

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, 3]. 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++ [4], Hermit [5], Pellet [6], and Racer [7].

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 [8]. 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 has to be performed relatively rarely, sometimes only once.

We present our technique in the context of the hypertableau calculus [5] 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 too 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 [9, 10], 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 [11], whereas with standard blocking the reasoner runs out of 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 [5], 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}_ℓ , derive an ABox $\mathcal{A}_{\ell+1}$ and thus explicate information implicit in \mathcal{T} and \mathcal{A}_ℓ . Derivation rules can be nondeterministic—that is, they can derive more than one $\mathcal{A}_{\ell+1}$ from \mathcal{A}_ℓ . 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}_ℓ 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* [5].

2.2 The Hypertableau Calculus

The formal definition of the hypertableau calculus is technically involved; therefore, we will introduce only those aspects needed to understand the idea behind core blocking, and leave the details to [5].

The calculus is applicable to a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ expressed in *SR \mathcal{O} I \mathcal{Q}* [12]—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 HT-clauses \mathcal{C} and an ABox \mathcal{A} . 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. 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 $B(x)$, $\geq n s.B(x)$, $B(y_i)$, $r(x, x)$, $r(x, y_i)$, $r(y_i, x)$, $r(x, z_j)$, $r(z_j, x)$, $x \approx z_j$, 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}_ℓ if a mapping σ from the variables in the clause to the individuals in \mathcal{A}_ℓ 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 $r(x, y) \rightarrow (\geq 1 r.A)(x) \vee D(y)$ and an ABox \mathcal{A}_ℓ containing $r(a, b)$, the Hyp-rule extends \mathcal{A}_ℓ with either $(\geq 1 r.A)(a)$ or $D(b)$.

The \geq -rule deals with existential quantifiers and number restrictions. Intuitively, an assertion $(\geq n s.B)(a)$ implies the existence of distinct individuals b_1, \dots, b_n such that a is connected through s with each b_i and that $B(b_i)$ holds. Thus, the \geq -rule is applicable to $(\geq n s.B)(a)$ in \mathcal{A}_ℓ if no individuals b_1, \dots, b_n exist such that $\text{ar}(s, a, b_i) \in \mathcal{A}_\ell$ 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 $\text{ar}(s, a, b) = s(a, b)$ if s is an atomic role and $\text{ar}(s, a, b) = r(b, a)$ if s is an inverse role

such that $s = r^-$. If that is the case, then \mathcal{A}_ℓ is extended to $\mathcal{A}_{\ell+1}$ by introducing fresh individuals c_1, \dots, c_n and adding assertions $\text{ar}(s, a, c_i)$ 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}_ℓ if $a \neq b$, in which case it derives $\mathcal{A}_{\ell+1}$ from \mathcal{A}_ℓ 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 \perp -rule detects obvious contradictions. If \mathcal{A}_ℓ contains $A(a)$ and $\neg A(a)$, or $a \approx a$, the rule derives $\mathcal{A}_{\ell+1} = \mathcal{A}_\ell \cup \{\perp\}$; then $\mathcal{A}_{\ell+1}$ is said to contain a *clash*.

2.3 Blocking

The \geq -rule is found in virtually all DL model construction calculi. Unrestricted application of the \geq -rule can lead to the introduction of infinitely many fresh individuals and thus prevent the calculus from terminating. To counteract that, the \geq -rule is applied to an assertion $(\geq n r.B)(a)$ 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 \geq -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:

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

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.

$$\mathcal{L}_{\mathcal{A}}(s) = \mathcal{L}_{\mathcal{A}}(t) \tag{1}$$

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

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

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

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 [5].

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 [5].

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 [5], as the construction may be time-consuming, and the reasoner may even run out of memory before terminating. Blocking prevents the application of the \geq -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 [8], 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 [13]; however, these techniques do not guarantee termination for DLs such as *SRQIQ* that provide for nominals, number restrictions, and inverse roles.

