The HermiT OWL Reasoner

Ian Horrocks, Boris Motik, and Zhe Wang

Oxford University Department of Computer Science
Oxford, OX1 3QD, UK
{ian.horrocks,boris.motik,zhe.wang}@cs.ox.ac.uk

Abstract. HermiT is the only reasoner we know of that fully supports the OWL 2 standard, and that correctly reasons about properties as well as classes. It is based on a novel "hypertableau" calculus that addresses performance problems due to nondeterminism and model size—the primary sources of complexity in state-of-the-art OWL reasoners. HermiT also incorporates a number of novel optimizations, including an optimized ontology classification procedure. Our tests show that HermiT performs well compared to existing tableau reasoners and is often much faster when classifying complex ontologies.

1 Introduction

HermiT is an OWL reasoning system based on a novel *hypertableau* calculus [12]. Like existing tableau based systems, HermiT reduces all reasoning tasks to ontology satisfiability testing, and proves the (un-)satisfiability of an ontology by trying to construct (an abstraction of) a suitable model. When compared to tableau calculi, however, the hypertableau technique can greatly reduce both the size of constructed models and the non-deterministic guessing used to explore all possible constructions. Moreover, HermiT employs a novel classification algorithm that greatly reduces the number of subsumption (and hence satisfiability) tests needed to classify a given ontology.

Our tests show that HermiT is as fast as other OWL reasoners when classifying relatively easy-to-process ontologies, and usually much faster when classifying more difficult ontologies. Moreover, HermiT is currently the only reasoner known to us that fully supports the OWL 2 standard: it supports all of the datatypes specified in the standard, and it correctly reasons about properties as well as about classes. Most other reasoners support only a subset of the OWL 2 datatypes [11], and all other OWL reasoners known to us implement only syntax based reasoning when classifying properties, and may thus fail to detect non-trivial but semantically entailed sub-property relationships [4].

HermiT also includes some nonstandard functionality that is currently not available in any other system. In particular, HermiT supports reasoning with ontologies containing *description graphs*. As shown in [10], description graphs allow for the representation of structured objects—objects composed of many parts interconnected in arbitrary ways. These objects abound in bio-medical ontologies such as FMA and GALEN, but they cannot be faithfully represented in OWL.

HermiT is available as an open-source Java library, and includes both a Java API and a simple command-line interface. We use the OWL API [6] both as part of the public Java interface and as a parser for OWL files; HermiT can thus process ontologies in any format handled by the OWL API, including RDF/XML, OWL Functional Syntax, KRSS, and OBO.

2 Architecture and Optimizations

On OWL ontology $\mathcal{O}$ can be divided into three parts: the property axioms, the class axioms, and the facts. These correspond to the RBox $\mathcal{R}$, TBox $\mathcal{T}$, and ABox $\mathcal{A}$ of a Description Logic knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$. All basic reasoning tasks, including subsumption checking, can be reduced to testing the satisfiability of such a knowledge base. For example, $\mathcal{K} \models A \sqsubseteq B$ iff $(\mathcal{R}, \mathcal{T}, \mathcal{A} \cup \{A \sqcap \neg B(s)\})$ is not satisfiable, where s is a "fresh" individual (i.e., one that does not occur in $\mathcal{K}$).

To show that a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ is satisfiable, a tableau algorithm constructs a *derivation*—a sequence of ABoxes $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n$, where $\mathcal{A}_0 = \mathcal{A}$ and each $\mathcal{A}_i$ is obtained from $\mathcal{A}_{i-1}$ by an application of one *inference rule*. The inference rules make the information implicit in the axioms of $\mathcal{R}$ and $\mathcal{T}$ explicit, and thus evolve the ABox $\mathcal{A}$ towards (an abstraction of) a model of $\mathcal{K}$. The algorithm terminates either if no inference rule is applicable to some $\mathcal{A}_n$, in which case $\mathcal{A}_n$ represents a model of $\mathcal{K}$, or if $\mathcal{A}_n$ contains an obvious contradiction, in which case the model construction has failed. The following inference rules are commonly used in DL tableau calculi.

