A Hypertableau Calculus for $\mathcal{SHIQ}$

Boris Motik, Rob Shearer, and Ian Horrocks

University of Manchester, UK

Abstract. We present a novel reasoning calculus for the Description Logic $\mathcal{SHIQ}$. In order to reduce the nondeterminism due to general inclusion axioms, we base our calculus on hypertableau and hyperresolution calculi, which we extend with a blocking condition to ensure termination. To prevent the calculus from generating large models, we introduce *"anywhere" pairwise blocking*. Our preliminary implementation shows significant performance improvements on several well-known ontologies. To the best of our knowledge, our reasoner is currently the only one that can classify the original version of the GALEN terminology.

1 Introduction

Modern Description Logic reasoners, such as Pellet [10], FaCT++ [15], and RACER [5], are typically based on tableau calculi [1, Chapter 2], which demonstrate (un)satisfiability of a knowledge base $\mathcal{K}$ via a constructive search for an abstraction of a model of $\mathcal{K}$. Despite numerous optimizations of the tableau procedure, ontologies are still encountered in practice that cannot be handled by existing systems. This is mainly because many different models might need to be examined, and each model might be very large [1, Chapter 3]. The former problem is due to *or-branching*: given a disjunctive assertion $C \sqcup D(s)$, a tableau algorithm nondeterministically guesses that either $C(s)$ or $D(s)$ holds. To show unsatisfiability of $\mathcal{K}$, *every* possible guess must lead to a contradiction: if assuming $C(s)$ leads to a contradiction, the algorithm must backtrack and assume $D(s)$. This can clearly result in exponential behavior. GCIs—axioms of the form $C \sqsubseteq D$—are the main source of disjunctions: to ensure that $C \sqsubseteq D$ holds, a tableau algorithm adds a disjunction $\neg C \sqcup D(s)$ to each individual s in the model. Various *absorption* optimizations [1, Chapter 9][7, 14] reduce the high degree of nondeterminism in such a procedure; however, they often fail to eliminate all sources of nondeterminism. This may be the case even for ontologies that can be translated into Horn clauses (such as GALEN, NCI, and SNOMED), for which reasoning without any nondeterminism should be possible in principle.

The size of the model being constructed is determined by *and-branching*—the expansion of a model due to existential quantifiers. Apart from memory consumption problems, and-branching can increase or-branching by increasing the number of individuals to which GCIs are applied.

In this paper, we present a reasoning calculus that addresses both sources of complexity. We focus on the DL $\mathcal{SHIQ}$; however, our calculus should be applicable to most DLs with known tableau algorithms. A $\mathcal{SHIQ}$ knowledge

base is first preprocessed into *DL-clauses*—universally quantified implications containing DL concepts and roles as predicates. The main inference rule for DL-clauses is hyperresolution: an atom from the head of a DL clause is derived *only if* all atoms from the clause body have been derived. On Horn clauses, this calculus is deterministic, which eliminates all or-branching. Our algorithm can be viewed as a hybrid of resolution and tableau, and is related to the hypertableau [2] and hyperresolution [12] calculi.

Hyperresolution decides many first-order fragments (see, e.g., [4, 3] for an overview). Unlike most of these fragments, $\mathcal{SHIQ}$ allows for cyclic GCIs of the form $C \sqsubseteq \exists R.C$, on which hyperresolution can generate infinite paths of successors. Therefore, to ensure termination, we use the pairwise blocking technique from [6] to detect cyclic computations. Due to hyper-inferences, the soundness and correctness proofs from [6] do not carry over to our calculus. In fact, certain simpler blocking conditions for weaker DLs cannot be applied in a straightforward manner in our setting. To limit and-branching, we extend the blocking condition from [6] to *anywhere pairwise blocking*: an individual can be blocked by an individual that is not necessarily an ancestor. This significantly reduces the sizes of the constructed models.

We have implemented our calculus in a new reasoner. Even with a relatively naïve implementation, our system outperforms existing reasoners on several real-world ontologies. For example, the deterministic treatment of GCIs significantly reduces the classification time for the NCI ontology. Furthermore, the pairwise anywhere blocking strategy seems to be very effective in limiting model sizes. To the best of our knowledge, our reasoner is currently the only one that can classify the original version of the GALEN terminology.

