HermiT: A Highly-Efficient OWL Reasoner

Rob Shearer, Boris Motik, and Ian Horrocks

Oxford University Computing Laboratory
Oxford, OX1 3QD, UK
{rob.shearer,boris.motik,ian.horrocks}@comlab.ox.ac.uk

Abstract. HermiT is a new OWL reasoner based on a novel "hypertableau" calculus. The new calculus addresses performance problems due to nondeterminism and model size—the primary sources of complexity in state-of-the-art OWL reasoners. The latter is particularly important in practice, and it is achieved in HermiT with an improved blocking strategy and and an optimization that tries to reuse existing individuals rather than generating new ones. HermiT also incorporates a number of other novel optimizations, such as a more efficient approach to handling nominals, and various techniques for optimizing ontology classification. Our tests show that HermiT is usually much faster than other reasoners when classifying complex ontologies, and it is already able to classify a number of ontologies which no other reasoner has been able to handle.

1 Introduction

Reasoning services for Description Logic ontologies, such as subsumption testing and classification, are usually performed by testing the consistency of a number of knowledge bases derived from the original ontology [1]. Satisfiability of a class, for example, is reduced to checking the consistency of a knowledge base in which at least one individual is an instance of that class. Tableau reasoners perform such consistency tests by attempting to construct a model for the knowledge base. The difficulties in constructing such models primarily arise from two sources. First, there are often a great number of different possible constructions which might be models; in general a tableau algorithm must analyze every one of these possibilities before concluding that no model is possible. Second, the models built by tableau reasoners can be extremely large, even for relatively small ontologies. These two sources of complexity also frequently interact: when the models constructed are large there are also usually more potential models which need to be considered, and reasoning can become impossible in practice.

HermiT is a Description Logic reasoning system based on an entirely new architecture which addresses both of these sources of complexity. HermiT implements a "hypertableau" calculus which greatly reduces the number of possible models which must be considered (down to only a single possibility for a significant subset of ontologies). HermiT also incorporates the "anywhere blocking" strategy, which limits the sizes of models which are constructed. Finally, HermiT makes use of a novel and highly-efficient approach to handling nominals in the

presence of number restrictions and inverse roles; we expect that this will allow ontology authors to make much freer use of nominals than has been possible to date. This combination of fundamental algorithmic improvements also enables a range of additional optimizations.

Our tests show that HermiT is as fast as other DL reasoners when classifying relatively easy-to-process ontologies, and usually much faster when classifying more difficult ontologies. In fact, HermiT can classify a number of ontologies which no other reasoner has previously been able to handle.

The HermiT system also serves as a platform for prototypical implementations of new language features. For example, HermiT already includes support for reasoning with ontologies which include description graphs [8].

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 both as part of the public Java interface and as a parser for OWL files; HermiT can 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.

2.1 Reducing Tableau Complexity

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 a (representation 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: if $(C_1 \sqcup C_2)(s)$ is true, then $C_1(s)$ or $C_2(s)$ or both are true. Therefore, tableau calculi make a nondeterministic guess and choose either C_1 or C_2 If choosing C_1 leads to a contradiction, the algorithm must backtrack and try C_2; this procedure is known as reasoning by case. The knowledge base $\mathcal{K}$ is unsatisfiable if and only if all choices fail to construct a model. We next discuss several sources of complexity in this procedure, and how HermiT addresses them.

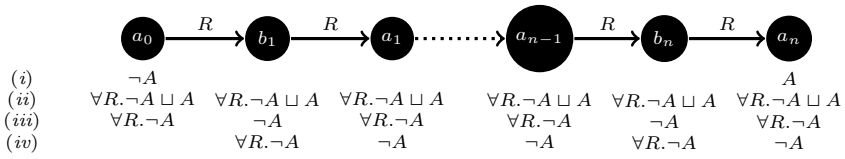


Fig. 1: Or-Branching Example

Or-Branching Handing disjunctions through reasoning by case is often 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 and can be a major source of inefficiency [1, Chapter 9]. For example, let $\mathcal{T}_1$ and $\mathcal{A}_1$ be a TBox and an ABox as specified in (1).

