HermiT: A Highly-Efficient Reasoner for Description Logics

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 reasoner for the Description Logic $\mathcal{SHOIQ}+$. The system is based on a novel "hypertableau" calculus and includes a number of new optimizations. HermiT demonstrates significantly faster performance when classifying large and complex ontologies than other state-of-the-art reasoners. HermiT is also the first reasoner able to classify a number of important knowledge bases, including the original version of the GALEN ontology.

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 consistency of a knowledge base in which an individual is a member 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 arise from two primary 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 build 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 possible 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 addressed both of the major sources of complexity. HermiT implements a *hypertableau* calculus which greatly reduces the number of models which must be considered (down to only a single model for a significant subset of ontologies). HermiT also incorporates the *anywhere blocking* strategy, which limits the sizes of the models which are constructed. Finally, HermiT incorporates a novel and highly-efficient approach to handling nominals in the presence of number restrictions and inverse roles; we expect that the ability to reason with nominals will allow ontology authors to make much freer use of nominals than has been possible to date. This combination of fundamental algorithmic improvements 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 than other reasoners when classifying more difficult ontologies. Furthermore, HermiT is able to classify a number of ontologies which no other reasoner has previously been able to handle.

The HermiT system also serves as a platform for implementation of reasoning for new ontologies features. HermiT already includes support for reasoning with ontologies which include *description graphs*.

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 ???

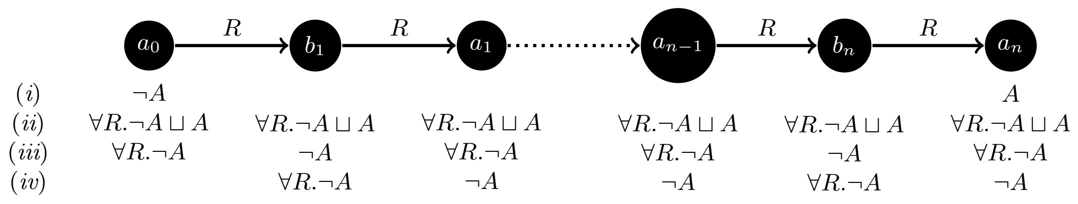
2.1 Reducing Tableau Complexity

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*.[1] 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. Thus, $\mathcal{K}$ is unsatisfiable only if all choices fail to construct a model. We next discuss several sources of complexity in this procedure, and how HermiT addresses them.

[1] Some formalizations of tableau algorithms work on *completion graphs* [5], which have a natural correspondence to ABoxes.



a_0
b_1
a_1
a_{n-1}
b_n
a_n
R
(i) $\neg A$ A
(ii) $\forall R.\neg A \sqcup A$
(iii) $\forall R.\neg A$ $\neg A$
(iv) $\forall R.\neg A$ $\neg A$

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 is thus 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; in fact, finding an order that exhibits good performance in practice requires advanced heuristics [15]. 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 $i \neq j$.

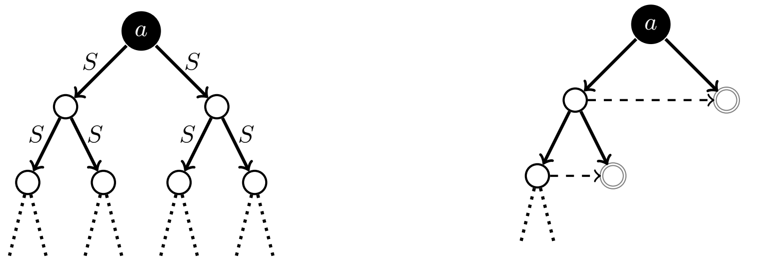
The axiom $\exists R.A \sqsubseteq A$, however, is not inherently nondeterministic: it is equivalent to the Horn clause $\forall x, y : [R(x, y) \wedge A(y) \rightarrow A(x)]$, which can be applied in a bottom-up manner to derive the assertions $A(b_n), A(a_{n-1}), \ldots, A(a_0)$ and reveal a contradiction 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* [14] 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 $\exists R.\top \sqsubseteq A \sqcap \forall R.\neg A$, 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 *DL-clauses* which allow 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 reducing nondeterminism: testing satisfiability of the concept $A \sqcup \neg B$ causes nondeterministic application of the $\sqcup$-rule in standard tableau reasoners; HermiT transforms this concept into an expression equivalent to $A \sqsubseteq B$, and thus is able to apply absorption-style optimizations much more pervasively than standard tableau reasoners.