Caching [14] 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 [15, 16]; 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.

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

$$\begin{aligned}\mathcal{L}_{\mathcal{A}}^{\text{core}}(s) &= \{A \in N_C \mid A(s) \text{ is a core assertion in } \mathcal{A} \text{ and } A \text{ is atomic}\} \text{ and} \\ \mathcal{L}_{\mathcal{A}}^{\text{core}}(s, t) &= \{r \in N_C \mid r(s, t) \text{ is a core assertion in } \mathcal{A}\}.\end{aligned}$$

Single and pairwise core blocking are obtained from the respective definitions given in Section 2.3 by using $\mathcal{L}_{\mathcal{A}}^{\text{core}}$ 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 [5] 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}_ℓ to a model. To this end, we define an ABox $\text{val}_{\mathcal{A}_\ell}(s)$ for a blockable individual s that, intuitively, contains the assertions from the unraveling of \mathcal{A}_ℓ that affect inferences involving s . If no derivation rule is applicable to $\text{val}_{\mathcal{A}_\ell}(s)$, we can conclude that no derivation rule is applicable to the model constructed from \mathcal{A}_ℓ as discussed in [5].

Definition 2. Let C be a set of HT clauses, and let \mathcal{A}_ℓ be an ABox. For an individual w , let $|w| = w$ if w is not blocked in \mathcal{A}_ℓ , and $|w| = w'$ if w is blocked in \mathcal{A}_ℓ by w' . For a blockable individual s , the ABox $\text{val}_{\mathcal{A}_\ell}(s)$ 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
$\{C(u) \mid C(u) \in \mathcal{A}_\ell\}$	$\{r(u, s) \mid r(u, s) \in \mathcal{A}_\ell\}$	$\{r(s, u) \mid r(s, u) \in \mathcal{A}_\ell\}$
$\{C(s) \mid C(s) \in \mathcal{A}_\ell\}$		
$\{C(v) \mid C(v) \in \mathcal{A}_\ell\}$	$\{r(s, v) \mid r(s , v) \in \mathcal{A}_\ell\}$	$\{r(v, s) \mid r(v, s) \in \mathcal{A}_\ell\}$
$\{C(b) \mid C(b) \in \mathcal{A}_\ell\}$	$\{r(s, b) \mid r(s , b) \in \mathcal{A}_\ell\}$	$\{r(b, s) \mid r(b, s) \in \mathcal{A}_\ell\}$

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

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

A directly blocked individual s with predecessor s' is validly blocked in \mathcal{A}_ℓ 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 [5], $\text{val}_{\mathcal{A}_\ell}(s)$ 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 and the block is identified as invalid, one should not immediately reconsider t as a potential blocker for s as this might prevent the calculus from terminating. Thus, core blocking cannot be applied straightforwardly in lieu of exact blocking. We deal with this problem by associating with each individual s in \mathcal{A}_ℓ a Boolean flag $\text{mod}_{\mathcal{A}_\ell}(s)$ that is updated as the derivation progresses. Intuitively, $\text{mod}_{\mathcal{A}_\ell}(s) = \text{true}$ means that $\text{val}_{\mathcal{A}_\ell}(s)$ has changed since the last time the blocks were checked for validity; thus, s may be reconsidered as a potential blocker. We also maintain a set S of pairs of validly blocked and blocking individuals, and use S -core blocking to ensure that the calculus only terminates when all blocks are valid.