$$\begin{aligned} \mathcal{T}_1 &= \{\exists R.A \sqsubseteq A\} \\ \mathcal{A}_1 &= \{\neg A(a_0),\ R(a_0, b_1),\ R(b_1, a_1), \ldots,\ R(a_{n-1}, b_n),\ R(b_n, a_n),\ A(a_n)\} \end{aligned} \tag{1}$$

The ABox $\mathcal{A}_1$ is graphically shown in Figure 1. The individuals occurring in the ABox are represented as black dots, an assertion of the form $A(a_0)$ is represented by placing A next to the individual a_0, and an assertion of the form $R(a_0, b_1)$ is represented as an R-labeled arrow from a_0 to b_1. Initially, $\mathcal{A}_1$ contains only the concept assertions shown in line (i).

To satisfy the axiom in $\mathcal{T}_1$, a tableau algorithm applies the $\sqsubseteq$-rule, thus adding the assertions shown in line (ii) of Figure 1. Tableau algorithms are usually free to choose the order in which they process the assertions in an ABox; tableau systems often use advanced heuristics to try to find an order that exhibits good performance in practice [14]. Let us assume that the algorithm chooses to process the assertions on a_i before those on b_j. Hence, by applying the rules to all a_i, the algorithm derives the assertions shown in line (iii) of Figure 1; after that, by applying the rules to all b_i, the algorithm derives the assertions shown in line (iv) of Figure 1. The ABox now contains both $A(a_n)$ and $\neg A(a_n)$, which is a contradiction. Thus, the algorithm needs to backtrack its most recent choice, so it flips its guess on b_{n-1} to $A(b_{n-1})$. This generates a contradiction on b_{n-1}, so the algorithm backtracks from all guesses for b_i, changes the guess on a_n to $A(a_n)$, and repeats the work for all b_i. This also leads to a contradiction, so the algorithm must revise its guess for a_{n-1}; but then, two guesses are again possible for a_n. In general, after revising a guess for a_i, all possibilities for a_j, $i < j \leq n$, must be reexamined, which results in exponential behavior. None of the standard backtracking optimizations [1, Chapter 9] are helpful: the problem arises because the order in which the individuals are processed makes the guesses on a_i independent from the guesses on a_j for $i \neq j$.

The axiom $\exists R.A \sqsubseteq A$, however, is not inherently nondeterministic: it is equivalent to the Horn clause $R(x, y) \wedge A(y) \rightarrow A(x)$, which can be applied bottom-up to derive the assertions $A(b_n), A(a_{n-1}), \ldots, A(a_0)$ and reveal a contradiction on a_0. These inferences are deterministic, so we can conclude that $\mathcal{K}_1$ is unsatisfiable without any backtracking. This example suggests that the processing of TBox axioms in tableau algorithms can be unnecessarily nondeterministic.

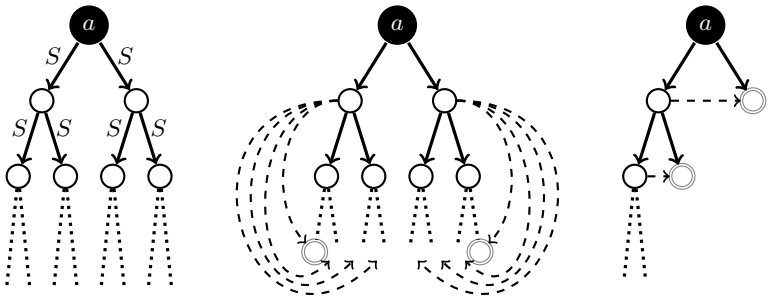
Various *absorption* optimizations [1, Chapter 9] have been developed to address this problem. The basic absorption algorithm tries to rewrite TBox axioms into the form $B \sqsubseteq C$ where B is an atomic concept. Then, instead of deriving $\neg B \sqcup C$ for each individual in an ABox, $C(s)$ is derived only if the ABox contains $B(s)$; thus, the absorbed axioms can be applied in a "more deterministic" way. This technique has been extended in several ways. *Role absorption* [13] rewrites axioms into the form $\exists R.\top \sqsubseteq C$; then, $C(s)$ is derived only if an ABox contains $R(s,t)$. *Binary absorption* [7] rewrites GCIs into the form $B_1 \sqcap B_2 \sqsubseteq C$; then, $C(s)$ is derived only if an ABox contains both $B_1(s)$ and $B_2(s)$. Neither of these two optimizations, however, helps us deal with the axiom in (1) directly. Role absorption produces an axiom $\exists R.\top \sqsubseteq A \sqcup \forall R.\neg A$, which still contains a disjunction in the consequent. Furthermore, binary absorption is not applicable to (1), since the axiom does not contain two concepts on the left-hand side of the implication symbol $\sqsubseteq$. The axiom (1) can be absorbed if it is rewritten as $A \sqsubseteq \forall R^{-}.A$. In practice, however, it is often unclear in advance which combination of transformation and absorption techniques will yield the best results; absorption algorithms are, therefore, typically guided primarily by heuristics and may not eliminate all nondeterminism.