2 Algorithm Overview

To see how GCIs can increase or-branching and thus cause performance problems, consider the following knowledge base $\mathcal{K}_1$:

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

To satisfy the GCI, a tableau algorithm derives $(\forall R.\neg A \sqcup A)(a_i)$, $0 \leq i \leq n$ and $(\forall R.\neg A \sqcup A)(b_j)$, $1 \leq j \leq n$. Assuming that a_i are processed before b_j, the algorithm derives $\forall R.\neg A(a_i)$, $0 \leq i \leq n$ and $\neg A(b_i)$, $1 \leq i \leq n$, after which it derives $\forall R.\neg A(b_i)$, $1 \leq i \leq n-1$ and $\neg A(a_i)$, $1 \leq i \leq n$. The ABox now contains a contradiction on a_n, so the algorithm 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. Next, the guess on a_n is changed to $A(a_n)$ and the work for all b_i is repeated. 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. Note that none of the standard backtracking optimizations [1, Chapter 9] help us avoid this problem. Namely, 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, $i \neq j$. It is difficult to estimate in advance which order is optimal; in fact, the processing order is typically determined by implementation side-effects (such as the data structures used to store $\mathcal{K}$).

The GCI $\exists R.A \sqsubseteq A$ is not inherently nondeterministic: it is equivalent to the Horn clause $\forall x, y : [R(x, y) \wedge A(y) \rightarrow A(x)]$. By hyperresolution, we derive the facts $A(b_n), A(a_{n-1}), \ldots, A(a_0)$, and eventually we drive 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 way tableau algorithms handle GCIs can be "unnecessarily" nondeterministic.

Absorption [1, Chapter 9] reduces the nondeterminism introduced by GCIs. If possible, it rewrites GCIs as $B \sqsubseteq C$ with B an atomic concept; then, during reasoning, it derives $C(s)$ only if the ABox contains $B(s)$. This localizes the applicability of the rewritten GCIs. Absorption has been extended to *binary absorption* [7], which rewrites a GCI to $B_1 \sqcap B_2 \sqsubseteq C$, and to *role absorption* [14], which rewrites a GCI to $\exists R.\top \sqsubseteq C$. Note, however, that the axiom $\exists R.A \sqsubseteq A$ cannot be absorbed directly. It can be absorbed if it is rewritten as $A \sqsubseteq \forall R^-.A$. In practice, it is often unclear in advance which combination of transformation and absorption techniques will yield the best results. Therefore, implemented absorption algorithms are guided primarily by heuristics.

Our algorithm can be seen as a generalization of absorption. It first translates GCIs into *DL-clauses*—universally quantified implications of the form $\bigwedge U_i \rightarrow \bigvee V_j$, where U_i are of the form $R(x, y)$ or $A(x)$, and V_j are of the form $R(x, y)$, $A(x)$, $\exists R.C(x)$, $\geqslant n\, R.C(x)$, or $x \approx y$. DL-clauses are used in *hyperresolution* inferences, which derive some V_j, but only if all U_i are matched to assertions in the ABox. This calculus is quite different from the standard DL tableau calculi. For example, it has no *choose*-rule for qualified number restrictions [13], and it can handle implications such as $R(x, y) \rightarrow B(x) \vee A(y)$ (obtained from $\exists R.\neg A \sqsubseteq B$) that contain several universally quantified variables.

It is easy to see that and-branching can cause the introduction of infinitely many new individuals. Consider the following (satisfiable) knowledge base $\mathcal{K}_2$:

$$(2)\quad \mathcal{T}_2 = \left\{ \begin{array}{l} A_1 \sqsubseteq \geqslant 2\, S.A_2,\ \ldots,\ A_{n-1} \sqsubseteq \geqslant 2\, S.A_n,\ A_n \sqsubseteq A_1, \\ A_i \sqsubseteq (B_1 \sqcup C_1) \sqcap \ldots \sqcap (B_m \sqcup C_m) \text{ for } 1 \leq i \leq n \end{array} \right\}\ \mathcal{A}_2 = \{A_1(a)\}$$

To check satisfiability of $\mathcal{K}_2$, a tableau algorithm builds a binary tree with each node labeled with some A_i and an element of $\Pi = \{B_1, C_1\} \times \ldots \times \{B_m, C_m\}$. A naïve algorithm would try to construct an infinite tree, so tableau algorithms employ *blocking* [6]: if a node a is labeled with the same concepts as some ancestor a' of a, then the existential quantifiers for a are not expanded. This ensures termination; however, the number of elements in Π is exponential, so, with "unlucky" guesses, the tree can be exponential in depth and doubly exponential in total. In the best case, the algorithm can, for example, choose B_j rather than C_j for each $1 \leq j \leq m$. It then constructs a polynomially deep binary tree and thus runs in exponential time.