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

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

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 $S := \emptyset$, $\mathcal{A}^a := \mathcal{A}$, and $\text{mod}_{\mathcal{A}^a}(s) := \text{true}$ for each individual s in \mathcal{A}^a .
2. Apply the hypertableau calculus exhaustively to \mathcal{A}^a and \mathcal{C} while using S -core blocking in the \geq -rule; furthermore, whenever $\mathcal{A}_{\ell+1}$ is derived from \mathcal{A}_ℓ , for each individual s in $\mathcal{A}_{\ell+1}$ set
 - (a) $\text{mod}_{\mathcal{A}_{\ell+1}}(s) := \text{true}$ if $\text{val}_{\mathcal{A}_{\ell+1}}(s) \neq \text{val}_{\mathcal{A}_\ell}(s)$ or if s does not occur in \mathcal{A}_ℓ , and
 - (b) $\text{mod}_{\mathcal{A}_{\ell+1}}(s) := \text{mod}_{\mathcal{A}_\ell}(s)$ otherwise.
 Let \mathcal{A}^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}^b by t and s is validly blocked in \mathcal{A}^b .
4. Set $\text{mod}_{\mathcal{A}^b}(s) := \text{false}$ for each individual s in \mathcal{A}^b .
5. If individual s exists such that s is core blocked in \mathcal{A}^b by t but $\langle s, t \rangle \notin S$, then set $\mathcal{A}^a := \mathcal{A}^b$ and go to Step 2.
6. Return \mathcal{A}^b .

Roughly speaking, our algorithm first applies the derivation rules as usual, but using core blocking (this is because $S = \emptyset$ in Step 1). After computing a candidate pre-model \mathcal{A}^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}^b as not changed. In Step 5, the algorithm checks whether \mathcal{A}^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. The following theorem shows that our algorithm is sound, complete, and terminating.

Theorem 1. *Let \mathcal{C} be a set of HT-clauses and \mathcal{A} an ABox.*

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

Proof (Sketch). For the first claim, assume that \mathcal{A}^b is an ABox computed in Step 2 such that, whenever s is directly blocked in \mathcal{A}^b by t for core blocking, then s is directly blocked in \mathcal{A}^b by t for standard blocking. Each individual s is then validly blocked in \mathcal{A}^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}^b in the usual way [5]. Furthermore, if an individual t does not validly block s in an ABox \mathcal{A}^b , then t can be considered again as a blocker for s only after $\text{val}_{\mathcal{A}^b}(s)$ or $\text{val}_{\mathcal{A}^b}(t)$ changes. Since \mathcal{A}^b is bounded in size, $\text{val}_{\mathcal{A}^b}(s)$ and $\text{val}_{\mathcal{A}^b}(t)$ 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 [5]. Finally, for the third claim, given an ABox \mathcal{A}^b computed by the calculus such that $\perp \notin \mathcal{A}^b$, we unravel \mathcal{A}^b into an interpretation in the standard way [5]. From the definition of unraveling in [5], one can see that, for each blockable individual s , the ABox $\text{val}_{\mathcal{A}^b}(s)$ contains the assertions that correspond to the part of the unraveled interpretation involving s . Since s is validly blocked in \mathcal{A}^b , all the relevant restrictions are satisfied for s . Since all blocks are valid in \mathcal{A}^b , the unraveled interpretation is a model of \mathcal{C} and \mathcal{A} . \square

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 \geq -rule to assertions of the form $(\geq n r.B)(s')$ and $(\geq m s.B)(t')$. 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 \geq -rule to an assertion of the form $(\geq n r.B)(a)$ 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 \geq -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 $\gamma = \bigwedge_{i=1}^m U_i \rightarrow \bigvee_{j=1}^n V_j$, 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} [10], 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.

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

$$B(x) \rightarrow (\geq 1 s.C)(x) \quad (6) \qquad B(x) \wedge s(x, y) \wedge E(y) \rightarrow \perp \quad (9)$$

$$C(x) \rightarrow (\geq 1 t.D)(x) \quad (7) \qquad A(a) \quad B(a') \quad (10)$$

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 $\text{val}_{\mathcal{A}_\ell}(b')$ shown below (since there are no inverse roles, $\text{val}_{\mathcal{A}_\ell}(b')$ 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.

