |
| **# Data Properties** | 63 | 63 |
| **# Equiv. class axioms** | 1010 | 0 |
| **# Subclass of axioms** | 999 | 386 |
| **# Prop. domain axioms** | 97 | 97 |
| **# Prop. range axioms** | 30 | 30 |

Furthermore, we represent a *patient record* as a set of OWL 2 individual axioms with respect to the terminological knowledge captured within the LUCADA and the integrated LUCADA-SNOMED CT ontologies as exemplified in [16]. According to this, a patient record is characterised (on average) by 25 class and property assertion axioms. An OWL 2 reasoner can be used to determine whether a specific patient is a member of a particular patient scenario class, and therefore, subject to the recommendations or actions of the respective guideline rule.

## 3 Evaluation

We evaluated the scalability of our guideline rule inference framework with off-the-shelf OWL 2 reasoners: HermiT 1.3.7 [10], Pellet 2.3.0 [17] and FaCT++ 1.6.2 [18]. The tests have been performed on a Windows 7 64-bit desktop computer with 15 GiB of RAM and an Intel Xeon 2.27 GHz CPU. Overall, we report two sets of experimental results as given below. Note that all results reported here have been acquired as averages of at least 10 repetitions of the described experimental setup.

### 3.1 Increasing the TBox with patient scenarios

In the first set of experiments we compared how the classification and realisation times of LUCADA and LUCADA-SNOMED CT ontologies varied as we increased the guideline rule coverage (i.e. patient scenarios classes). To this end, we incrementally added to each ontology 40 patient scenarios, represented as equivalent class axioms (see Section 2), and recorded the times taken by each reasoner to perform classification (i.e. execution of precomputeInferences(CLASS_HIERARCHY) method) and realisation of only one patient individual (i.e. execution of the method getTypes()).

Figures 1 and 2 summarise the reasoning times obtained for the LUCADA and LUCADA-SNOMED CT ontologies respectively. In both figures, we only report the total inference times (classification + realisation) for FaCT++ and Pellet since the individual realisation times for these two reasoners were negligible. However, for HermiT

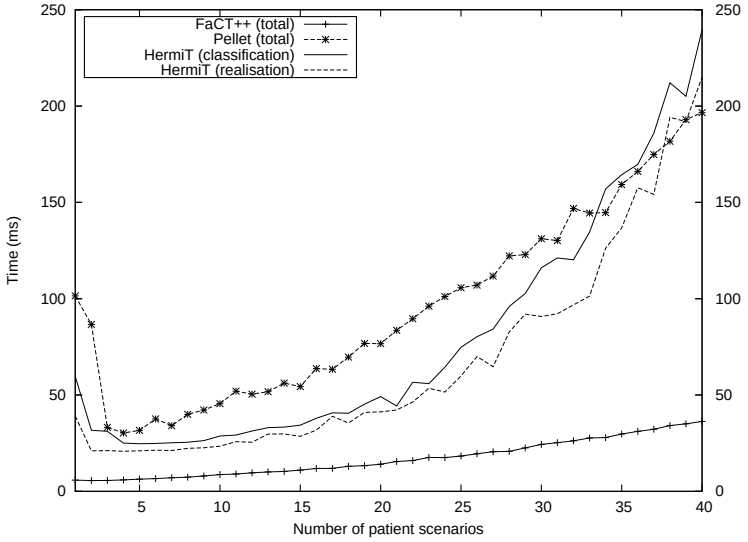



Fig. 1: Reasoning times for LUCADA containing 1 to 40 patient scenarios

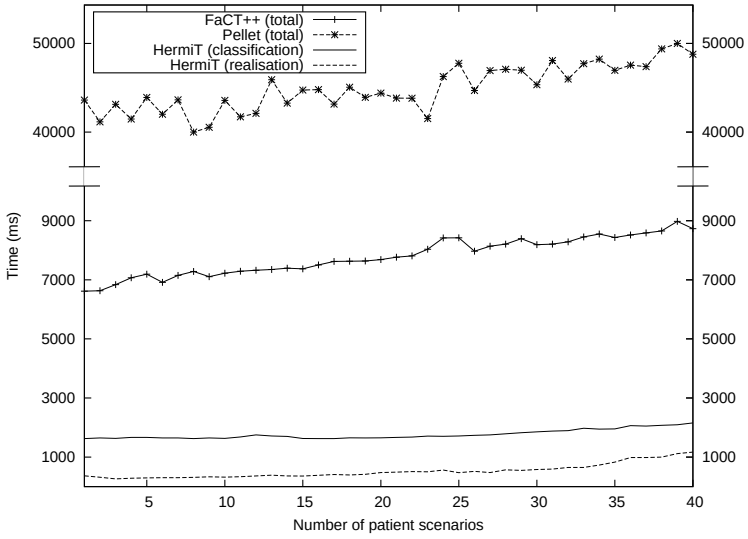



Fig. 2: Reasoning times for LUCADA-SNOMED CT containing 1 to 40 patient scenarios

we present classification and realisation times separately, since realisation takes up a significant portion of the total inference time (up to 0.2ms for LUCADA and 1s for LUCADA-SNOMED CT). We note that the classification times for all three reasoners are below one second for the LUCADA ontology, whereas they rise to 9 and 50 seconds re-