- $\sqcup$-rule: Given $(C_1 \sqcup C_2)(s)$, derive either $C_1(s)$ or $C_2(s)$.
- $\sqcap$-rule: Given $(C_1 \sqcap C_2)(s)$, derive $C_1(s)$ and $C_2(s)$.
- $\exists$-rule: Given $(\exists R.C)(s)$, derive $R(s,t)$ and $C(t)$ for t a fresh individual.
- $\forall$-rule: Given $(\forall R.C)(s)$ and $R(s,t)$, derive $C(t)$.
- $\sqsubseteq$-rule: Given an axiom $C \sqsubseteq D$ and an individual s, derive $(\neg C \sqcup D)(s)$.

The $\sqcup$-rule is nondeterministic, and the knowledge base $\mathcal{K}$ is unsatisfiable if and only if all choices lead to a contradiction.

Or-Branching This "case based" procedure for handing disjunctions is sometimes called *or-branching*. The $\sqsubseteq$-rule is the main source of or-branching, as it adds a disjunction for each TBox axiom to each individual in an ABox, even if the corresponding axiom is equivalent to a Horn clause, and so inherently deterministic. Such indiscriminate application of the $\sqsubseteq$-rule can be a major source of inefficiency, and this has been addressed by various *absorption* optimizations [2, Chapter 9], including role absorption [14] and binary absorption [8].

HermiT's hypertableau algorithm generalizes these optimizations by rewriting description logic axioms into a form which allows all such absorptions be performed simultaneously, as well as allowing additional types of absorption impossible in standard tableau calculi. Furthermore, HermiT actually rewrites DL concepts to further reduce nondeterminism, and is thus able to apply absorption-style optimizations much more pervasively.

And-Branching The introduction of new individuals in the $\exists$-rule is sometimes called *and-branching*, and it is another major source of inefficiency in tableau algorithms [2]. To ensure termination, tableau algorithms employ *blocking* to prevent infinitely repeated application of the $\exists$-rule [7]. Standard blocking is applied only along a single "branch" of fresh individuals. HermiT uses a more aggressive *anywhere blocking* strategy [12] that can reduce the size of generated models by an exponential factor, and this substantially improves real-world performance on many difficult and complex ontologies.

HermiT also tries to further reduce the size of the generated model using a technique called *individual reuse*: when expanding an existential $\exists R.C$, it first attempts to re-use some existing individual labeled with C to construct a model, and only if this model construction fails does it introduce a new individual. This approach allows HermiT to consider non-tree-shaped models, and drastically reduces the size of models produced for ontologies which describe complex structures, such as ontologies of anatomy. "Reused" individuals, however, are semantically equivalent to nominal concepts, and thus performance gains due to individual reuse are highly dependent upon the efficient handling of nominals. HermiT therefore uses an optimised *nominal introduction rule* that reduces nondeterminism, and is more conservative in its introduction of new nominals.

Caching Blocking Labels Anywhere blocking avoids repetitive model construction in the course of a single satisfiability test. HermiT further extends this approach to avoid repetitive construction across an entire set of satisfiability tests. Conceptually, instead of performing n different tests by constructing n different models, it performs a single test which constructs a single model containing n independent fragments. Although no two fragments are connected, the individuals in one fragment can block those in another, greatly reducing the size of the combined model. In practice, tests are not actually performed simultaneously. Instead, after each test a compact representation of the model generated is retained for the purpose of blocking in future tests. This naïve strategy is, however, not compatible with ontologies containing nominals, which could connect the models from independent tests.

This optimization has been key to obtaining the results that we present in Section 3. For example, on GALEN only one satisfiability test is costly because it computes a substantial part of a model of the TBox; all subsequent satisfiability tests reuse large parts of that model.

Classification Optimizations DL reasoning algorithms are often used in practice to compute a *classification* of a knowledge base $\mathcal{K}$—that is, to determine whether $\mathcal{K} \models A \sqsubseteq B$ for each pair of atomic concepts A and B occurring in $\mathcal{K}$. Clearly, a naïve classification algorithm would involve a quadratic number of subsumption tests, each of which can potentially be expensive. To obtain acceptable levels of performance, various optimizations have been developed that reduce the number of subsumption tests [3] and the time required for each test [2, Chapter 9].