HermiT's hypertableau algorithm generalizes these absorption optimizations by rewriting description logic axioms into a form which allows standard absorption, role absorption, and binary absoprtion to be performed simultaneously, as well as allowing additional types of "absorption" impossible in standard tableau calculi. In the hypertableau calculus, an axiom $A \sqcap B \sqcap \exists R.C \sqsubseteq D$ would only introduce $D(s)$ if the ABox already contained $A(s)$, $B(s)$, $R(s,t)$, and $C(t)$ for some t. Furthermore, HermiT actually rewrites DL concepts to further reduce nondeterminism. Testing satisfiability of the concept $\neg A \sqcup B$ causes nondeterministic application of the $\sqcup$-rule in standard tableau reasoners; HermiT transforms this concept into an expression equivalent to $A \sqsubseteq B$, and is thus able to apply absorption-style optimizations much more pervasively than standard tableau reasoners can.

And-Branching The introduction of new individuals in the $\exists$-rule is called *and-branching*, and it is another major source of inefficiency in tableau algorithms [1]. Consider, for example, the following (satisfiable) knowledge base $\mathcal{K}_2$.

$$\begin{aligned} \mathcal{T}_2 &= \{\ A_0 \sqsubseteq \geq 2\, S.A_1,\ \ldots,\ A_{n-1} \sqsubseteq \geq 2\, S.A_n,\ A_n \sqsubseteq A_1\ \} \\ \mathcal{A}_2 &= \{\ A_0(a)\ \} \end{aligned} \tag{2}$$

At-least restrictions are dealt with in tableau algorithms by the $\geq$-rule, which is quite similar to the $\exists$-rule: from $\geq n\, R.C(s)$, the $\geq$-rule derives $R(s,t_i)$ and $C(t_i)$ for $1 \leq i \leq n$, and $t_i \not\approx t_j$ for $1 \leq i < j \leq n$. Thus, the assertion $A_0(a)$ implies the existence of at least two individuals in A_1, each of which imply the existence of at least two individuals in A_2, and so on. Given $\mathcal{K}_2$, a tableau algorithm thus constructs a binary tree, shown in Figure 2a. Each individual at depth n is an instance of A_n; because of the GCI $A_n \sqsubseteq A_1$, this individual



(a) Without blocking (b) Ancestor Blocking (c) Anywhere Blocking

Fig. 2: And-Branching Example

must be an instance of A_1 as well, so we can repeat the whole construction and generate an even deeper tree. Clearly, a naïve application of the tableau rules does not terminate if the TBox contains existential quantifiers in cycles.

To ensure termination in such cases, tableau algorithms employ *blocking* [6]: if the two individuals s and t are identical[1], then we say that s *directly blocks* t and we need not apply further expansion rules to t. Intuitively, blocking ensures that the part of the ABox rooted at s "behaves" just like the part rooted at t, so we can generate a model by replacing the individual t with a copy of the tree rooted at s. The model may be infinite (the copy of the s subtree may include a copy of t, which will be replaced with another copy of s, and so on), but we need never actually construct it—an ABox with blocked individuals is sufficient to prove that such a model exists.