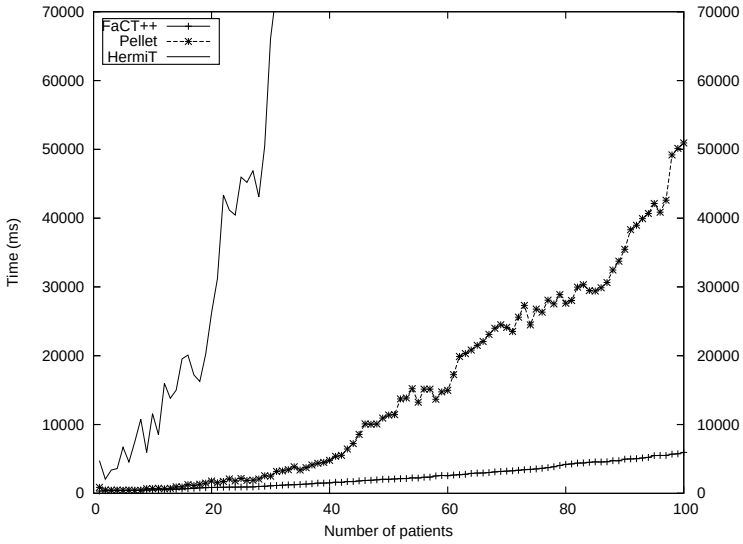



Fig. 3: Realisation times in LUCADA with 1 to 100 patient records

spectively with FaCT++ and Pellet for the integrated LUCADA-SNOMED CT ontology. Note that HermiT classifies the integrated ontology the fastest, with classification times ranging from 1.6s to 2.2s.

### 3.2 Increasing the ABox with patient records

In the second set of experiments, we incrementally added 100 patient records, represented as OWL 2 individuals axioms (see Section 2), to the LUCADA ontology which contained 40 patient scenarios. Figure 3 compares the realisation times (i.e. execution of the method getTypes() for each patient individual) obtained by all three reasoners. As expected, realisation times increase as more patients are added to the ontology. It is noticeable that FaCT++ and HermiT have very disparate behaviours. While the increase in realisation times with respect to the number of patient individual in the ontology is fairly gradual and linear for FaCT++, the realisation times for HermiT increase very quickly and clearly in a non-linear fashion. Although not as severe as the realisation times achieved by HermiT, Pellet realisation times are also considerably slower compared to FaCT++ and seem to increase non-linearly.

## 4 Conclusions

In this paper we evaluated empirically the classification and realisation performances of the three most commonly used OWL 2 reasoners within our guideline rule inference framework. We found that FaCT++ is the best choice for our application since it provides very fast inference times for both classification and realisation. We also found

that HermiT provides the fastest TBox reasoning times for the integrated LUCADA-SNOMED CT ontology; but it performs poorly in ABox reasoning with both ontologies. Finally, we found that Pellet performs well in classifying the LUCADA ontology but struggles with the LUCADA-SNOMED CT ontology, which contains many axioms inherited from SNOMED CT.

## Acknowledgements

The LCA project was funded by the CDT in Healthcare Innovation programme within the Institute of Biomedical Engineering, Oxford University. We would also like to acknowledge the clinical inputs from our collaborators Dr Michael Peake, Prof Fergus Gleeson and Dr Donald Tse during the elicitation of guideline rules from the literature. Jiménez-Ruiz was partially supported by the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, "Optique", and the EPSRC projects Score!, ExODA and MaSI$^3$.

## References

1. Austin, M.: Information Integration and Decision Support for Multidisciplinary Team Meetings on Colorectal Cancer. Ph.D. thesis, University of Oxford (2008)
2. Cuenca Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P.F., Sattler, U.: OWL 2: The next step for OWL. J. Web Sem. 6(4), 309–322 (2008)
3. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: Theory and practice. J. Artif. Intell. Res. 31, 273–318 (2008)
4. Fox, J., Johns, N., Lyons, C., Rahmanzadeh, A., Thomson, R., Wilson, P.: PROforma: a general technology for clinical decision support systems. Computer Methods and Programs in Biomedicine 54(12), 59 – 67 (1997)
5. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Tech. rep., World Wide Web Consortium (2004)
6. Isern, D., Sánchez, D., Moreno, A.: HeCaSe2: A Multi-agent Ontology-Driven Guideline Enactment Engine. In: 5th International Central and Eastern European Conference on Multi-Agent Systems. pp. 322–324 (2007)
7. Jiménez-Ruiz, E., Cuenca Grau, B.: LogMap: Logic-based and Scalable Ontology Matching. In: Int'l Sem. Web Conf. (ISWC). pp. 273–288 (2011)
8. Jiménez-Ruiz, E., Cuenca Grau, B., Zhou, Y., Horrocks, I.: Large-scale interactive ontology matching: Algorithms and implementation. In: European Conf. on Artif. Intell. (ECAI). pp. 444–449 (2012)
9. Miksch, S., Shahar, Y., Johnson, P.D.: Asbru: A Task-Specific, Intention-Based, and Time-Oriented Language for Representing Skeletal Plans. In: 7th Workshop on Knowledge Engineering: Methods & Languages (KEML-97) (1997)
10. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. Artif. Intell. Res. 36, 165–228 (2009)
11. Musen, M.A., Tu, S.W., Das, A.K., Shahar, Y.: EON: A Component-Based Approach to Automation of Protocol-Directed Therapy. Journal of the American Medical Informatics Association 3(6) (1996)