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

$$\begin{aligned} \mathcal{T}_2 = \{\ & A_1 \sqsubseteq\ \geq 2\, S.A_2,\ \ldots,\ A_{n-1} \sqsubseteq\ \geq 2\, S.A_n,\ A_n \sqsubseteq A_1, \\ & A_i \sqsubseteq (B_1 \sqcup C_1) \sqcap \ldots \sqcap (B_m \sqcup C_m) \text{ for } 1 \leq i \leq n\ \} \\ \mathcal{A}_2 = \{\ & A_1(a)\ \} \end{aligned} \tag{2}$$



(a) Ancestor Blocking (b) Anywhere Blocking

Fig. 2: And-Branching Example

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 $A_1(a)$ implies the existence of at least two individuals in A_2, which imply the existence of at least two individuals in A_3, and so on. Given $\mathcal{K}_2$, a tableau algorithm thus constructs a binary tree, shown in Figure 2a, in which with each individual is labeled with some A_i and an element of $\mathit{\Pi} = \{B_1, C_1\} \times \ldots \times \{B_m, C_m\}$. Each individual at depth n is an instance of A_n; because of the GCI $A_n \sqsubseteq A_1$, this individual 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 is such cases, tableau algorithms employ *blocking* [6], which is based on an important observation about the shape of ABoxes that can be derived from some input ABox $\mathcal{A}$. The individuals in $\mathcal{A}$ are called *named* (shown as black circles), and they can be connected by role assertions in an arbitrary way. The individuals introduced by the $\exists$- and $\geq$-rules are called *blockable* (shown as white circles). For example, if $\exists R.C(a)$ is expanded into $R(a, s)$ and $C(s)$, then s is called a blockable individual and it is an *R-successor* of a. It is not difficult to see that, if the knowledge base does not contains nominals, no tableau inference rule can connect s with some element of $\mathcal{A}$: the individual s can participate only in inferences that derive an assertion of the form $D(s)$, create a new successor of s, or, in the presence of (local) reflexivity, connect s to itself. Hence, each ABox $\mathcal{A}'$ obtained from $\mathcal{A}$ can be seen as a "forest" of the form shown in Figure 3: each named individual can be arbitrarily connected to other named individuals and to a tree of blockable successors.

The forest-like structure of ABoxes enables *ancestor blocking*: an individual s is *directly blocked* if s has an ancestor t identical to s;[2] all successors of s are

[2] The definition of "identical" depends on the logic used.

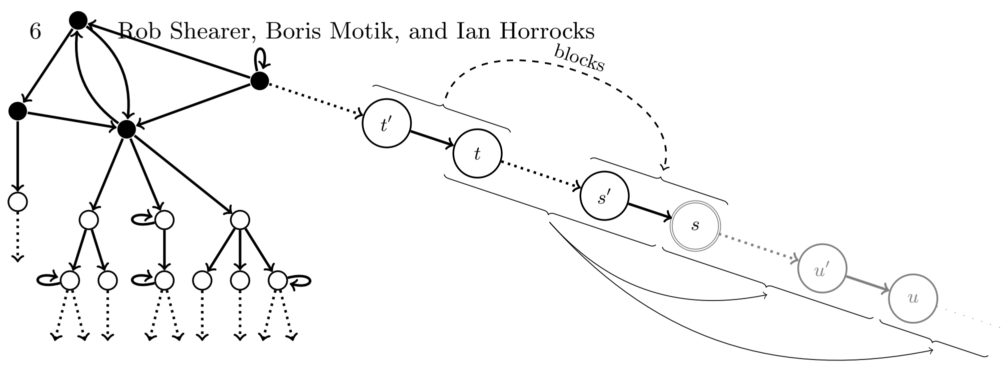


Fig. 3: Forest-Like Shape of ABoxes

then *indirectly blocked*. In standard tableau algorithms, the $\exists$- and $\geq$-rules are applicable only to nonblocked individuals, which ensures termination. 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 s with a copy of the tree rooted at t. The model will be infinite (the copy of the t subtree will include another copy of s, which will be replaced with another copy of t, and so on), but we need never actually construct it—an ABox with blocked individuals is sufficient to prove that such a model exists.

