Optimized Description Logic Reasoning via Core Blocking

Birte Glimm, Ian Horrocks, and Boris Motik

Oxford University Computing Laboratory, UK

Abstract. State of the art reasoners for expressive description logics, such as those that underpin the OWL ontology language, are typically based on highly optimized implementations of (hyper)tableau algorithms. Despite numerous optimizations, certain ontologies encountered in practice still pose significant challenges to such reasoners, mainly because of the size of the model abstractions that they construct. To address this problem, we propose a new blocking technique that tries to identify and halt redundant construction at a much earlier stage than standard blocking techniques. An evaluation of a prototypical implementation in the HermiT reasoner shows that our technique can dramatically reduce the size of constructed model abstractions and reduce reasoning time.

1 Introduction

Description logics (DLs) are a family of logic based knowledge representation formalisms [1] widely used in conceptual modeling, and they provide the formal basis for the OWL 2 ontology language [2]. DL reasoners support the development and deployment of ontologies in numerous tools and applications. State of the art reasoners for expressive DLs are typically based on highly optimized variants of (hyper)tableau algorithms—model building procedures that decide the (un)satisfiability of a knowledge base $\mathcal{K}$ via a constructive search for an abstraction of a model for $\mathcal{K}$. Examples of such reasoners include FaCT++ [3], HermiT [4], Pellet [5], and Racer [6].

Despite numerous optimizations, certain existing and emerging knowledge bases still pose significant challenges to such reasoners. In particular, state of the art (hyper)tableau reasoners use a cycle detection technique called *blocking* to ensure that only finite model abstractions are constructed. These abstractions can, however, be very large, which can be problematical with respect to space and time: the available memory may be exhausted, and even if this is not the case, building such large structures, and potentially performing backtracking search over them, can be time consuming.

It has already been demonstrated that using a more fine-grained blocking condition can make the constructed abstractions smaller, resulting in a significant speedup [7]. Even with such a blocking condition, however, the constructed model abstractions can be very large; furthermore, checking such fine-grained conditions can itself be costly.

To address these problems, we propose a new *core blocking* technique. Our technique first employs an easy-to-check and very "aggressive" blocking condition that can halt the model construction much earlier than existing techniques. This condition is so aggressive that, if used alone, it is not necessarily the case that the constructed abstraction can be expanded into a model. Therefore, after a model abstraction has been

constructed, a detailed check is performed to ensure that all blocks are indeed valid, and model construction terminates only if all blocks pass this check. Checking blocks for validity can be costly, but it has to be performed relatively rarely, sometimes only once.

We present our technique in the context of the hypertableau calculus [4] and show that the resulting calculus is sound, complete, and terminating. As we discuss in Section 3.4, however, our idea can easily be transferred to standard tableau calculi.

To make our technique effective in practice, one must strike a balance between model expansion and blocking validation: if the core blocking condition is too strict, then it will offer little advantage over standard blocking, and model abstractions may still grow very large; if the condition is not strict enough, then it may stop the construction too early, and the reasoner will spend most of its time in validating blocks. We therefore present several variants of core blocking inspired by our observations about reasoning algorithms for $\mathcal{EL}$-like DLs [8, 9], and we present an empirical evaluation using a prototypical implementation of our technique in the HermiT reasoner. The evaluation compares the performance of the hypertableau algorithm employing the original blocking condition and each of the core blocking variants on several widely used ontologies. As might be expected, the effects of core blocking are most pronounced with large and complex ontologies such as DOLCE or GALEN, on which it significantly reduces the sizes of the constructed model abstractions. Furthermore, core blocking allows HermiT to classify an OWL version of the FMA ontology [10], whereas with standard blocking the reasoner runs out or memory. Finally, with simple ontologies such as Wine, core blocking may have little or no effect on the size of abstractions, but it incurs little or no additional cost compared to the standard blocking technique.

2 Preliminaries

We present our results in the context of the hypertableau calculus, which preprocesses a DL knowledge base into a particular clausal form. Therefore, the precise definitions of DLs [1] are not relevant. The basic elements of DL knowledge bases are *atomic concepts*, *atomic roles*, and *individuals*, which correspond to unary predicates, binary predicates, and constants, respectively. Using various *constructors*, one can construct complex *concepts*, which can be transformed into formulae of two-variable first-order logic with counting. For example, the concept $Man \sqcap \exists hasChild.Man$, representing the set of all men with a male child, corresponds to $Man(x) \wedge \exists y : hasChild(x, y) \wedge Man(y)$. Concepts can be used in axioms of the form $C \sqsubseteq D$, which state that the extension of C is contained in the extension of D. For example, $Man \sqsubseteq Person$ expresses the fact that all men are persons. Various DLs allow for other types of axioms, such as statements that a particular role is reflexive, symmetric, or transitive. A TBox $\mathcal{T}$ is a set of axioms. An ABox $\mathcal{A}$ is a set of assertions of the form $C(a)$, $R(a, b)$, $a \approx b$, and $a \not\approx b$, where C is a concept, R is an atomic role, and a and b are individuals. A knowledge base is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$; it is satisfiable if a first-order interpretation satisfying $\mathcal{T}$ and $\mathcal{A}$ exists.

2.1 Model Construction Calculi

Model construction calculi, such as tableau and hypertableau [4], decide the satisfiability of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ by trying to construct a model of $\mathcal{K}$. To this end, they use *derivation*

rules that, given an ABox $\mathcal{A}_\ell$, derive an ABox $\mathcal{A}_{\ell+1}$ and thus explicate information implicit in $\mathcal{T}$ and $\mathcal{A}_\ell$. Derivation rules can be nondeterministic—that is, they can derive more than one $\mathcal{A}_{\ell+1}$ from $\mathcal{A}_\ell$. To check the satisfiability of $\mathcal{K}$, model construction calculi construct a *derivation* for $\mathcal{K}$, which is a sequence of ABoxes $\mathcal{A}_0, \dots, \mathcal{A}_k$ such that $\mathcal{A}_0 = \mathcal{A}$, the ABox $\mathcal{A}_{\ell+1}$ is obtained from $\mathcal{A}_\ell$ by applying a derivation rule for each $0 \leq \ell < k$, and no derivation rule is applicable to $\mathcal{A}_k$. The satisfiability of $\mathcal{K}$ can be decided by checking whether a derivation exists such that $\mathcal{A}_k$ does not contain an obvious contradiction; such an $\mathcal{A}_k$ is called a *pre-model*. Given such an $\mathcal{A}_k$, a model for $\mathcal{K}$ can be constructed by a process called *unraveling* [4].