12. NICE: The Diagnosis and Treatment of Lung Cancer (Update). National Collaborating Centre for Cancer (UK). NICE Clinical Guidelines, No. 121. (2011), available from: http://www.ncbi.nlm.nih.gov/books/NBK99021/
13. NLCA: The National Clinical Lung Cancer Audit (LUCADA) Data Manual (2010), available from: http://www.hscic.gov.uk/lung
14. Riaz, S.P., Lchtenborg, M., Jack, R.H., Coupland, V.H., Linklater, K.M., Peake, M.D., Mller, H.: Variation in surgical resection for lung cancer in relation to survival: Population-based study in england 20042006. European Journal of Cancer 48(1), 54 – 60 (2012)
15. Schulz, S., Cornet, R., Spackman, K.A.: Consolidating SNOMED CT's ontological commitment. Applied Ontology 6(1), 1–11 (2011)
16. Sesen, M.B., Bañares-Alcántara, R., Fox, J., Kadir, T., Brady, J.M.: Lung Cancer Assistant: An ontology-driven, online decision support prototype for lung cancer treatment selection. In: OWL: Experiences and Directions Workshop (OWLED) (2012)
17. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. Web Sem. 5(2), 51–53 (2007)
18. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: Third International Joint Conference on Automated Reasoning, IJCAR. pp. 292–297 (2006)
19. Tu, S.W., Campbell, J.R., Glasgow, J., Nyman, M.A., McClure, R., McClay, J., Parker, C., Hrabak, K.M., Berg, D., Weida, T., Mansfield, J.G., Musen, M.A., Abarbanel, R.M.: The SAGE Guideline Model: Achievements and Overview. Journal of the American Medical Informatics Association 14(5), 589 – 598 (2007)
20. Wang, D., Peleg, M., Tu, S.W., Boxwala, A.A., Ogunyemi, O., Zeng, Q.T., Greenes, R.A., Patel, V.L., Shortliffe, E.H.: Design and implementation of the GLIF3 guideline execution engine. Journal of Biomedical Informatics 37(5), 305–318 (2004)

# Genomic CDS: an example of a complex ontology for pharmacogenetics and clinical decision support

Matthias Samwald[1]

[1] Medical University of Vienna, Vienna, Austria
matthias.samwald@meduniwien.ac.at

**Abstract.** Individual genetic data can be used to better predict the efficacy and safety of medications for individual patients. The Genomic Clinical Decision Support (Genomic CDS) ontology aims to utilize advanced Web Ontology Language 2 (OWL 2) reasoning for this task. The important, clear-cut medical use case, the complex axioms in the ontology and the heavy use of qualified cardinality restrictions make the ontology an interesting test object for new OWL 2 reasoners with improved performance.

**Keywords:** OWL, pharmacogenetics, clinical decision support

## 1 Motivation

Different patients can react drastically different to the same type of medication (**Fig. 1**). The goal of personalized medicine and pharmacogenetics is to predict an individual patient's response by analyzing genetic markers that influence how medications are metabolized or able to bind to their targets.

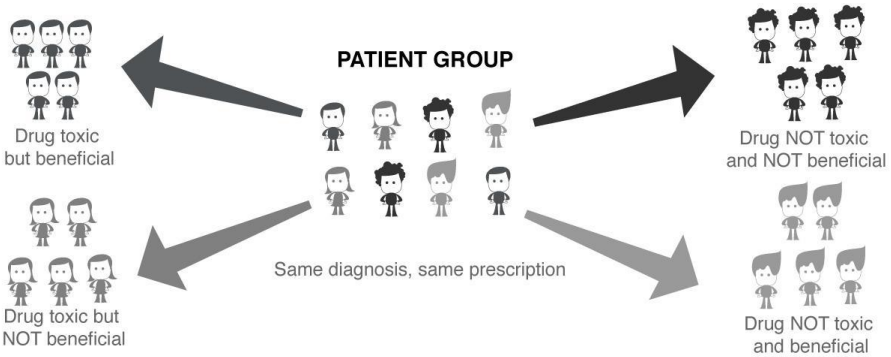



**Fig. 1.** The efficacy and safety of medications can drastically vary between patients. The goal of pharmacogenetics is to classify patients into subgroup based on genetic markers, to better predict which treatments could help and which could do harm.