To curb and-branching, we extend pairwise blocking [6] to *anywhere pairwise blocking*, in which an individual can be blocked not only by an ancestor, but by

any individual satisfying certain ordering requirements. This reduces the worst-case complexity of the algorithm by an exponential factor; for example, on $\mathcal{K}_2$, after we exhaust all members of Π, all subsequently created individuals will be blocked. Such blocking can sometimes also improve the best-case complexity; for example, on $\mathcal{K}_2$ our algorithm can create a polynomial path and then use the individuals from that path to block their siblings.

3 The Satisfiability Checking Algorithm

Our algorithm consists of two phases: preprocessing and inferencing.

3.1 Preprocessing

The goal of the preprocessing phase is to transform a $\mathcal{SHIQ}$ knowledge base into a *normalized* ABox (in which all concept assertions are of the form $B(s)$ or $\geqslant n\, R.B(s)$ and all role assertions involve only atomic roles), and a collection of *DL-clauses*, which we denote as $\Xi(\mathcal{K})$ in the rest of this paper. We omit the details of the transformation due to lack of space; the complete algorithm is described in [9] and illustrated here by example.

Our calculus does not deal with transitive roles, so we transform the $\mathcal{SHIQ}$ knowledge base into an equisatisfiable $\mathcal{ALCHIQ}$ knowledge base using the well-known encoding described in [8, Section 5.2]. The next problem is that concepts in $\mathcal{ALCHIQ}$ axioms can occur under implicit negation. We make negation explicit by moving all concepts to the right-hand side of the implication and putting all concepts into negation-normal form. For example, the axiom

$$\exists R.(C \sqcap D) \sqsubseteq \exists S.(E \sqcup F) \tag{3}$$

is rewritten as follows:

$$\top \sqsubseteq \forall R.(\neg C \sqcup \neg D) \sqcup \exists S.(E \sqcup F) \tag{4}$$

It is well known that naïve clausification of $\mathcal{ALCHIQ}$ axioms would result in exponential blowup. We instead apply a variant of the well-known structural transformation [11], which replaces complex concepts with new names and introduces new axioms to define these names. Our goal, however, is to obtain Horn DL-clauses whenever possible. As discussed in [9], if we are not careful the structural transformation can destroy the Horn-ness of an axiom. Therefore, we modify the transformation to replace a complex concept C with either A or $\neg A$, where A is a fresh atomic name. The polarity of the replacement concept is chosen such that the axiom that defines the replacement will not introduce additional nondeterminism into the final clause set; this condition is detected by analyzing the structure of literals within the replaced concept. Applying our structural transformation to (4) gives us the following axioms:

$$\top \sqsubseteq \forall R.\neg Q_1 \sqcup \exists S.Q_2 \tag{5}$$

$$\neg Q_1 \sqsubseteq \neg C \sqcup \neg D \tag{6}$$

$$Q_2 \sqsubseteq E \sqcup F \tag{7}$$

Without complex subexpressions or implicit negation, the transformation to DL-clauses is straightforward. Universal restrictions are rewritten using their first-order-logic interpretations; e.g. $\forall R.C$ is rewritten as $\neg R(x,y) \vee C(y)$. Negated atoms are then moved to the antecedent of the DL-clause, and positive atoms are moved to the consequent. The DL-clauses derived from (3) are as follows:

$$R(x,y) \wedge Q_1(y) \rightarrow \exists S.Q_2(x) \tag{8}$$

$$C(x) \wedge D(x) \rightarrow Q_1(x) \tag{9}$$

$$Q_2(x) \rightarrow E(x) \vee F(x) \tag{10}$$

3.2 The Hypertableau Calculus for DL-Clauses

We now present our hypertableau calculus for deciding satisfiability of $\mathcal{A} \cup \Xi(\mathcal{K})$.