$$\text{val}_{\mathcal{A}_\ell}(b') = \{ B(a'), \geq 1 s.C(a'), s(a', b'), C(b'), \geq 1 t.D(b'), E(b'), D(c), t(b', c), A(a), \geq 1 r.C(a) \} \quad (11)$$

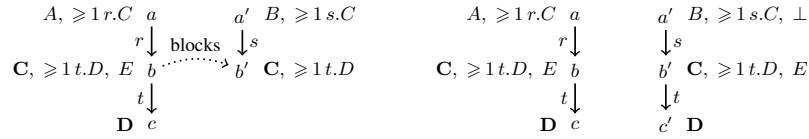


Fig. 1. Left is the first pre-model for the knowledge base (5)–(10) and single simple core blocking, and right is the final pre-model. Core concepts are shown in bold.

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 $\text{val}_{\mathcal{A}_\ell}(s)$ 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 $\text{val}_{\mathcal{A}_\ell}(a)$ 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 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 MacBook Air laptop with 2 GB of RAM running Mac OS 10.6.1. 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 pre-model sizes under the standard blocking strategy. The y-axis either displays the number of individuals in the pre-models or the reasoning times in milliseconds. All reasoning times do not include 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 anywhere blocking strategy, *complex pairwise* refers to pairwise core blocking with the complex core policy, and so on.

Tables 1–3 show measurements taken while testing the satisfiability of all concepts in an ontology averaged over all concepts. The meaning of various rows is as follows:

¹ <http://www.hermit-reasoner.com/coreBlocking.html>

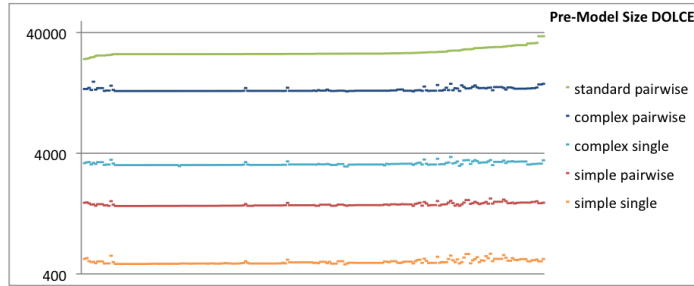


Fig. 2. The number of individuals in the pre-models for all concepts in DOLCE

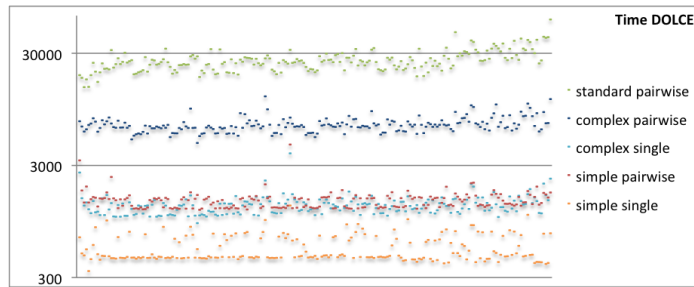


Fig. 3. The reasoning times in ms for testing the satisfiability of all concepts in DOLCE

initial pre-model size shows the average size of the first pre-model found; *initially (invalidly) blocked* shows the average number of (invalidly) core-blocked individuals in the first pre-model; *final pre-model size* 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; and *number of validations* shows the average number of validations needed to decide concept satisfiability. Finally, all tables show the time in seconds needed to classify the ontology using the various blocking approaches.

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 single simple core blocking strategy gives the best results overall even though the ontology normally requires pairwise blocking. DOLCE does not seem to be very highly constrained, so most 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. 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.