To produce clinically valid and trustworthy predictions, no errors or ambiguities should arise in the process of inferring a patient's likely response from raw genetic

data. Current formalisms, data infrastructures and software applications leave many opportunities for introducing such errors and ambiguities. Ontologies formalized with the Web Ontology Language 2 (OWL 2) could be an excellent choice for tackling this problem, but the complexity and potentially large scale of ontologies in this domain also pose formidable challenges to currently available OWL 2 reasoners.

## 2 The Genomic CDS ontology

The Genomic Clinical Decision Support (Genomic CDS) ontology is an OWL 2 ontology aimed at representing pharmacogenetic knowledge and providing clinical decision support based on pharmacogenetic data. It is being developed by members of the Clinical Pharmacogenomics Task Force, which is part of the Health Care and Life Science Interest Group of the World Wide Web Consortium (W3C). The OWL files of the ontology, as well as 'demo' files containing example patient data can be downloaded from http://www.genomic-cds.org/ont/snapshot-june-2013
We also created a simplified version of the Genomic CDS ontology, called 'Genomic CDS light', which does not contain some of the axioms of the full ontology. Both versions of the ontology have ALCQ expressivity. They are characterized by extensive use of qualified cardinality restrictions.

The goals of developing the ontology are:

- Providing a simple and concise formalism for representing pharmacogenetic knowledge,
- Finding errors and lacking definitions in pharmacogenetic knowledge bases
- Automatically assigning alleles and phenotypes to patients
- Matching patients to clinically appropriate pharmacogenetic guidelines and clinical decision support messages

In the most common scenario, genetic patient data in OWL format is combined with the axioms of the Genomic CDS ontology, and an OWL reasoner is used to infer matching pharmacogenetic treatment recommendations. Several inference steps are needed to derive matching treatment recommendations from raw data about genetic markers (**Fig. 2**). The raw data consists of small variants in the genetic code, which in most cases are so-called *single nucleotide polymorphisms* (SNPs), such as an 'A' instead of a 'G'. *Alleles* are variants of a gene that are defined by containing sets of such small variants. *Phenotypes* are referring to the specific effects that certain small variants and alleles can have on the organism, e.g., how quickly a patient metabolizes a specific drug. Clinical guidelines can use small variants, alleles and/or phenotypes to match patients with treatment recommendations

The human genome usually contains two copies of each gene (one from the father, one from the mother), with each copy potentially bearing multiple genetic variants. Because of this, the ontologies rely heavily on qualified cardinality restrictions with

cardinalities of two, which seems to cause performance issues with most current OWL reasoners.

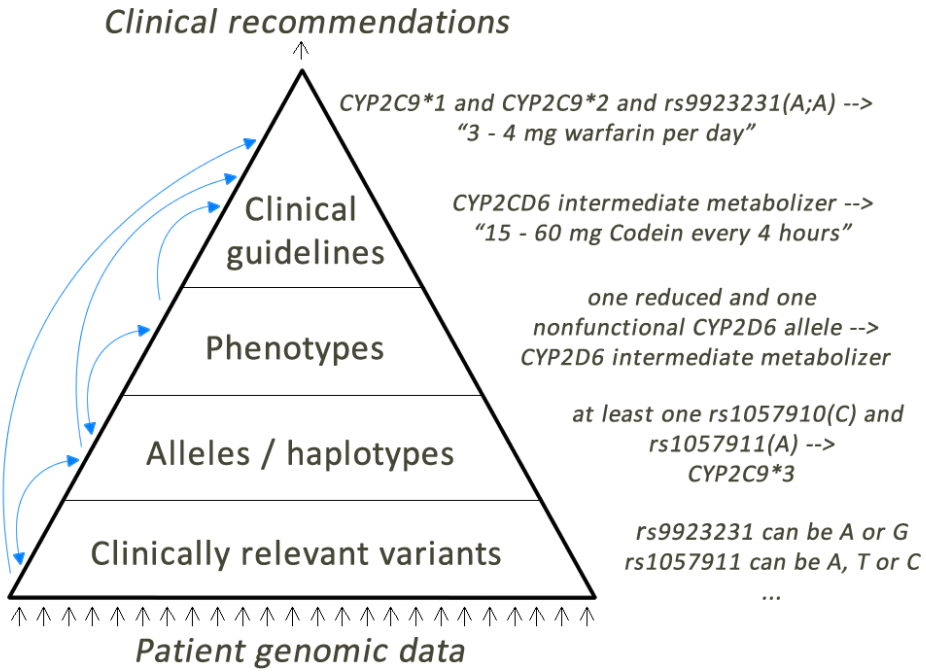



**Fig. 2.** : Through a series of inference steps, matching pharmacogenetic treatment guidelines are inferred from raw genetic patient data.