HermiT employs a novel classification procedure, and exploits the unique properties of the system's new calculus to further optimise the procedure. In

Table 1: Statics of the ontologies

Ontology Name	DL Expressivity	Classes	Properties	TBox	RBox
EMap (Feb09)	$\mathcal{EL}$	13737	2	13730	0
GO Term DB (Feb06)	$\mathcal{EL}++$	20526	1	28997	1
DLP ExtDnS 397	$\mathcal{SHIN}$	96	186	232	675
LUBM (one university)	$\mathcal{ALEHI}^+(D)$	43	32	142	51
Biological Process (Feb09)	$\mathcal{EL}++$	16303	5	32286	3
MGED Ontology	$\mathcal{ALCOF}(D)$	229	104	452	21
RNA With Individual (Dec09)	$\mathcal{SRIQ}(D)$	244	93	364	310
NCI Thesaurus (Feb09)	$\mathcal{ALCH}(D)$	70576	189	100304	290
OBI (Mar10)	$\mathcal{SHOIN}(D)$	2638	83	9747	150
FMA Lite (Feb09)	$\mathcal{ALEI}^+$	75145	3	119558	3
FMA-constitutional part (Feb06)	$\mathcal{ALCOIF}(D)$	41648	168	122695	395
GALEN-doctored	$\mathcal{ALEHIF}^+$	2748	413	3937	799
GALEN-undoctored	$\mathcal{ALEHIF}^+$	2748	413	4179	800
GALEN-module1	$\mathcal{ALEHIF}^+$	6362	162	14515	219
GALEN-full	$\mathcal{ALEHIF}^+$	23136	950	35531	2165

particular, when it tries to construct a model I of $\mathcal{K} \cup \{A(a)\}$ (in order to determine the satisfiability of A), HermiT is able to exploit the information in I to derive a great deal of information about both subsumers and non-subsumers of A, information that can be efficiently exploited by the new classification procedure [4].

3 Empirical Results

To evaluate our reasoning algorithm in practice, we compared HermiT with the state-of-the-art tableau reasoners Pellet 2.3.0 [13], and FaCT++ 1.5.3 [15].

We selected a number of standard test ontologies, and measured the time needed to classify them using each of the mentioned reasoners. Unlike Pellet and FaCT++, HermiT does not include a dedicated reasoner for any tractable fragment of OWL 2. Hence, we mainly focus on ontologies that exploit most or all of the expressive power available in OWL 2. All tests were performed on a 2.7 GHz MacBook Pro with 8 GB of physical memory. A classification attempt was aborted if it exhausted all available memory (Java tools were allowed to use 2 GB of heap space), or if it exceeded a timeout of 20 minutes.

The majority of the test ontologies were classified very quickly by all three reasoners. For these "trivial" ontologies, the performance of HermiT was comparable to that of the other reasoners. Therefore, we consider here only the test results for "interesting" ontologies—that is, ontologies that are either not trivial or on which the tested reasoners exhibited a significant difference in performance (see Table 1 for details of these ontologies).

Table 2 summarizes the results of our tests on these "interesting" ontologies. Since HermiT has no special handling for tractable fragments of OWL 2, the performance of HermiT on such ontologies may not be competitive. For example,

Table 2: Results of Performance Evaluation

Ontology Name	Classification Times (seconds)		
	HermiT	Pellet	FaCT++
EMap (Feb09)	1.1	0.4	34.2
GO Term DB (Feb06)	1.3	1.3	6.1
DLP ExtDnS 397	1.3	timeout	0.05
LUBM (one university)	1.7	0.7	152.7
Biological Process (Feb09)	1.8	4.0	8.0
MGED Ontology	2.1	19.6	0.04
RNA With Individual (Dec09)	2.7	0.8	102.9
NCI Thesaurus (Feb09)	58.2	12.3	4.4
OBI (Mar10)	150.0	timeout	17.2
FMA Lite (Feb09)	211.1	timeout	timeout
FMA-constitutional part (Feb06)	1638.3	timeout	396.9
GALEN-doctored	1.8	timeout	2.5
GALEN-undoctored	6.7	out of mem.	11.6
GALEN-module1	out of mem.	timeout	timeout
GALEN-full	out of mem.	timeout	timeout