Standard tableau algorithms only allow individuals to be blocked by their ancestors—this is called *ancestor blocking*. This causes the derivation for $\mathcal{K}_2$ to terminate but results in the exponentially-large construction shown in Figure 2b, where blocking is indicated by dashed lines. HermiT extends this blocking strategy such that an individual can be blocked by (almost) any other individual. On $\mathcal{K}_2$ our improved *anywhere blocking* approach results in the construction shown in Figure 2c. Anywhere blocking can reduce the size of generated models by an exponential factor, and this substantially improves real-world performance on many difficult and complex ontologies.

Although anywhere blocking can often prevent the creation of multiple copies of identical individuals, it is not uncommon for tableau procedures to produce models containing a great many very similar individuals. If an expression $\exists R.C$ occurs in different parts of a partially-constructed model, then multiple individuals labeled with C will be created, and if the structures surrounding these new individuals differ in any way then one will not block the other. In many cases this results in unnecessary replication. HermiT takes advantage of this observation through *individual reuse*: when we expand an existential $\exists R.C$ we first attempt to re-use some existing individual labeled with C to construct a model, and only

[1] The definition of "identical" depends upon the logic used.

if this model construction fails do we 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 efficient handling of nominals.

Nominal Generation In logics which include both inverses and number restrictions, *nominal* concepts—concepts which refer to a particular individual in the ABox—make the blocking rules more complex. Because each nominal has a unique identity, it cannot be copied, and thus cannot appear in a part of a subtree which occurs multiple times in a model due to blocking. In fact, the combination of inverse roles and number restrictions can limit the number of neighbors of a nominal node, making them "unique" and uncopyable as well; these neighbors can impose uniqueness constraints on neighbors of neighbors, and so on. In order to ensure the correctness of blocking, it is necessary to identify precisely which individuals can be copied and which are unique.

Standard tableau algorithms identify unique individuals, called *root individuals*, recursively, beginning with the individuals in the initial ABox. Whenever there are number restrictions and inverse roles which limit the number of neighbors of such a root individual, the tableau NN-rule guesses exactly how many such neighbors will exist in the final model and constructs an appropriate number of root individuals. Number restrictions on these new individuals can cause another application of the NN-rule to produce new root neighbors-of-neighbors, and so on. This procedure will eventually terminate, but each possible "guess" for each NN-rule application must be explored if a model cannot be found, and each application can produce a large number of new individuals, leading to larger models. A single large number in a number restriction can make reasoning using the NN-rule completely impractical.

HermiT addresses this problem by replacing the NN-rule with an NI-rule which does not introduce new root individuals but instead simply labels existing individuals as roots. By keeping track of unique identifiers for each root individual this approach is able to ensure correctness of the algorithm without increasing the sizes of the models constructed.

2.2 Additional 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 calls to the subsumption checking algorithm, each of which can potentially be highly expensive. To obtain acceptable levels of performance, various optimizations have been developed that reduce the number of subsumption checks [2] and the time required for each check [1, Chapter 9]. Along these lines, HermiT implements a number of new

classification optimizations which exploit unique properties of the system's new model construction calculus.

Reading Classification Relationships from ABox Labels In order to check satisfiability of a concept A in knowledge base $\mathcal{K}$, HermiT creates the knowledge base $\mathcal{K}' = \mathcal{K} \cup \{A(a)\}$, where a is a new individual, and attempts to construct a model of $\mathcal{K}'$. If A is satisfiable, then the construction will yield a model I. HermiT is able to exploit the information in I to derive information about A beyond its satisfiability.

If the assertion $B(a)$ does not occur in I, then it is clearly possible for an individual to be a member of A without being a member of B, thus we can conclude $A \not\sqsubseteq B$. If the assertion $B(a)$ does occur in I, and the derivation of $B(a)$ does not depend on any nondeterministic choice (and thus $B(a)$ would appear in any model), this proves that $A \sqsubseteq B$. HermiT's ability to minimize nondeterminism often makes it possible to perform most of the classification of a concept A using only a single satisfiability check.

Caching Blocking Labels HermiT's anywhere blocking technique, described in Section 2.1, avoids the creation of identical sub-models in the course of a consistency test; we further extend this approach to avoid the creation of identical sub-models across an entire set of consistency tests. Conceptually, instead of performing n different tests by constructing n different models, we perform 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 not compatible with ontologies containing nominals, however, 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 subsumption test is costly because it computes a substantial part of a model of the TBox; all subsequent subsumption tests reuse large parts of that model.