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	27,505	13,406	3,263	1,502	502
finally blocked	27,066	12,909	3,167	1,115	418
initial pre-model size	—	13,406	3,195	1,493	420
initially blocked	—	12,908	3,088	1,106	336
initially invalidly blocked	—	0.11	10	1.22	12
number of validations	—	1.09	2.08	2.04	2.96
time in ms	26,168	6,946	1,336	1,470	552
classification time	1h35m26s	25m27s	5m26s	5m59s	2m49s

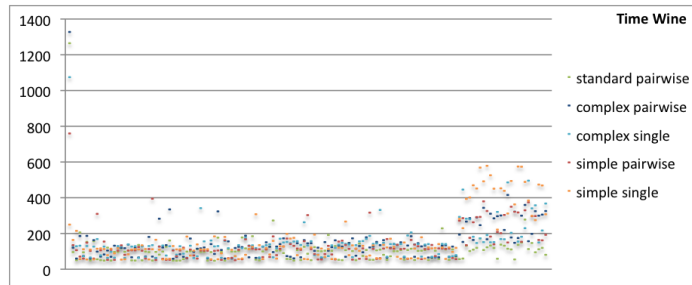


Fig. 4. The reasoning times for testing the satisfiability of all concepts in the Wine ontology

GALEN is the original version of the GALEN medical ontology dating from about 10 years ago. Apart from CB [10], which implements an extension of an \mathcal{EL} -style algorithm to Horn-*SHIQ* [10], HerMiT is currently the only reasoner that can classify this ontology. GALEN is a Horn-*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). 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. On all concepts, all core blocking strategies lead to the construction of significantly smaller pre-models (see Figure 5). Interestingly, only the pairwise strategies improve the reasoning times compared to the standard blocking strategy (see Figure 6); this is due to a large number of expensive validity checks needed to find a valid pre-model. This is in contrast to DOLCE, which also normally requires pairwise blocking, but for which the single core blocking strategies work best.

On ontologies without nominals, HerMiT’s classification algorithm uses a *blocking cache*: once a pre-model for a concept is constructed, parts of the pre-model are reused in the remaining subsumption tests [5]. This dramatically reduces the overall classification time. Although the blocking cache can in principle be used with core blocking, our implementation does not yet implement such an optimization; therefore, the blocking cache has been switched off for the classification of GALEN under the standard block-

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	226	226	226	226	223
finally blocked	18	1	21	1	18
initial pre-model size	—	225	225	225	222
initially blocked	—	1	21	1	17
initially invalidly blocked	—	0.19	0.19	0.06	0.06
number of validations	—	1.12	1.13	1.37	1.37
time in ms	108	152	154	141	168
classification time	1m17s	1m29s	1m24s	1m17s	1m19s

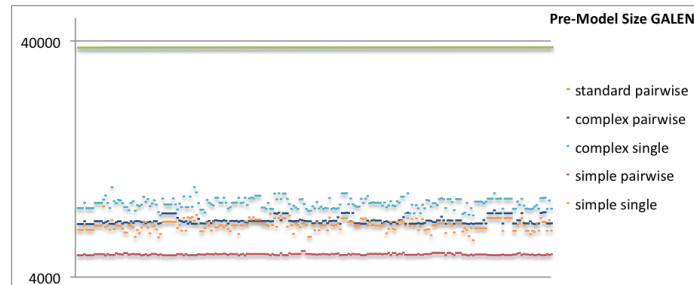


Fig. 5. The number of individuals in the pre-models for all concepts in GALEN

ing strategy. Ontologies in our test suite other than GALEN contain nominals, which prevents the application of the blocking cache with standard blocking anyway.

The foundational model of anatomy (FMA) is a domain ontology about human anatomy. The ontology is one of the largest OWL ontologies available, containing 41,648 concepts and 122,617 $ALCOIF(D)$ axioms, and it is transformed into 125,346 HT-clauses and 3,740 ABox assertions. To the best of our knowledge, no reasoner could so far classify the ontology. 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 was significantly slower and suffered from 5 timeouts due to 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. Hermit was able to classify the entire ontology in about 6.5 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 into the final pre-model size in Table 4; however, these ABoxes are much larger than the ones for satisfiable concepts, which explains the apparent gap between the initial and the final pre-model size.