Consider now an "unlucky" run of a tableau algorithm with ancestor blocking on $\mathcal{K}_2$. The number of elements in Π is exponential, so it can happen that blocking comes into effect only after the algorithm constructs an exponentially deep tree; since the tree is binary, it is doubly exponential in total. In a "lucky" run, the algorithm can always pick B_j instead of C_j; then, the algorithm constructs a polynomially deep binary tree, so the tree is exponential in total. Thus, the and-branching caused by the $\exists$- and $\geq$-rules can lead to unnecessary generation of an ABox that is doubly exponential in the size of the input, which limits the scalability of tableau algorithms in practice.

We employ pairwise blocking from Section 2.1 to ensure termination of the calculus; to curb and-branching, however, we extend it to *anywhere pairwise blocking*. The key idea is to extend the set of potential blockers for s beyond the ancestors of s. In doing so, we must avoid cyclic blocks: if s is allowed to block t and t can block s, then neither s nor t is guaranteed to have all its successors constructed, which would render the calculus incomplete. Therefore, we parameterize our algorithm with a strict ordering $\prec$ on individuals that contains the ancestor relation. We allow t to block s only if, in addition to conditions mentioned in Section 2.1, we have $t \prec s$. If $\prec$ coincides with the ancestor relation, anywhere blocking becomes equivalent to the ancestor blocking.

Anywhere blocking can reduce and-branching in practice. Consider again the knowledge base $\mathcal{K}_2$ from Section 2.1. After we exhaust the exponentially many members of Π, all subsequently created individuals will be blocked. In the best case, we can always choose B_j instead of C_j, so we create a polynomial path in the tree and then use the individuals from that path to block their siblings, as shown in Figure 2b. Hence, with anywhere blocking, satisfiability of $\mathcal{K}_2$ can be checked in polynomial time.

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, so it cannot be considered 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 unique individual, the tableau NN-rule guesses exactly how many such neighbors will exist in the final model and constructs these individuals. Number restrictions on these individuals can cause another application of the NN-rule to produce unique 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 addressed 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 uncopyable. By keeping track of unique identifiers for each uncopyable 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, we have developed two simple optimizations that, to the best of our knowledge, have not been considered previously in the literature.

Reading Classification Relationships from Concept Labels Let $(\mathcal{A}, \mathcal{C})$ be an ABox and a set of HT-clauses obtained by clausifying a knowledge base $\mathcal{K}$, and let A and B be atomic concepts for which we want to check whether $\mathcal{K} \models A \sqsubseteq B$; since A and B are atomic, this is the case if and only if $(\mathcal{A}', \mathcal{C})$ is unsatisfiable where $\mathcal{A}' = \mathcal{A} \cup \{A(a), \neg B(a)\}$ and a is a fresh individual. Let $\mathcal{A}_1$ be a clash-free ABox labeling a leaf in a derivation from $(\mathcal{A}', \mathcal{C})$. We can use $\mathcal{A}_1$ to learn the following things about subsumption in $\mathcal{K}$. The proof of these claims is trivial, and we omit if for the sake of brevity.

1. If $C(a) \in \mathcal{A}_1$ and the derivation of $C(a)$ does not depend on a nondeterministic choice, then $\mathcal{K} \models A \sqsubseteq C$.
2. If $\mathcal{A}_1$ has been obtained from $\mathcal{A}'$ deterministically, then $\mathcal{K} \models A \sqsubseteq C$ only if $C(a) \in \mathcal{A}_1$.
3. If $C(b) \in \mathcal{A}_1$ but $D(b) \notin \mathcal{A}_1$ for C and D concepts and b an individual that is not blocked, then $\mathcal{K} \not\models C \sqsubseteq D$.