2.2 The Hypertableau Calculus

The formal definition of the hypertableau calculus is technically involved; therefore, we present here only the aspects that are needed to understand the idea behind core blocking. For further details and the precise definitions, please refer to [4].

The calculus is applicable to a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ expressed in $\mathcal{SROIQ}$ [11]—the DL underpinning OWL 2. The calculus does not operate on $\mathcal{K}$ directly; rather, in order to reduce nondeterminism, it first translates $\mathcal{K}$ into a set of clauses $\mathcal{C}$ and an ABox $\mathcal{A}$. The class of clauses on which the hypertableau calculus operates is called HT-clauses. An HT-clause is an implication of the form $\bigwedge_{i=1}^{m} U_i \rightarrow \bigvee_{j=1}^{n} V_j$, where U_i and V_j are called the *antecedent* and the *consequent* atoms, respectively. Most notably, the atoms in an HT-clause must satisfy the following restrictions, where r is an atomic role, s is a role, A is an atomic concept, and B is a possibly negated atomic concept.

- Each *antecedent* atom is of the form $A(x)$, $r(x, x)$, $r(x, y_i)$, $r(y_i, x)$, $A(y_i)$, or $A(z_j)$.
- Each *consequent* atom is of the form $B(x)$, $\geqslant n\, s.B(x)$, $B(y_i)$, $r(x, x)$, $r(x, y_i)$, $r(y_i, x)$, $r(x, z_j)$, $r(z_j, x)$, $x \approx z_j$, or $y_i \approx y_j$.

These syntactic restrictions reflect the structure of DL axioms and ultimately ensure termination of the calculus. HT-clauses are straightforwardly interpreted in first-order logic, and they intuitively state that at least one consequent atom must be true whenever all atoms in the antecedent are true. We next present the derivation rules of the calculus.

The Hyp-rule is the main derivation rule. For σ a mapping of variables to individuals and U an atom, let $\sigma(U)$ be the atom obtained from U by replacing each variable x with $\sigma(x)$. The Hyp-rule is applicable to an HT-clause $\bigwedge_{i=1}^{m} U_i \rightarrow \bigvee_{j=1}^{n} V_j$ and an ABox $\mathcal{A}_\ell$ if a mapping σ from the variables in the clause to the individuals in $\mathcal{A}_\ell$ exists such that $\sigma(U_i) \in \mathcal{A}_\ell$ for each $1 \leq i \leq m$, but $\sigma(V_j) \notin \mathcal{A}_\ell$ for each $1 \leq j \leq n$. If that is the case, the rule nondeterministically derives $\mathcal{A}_{\ell+1} = \mathcal{A}_\ell \cup \{\sigma(V_j)\}$ for some $1 \leq j \leq n$. For example, when applied to the HT-clause $r(x, y) \rightarrow (\geqslant 1\, r.A)(x) \vee D(y)$ and an ABox $\mathcal{A}_\ell$ containing $r(a, b)$, the Hyp-rule extends $\mathcal{A}_\ell$ with either $(\geqslant 1\, r.A)(a)$ or $D(b)$.

The $\geqslant$-rule deals with existential quantifiers and number restrictions. Intuitively, an assertion $(\geqslant n\, s.B)(a)$ implies the existence of distinct individuals $b_1, \dots, b_n$ such that role s connects a with each b_i and that $B(b_i)$ holds. Thus, the $\geqslant$-rule is applicable to $(\geqslant n\, s.B)(a)$ in $\mathcal{A}_\ell$ if no individuals $b_1, \dots, b_n$ exist such that $\mathsf{ar}(s, a, b_i) \in \mathcal{A}_\ell$ and $B(b_i) \in \mathcal{A}_\ell$ for each $1 \leq i \leq n$, and $b_i \not\approx b_j \in \mathcal{A}_\ell$ for each $1 \leq i < j \leq n$, where

$\mathsf{ar}(s, a, b) = s(a, b)$ if s is an atomic role and $\mathsf{ar}(s, a, b) = r(b, a)$ if s is an inverse role such that $s = r^-$. If that is the case, then $\mathcal{A}_\ell$ is extended to $\mathcal{A}_{\ell+1}$ by introducing fresh individuals $c_1, \ldots, c_n$ and adding assertions $\mathsf{ar}(s, a, c_i)$ and $B(c_i)$ for $1 \leq i \leq n$, and $c_i \not\approx c_j$ for $1 \leq i < j \leq n$. The individual a is called a *predecessor* of each c_i; each c_i is called a *successor* of a; and *ancestor* and *descendant* are transitive closures of the predecessor and successor relations, respectively.

The $\approx$-rule deals with equality. Intuitively, an assertion $a \approx b$ states that individuals a and b are equal, so one can be treated as a synonym for the other. The $\approx$-rule is applicable to an assertion $a \approx b$ in $\mathcal{A}_\ell$ if $a \neq b$, in which case it derives $\mathcal{A}_{\ell+1}$ from $\mathcal{A}_\ell$ by a process called *merging*: a is replaced with b in all assertions, certain assertions are removed, and some bookkeeping information is added in order to ensure termination.

The NI-rule deals with certain complications due to an interaction between number restrictions, nominals, and inverse roles. The details of this derivation rule are not relevant for core blocking, so we omit it for the sake of brevity.

The $\bot$-rule detects obvious contradictions. If $\mathcal{A}_\ell$ contains $A(a)$ and $\neg A(a)$, or $a \not\approx a$, the rule derives $\mathcal{A}_{\ell+1} = \mathcal{A}_\ell \cup \{\bot\}$; then $\mathcal{A}_{\ell+1}$ is said to contain a *clash*.

2.3 Blocking

The $\geqslant$-rule is found in virtually all DL model construction calculi. Unrestricted application of the $\geqslant$-rule can lead to the introduction of infinitely many fresh individuals and thus prevent the calculus from terminating. To counteract that, the $\geqslant$-rule is applied to an assertion $(\geqslant n\, r.B)(a)$ only if the individual a is not *blocked*, as described next.

To apply blocking, the individuals are split into two sets. *Root* individuals either occur in the input or are introduced by the NI-rule, and they are never blocked. In contrast, *blockable* individuals are introduced by the $\geqslant$-rule and they can be blocked. Blocking uses labels of an individual and an individual pair defined as follows, where A is an atomic concept and r is an atomic role:

$$\mathcal{L}_\mathcal{A}(s) = \{A \mid A(s) \in \mathcal{A}\} \qquad \mathcal{L}_\mathcal{A}(s, t) = \{r \mid r(s, t) \in \mathcal{A}\}$$

To prevent cyclic blocks, one needs to pick a strict order $\prec$ over all individuals satisfying certain restrictions. In practice, $\prec$ usually coincides with the order in which individuals are inserted into an ABox.