Definition 1. ***Unnamed Individuals.*** *For a set of* named *individuals N_I, the set of* all individuals *N_X is inductively defined as $N_I \subseteq N_X$ and, if $x \in N_X$, then $x.i \in N_X$ for each integer i. The individuals in $N_X \setminus N_I$ are* unnamed. *An individual $x.i$ is a* successor *of x, and x is a* predecessor *of $x.i$;* descendant *and* ancestor *are the transitive closures of successor and predecessor, respectively.*

Pairwise Anywhere Blocking. *A concept is* blocking-relevant *if it is of the form A, $\geqslant n\, R.A$, or $\geqslant n\, R.\neg A$, for A an atomic concept. The* label of an individual *s and* of an individual pair *$\langle s,t\rangle$ in an ABox $\mathcal{A}$ are defined as follows:*

$$\mathcal{L}_{\mathcal{A}}(s) = \{C \mid C(s) \in \mathcal{A} \text{ and } C \text{ is a blocking-relevant concept}\}$$
$$\mathcal{L}_{\mathcal{A}}(s,t) = \{R \mid R(s,t) \in \mathcal{A}\}$$

Let $\prec$ be a strict ordering (i.e., a transitive and irreflexive relation) on N_X containing the ancestor relation—that is, if s' is an ancestor of s, then $s' \prec s$. By induction on $\prec$, we assign to each individual s in $\mathcal{A}$ a status as follows:

- *s is* directly blocked by *an individual s' iff both s and s' are unnamed, s' is not blocked, $s' \prec s$, $\mathcal{L}_{\mathcal{A}}(s) = \mathcal{L}_{\mathcal{A}}(s')$, $\mathcal{L}_{\mathcal{A}}(t) = \mathcal{L}_{\mathcal{A}}(t')$, $\mathcal{L}_{\mathcal{A}}(s,t) = \mathcal{L}_{\mathcal{A}}(s',t')$, and $\mathcal{L}_{\mathcal{A}}(t,s) = \mathcal{L}_{\mathcal{A}}(t',s')$, for t and t' the predecessors of s and s', resp.*
- *s is* indirectly blocked *iff its predecessor is blocked.*
- *s is* blocked *iff it is either directly or indirectly blocked.*

Pruning. *The ABox $\mathsf{prune}_{\mathcal{A}}(s)$ is obtained from $\mathcal{A}$ by removing all assertions of the form $R(t,t.i)$, $R(t.i,t)$, $C(t.i)$, $u \approx t.i$, and $u \not\approx t.i$, where t is either s or some descendant of s, i is an integer, and u is an arbitrary individual.*

Merging. *The ABox $\mathsf{merge}_{\mathcal{A}}(s \rightarrow t)$ is obtained from $\mathsf{prune}_{\mathcal{A}}(s)$ by replacing the individual s with the individual t in all assertions.*

Derivation Rules. *Table 1 specifies* derivation rules *that, given an ABox $\mathcal{A}$ and a set of DL-clauses $\Xi(\mathcal{K})$, derive the ABoxes $\mathcal{A}_1, \ldots, \mathcal{A}_n$. In the Hyp-rule, σ is a mapping from N_V to the individuals occurring in $\mathcal{A}$, and $\sigma(U)$ is the atom obtained from U by replacing each variable x with $\sigma(x)$.*

Derivation. *For a normalized $\mathcal{ALCHIQ}$ knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, a* derivation *is a pair (T, λ) where T is a finitely branching tree and λ is a function*