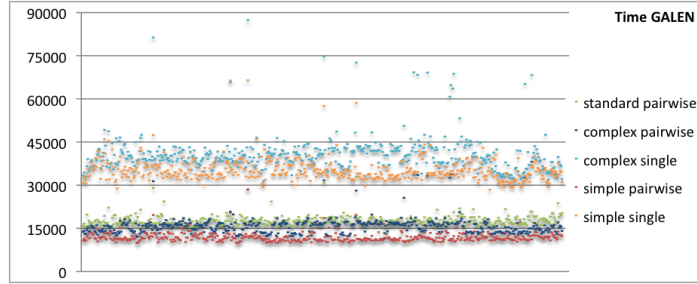


Fig. 6. The reasoning times for testing the satisfiability of all concepts in GALEN

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	37,673	7,689	8,695	5,012	6,640
finally blocked	36,231	6,103	7,969	3,456	5,922
initial pre-model size	—	6,971	1,744	4,646	1,458
initially blocked	—	5,328	1,108	3,087	873
initially invalidly blocked	—	147	216	126	206
number of validations	—	9	18	6	15
time in ms	17,687	15,979	40,655	11,821	35,188
classification time	2h38m46s	4h9m27s	14h2m7s	2h1m7s	4h56m14s

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 doubled 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. In general, simple core blocking seems to lead to the smallest pre-models and the best performance, which makes it a sensible default setting. 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 results are less clear regarding the choice of pairwise vs. single core blocking, which seems to be ontology dependent: pairwise core blocking works best on GALEN, whereas single core blocking is a better choice for DOLCE and FMA. This is a result of the highly constrained nature of the GALEN ontology, which means that single blocks are frequently invalid, and many expensive validity checks are thus needed. If validity tests were less costly, then single core blocking might be the best choice in all cases.

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

final pre-model size	1,638	finally blocked	996
initial pre-model size	89	initially blocked	0.05
initially invalidly blocked	0.01	number of validations	0.2
time in ms	478	classification time	6h30m33s

We have not highly optimized blocking validation in our prototypical implementation, and we believe we can significantly improve its performance in future. Furthermore, we identified two most common reasons for invalid blocks: the \geq -rule is applicable to an assertion from $\text{val}_{\mathcal{A}_\ell}(s)$ 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. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL web ontology language semantics and abstract syntax. URL (2004) <http://www.w3.org/TR/2004/REC-owl-semantics-20040210/>.
3. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 web ontology language document overview. URL (2009) <http://www.w3.org/TR/owl2-overview/>.
4. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Proc. of IJCAR-06. Volume 4130 of LNCS. (2006) 292 – 297
5. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. of Artificial Intelligence Research **173**(14) (2009) 1275–1309
6. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. of Web Semantics **5**(2) (2007)
7. Haarslev, V., Möller, R.: Description of the RACER system and its applications. In: Proc. of DL-01. (2001)
8. Horrocks, I., Sattler, U.: Optimised reasoning for *SHIQ*. In: Proc. of ECAI-02. (2002)
9. Baader, F., Brandt, S., Lutz, C.: Pushing the \mathcal{EL} envelope. In: Proc. of IJCAI-05. (2005)
10. Kazakov, Y.: Consequence-driven reasoning for horn SHIQ ontologies. In: Proc. of IJCAI-09. (2009) 2040–2045
11. 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
12. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible *SROIQ*. In: Proc. KR-06. (2006) 57–67
13. 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
14. Ding, Y., Haarslev, V.: Tableau Caching for Description Logics with Inverse and Transitive Roles. In: Proc. DL. (2006)
15. Goré, R., Widmann, F.: Sound global state caching for *ALC* with inverse roles. In: Proc. of TABLEAUX 2009. LNCS (2009) 205–219
16. Donini, F.M., Massacci, F.: EXPTIME tableaux for *ALC*. Artificial Intelligence Journal **124**(1) (2000) 87–138