Pairwise anywhere blocking is necessary for knowledge bases that use inverse roles and number restrictions. Each individual s in an ABox $\mathcal{A}$ is assigned by induction on $\prec$ a status as follows: s is *blocked* if it is directly or indirectly blocked; s is *indirectly blocked* if it has a blocked ancestor; and s is *directly blocked* by an individual t if, for s' and t' the predecessors of s and t, respectively, s, t, s', and t' are all blockable, t is not blocked, $t \prec s$, and (1)–(4) hold.

$$\mathcal{L}_\mathcal{A}(s) = \mathcal{L}_\mathcal{A}(t) \tag{1}$$
$$\mathcal{L}_\mathcal{A}(s') = \mathcal{L}_\mathcal{A}(t') \tag{2}$$
$$\mathcal{L}_\mathcal{A}(s, s') = \mathcal{L}_\mathcal{A}(t, t') \tag{3}$$
$$\mathcal{L}_\mathcal{A}(s', s) = \mathcal{L}_\mathcal{A}(t', t) \tag{4}$$

The simpler *single anywhere blocking* can be used on knowledge bases without inverse roles, and it differs from the above definition in that s is *directly blocked* by an individual t if s and t are blockable, t is not blocked, $t \prec s$, and (1) holds.

A pre-model $\mathcal{A}'$ can be extended to a model for $(\mathcal{A}, \mathcal{C})$ by unraveling. Roughly speaking, each individual s that is directly blocked in $\mathcal{A}'$ by t is replaced by a "copy" of t; a precise account of this process is given in [4].

Tableau algorithms for DLs without inverse roles can use *single subset blocking*, in which s is directly blocked by t if $\mathcal{L}_{\mathcal{A}}(s) \subseteq \mathcal{L}_{\mathcal{A}}(t)$. This can result in smaller pre-models; however, in the hypertableau calculus single subset blocking does not guarantee that a pre-model can be expanded into a model [4].

3 Optimized Blocking Strategies

On complex knowledge bases, reasoners based on model construction calculi may construct very large pre-models. This is one of the main limiting factors in practical DL reasoning [4], as the construction may be time-consuming, and the reasoner may even run out of memory before terminating. Blocking prevents the application of the $\geqslant$-rule and thus constrains the size of pre-models, so some effort has been devoted to detecting situations in which a block can be established earlier than with the standard single or pairwise blocking, but without sacrificing soundness.

For tableau algorithms that normally require pairwise blocking, Horrocks and Sattler proposed a more precise blocking condition [7], which amounts to single subset blocking with additional constraints on the predecessor of the individual that is to be blocked and on the blocker itself. For example, if an individual s with predecessor s' is to be blocked by an individual t, and $\forall r.C$ is in the label of the blocker t, then either s should be not an r^--successor of s' or C should be in the label of s'. Although checking the blocking conditions is quite expensive, the optimization exhibits substantial improvements in reasoning performance due to the significantly smaller pre-models.

Related blocking optimizations were proposed in the context of first-order theorem proving [12]; however, these techniques do not guarantee termination for DLs such as $\mathcal{SROIQ}$ that provide for nominals, number restrictions, and inverse roles.

Caching [13] is an orthogonal approach for reducing the pre-model size by reusing already constructed pre-model fragments. In fact, caching techniques can be used to obtain a worst-case optimal algorithm for certain DLs [14, 15]; in contrast, standard (hyper)tableau algorithms are usually not worst-case optimal.

3.1 Core Blocking

Unlike existing blocking techniques, core blocking is approximate rather than exact: applying core blocking alone does not guarantee that a pre-model can indeed be unraveled into a model. To ensure the latter, a pre-model needs to be checked to discover invalid blocks; if such blocks are found, the derivation is continued until either a contradiction is derived or all blocks become valid. The latter condition is satisfied at latest when the standard blocking condition is satisfied, which ensures termination.

To formalize the process of discovering approximate blocks, we assume that each assertion α in an ABox is associated with a Boolean flag that determines whether α is a *core assertion*. A *core blocking policy* will be used to determine which assertions are core. A credulous policy would make no assertions core, thus allowing any individual to potentially block any other individual. While this might generate very small pre-models, it is unlikely to be practicable because most blocks are likely to be invalid, so a reasoner would spend a lot of time in validation. In contrast, a conservative policy would make all assertions core, thus making core blocking exact. In Section 3.3 we present two policies that strike a balance between the potential for reduction in the pre-model size and the cost of validating blocks. Before that, however, we introduce a general notion of core blocking that is applicable to any policy.

Definition 1. *For an ABox $\mathcal{A}$ and a pair of individuals s and t, let*

$$\mathcal{L}^{\mathsf{core}}_{\mathcal{A}}(s) = \{A \mid A \in \mathcal{L}_{\mathcal{A}}(s) \text{ and } A(s) \text{ is a core assertion in } \mathcal{A}\} \text{ and}$$
$$\mathcal{L}^{\mathsf{core}}_{\mathcal{A}}(s,t) = \{r \mid r(s,t) \in \mathcal{L}_{\mathcal{A}}(s,t) \text{ and } r(s,t) \text{ is a core assertion in } \mathcal{A}\}.$$

Single *and* pairwise core blocking *are obtained from the respective definitions given in Section 2.3 by using $\mathcal{L}^{\mathsf{core}}_{\mathcal{A}}$ instead of $\mathcal{L}_{\mathcal{A}}$ in conditions* (1)–(4)*; furthermore, in single core blocking, for s to be directly blocked by t we additionally require both s and t to be successors of blockable individuals.*

On knowledge bases that normally require pairwise anywhere blocking (i.e., that contain inverse roles), the model construction from [4] requires individuals s and t involved in direct blocking to be successors of blockable individuals. This is reflected in the notion of single core blocking in Definition 1, which allows single core blocking to be used with knowledge bases that contain inverse roles.

Since core blocking may halt the model expansion too early, we introduce a blocking validation test that checks whether any of the derivation rules would be applicable if we were to unravel a candidate pre-model $\mathcal{A}_\ell$ to a model. To this end, we define an ABox $\mathsf{val}_{\mathcal{A}_\ell}(s)$ for a blockable individual s that, intuitively, contains the assertions from the unraveling of $\mathcal{A}_\ell$ that affect inferences involving s. If no derivation rule is applicable to $\mathsf{val}_{\mathcal{A}_\ell}(s)$, we can conclude that no derivation rule is applicable to the model constructed from $\mathcal{A}_\ell$ as discussed in [4].