Table 1. Derivation Rules of the Tableau Calculus

Hyp-rule	If 1. $U_1 \wedge ... \wedge U_m \rightarrow V_1 \vee ... \vee V_n \in \Xi(\mathcal{K})$, 2. a mapping $\sigma : N_V \rightarrow N_\mathcal{A}$ exists, for $N_\mathcal{A}$ the set of individuals in $\mathcal{A}$, 3. $\sigma(U_i) \in \mathcal{A}$ for each $1 \leq i \leq m$, 4. $\sigma(V_j) \notin \mathcal{A}$ for each $1 \leq j \leq n$, then if $n = 0$, then $\mathcal{A}_1 = \mathcal{A} \cup \{\bot\}$, otherwise $\mathcal{A}_j := \mathcal{A} \cup \{\sigma(V_j)\}$ for $1 \leq j \leq n$.
$\geq$-rule	If 1. $\geqslant n\, R.C(s) \in \mathcal{A}$, 2. s is not blocked in $\mathcal{A}$, and 3. there are no individuals $u_1, \ldots, u_n$ such that $\{\mathsf{ar}(R, s, u_i), C(u_i) \mid 1 \leq i \leq n\} \cup \{u_i \not\approx u_j \mid 1 \leq i < j \leq n\} \subseteq \mathcal{A}$, then $\mathcal{A}_1 := \mathcal{A} \cup \{\mathsf{ar}(R, s, t_i),\ C(t_i) \mid 1 \leq i \leq n\} \cup \{t_i \not\approx t_j \mid 1 \leq i < j \leq n\}$ where $t_1, \ldots, t_n$ are fresh pairwise distinct successors of s.
$\approx$-rule	If 1. $s \approx t \in \mathcal{A}$ and 2. $s \neq t$ then $\mathcal{A}_1 := \mathsf{merge}_\mathcal{A}(s \rightarrow t)$ if t is named or if s is a descendant of t, $\mathcal{A}_1 := \mathsf{merge}_\mathcal{A}(t \rightarrow s)$ otherwise.
$\bot$-rule	If 1. $s \not\approx s \in \mathcal{A}$ or $\{A(s), \neg A(s)\} \subseteq \mathcal{A}$ and 2. $\bot \notin \mathcal{A}$ then $\mathcal{A}_1 := \mathcal{A} \cup \{\bot\}$.

$$\mathsf{ar}(R, s, t) = \begin{cases} R(s, t) \text{ if } R \text{ is an atomic role} \\ S(t, s) \text{ if } R \text{ is an inverse role and } R = S^- \end{cases}$$

*that labels the nodes of T with ABoxes such that (*i*) $\lambda(\epsilon) = \mathcal{A}$ for ϵ the root of the tree, and (*ii*) for each node t, if one or more derivation rules are applicable to $\lambda(t)$ and $\Xi(\mathcal{K})$, then t has children $t_1, \ldots, t_n$ such that $\lambda(t_1), \ldots, \lambda(t_n)$ are the result of applying one (arbitrarily chosen) applicable rule to $\lambda(t)$ and $\Xi(\mathcal{K})$.*

Clash. *An ABox $\mathcal{A}$ contains a* clash *iff $\bot \in \mathcal{A}$; otherwise, $\mathcal{A}$ is* clash-free*.*

In [6], the successor relation is encoded using role arcs, which point only from predecessors to successors. Since our ABoxes contain only atomic roles, role arcs can point in both directions, so we encode the successor relation in the individuals. The ordering $\prec$ ensures that there are no cyclic blocks, so all successors of nonblocked individuals have been constructed. *Ancestor pairwise blocking* from [6] is obtained if $\prec$ is exactly the descendant relation.

Pruning prevents infinite loops of merge-create rule applications—the so-called "yo-yo" effect. Intuitively, merging ensures that no individual "inherits" successors through merging. In [6], the successors are not physically removed, but are marked as "not present" by setting their edge labels to $\emptyset$. This has exactly the same effect as pruning.

The relationship between our new calculus and knowledge base satisfiability is given by the following theorem:

Theorem 1. *A $\mathcal{SHIQ}$ knowledge base $\mathcal{K}$ is satisfiable if and only if each derivation from $\mathcal{K}' = \Xi(\mathcal{K})$ contains a leaf node t such that $\lambda(t)$ is clash-free; furthermore, the construction of each such derivation terminates.*

Table 2. Results of Performance Evaluation

Ontology	HT	HT-anc	Pellet	FaCT++	Racer
NCI	8 s	9 s	44 min	32 s	36 s
GALEN original	44 s	—	—	—	—
GALEN simplified	7 s	104 s	—	859 s	—

4 Implementation

Based on the calculus from Section 3, we have implemented a prototype DL reasoner.[1] Currently, it can only handle Horn DL-clauses—our main goal was to show that significant performance improvements can be gained by exploiting the deterministic nature of many ontologies.