A simplified example of a rule for inferring an allele (CYP2C9*3) and its single nucleotide polymorphisms (SNPs) from a so-called 'tagging SNP' (a SNP that is necessary and sufficient for inferring the presence of the allele) looks like this in Manchester syntax:

```
Class: 'human with CYP2C9*3'
    EquivalentTo:
           has some rs1057910_C
    SubClassOf:
       has some 'CYP2C9 *3',
       (has some rs1057910_C)
        and (has some rs1057911_A)
        and (has some rs1799853_C)
        and (has some rs2256871_A)
        and (has some rs72558188_AGAAATGGAA)
```

An example of an axiom for inferring an adequate clinical decision support message for the anticoagulant drug warfarin (based on a combination of alleles and SNPs according to an official recommendation in the drug label):

```
Class: 'human triggering CDS rule 7'
  EquivalentTo:
   (has some 'CYP2C9*1') and (has some 'CYP2C9*3')
    and (has exactly 2 rs9923231_C)
   Annotations:
     label "human triggering CDS rule 7",
     CDS_message "3-4 mg warfarin per day should
       be considered as a starting dose range for
       a patient with this genotype according to
       the Warfarin drug label (Bristol-Myers
       Squibb)."
```

We used two OWL 2 reasoners with our ontology: TrOWL[1] [1] and HermiT[2] [2]. We also evaluated other OWL 2 reasoners (Fact++[3], Pellet[4]) in early stages of the project, but excluded them from further tests because they did not terminate or crashed even with small, preliminary versions of the ontology we developed. We compared the performance of the two reasoners on a virtual machine running on the Amazon Elastic Cloud Computing (EC2) cloud[5]. The machine was of the "High-Memory Extra Large Instance" type, running Microsoft Windows Server 2008, with 17.1 GB of memory, a 64-bit platform, and two virtual cores with 3.25 EC2 compute units each.
The reasoners were run as plugins in the 64 bit version of the Protégé 4.2 ontology editor. The initial heap size for Protégé was $10^{10}$ bytes (10 GB), and the maximum allowed heap size was $1.5x10^{10}$ bytes (15 GB). The TrOWL reasoner plugins with version 0.6 and 1.1 were each run three times for each ontology, and the mean of the time needed for classification was calculated. The HermiT 1.3.8 plugin was run once for each version of the ontology.

These preliminary tests showed TrOWL to be significantly more performant than HermiT for classifying the ontologies (
**Table 1**). However, HermiT was able to identify biologically meaningful inconsistencies present in genomic-cds-demo.owl (but not present in the light version of the ontology). TrOWL did not recognize these inconsistencies, most likely because it only partially covers the OWL 2 DL ruleset. These results show that only TrOWL is performant enough to be used in realistic settings (e.g. for clinical decision support), but that HermiT could serve to test and validate the results from TrOWL during develop-

[1] http://trowl.eu
[2] http://www.hermit-reasoner.com/
[3] http://code.google.com/p/factplusplus/
[4] http://clarkparsia.com/pellet/
[5] http://aws.amazon.com/en/ec2/

ment (possibly comparing the results of the two reasoners for smaller ontology fragments).

**Table 1.** Reasoning performance: TrOWL is significantly more performant than HermiT in classifying our demo ontology (OWL 2 DL with ALCQ expressivity)

| | **HermiT 1.3.8** | **TrOWL 1.1** | **TrOWL 0.6** |
|---|---|---|---|
| **genomic-cds-light-demo.owl (2150 classes, 9500 axioms)** | 3 hours 48 minutes | 1.5 seconds | 18 seconds |
| **genomic-cds-demo.owl (2300 classes, 11000 axioms)** | detected inconsistencies | 5.8 seconds | 54 seconds |

## 3 Conclusions and outlook

The Genomic CDS ontology is an example of an OWL 2 ontology for clinical genetics and decision support. Even though it is focused on a relatively small set of the most important pharmacogenetic markers, the ontology poses a significant challenge to currently available OWL 2 reasoners. There is great need for reasoners that are optimized for the kinds of OWL axioms encountered in ontologies dealing with clinical genomics.

## 4 Acknowledgements

The research leading to these results has received funding from the Austrian Science Fund (FWF): [PP 25608-N15].

## References

1. Thomas, E., Pan, J.Z., Ren, Y.: TrOWL: Tractable OWL 2 Reasoning Infrastructure. the Proc. of the Extended Semantic Web Conference (ESWC2010) (2010).
2. Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. J. Artif. Intell. Res. 36, 165–228 (2009).

# A large-scale gene-centric semantic web knowledge base for molecular biology

José Cruz-Toledo[1§], Alison Callahan[1§], and Michel Dumontier[1]

[1]Department of Biology, Carleton University, Ottawa, Canada,
{acallaha, jctoledo}@connect.carleton.ca, michel_dumontier@carleton.ca
[§]These authors contributed equally to this work

**Abstract.** The discovery of the central role of genes in regulating the fundamental biochemical processes of living things has driven biologists to collect, analyze and re-use enormous amounts of information, and to make this information available in thousands of curated databases. The increasingly popular use of specialized terminologies, often organized into hierarchical taxonomies or more formal ontologies, to describe this data indicates that managing the total amount of resources available (big data) will surely continue to be an ongoing challenge. Here, we describe a biological gene-centric dataset (available at http://semanticscience.org/projects/gene-world), aimed at providing the reasoner community with a fully connected graph of data and ontologies of value to the bioinformatics community and for which there currently exists significant challenges in using automated reasoning for consistency checking and query answering of large ontology-mapped linked data.