Definition 2. *Let $\mathcal{C}$ be a set of HT clauses, and let $\mathcal{A}_\ell$ be an ABox. For an individual w, let $|w| = w$ if w is not blocked in $\mathcal{A}_\ell$, and $|w| = w'$ if w is blocked in $\mathcal{A}_\ell$ by w'. For a blockable individual s, the ABox $\mathsf{val}_{\mathcal{A}_\ell}(s)$ is the union of the sets shown in the following table, where u denotes the predecessor of s, v denotes a successor of $|s|$, b denotes a root individual, C denotes a concept, and r denotes an atomic role.*

1	2	3
$\{C(u) \mid C(u) \in \mathcal{A}_\ell\}$	$\{r(u,s) \mid r(u,s) \in \mathcal{A}_\ell\}$	$\{r(s,u) \mid r(s,u) \in \mathcal{A}_\ell\}$
$\{C(s) \mid C(\lvert s\rvert) \in \mathcal{A}_\ell\}$		
$\{C(v) \mid C(\lvert v\rvert) \in \mathcal{A}_\ell\}$	$\{r(s,v) \mid r(\lvert s\rvert, v) \in \mathcal{A}_\ell\}$	$\{r(v,s) \mid r(v, \lvert s\rvert) \in \mathcal{A}_\ell\}$
$\{C(b) \mid C(b) \in \mathcal{A}_\ell\}$	$\{r(s,b) \mid r(\lvert s\rvert, b) \in \mathcal{A}_\ell\}$	$\{r(b,s) \mid r(b, \lvert s\rvert) \in \mathcal{A}_\ell\}$

A blockable individual s is safe for blocking *in an ABox $\mathcal{A}_\ell$ if the following conditions are satisfied:*

– *the* Hyp*-rule is not applicable to an HT-clause* $\gamma \in \mathcal{C}$ *and* $\mathsf{val}_{\mathcal{A}_\ell}(s)$ *with a mapping* σ *such that* $\sigma(x) = s$*, and*
– *the* $\geqslant$*-rule is not applicable to an assertion* $(\geqslant n\, r.B)(s)$ *in* $\mathsf{val}_{\mathcal{A}_\ell}(s)$.

A directly blocked individual s *with predecessor* s' *is* validly blocked *in* $\mathcal{A}_\ell$ *if both* s *and* s' *are safe for blocking.*

We finish this section with a note that, on knowledge bases that normally require single blocking (i.e., that do not contain inverse roles), Definitions 1 and 2 can be simplified. By the model construction from [4], $\mathsf{val}_{\mathcal{A}_\ell}(s)$ then needs to contain only sets from columns 1 and 2 in Definition 2; this, in turn, allows us to drop the extra requirement on the predecessors of s and t in Definition 1 in the case of single core blocking.

3.2 Applying Core Blocking in a Derivation

If an individual s is core-blocked by an individual t but the block is identified as invalid, one should reconsider t as a potential blocker for s only after $\mathsf{val}_{\mathcal{A}_\ell}(s)$ changes; otherwise, the calculus might get stuck in an endless loop trying to block s by t and subsequently discovering the block to be invalid. We deal with this problem by associating with each individual s in $\mathcal{A}_\ell$ a Boolean flag $\mathsf{mod}_{\mathcal{A}_\ell}(s)$ that is updated as the derivation progresses. Intuitively, $\mathsf{mod}_{\mathcal{A}_\ell}(s) = \mathsf{true}$ means that $\mathsf{val}_{\mathcal{A}_\ell}(s)$ has changed since the last time blocks were checked for validity. We also maintain a set S of pairs of validly blocked and blocking individuals, which we to ensure that the calculus terminates only when all blocks are valid.

Definition 3. *Let* S *be a set of pairs of individuals; let* $\mathcal{A}_\ell$ *be an ABox; and let* s *and* t *be individuals occurring in* $\mathcal{A}_\ell$*. Then,* s *is directly blocked by* t *in* $\mathcal{A}_\ell$ *for* S-core blocking *iff* s *is directly blocked by* t *in* $\mathcal{A}_\ell$ *for core blocking and*

$$\langle s, t\rangle \in S \quad \textit{or} \quad \mathsf{mod}_{\mathcal{A}_\ell}(s) = \mathsf{true} \quad \textit{or} \quad \mathsf{mod}_{\mathcal{A}_\ell}(t) = \mathsf{true}.$$

A derivation by the hypertableau calculus with core blocking *for a set of HT-clauses* $\mathcal{C}$ *and an ABox* $\mathcal{A}$ *is constructed by applying the following steps.*

1. *Set* $S := \emptyset$, $\mathcal{A}^{\mathsf{a}} := \mathcal{A}$, *and* $\mathsf{mod}_{\mathcal{A}^{\mathsf{a}}}(s) := \mathsf{true}$ *for each individual* s *in* $\mathcal{A}^{\mathsf{a}}$.
2. *Apply the hypertableau calculus exhaustively to* $\mathcal{A}^{\mathsf{a}}$ *and* $\mathcal{C}$ *while using* S*-core blocking in the* $\geqslant$*-rule; furthermore, whenever* $\mathcal{A}_{\ell+1}$ *is derived from* $\mathcal{A}_\ell$*, for each individual* s *in* $\mathcal{A}_{\ell+1}$ *set*
   (a) $\mathsf{mod}_{\mathcal{A}_{\ell+1}}(s) := \mathsf{true}$ *if* $\mathsf{val}_{\mathcal{A}_{\ell+1}}(s) \neq \mathsf{val}_{\mathcal{A}_\ell}(s)$ *or if* s *does not occur in* $\mathcal{A}_\ell$*, and*
   (b) $\mathsf{mod}_{\mathcal{A}_{\ell+1}}(s) := \mathsf{mod}_{\mathcal{A}_\ell}(s)$ *otherwise.*
   Let $\mathcal{A}^{\mathsf{b}}$ *be a resulting ABox to which no derivation rule is applicable.*
3. *Set* S *to be equal to the set of pairs* $\langle s, t\rangle$ *of individuals such that* s *is directly blocked in* $\mathcal{A}^{\mathsf{b}}$ *by* t *and* s *is validly blocked in* $\mathcal{A}^{\mathsf{b}}$.
4. *Set* $\mathsf{mod}_{\mathcal{A}^{\mathsf{b}}}(s) := \mathsf{false}$ *for each individual* s *in* $\mathcal{A}^{\mathsf{b}}$.
5. *If an individual* s *exists such that* s *is core blocked in* $\mathcal{A}^{\mathsf{b}}$ *by* t *but* $\langle s, t\rangle \notin S$*, then set* $\mathcal{A}^{\mathsf{a}} := \mathcal{A}^{\mathsf{b}}$ *and go to Step 2.*
6. *Return* $\mathcal{A}^{\mathsf{b}}$.