From the first two properties, if $\mathcal{K}$ is a deterministic knowledge base such as GALEN, we can classify it using a *linear* number of calls to the hypertableau algorithm: for each atomic concept A, we check satisfiability of $(\mathcal{A} \cup \{A(a)\}, \mathcal{C})$; if the algorithm produces a clash-free ABox $\mathcal{A}_1$, the superclasses of A are contained in $\mathcal{L}_{\mathcal{A}_1}(a)$. These optimizations are applicable in the case of tableau algorithms as well; however, due to increased or-branching, they are likely to be less effective.

Caching Blocking Labels Let $\mathcal{T}$ and $\mathcal{R}$ be a $\mathcal{SHIQ}$ TBox and RBox, respectively, and let $\mathcal{C} = \Xi_{\mathcal{TR}}(\mathcal{T} \cup \mathcal{R})$. Furthermore, let us assume that the classification of $\mathcal{T} \cup \mathcal{R}$ involves n consecutive calls to the hypertableau algorithm for $(\{A_i(a_i), \neg B_i(a_i)\}, \mathcal{C})$. Then, if a derivation from $(\{A_i(a_i), \neg B_i(a_i)\}, \mathcal{C})$ contains a leaf node labeled with a clash-free ABox $\mathcal{A}_i$, we can use the nonblocked individuals from $\mathcal{A}_i$ as potential blockers in all subsequent satisfiability satisfiability checks of $(\{A_j(a_j), \neg B_j(a_j)\}, \mathcal{C})$ for $j > i$.

This is a simple consequence of the following fact. Let I_1 and I_2 be two models of $\mathcal{T} \cup \mathcal{R}$ such that $\triangle^{I_1} \cap \triangle^{I_2} = \emptyset$; furthermore, let I be defined as $\triangle^I = \triangle^{I_1} \cup \triangle^{I_2}$, $A^I = A^{I_1} \cup A^{I_2}$, and $R^I = R^{I_1} \cup R^{I_2}$, for each atomic concept A and each atomic role R. Then, by a simple induction on the structure of axioms in $\mathcal{T} \cup \mathcal{R}$, it is trivial to show that $I \models \mathcal{T} \cup \mathcal{R}$. This property does not hold in the presence of nominals, which can impose a bound on the number of elements in the interpretation of a concept; the bound could be satisfied in I_1 and I_2 individually, but violated in I.

Our optimization is correct because, instead of $(\{A_i(a_i), \neg B_i(a_i)\}, \mathcal{C})$, we can check the satisfiability of $(\mathcal{A}_i \cup \{A_i(a_i), \neg B_i(a_i)\}, \mathcal{C})$, and in doing so we can use the individuals from $\mathcal{A}_i$ as potential blockers due to anywhere blocking. This optimization can be seen as a very simple form of model caching [1, Chapter 9], and it has been key to obtaining the results that we present in Section **??**. 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.

In practice, we do not need to keep the entire ABox $\mathcal{A}_i$ around; rather, for each nonblocked blockable individual t with a predecessor t', we simply need to retain the sets $\mathcal{L}_{\mathcal{A}_i}(t)$, $\mathcal{L}_{\mathcal{A}_i}(t')$, $\mathcal{L}_{\mathcal{A}}(t,t)$, $\mathcal{L}_{\mathcal{A}_i}(t,t')$, and $\mathcal{L}_{\mathcal{A}_i}(t',t)$.

Individual Re-use [Find some content for this.]

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 [9] and [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 and they cannot be faithfully represented in OWL. For example, FMA models the human hand as consisting of the fingers, the palm, various bones, blood vessels and so on, all of which are highly interconnected.

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 [11], and FaCT++ 1.1.10 [16]. 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]—a collection of OWL ontologies gathered from the Web; the Open Biological Ontologies (OBO) Foundry[3]—a collection of biology and life science ontologies; and several variants of the GALEN ontology [12]—a large and complex biomedical ontology. 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. After eliminating syntactically incorrect OWL ontologies, we obtained a test suite consisting of 139 ontologies.

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—126 of them, to be precise—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

[3] `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

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 expect our development version of HermiT to exhibit substantially improved performance on ontologies with many role axioms.

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. The *completion graph caching* optimization [13] can be used to avoid repeating the same reasoning steps over the ABox for each independent subsumption test; however, HermiT does not yet implement this optimization, which may explain its worse performance. HermiT completes each subsumption test quite quickly (on the order of 30ms), suggesting that HermiT should achieve performance levels comparable to other reasoners once completion graph caching is implemented.