**Keywords.** Semantic Web, Bioinformatics, DL reasoning, SPARQL

## 1 Motivation

The central dogma of molecular biology states that regions of DNA (genes) are responsible for encoding molecular machines called proteins, which participate in and control the biochemical reactions essential to sustaining life. The discovery of the central role of genes as the blueprint of our evolutionary history and their involvement in health and disease has driven biologists to characterize these very important entities. Enormous amounts of information have been collected, analyzed, summarized and re-published in thousands of curated databases [1] and large central hubs such as the databases of the National Center for Biotechnology Information (NCBI).

The exponential growth of available molecular data clearly yields enormous benefits to biologists attempting to elucidate the functioning of genes in related systems, but it also presents significant challenges for modern biology. Consider that the amount of data collected in this year alone to characterize a collection of biochemical reactions (a pathway) will be on par with the amount of data that has ever been collected about that pathway in the history of the field [2]. Moreover, the use of specialized terminologies, often organized into hierarchical taxonomies or more formal ontologies, indicates that managing the total amount of data (big data) will surely con-

tinue to be an ongoing challenge. Indeed, it is the overall organization and interpretation of this vast deluge of information that presents the greatest challenge.

Motivated by this challenge, we present a preliminary version of a biological gene-centric dataset aimed at providing the reasoner community with a fully connected graph of data and ontologies of value to the bioinformatics community, for which there currently exists significant challenges in using automated reasoning for consistency checking and query answering of large ontology-mapped linked data. We focus our attention on one of the larger datasets in the Bio2RDF project [3] - NCBI Gene - and consider queries that extend from this dataset into other datasets and ontologies that together form a large 'Gene-World' knowledge base. Our Gene-World knowledge base contrasts other ontologies and datasets that have been used to benchmark OWL reasoners [4], such as LUBM [5] and SNOMED-CT [6], in several respects: (i) Gene-World is a 'real world' knowledge base composed of existing resources used by biologists and bioinformaticians on a daily basis, as opposed to an arbitrary automatically generated knowledge base, (ii) it consists of a very large T-box and A-box and (iii) its T-Box consists of multiple ontologies with differing DL expressivity. The datasets and ontologies described are available at [7]. Example queries that can be used to evaluate RDF/OWL based reasoner performance over this knowledge base are also described.

## 2 Datasets and Ontologies

All Gene-World datasets are drawn from Bio2RDF Release 2 (released January 2013). The NCBI Gene [8] Bio2RDF dataset consists of 394,026,267 triples with 12,543,449 unique subjects, 60 unique predicates, and 121,538,103 unique objects. NCBI Gene describes genes including their names, reference sequences, variants, phenotypes, pathways and cross-references to related resources. HomoloGene [9] is a database of programmatically generated clusters of homologous, including paralogous and orthologous, genes from a set of 21 completely sequenced eukaryotic genomes. The HomoloGene Bio2RDF dataset consists of 1,281,881 triples with 43,605 unique subjects, 17 unique predicates and 1,011,783 unique objects, and uses NCBI Gene identifiers to refer to the genes it clusters. NCBI Gene makes reference to three ontologies: the Gene Ontology (GO) for asserting function, process or location annotations about genes, the Evidence Code Ontology (ECO) for qualifying the source of these GO annotations, the NCBI Taxonomy (TAXON) for asserting the species of a gene. The Semanticscience Integrated Ontology (SIO) and the Sequence Ontology (SO) have been mapped to NCBI Gene Bio2RDF vocabulary classes and relations (Table 1) to ground the dataset types and predicates in domain-specific ontologies.

The Gene Ontology (GO) [10] [11] is a hierarchy of controlled biological terms that is organized into three orthogonal ontologies which capture knowledge about cellular locations, biological processes and molecular functions. The terms and relations contained in GO are serialized as a directed acyclic graph where concepts are organized into a hierarchy in which more specific GO terms are subsumed by more general terms by following *is a* or in some cases *part of* relationships. The Evidence

Code Ontology (ECO) is a controlled vocabulary used for describing the scientific evidence that supports an assertion. ECO's 290+ terms include descriptions of laboratory experiments, computational methods and literature annotation terminology. The NCBI Taxonomy (TAXON) [9] is a database of taxonomic lineage obtained from a variety of sources, including primary literature, external databases and expert human curation efforts for databases hosted by the NCBI. The Sequence Ontology (SO) [12] describes a rich set of features and attributes of biological sequences. The terms and relations included in this ontology characterize both physical attributes of biological sequences (*i.e.* binding sites, exons) and the processes in which biological sequences may be involved in (*i.e.* translational frameshifts, transitions, deletions, *etc*). The Semanticscience Integrated Ontology (SIO) provides a basic set of types and relations for describing objects, processes and attributes of biological entities. Fig. 1 shows how these ontologies are used within the Gene dataset, or are linked to the Gene dataset by virtue of mappings to SIO.