Roughly speaking, our algorithm first applies the derivation rules as usual, with the difference that core blocking is used (this is because $S = \emptyset$ in Step 1). After computing a candidate pre-model $\mathcal{A}^{\mathsf{b}}$ in Step 2, in Step 3 the algorithm updates S to the set of pairs of valid blocks, and in Step 4 it marks all individuals in $\mathcal{A}^{\mathsf{b}}$ as not changed. In Step 5, the algorithm checks whether $\mathcal{A}^{\mathsf{b}}$ contains invalid blocks. If that is the case, the process is repeated; but then, S-core blocking ensures that only those blocks are considered that are known to be valid or for which at least one of the individuals has changed since the last validation. Theorem 1 shows that the calculus is sound, complete, and terminating.

Theorem 1. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a $\mathcal{SROIQ}$ knowledge base and $\mathcal{C}$ the set of HT-clauses for $\mathcal{K}$.*

1. *The hypertableau calculus with core blocking terminates.*
2. *If $\mathcal{C}$ and $\mathcal{A}$ are satisfiable, then $\bot \notin \mathcal{A}^{\mathsf{b}}$ for some $\mathcal{A}^{\mathsf{b}}$ computed by the calculus.*
3. *If $\mathcal{C}$ and $\mathcal{A}$ are unsatisfiable, then $\bot \in \mathcal{A}^{\mathsf{b}}$ for each $\mathcal{A}^{\mathsf{b}}$ computed by the calculus.*

Proof (Sketch). For the first claim, assume that $\mathcal{A}^{\mathsf{b}}$ is an ABox computed in Step 2 such that, whenever s is directly blocked in $\mathcal{A}^{\mathsf{b}}$ by t for core blocking, then s is directly blocked in $\mathcal{A}^{\mathsf{b}}$ by t for standard blocking. Each individual s is then validly blocked in $\mathcal{A}^{\mathsf{b}}$, so $\langle s, t\rangle \in S$ at Step 3 and the condition at Step 5 is not satisfied, so the calculus terminates. Thus, in the worst case, core blocking reduces to standard blocking, which implies a bound on the size of $\mathcal{A}^{\mathsf{b}}$ in the usual way [4]. Furthermore, if an individual t does not validly block s in an ABox $\mathcal{A}^{\mathsf{b}}$, then t can be considered again as a blocker for s only after $\mathsf{val}_{\mathcal{A}^{\mathsf{b}}}(s)$ or $\mathsf{val}_{\mathcal{A}^{\mathsf{b}}}(t)$ changes. Since $\mathcal{A}^{\mathsf{b}}$ is bounded in size, $\mathsf{val}_{\mathcal{A}^{\mathsf{b}}}(s)$ and $\mathsf{val}_{\mathcal{A}^{\mathsf{b}}}(t)$ can change only a bounded number of times; hence, t is considered as a candidate blocker for s only a finite number of times, which implies termination.

The second claim holds in the same way as in [4]. Finally, for the third claim, given an ABox $\mathcal{A}^{\mathsf{b}}$ computed by the calculus such that $\bot \notin \mathcal{A}^{\mathsf{b}}$, we unravel $\mathcal{A}^{\mathsf{b}}$ into an interpretation in the standard way [4]. From the definition of unraveling in [4], one can see that, for each blockable individual s, the ABox $\mathsf{val}_{\mathcal{A}^{\mathsf{b}}}(s)$ contains the assertions that correspond to the part of the unraveled interpretation involving s. Since s is validly blocked in $\mathcal{A}^{\mathsf{b}}$, all the relevant restrictions are satisfied for s. Since all blocks are valid in $\mathcal{A}^{\mathsf{b}}$, the unraveled interpretation is a model of $\mathcal{C}$ and $\mathcal{A}$. □

3.3 Core Blocking Policies

We now present two policies for identifying core assertions. Each policy can be used with either single or pairwise core blocking.

The *simple core policy* is inspired by the following observation. Let $\mathcal{A}$ be a potentially infinite ABox obtained by applying the hypertableau calculus without blocking to an $\mathcal{EL}$ knowledge base $\mathcal{K}$, and let s and t be two individuals introduced by applying the $\geqslant$-rule to assertions of the form $(\geqslant n\, r.B)(s')$ and $(\geqslant m\, r'.B)(t')$. Then, $\mathcal{L}_{\mathcal{A}}(s) = \mathcal{L}_{\mathcal{A}}(t)$; in fact, the concept labels $\mathcal{L}_{\mathcal{A}}(s)$ and $\mathcal{L}_{\mathcal{A}}(t)$ depend only on the concept B. The policy thus makes such assertions $B(s)$ and $B(t)$ core in the hope that, if a knowledge base is sufficiently "$\mathcal{EL}$-like," then s would validly block t.

Definition 4. *The* simple core *policy marks all assertions as not core unless they are covered by one of the following rules.*

- *Each assertion $B(c_j)$ derived by applying the $\geqslant$-rule to an assertion of the form $(\geqslant n\, r.B)(a)$ is marked as core.*
- *Each assertion α' derived by the $\approx$-rule from an assertion α via merging is marked as core if and only if α is core.*
- *If an ABox contains α as a noncore assertion but some derivation rule derives α as a core assertion, the former assertion is replaced with the latter.*

Simple core blocking generates very small cores, but it can be imprecise and can therefore lead to frequent validation of blocks. For example, if s and t are individuals introduced by applying the $\geqslant$-rule as above, then inferences involving the predecessor of s can cause the propagation of new concepts to s, which might invalidate blocking. Furthermore, if the knowledge base contains nondeterministic concepts, then nondeterministic inferences involving s and t may cause $\mathcal{L}_{\mathcal{A}}(s)$ and $\mathcal{L}_{\mathcal{A}}(t)$ to diverge, which can also invalidate blocking. We therefore define the following, stronger notion of cores.