To classify a knowledge base $\mathcal{K}$, we run our algorithm on $\mathcal{K}_i = \mathcal{K} \cup \{C_i(a_i)\}$ for each concept C_i, obtaining an ABox $\mathcal{A}_i$. If $D(a_i) \in \mathcal{A}_i$ and $D(a_i)$ was derived without making any nondeterministic choices, then $\mathcal{K} \models C_i \sqsubseteq D$. Since our test ontologies are translated to Horn DL-clauses on which our algorithm is deterministic, $D(a_i) \in \mathcal{A}_i$ iff $\mathcal{K} \models C_i \sqsubseteq D$. Thus, we can classify $\mathcal{K}$ with a linear number of calls to our algorithm. This optimization is also applicable in standard tableau calculi; the nondeterministic handling of GCIs, however, diminishes its value. We also developed an optimization of anywhere blocking which caches the signatures of unblocked nodes in completed models and uses them as blocking candidates in new models; full details can be found in [9].

Table 2 shows the times that our reasoner, Pellet 1.3, FaCT++ 1.1.4, and Racer 1.9.0 take to classify our test ontologies. To isolate the improvements due to each of the two innovations of our algorithm, we evaluated our system with anywhere blocking (denoted as HT), as well as with ancestor blocking [6] (denoted as HT-anc). All ontologies are available from our reasoner's Web page.

NCI is a relatively large (about 23000 atomic concepts) but simple ontology. FaCT++ and RACER can classify NCI in a short time mainly due to an optimization which eliminates many unnecessary tests, and the fact that all axioms in NCI are definitional so they are handled efficiently by absorption. We conjecture that Pellet is slower by two orders of magnitude because it does not use these optimizations, so it must deal with disjunctions.

GALEN has often been used as a benchmark for DL reasoning. The original version of GALEN contains about 2700 atomic concepts and many GCIs similar to (2). Most GCIs cannot be absorbed without any residual nondeterminism. Thus, the ontology is hard because it requires the generation of large models with many nondeterministic choices. Hence, GALEN has been simplified by removing 273 axioms, and this simplified version of GALEN has commonly been used for performance testing. As Table 2 shows, only HT can classify the original version of GALEN. In particular, anywhere blocking prevents our reasoner from generating the same fragments of a model in different branches.

[1] `http://www.cs.man.ac.uk/~bmotik/HermiT/`

5 Conclusion

In this paper, we presented a novel reasoning algorithm for DLs that combines hyper-inferences to reduce the nondeterminism due to GCIs with anywhere blocking to reduce the sizes of generated models. In future, we shall extend our reasoner to handle disjunction and conduct a more comprehensive performance evaluation. Furthermore, we shall investigate the possibilities of optimizing the blocking condition and heuristically guiding the model construction to further reduce the sizes of the models created. Finally, we shall try to extend our approach to the DLs $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$, which provide the logical underpinning of the Semantic Web ontology languages.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
2. P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *Proc. JELIA '96*, pages 1–17, Évora, Portugal, September 30–October 3 1996.
3. C. Fermüller, T. Tammet, N. Zamov, and A. Leitsch. *Resolution Methods for the Decision Problem*, volume 679 of *LNAI*. Springer, 1993.
4. C. G. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. Resolution Decision Procedures. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 25, pages 1791–1849. Elsevier Science, 2001.
5. V. Haarslev and R. Möller. RACER System Description. In *Proc. IJCAR 2001*, pages 701–706, Siena, Italy, June 18–23 2001.
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. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univesität Karlsruhe, Germany, 2006.
9. Boris Motik, Rob Shearer, and Ian Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In *Proc. of the 16th Int. Conf. on Automated Deduction (CADE-16)*, July 17–20 2007. 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. D. A. Plaisted and S. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Symbolic Logic and Computation*, 2(3):293–304, 1986.
12. A. Robinson. Automatic Deduction with Hyper-Resolution. *Int. Journal of Computer Mathematics*, 1:227–234, 1965.
13. S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, Germany, 2001.
14. D. Tsarkov and I. Horrocks. Efficient Reasoning with Range and Domain Constraints. In *Proc. DL 2004*, Whistler, BC, Canada, June 6–8 2004.
15. D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. IJCAR 2006*, pages 292–297, Seattle, WA, USA, 2006.