2.3 Features

HermiT 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 [8], 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.

3 Empirical Results

To evaluate our reasoning algorithm in practice, we compared HermiT with the state-of-the-art tableau reasoners Pellet 1.5.1 [10], and FaCT++ 1.1.10 [15]. Pellet and FaCT++ are based on the existing reasoning algorithms [5], so they differ from HermiT in both derivation rules and blocking strategy. (Both Pellet and FaCT++ employ ancestor blocking.) In order to estimate the practical impact of these two differences separately, we implemented a version of HermiT with ancestor blocking, which we call HermiT-Anc.

We selected test ontologies from the Gardiner ontology suite [3], the Open Biological Ontologies (OBO) Foundry[2], and several variants of the GALEN ontology [11]. Most ontologies from the Gardiner and OBO collections contain datatypes, which are currently not supported in HermiT; therefore, we have converted datatypes in these ontologies to atomic classes.

We measured the time needed to classify each test ontology using all of the mentioned reasoners. All tests were performed on a 2.2 GHz MacBook Pro with 2 GB of physical memory. A classification attempt was aborted if it exhausted all available memory (Java tools were allowed to use 1.5 GB of heap space), or if it exceeded a timeout of 20 minutes.

The majority of the test ontologies were classified in under a second by HermiT, and under ten seconds by Pellet and FaCT++. For these "trivial" ontologies, the performance of HermiT was comparable to that of the other reasoners. Therefore, we consider here only the tests results for "interesting" ontologies—that is, ontologies that are either not trivial or on which the tested reasoners exhibited a significant difference in performance.

Table 1 summarizes the results of tests on the "interesting" ontologies. In most cases, HermiT performs as well as or better than the other reasoners.

HermiT performs worse than Pellet and FaCT++ on the DLP ExtDnS ontology. This ontology includes a substantially more complex RBox than most other ontologies in the test suite, with 384 role axioms. The tested version of HermiT implements transitivity through axiom rewriting; our analysis revealed that HermiT's poor performance on DLP ExtDnS is due to inefficiencies in this rewriting. We are currently working to develop more efficient role encodings.

HermiT also performs worse than Pellet and FaCT++ on the MGED ontology. This ontology contains nominals, as well as a moderately complex ABox (over 600 assertions). Since the ontology uses nominals, the ABox must be taken into account when classifying the ontology. HermiT is not yet optimized for ABox reasoning.

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. To simplify the ontology, we extracted a module (called GALEN-module1) from GALEN-full using the techniques from [4]. Although the module is much smaller than the full ontology,

[2] `http://obofoundry.org/`

Table 1: Results of Performance Evaluation

Ontology Name	Classification Times (seconds)			
	HermiT	HermiT-Anc	Pellet	FaCT++
Fly Taxonomy	1.1	1.2	1.2	5.3
GO Term DB	1.6	1.8	36.4	19.2
Biological Process	2.4	1.6	10.7	79.2
NCI	2.8	3.7	17.0	30.2
MGED	5.7	11.2	0.8	0.249
BP XP OBOL	8.7	8.5	505.1	1742.3
OWL Guide Food	19.3	29.6	14.2	1388.1
FMA Lite	43.8	error	error	error
DLP ExtDnS	95.8	error	7.1	0.1
FMA-constitutional part	error	error	error	error
GALEN-horrocks	1.5	1.5	13.5	156.9
Not-GALEN	1.6	1.8	54.1	200.4
GALEN-doctored	3.9	4.9	error	2836.1
GALEN-original	11.9	error	error	error
GALEN-module1	error	error	error	error
GALEN-full	error	error	error	error

no reasoner was able to classify it either. Similarly, no reasoner could classify FMA-constitutional part. Our analysis has shown that, due to a large number of cyclic axioms, on these ontologies reasoners construct extremely large ABoxes and eventually exhaust all available memory. Our individual reuse technique is designed to address this issue; the above tests were conducted using a version of HermiT which did not include this optimization.