Definition 5. *The* complex core *policy is the extension of the simple core policy in which, whenever the* Hyp*-rule derives an assertion $\sigma(V_j)$ using a mapping σ and an HT-clause $\gamma = \bigwedge_{i=1}^{m} U_i \to \bigvee_{j=1}^{n} V_j$, the assertion $\sigma(V_j)$ is marked as core if and only if $\sigma(V_j)$ is a concept assertion and*

- *$n > 1$, or*
- *$\sigma(V_j)$ is of the form $B(s)$ with s a successor of $\sigma(w)$ for some variable w in γ.*

The complex core policy is motivated by the fact that, when $\mathcal{EL}$-style algorithms are extended to expressive but deterministic DLs such as Horn-$\mathcal{SHIQ}$ [9], the concepts that are propagated to an individual from its predecessor uniquely determine the individual's label, so we mark all such assertions as core.

From the above discussion, one might expect simple core blocking to be exact on HT-clauses obtained from an $\mathcal{EL}$ knowledge base. The knowledge base consisting of axioms (5)–(10), however, shows that this is not the case.

$$A(x) \to (\geqslant 1\, r.C)(x) \quad (5) \qquad\qquad t(x,y) \wedge D(y) \to E(x) \quad (8)$$

$$B(x) \to (\geqslant 1\, s.C)(x) \quad (6) \qquad\qquad B(x) \wedge s(x,y) \wedge E(y) \to \bot \quad (9)$$

$$C(x) \to (\geqslant 1\, t.D)(x) \quad (7) \qquad\qquad A(a) \quad B(a') \quad (10)$$

The algorithm initially produces the pre-model shown on the left-hand side of Figure 1, in which b directly core-blocks b' (since there are no inverse roles, it does not matter that a and a' are root individuals). If single core blocking were exact, the algorithm would terminate and declare the knowledge base satisfiable. This block, however, is invalid since the Hyp-rule is applicable to HT-clause (9) and ABox $\mathsf{val}_{\mathcal{A}_\ell}(b')$ shown below (since there are no inverse roles, $\mathsf{val}_{\mathcal{A}_\ell}(b')$ does not contain assertions from column 3 of Definition 2) for $\sigma(x) = a'$ and $\sigma(y) = b'$. Therefore, the model expansion continues as shown on the right-hand side of Figure 1, which ultimately leads to a contradiction.

$$\begin{aligned}\mathsf{val}_{\mathcal{A}_\ell}(b') = \{\ & B(a'),\ \ \geqslant 1\, s.C(a'),\ \ s(a',b'),\ \ C(b'),\ \ \geqslant 1\, t.D(b'),\ \ E(b'),\\ & D(c),\ \ t(b',c),\ \ A(a),\ \ \geqslant 1\, r.C(a)\ \ \}\end{aligned} \quad (11)$$

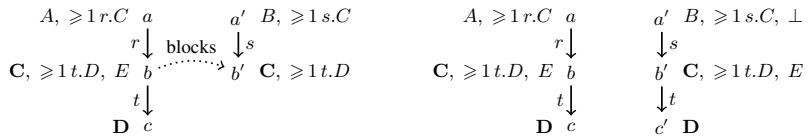

Fig. 1. Left is the first pre-model for the knowledge base (5)–(10) and single simple core blocking, and right is the final pre-model. Core concepts are shown in bold.

3.4 Applying Core Blocking in Tableau Calculi

Although we presented our idea in the context of the hypertableau calculus, core blocking can be straightforwardly adapted to tableau calculi. This would require adapting the definition of $\mathsf{val}_{\mathcal{A}_\ell}(s)$ to reflect the model construction used in the soundness proof of the calculus, and adapting the notion of safe blocking and Definition 5 to reflect the inference rules of the calculus. For example, most tableau calculi use a $\forall$-rule that from $(\forall r.A)(a)$ and $r(a, b)$ derives $A(b)$. Consequently, an individual a should be considered safe for blocking if the $\forall$-rule is not applicable to an assertion in $\mathsf{val}_{\mathcal{A}_\ell}(a)$ involving a, and $A(b)$ should be made core if b is a successor of a.

4 Empirical Evaluation

We implemented the different core blocking strategies in our HermiT reasoner and carried out a preliminary empirical evaluation. For the evaluation, we selected several ontologies commonly used in practice. We classified each ontology and tested the satisfiability of all concepts from the ontologies with the different blocking strategies. We were mainly interested in the number of individuals in a candidate pre-model, which we expected to be smaller due to core blocking. Since the number of individuals in a pre-model directly relates to the amount of memory required by the reasoner, smaller pre-models can make the difference between being able to process an ontology or not.

We conducted our tests on a 2.6 GHz Windows 7 Desktop machine with 8 GB of RAM. We used Java 1.6 allowing for 1 GB of heap space in each test. All tested ontologies, a version of HermiT that supports core blocking, and Excel spreadsheets containing test results are available online.[1]

Figures 2–6 contain concepts on the x-axis; however, concept names are not shown due to the high number of concepts. The concepts are ordered according to the performance of the standard blocking strategy reasoner. The y-axis either displays the number of individuals in the pre-models or the reasoning times in milliseconds. All reasoning times exclude loading and preprocessing times, since these are independent of the blocking strategy. Some figures employ a logarithmic scale to improve readability. The label *standard pairwise* refers to the standard pairwise anywhere blocking strategy, *complex pairwise* refers to pairwise core blocking with the complex core policy, etc.

Tables 1–3 show average measurements taken while testing the satisfiability of all concepts in an ontology. The meaning of various rows is as follows: *final pre-model size*

[1] http://www.hermit-reasoner.com/coreBlocking.html

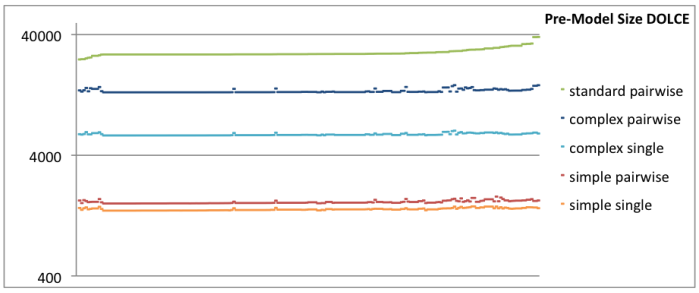

Fig. 2. The number of individuals in the pre-models for all concepts in DOLCE

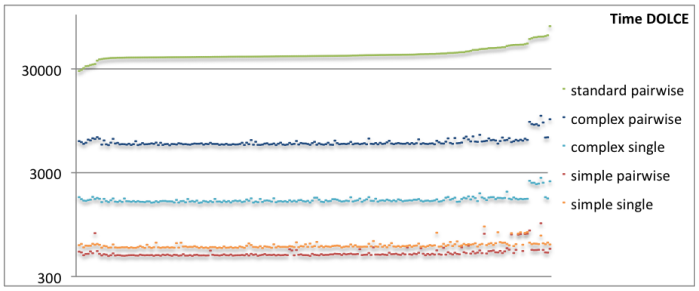

Fig. 3. The reasoning times in ms for testing the satisfiability of all concepts in DOLCE

shows the average number of individuals in the final pre-model; *finally blocked* shows the average number of blocked individuals in the final pre-model; *number of validations* shows the average number of validations before a pre-model was found in which all blocks are valid; *time in ms* shows the average time to test concept satisfiability; and *validation part* shows the percentage of this time taken to validate blocks. Finally, all tables show the time needed to classify the ontology in the format *hours:minutes:seconds*.