FaCT++ shows advantages when classifying ontologies which fall into a predefined syntactic fragment for which it uses a more efficient reasoning technique [16]. Different versions of GALEN have commonly been used for testing the performance of DL reasoners. The full version of the ontology (called GALEN-full) cannot be processed by any of the reasoners. Thus, we extracted a module (called GALEN-module1) based on a single concept from GALEN-full using the techniques from [5] in order to determine if modularisation techniques might make classification feasible. However, although the module is much smaller than the full ontology, no reasoner was able to classify it either. Our analysis has shown that, due to a large number of cyclic axioms, the reasoners construct extremely large ABoxes and eventually exhaust all available memory (or get lost in the resulting large search space). FMA-constitutional part exhibits similar features, but to a lesser extent, and both HermiT and FaCT++ were able to classify it. Because of the failure of DL reasoners to process GALEN-full, various simplified versions of GALEN have often been used in practice. As Table 2 shows, these ontologies can still be challenging for state-of-the-art reasoners. HermiT, however, can classify them quite efficiently.

4 Conclusions and Future Directions

We have described HermiT, an OWL reasoner based on novel algorithms and optimizations. HermiT fully supports the OWL 2 standard, and shows significant performance advantages over other reasoners across a wide range of expressive real-world ontologies. Although not always the fastest, HermiT exhibits relatively robust performance on our tested ontologies, and as shown in our results, it never failed to classify an ontology in the test corpus that was successfully handled by

one of the other reasoners. HermiT also includes support for some non-standard ontology features, such as description graphs.

We are continuing to develop HermiT, and to explore new and refined optimization techniques. We also continue to extend its functionality: the latest version, for example, provides support for the SPARQL 1.1 query language [1]. We are also investigating techniques for exploiting specialized reasoning techniques, such as those implemented in the ELK system [9], to speed up the classification of ontologies that are (largely) within a fragment of OWL that such techniques can handle.

Acknowledgments This work was supported by the EU FP7 project SEALS and by the EPSRC projects ConDOR, ExODA, and LogMap.

References

1. SPARQL 1.1 Query Language. W3C Working Draft, 12 May 2011.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. 2nd edition, 2007.
3. F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, and E. Franconi. Making KRIS Get a Move on. *Applied Intelligence*, 4(2):109–132, 1994.
4. B. Glimm, I. Horrocks, B. Motik, R. Shearer, and G. Stoilos. A Novel Approach to Ontology Classification. *J. of Web Semantics*, 10(1), 2011.
5. B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular Reuse of Ontologies: Theory and Practice. *JAIR*, 31:273–318, 2008.
6. M. Horridge and S. Bechhofer. The OWL API: A Java API for Working with OWL 2 Ontologies. In *Proc. OWLED 2009*, 2009.
7. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$. In *Proc. CADE-17*, pages 482–496, 2000.
8. A. K. Hudek and G. Weddell. Binary Absorption in Tableaux-Based Reasoning for Description Logics. In *Proc. DL 2006*, 2006.
9. Y. Kazakov, M. Krötzsch, and F. Simančík. Concurrent Classification of EL Ontologies. In *Proc. of ISWC 2011*, pages 305–320, 2011.
10. B. Motik, B. Cuenca Grau, I. Horrocks, and U. Sattler. Representing Ontologies Using Description Logics, Description Graphs, and Rules. *Artificial Intelligence*, 173(14):1275–1309, 2009.
11. B. Motik and I. Horrocks. OWL Datatypes: Design and Implementation. In *Proc. of ISWC 2008*, pages 307–322, 2008.
12. B. Motik, R. Shearer, and I. Horrocks. Hypertableau Reasoning for Description Logics. *JAIR*, 36:165–228, 2009.
13. E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A Practical OWL-DL Reasoner. *J. of Web Semantics*, 5(2):51–53, 2007.
14. D. Tsarkov and I. Horrocks. Efficient Reasoning with Range and Domain Constraints. In *Proc. DL 2004*, 2004.
15. D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. IJCAR 2006*, pages 292–297, 2006.
16. D. Tsarkov, I. Horrocks, and P. F. Patel-Schneider. Optimizing Terminological Reasoning for Expressive Description Logics. *J. of Automated Reasoning*, 39(3):277–316, 2007.