Because of the failure of DL reasoners to process GALEN-full, various simplified versions of GALEN have often been used in practice. As Table 1 shows, these ontologies are still challenging for state-of-the-art reasoners. HermiT, however, can classify them quite efficiently; in fact, HermiT is the only reasoner that can classify GALEN-original. All the other reasoners, including HermiT-Anc, quickly run out of memory on GALEN-original; this suggests that, by drastically reducing the sizes of generated ABoxes, anywhere blocking can mean the difference between success and failure on complex ontologies.

On ontologies that can be processed by both HermiT and HermiT-Anc, both reasoners show comparable performance, suggesting that the ABoxes generated on these ontologies are not particularly large. On some of these ontologies (e.g., BP XP OBOL and OWL Guide Food), other reasoners can perform significantly more slowly; this suggests that the increase in HermiT's performance is mainly due to the hypertableau rule application strategy and reduced nondeterminism. Thus, while the hypertableau strategy may not be as important as anywhere blocking in determining the practical limits of DL reasoners, it can still lead to significant performance improvements in practice.

4 Conclusions and Future Directions

We have described HermiT, a new reasoner for SHOIQ+ (and OWL) based on novel algorithms and optimizations. HermiT shows significant performance advantages over other reasoners across a wide range of real-world ontologies. In several cases, HermiT is able to classify ontologies that no other reasoner can process. HermiT also includes support for some non-standard ontology features, such as description graphs.

We intend to continue to develop HermiT to track the emerging OWL 2.0 standard, including extended datatype support. We expect the performance of HermiT to continue to improve as we refine our optimization techniques, including the development of heuristics to maximize the benefit of our individual reuse technique.

In our future work, we intend to extend the ABox reasoning capabilities of HermiT with both a more expressive ABox query interface as well as new optimization techniques which allow reasoning with extremely large ABoxes.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. 2nd edition, 2007.
2. 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.
3. T. Gardiner, I. Horrocks, and D. Tsarkov. Automated Benchmarking of Description Logic Reasoners. In *Proc. DL 2006*, volume 189 of *CEUR Workshop Proceedings*.
4. B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular Reuse of Ontologies: Theory and Practice. *JAIR*, 31:273–318, 2008.
5. I. Horrocks and U. Sattler. A Tableau Decision Procedure for $\mathcal{SHOIQ}$. *Journal of Automated Reasoning*, 39(3):249–276, 2007.
6. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$. In *Proc. CADE-17*, pages 482–496, Pittsburgh, USA, 2000.
7. A. K. Hudek and G. Weddell. Binary Absorption in Tableaux-Based Reasoning for Description Logics. In *Proc. DL 2006*, Windermere, UK, May 30-June 1 2006.
8. B. Motik, B. Cuenca Grau, and U. Sattler. Structured Objects in OWL: Representation and Reasoning. In *Proc. WWW 2008*, Beijing, China, 2008.
9. B. Motik and I. Horrocks. Individual Reuse in Description Logic Reasoning. In *Proc. IJCAR 2008)*, Sydney, Australia, August 10–15 2008. To appear.
10. B. Parsia and E. Sirin. Pellet: An OWL-DL Reasoner. Poster, In Proc. ISWC 2004, Hiroshima, Japan, November 7–11, 2004.
11. A. L. Rector, W. A. Nowlan, and A. Glowinski. Goals for concept representation in the GALEN project. In *Proc. SCAMC '93*, Washington DC, USA, 1993.
12. E. Sirin, B. Cuenca Grau, and B. Parsia. From Wine to Water: Optimizing Description Logic Reasoning for Nominals. In *Proc. KR 2006*, Lake District, UK.
13. D. Tsarkov and I. Horrocks. Efficient Reasoning with Range and Domain Constraints. In *Proc. DL 2004*, Whistler, BC, Canada, June 6–8 2004.
14. D. Tsarkov and I. Horrocks. Ordering Heuristics for Description Logic Reasoning. In *Proc. IJCAI 2005*, pages 609–614, Edinburgh, UK, 2005.
15. D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. IJCAR 2006*, pages 292–297, Seattle, WA, USA, 2006.