DOLCE is a small but complex $\mathcal{SHOIQ}(\mathcal{D})$ ontology containing 209 concepts and 1,537 axioms that produce 2,325 HT-clauses. Core blocking works particularly well on DOLCE. The pre-model sizes (see Figure 2) and the reasoning times (see Figure 3) for all core blocking variants are consistently below those obtained with the standard anywhere blocking strategy. The simple single core blocking strategy gives the smallest pre-models but the reasoning times are slightly smaller for the simple pairwise strategy. This is because the simple single strategy produces more invalid blocks and, consequently, requires more expansion and (expensive) validation cycles before a final pre-model is found (see Table 1). Overall, the strategies work very well because DOLCE does not seem to be highly constrained and many blocks are valid immediately.

Wine has often been used to demonstrate various DL features. The ontology contains 139 concepts and 393 $\mathcal{SHOIN}(\mathcal{D})$ axioms that are converted to 627 HT-clauses. State of the art reasoners can routinely process the ontology. Satisfiability for almost all concepts can be checked using pre-models with very few blocks and almost all blocks are valid immediately, so we have not provided a figure showing the pre-model sizes.

Table 1. Average measurements over all concepts in DOLCE and the classification time

	standard pairwise	complex pairwise	complex single	simple pairwise	simple single
final pre-model size	28,310	13,583	5,942	1,634	1,426
finally blocked	19,319	9,341	4,241	1,207	1,046
number of validations	—	1.03	1.06	1.09	2.09
time in ms	41,821	5,970	1,663	511	601
validation part	—	2.17%	3.98%	48.84%	65.63%
classification time	01:18:32	00:24:03	00:08:43	00:03:45	00:05:29

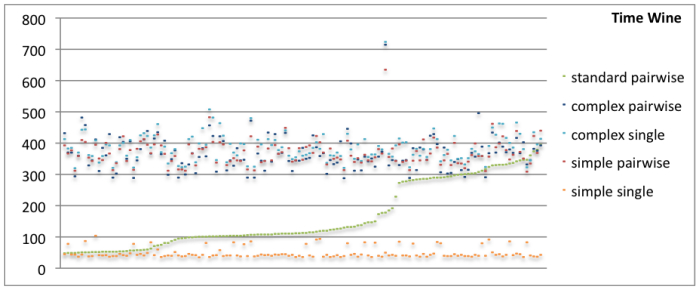

Fig. 4. The reasoning times for testing the satisfiability of all concepts in the Wine ontology

Different blocking strategies (cf. Figure 4 and Table 2) exhibit little difference in performance, and slight variations can be attributed to Java's inconsistent runtime behavior.

GALEN is the original version of the GALEN medical ontology dating from about 10 years ago. Apart from CB [9], which implements an extension of an $\mathcal{EL}$-style algorithm to Horn-$\mathcal{SHIQ}$ [9], HermiT is currently the only reasoner that can classify this ontology. GALEN is a Horn-$\mathcal{SHIF}$ ontology containing 2,748 concepts and 4,979 axioms that produce 8,189 HT-clauses, and it normally requires pairwise blocking. GALEN is unusual in that it contains 2,256 "easy" concepts that are satisfied in very small pre-models ($<$ 200 individuals) and 492 "hard" concepts that are satisfied in very large pre-models ($>$ 35,000 individuals) for the standard blocking strategy. The classification times in Table 3 take all concepts into account; in all other cases we omit the measurements for the "easy" concepts since they do not show much difference between the different blocking strategies and just clutter the presentation. As for DOLCE, the simple single core blocking strategy produces the most significant reduction in model size (see Figure 5). Although this strategy requires the most validation rounds, and these take up 86% of the overall reasoning time, this strategy is still the fastest (see Figure 6) since the reduction in model sizes compensates for the expensive block validations.

The only optimization in HermiT that needs adapting in order to work with core blocking is the *blocking cache*: once a pre-model for a concept is constructed, parts of the pre-model are reused in the remaining subsumption tests [4]. This dramatically reduces the overall classification time. The blocking cache can only be used on ontologies without nominals; in out test suite only GALEN falls into that category. Although the

Table 2. Average measurements over all concepts in Wine and the classification time

	standard pairwise	complex pairwise	complex single	simple pairwise	simple single
final pre-model size	204.56	204.56	204.56	204.56	219.40
finally blocked	0	0	0	0	0
number of validations	—	1.00	1.00	1.01	1.01
time in ms	108	152	154	141	168
validation part	—	0.00%	0.01%	0.01%	0.06%
classification time	00:00:38	00:00:41	00:00:42	00:00:40	00:00:41

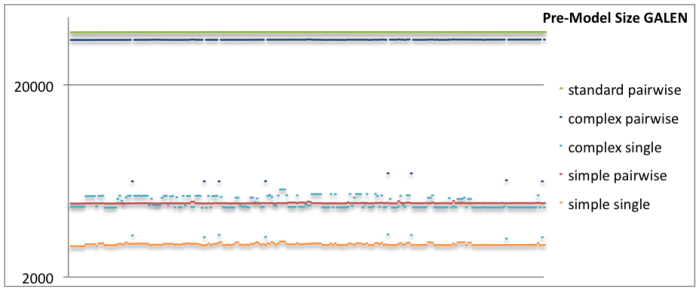

Fig. 5. The number of individuals in the pre-models for the (hard) concepts in GALEN

blocking cache could in principle be adapted for use with core blocking, this has not yet been implemented, so we switched this optimization off.