**Table 1.** Summary metrics of ontologies that can be used to reason over the NCBI Gene dataset: the Evidence Code Ontology (ECO), the Gene Ontology (GO), the NCBI Taxonomy (NT), the SemanticScience Integrated Ontology (SIO) and the Sequence Ontology (SO)

| **Ontology** | **Classes** | **Object properties** | **subClassOf axioms** | **subPropertyOf axioms** | **DL expressivity** |
|---|---|---|---|---|---|
| ECO | 297 | 2 | 453 | 0 | ALC |
| GO | 34403 | 6 | 63375 | 0 | ALE |
| TAXON | 1018210 | 15 | 1018204 | 0 | AL(D) |
| SIO | 1385 | 201 | 1729 | 207 | SRIQ(D) |
| SO | 2151 | 74 | 2602 | 9 | SHI |

## 3 Reasoning Tasks

In this section, we describe reasoning tasks over the Gene-World knowledge base that can be used to benchmark the performance of an OWL reasoner or SPARQL query system. After loading all the triples for the NCBI Gene and HomoloGene RDF datasets, as well as all ontologies listed in Table 1, the first benchmark task for an OWL reasoner would be to check the consistency of the combined knowledge base. While each component is expected to not contain any unsatisfiable classes, mappings between SIO and Gene, SO and Gene or the disjoint class axioms for the NCBI taxonomy ontology may lead to class or property unsatisfiability.

Below, we present a set of DL and SPARQL-DL queries over the combined knowledge base that may not give the complete set of results without reasoning support for some portion of OWL2-DL (there are no nominals in the knowledge base). GitHub Gists of all queries are available at [13].

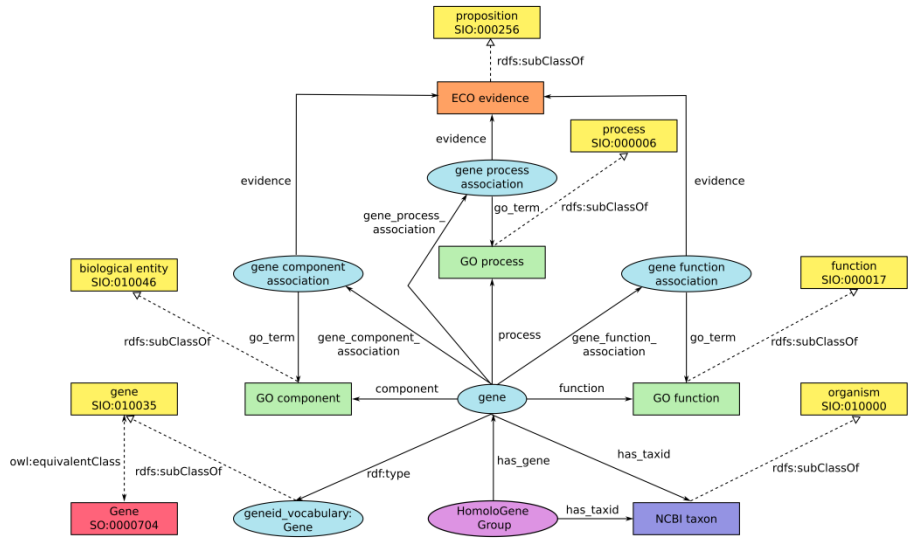



**Fig. 1.** Links between a gene in the NCBI Gene dataset and annotations of its function, associated cellular components and/or processes. Functions, cellular components, and processes are described using the Gene Ontology (GO, in green), while the associated evidence type for an association is described using the Evidence Codes Ontology (ECO, in orange). The taxonomic group for a gene is described using NCBI Taxonomy (in blue). HomoloGene (in purple) groups related genes and taxa. Each part of an NCBI Gene record is mapped to the Semanticscience Integrated Ontology (SIO, in yellow), which also has mappings to the Sequence Ontology (SO, in pink). Ellipses represent resources. Boxes represent ontology classes.

### 3.1 Query answering

**Q1:** retrieve transfer RNA genes
DL query: tRNA-gene
*simple query that retrieves a type assertion in NCBI gene data*

**Q2:** retrieve human genes
DL query: gene that has_taxid some ‘Homo sapiens [taxid:9606]’
*conjunctive query over NCBI Gene and NCBI taxonomy*

**Q3:** retrieve genes that are from any mammal but human
DL query: gene that has_taxid some (‘Mammalia [taxid: 40674]’ and not ‘Homo sapiens [taxid:9606]’)
*conjunctive query with negation, and subclass reasoning over asserted hierarchy and class and relation mappings to upper level ontology*

**Q4**: retrieve genes that are annotated with a specific enzymatic function:

DL query: gene that ‘has function’ some ‘acetylglucosaminyltransferase activity [go: 0008375]’
*simple conjunctive query with subclass reasoning*

**Q5:** retrieve genes that are annotated with a specific function that was not inferred by computational analysis.
DL query: gene that ‘has function’ some function that inverse(go_term) some (‘has evidence’ some (not ‘inferred from electronic annotation’))
*conjunctive query using negation, mappings, inverse*

**Q6:** retrieve organisms that have genes with an enzymatic activity that was not obtained by computational analysis
DL query: ‘Mammalia [taxid: 40674]’ that inverse(has_taxid) some (gene that 'has function' some (function that inverse(go_term) some ('has evidence' some (not 'inferred from electronic annotation')))
*conjunctive query with negation, inverse, mappings*

### 3.2 Querying using class axioms

All of the ontologies listed in Table 1 have rich class hierarchies. SIO and the Sequence Ontology (SO) also have axiomatic class definitions. DL queries can thus leverage the axioms used to define classes, as well as the class hierarchy.