Different versions of GALEN have commonly been used for testing 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, 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. To alleviate this problem, we are currently developing a reasoning technique in which the $\exists$-rule is modified such that it nondeterministically tries to reuse individuals from the ABox generated thus far. Our initial experiments have shown very promising results [8].

Because of the failure of DL reasoners to process GALEN-full, various simplified versions of GALEN have often been used in practice. GALEN-horrocks and Not-GALEN are two versions found in the Gardiner suite, GALEN-original is a version of GALEN from roughly 10 years ago, and GALEN-doctored has been obtained from GALEN-original by removing about a 100 "difficult" axioms. 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), Pellet and FaCT++ 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: Theory, Implementation and Applications.* Cambridge University Press, 2nd edition, August 2007.
2. F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, and E. Franconi. An Empirical Analysis of Optimization Techniques for Terminological Representation systems or: "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. of the 2006 Description Logic Workshop (DL 2006)*, volume 189 of *CEUR Workshop Proceedings*, 2006.
4. B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular Reuse of Ontologies: Theory and Practice. *Journal of Artificial Intelligence Research*, 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 D. MacAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE-17)*, volume 1831 of *LNAI*, pages 482–496, Pittsburgh, USA, June 17–20 2000. Springer.
7. A. K. Hudek and G. Weddell. Binary Absorption in Tableaux-Based Reasoning for Description Logics. In B. Parsia, U. Sattler, and D. Toman, editors, *Proc. of the 2006 Int. Workshop on Description Logics (DL 2006)*, volume 189 of *CEUR Workshop Proceedings*, Windermere, UK, May 30-June 1 2006.
8. B. Motik and I. Horrocks. Individual Reuse in Description Logic Reasoning. In *Proc. of the 4th Int. Joint Conf. on Automated Reasoning (IJCAR 2008)*, Sydney, Australia, August 10–15 2008. Springer. To appear.
9. Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, and Ulirke Sattler. Representing Structured Objects using Description Graphs. In *Proc. of the 11th Int. Joint Conf. on Principles of Knowledge Representation and Reasoning (KR 2008)*, Sydney, Australia, August 16–19 2008. AAAI Press. To appear.
10. Boris Motik, Bernardo Cuenca Grau, and Ulrike Sattler. Structured Objects in OWL: Representation and Reasoning. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proc. of the 17th Int. World Wide Web Conference (WWW 2008)*, pages 555–564, Beijing, China, April 21–25 2008. ACM Press.
11. B. Parsia and E. Sirin. Pellet: An OWL-DL Reasoner. Poster, In Proc. of the 3rd Int. Semantic Web Conference (ISWC 2004), Hiroshima, Japan, November 7–11, 2004.
12. A. L. Rector and J. Rogers. Ontological and Practical Issues in Using a Description Logic to Represent Medical Concept Systems: Experience from GALEN. In P. Barahona, F. Bry, E. Franconi, N. Henze, and U. Sattler, editors, *Tutorial Lectures of the 2nd Int. Summer School 2006*, volume 4126 of *LNCS*, pages 197–231, Lisbon, Portugal, September 4–8 2006. Springer.

13. E. Sirin, B. Cuenca Grau, and B. Parsia. From Wine to Water: Optimizing Description Logic Reasoning for Nominals. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 90–99, Lake District, UK, June 2–5 2006. AAAI Press.
14. D. Tsarkov and I. Horrocks. Efficient Reasoning with Range and Domain Constraints. In V. Haarslev and R. Möller, editors, *Proc. of the 2004 Int. Workshop on Description Logics (DL 2004)*, volume 104 of *CEUR Workshop Proceedings*, Whistler, BC, Canada, June 6–8 2004.
15. D. Tsarkov and I. Horrocks. Ordering Heuristics for Description Logic Reasoning. In L. Pack Kaelbling and A. Saffiotti, editors, *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 609–614, Edinburgh, UK, July 30–August 5 2005. Morgan Kaufmann Publishers.
16. D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *LNAI*, pages 292–297, Seattle, WA, USA, August 17–20 2006. Springer.