The foundational model of anatomy (FMA) is a domain ontology about human anatomy [10]. The ontology is one of the largest OWL ontologies available, containing 41,648 concepts and 122,617 $\mathcal{ALCOIF}(\mathcal{D})$ axioms, and it is transformed into 125,346 HT-clauses and 3,740 ABox assertions. We initially tried to take the same measurements for FMA as for the other ontologies; however, after 20 hours we processed only about 10% of the concepts (5,288 out of 41,648), so we aborted the test. Only the single simple core blocking strategy was able to process all 5,288 concepts. The pairwise simple core strategy stayed within the memory limit, but it was significantly slower and it suffered from 5 timeouts due to our imposition of a 2 minute time limit per concept. The standard blocking strategy exceeded either the memory or the time limit on 56 concepts, the pairwise complex core strategy on 70, and the single complex core strategy on 37 concepts. Therefore, we produced complete measurements only with single simple core blocking, using which HermiT was able to classify the entire ontology in about 5.5 hours, discovering 33,431 unsatisfiable concepts. The ontology thus seems to contain modeling errors that went undetected so far due to lack of adequate tool support. The unsatisfiability of all of these concepts was detected before blocking validation was required. The sizes of the ABoxes constructed while processing unsatisfiable concepts is included in the final pre-model size in Table 4, although these are not strictly pre-models since they contain a clash.

We also tested how much memory is necessary to construct all pre-models for DOLCE and GALEN under different blocking strategies. Starting with 16MB, we dou-

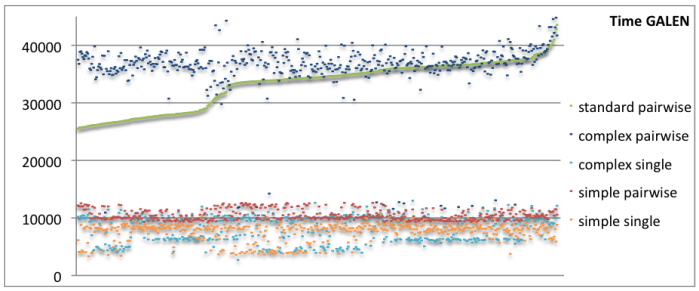

Fig. 6. The reasoning times for testing the satisfiability of the (hard) concepts in GALEN

Table 3. Average measurements over (hard) concepts in GALEN and the classification time

	standard pairwise	complex pairwise	complex single	simple pairwise	simple single
final pre-model size	37,747	33,557	4,876	4,869	2,975
finally blocked	19,290	19,726	2,234	1,896	1,247
number of validations	—	9.18	12.65	8.87	13.91
time in ms	33,293	36,213	8,050	10,485	7,608
validation part	—	27.47%	74.91%	81.47%	86.78%
classification time	03:50:01	04:35:12	01:07:18	01:27:50	01:02:44

bled the memory until the tested strategy could build all pre-models. The simple and complex core blocking strategies require as little as 64MB and 128MB of memory, respectively, whereas the standard blocking technique requires 512MB.

5 Discussion

In this paper we presented several novel blocking strategies that can improve the performance of DL reasoners by significantly reducing the size of the pre-models generated during satisfiability tests. Although we expected complex core blocking to work better on knowledge bases in expressive DLs, the evaluation shows that the simple core policy clearly outperforms the complex core policy regarding space and time on all tested ontologies. On more complex ontologies, the memory requirement with core blocking seems to decrease significantly.

On ontologies such as Wine where very few individuals are blocked, the new strategies cannot really reduce the sizes of the pre-models; however, they do not seem to have a negative effect on the reasoning times either. On more complex ontologies, the memory requirement with core blocking seems to decrease significantly.

The current publicly available version of HermiT (1.2.2) uses simple single core blocking as its default blocking strategy for ontologies with nominals; for ontologies without nominals it uses standard anywhere blocking with the blocking cache optimization, an optimization that has not yet been extended to core blocking.

Blocking validation is not highly optimized in our prototypical implementation. This is most apparent for the single simple core strategy that causes the most invalid

Table 4. Average measurements over FMA with the single simple core strategy

final pre-model size	1,747	finally blocked	1,074
time in ms	518	validation part	0.00%
number of validations	0.2	classification time	05:31:23

blocks and where block validation takes 86% of the time for GALEN. Only the significant model size reductions allows this strategy to nevertheless be the fastest. We believe that we can significantly improve the performance in the future. We identified the two most common reasons for invalid blocks: the $\geqslant$-rule is applicable to an assertion from $\mathsf{val}_{\mathcal{A}_\ell}(s)$ of a blocked individual, or the Hyp-rule is applicable to the assertions from the temporary ABox of the predecessor of a directly blocked individual. Testing for these two cases first should reduce the overall time of validity tests. Finally, we shall adapt the blocking cache technique to core blocking.

Acknowledgements The presented work is funded by the EPSRC project HermiT: Reasoning with Large Ontologies.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook. Cambridge University Press (2003)
2. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 web ontology language document overview. URL (2009) http://www.w3.org/TR/owl2-overview/.
3. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Proc. of IJCAR-06. Volume 4130 of LNCS. (2006) 292 – 297
4. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. of Artificial Intelligence Research **173**(14) (2009) 1275–1309
5. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. of Web Semantics **5**(2) (2007)
6. Haarslev, V., Möller, R.: Description of the RACER system and its applications. In: Proc. of DL-01. (2001)
7. Horrocks, I., Sattler, U.: Optimised reasoning for $\mathcal{SHIQ}$. In: Proc. of ECAI-02. (2002) 277–281
8. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of IJCAI-05. Volume 19. (2005) 364–369
9. Kazakov, Y.: Consequence-driven reasoning for horn SHIQ ontologies. In: Proc. of IJCAI-09. (2009) 2040–2045
10. Golbreich, C., Zhang, S., Bodenreider, O.: The foundational model of anatomy in owl: Experience and perspectives. J. of Web Semantics: Science, Services and Agents on the World Wide Web **4**(3) (2006) 181–195
11. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. KR-06. (2006) 57–67
12. Baumgartner, P., Schmidt, R.A.: Blocking and other enhancements for bottom-up model generation methods. In: Proc. of IJCAR-06. Volume 4130 of LNCS. (2006) 125–139
13. Ding, Y., Haarslev, V.: Tableau caching for description logics with inverse and transitive roles. In: Proc. of DL-06. (2006)
14. Goré, R., Widmann, F.: Sound global state caching for $\mathcal{ALC}$ with inverse roles. In: Proc. of TABLEAUX-09. LNCS (2009) 205–219
15. Donini, F.M., Massacci, F.: EXPTIME tableaux for $\mathcal{ALC}$. Artificial Intelligence Journal **124**(1) (2000) 87–138