**Q7:** retrieve a gene that encodes for a certain kind of molecule using SIO
DL query: gene and (encodes some ‘small cytoplasmic RNA (scRNA)’)
*reasoning with subclass axioms from mapped ontology*

**Q8**: retrieve a gene that encodes for a certain kind of molecule using SO
DL query: gene and (has_quality scRNA_encoding)
*reasoning with subclass axioms from mapped ontology*

### 3.3 SPARQL DL queries

SPARQL DL [14] is a subset of SPARQL that allows the formulation of queries using combination of OWL semantics and SPARQL variables. SPARQL DL is particularly useful in cases where one wishes to retrieve instances that are linked to some other resource, but also take advantage of DL reasoning. This is possible by using SPARQL variable bindings.

**Q9:** retrieve orthologous human and mouse genes annotated with function to bind ATP
Type(?human_gene, ‘gene’), Type(?mouse_gene, ‘gene’), Type(?homologene_group, HomoloGene_Group), PropertyValue(?human_gene, has_taxid, ‘Homo sapiens’), PropertyValue(?mouse_gene, has_taxid, ‘Mus musculus’), PropertyValue(?human_gene, ‘has function’, ‘ATP binding’),

PropertyValue(?mouse_gene, 'has function', 'ATP binding'), PropertyValue(?homologene_group, has_gene, ?human_gene), PropertyValue(?homologene_group, has_gene, ?mouse_gene)

## 4 Summary

We have described Gene-World, a large gene-centric knowledge base consisting of Bio2RDF datasets with over 395 million statements linked to five bio-ontologies with varying degrees of DL expressivity. The size and complexity of this dataset in addition to the provided DL and SPARQL-DL queries may provide a useful benchmark against which to evaluate OWL reasoner capability and efficiency for life science datasets. Should this preliminary knowledge base become useful in reasoner evaluation, we expect to extend it include more of the 20+ datasets and hundreds of ontologies in Bio2RDF.

## 5 References


1. Fernandez-Suarez XM, Galperin MY: **The 2013 Nucleic Acids Research Database Issue and the online molecular biology database collection**. *Nucleic Acids Res* 2013, **41**(Database issue):D1-7.
2. Chuang HY, Hofree M, Ideker T: **A decade of systems biology**. *Annu Rev Cell Dev Biol* 2010, **26**:721-744.
3. Callahan A, Cruz-Toledo J, Dumontier M: **Ontology-Based Querying with Bio2RDF's Linked Open Data**. *Journal of Biomedical Semantics* 2013, **4**(Supplement 1):S1.
4. Dentler K, Cornet R, ten Teije A, de Keizer N: **Comparison of reasoners for large ontologies in the OWL 2 EL profile**. *Semantic Web* 2011, **2**(2):71-87.
5. Guo Y, Pan Z, Heflin J: **LUBM: A benchmark for OWL knowledge base systems**. *Web Semantics: Science, Services and Agents on the World Wide Web* 2005, **3**(2-3):158-182.
6. Stearns MQ, Price C, Spackman KA, Wang AY: **SNOMED clinical terms: overview of the development process and project status**. *Proc AMIA Symp* 2001:662-666.
7. **Gene-World: A large-scale gene-centric semantic web knowledge base for molecular biology** [http://semanticscience.org/projects/gene-world]
8. Maglott D, Ostell J, Pruitt KD, Tatusova T: **Entrez Gene: gene-centered information at NCBI**. *Nucleic Acids Res* 2011, **39**(Database issue):D52-57.
9. **Database resources of the National Center for Biotechnology Information**. *Nucleic Acids Res* 2013, **41**(Database issue):D8-D20.
10. Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, Cherry JM, Davis AP, Dolinski K, Dwight SS, Eppig JT *et al*: **Gene ontology: tool for the unification of biology. The Gene Ontology Consortium**. *Nat Genet* 2000, **25**(1):25-29.
11. **OWL Export of GO DATABASE DAILY TERMDB** [http://archive.geneontology.org/latest-termdb/go_daily-termdb.owl.gz]
12. Eilbeck K, Lewis SE: **Sequence ontology annotation guide**. *Comp Funct Genomics* 2004, **5**(8):642-647.
13. **Gene-World DL and SPARQL-DL Queries** [http://semanticscience.org/projects/gene-world/gene-world-query-gists.html]
14. Sirin E, Parsia B: **SPARQL-DL: SPARQL Query for OWL-DL**. In: *3rd OWL Experiences and Directions Workshop (OWLED-2007).* Innsbruck, Austria; 2007.