J Autom Reasoning (2013) 50:355–381
DOI 10.1007/s10817-011-9243-0

ExpTime Tableaux for $\mathcal{ALC}$ Using Sound Global Caching

Rajeev Goré · Linh Anh Nguyen

Received: 17 April 2009 / Accepted: 18 November 2011 / Published online: 10 December 2011

Abstract We present a simple ExpTime (complexity-optimal) tableau decision procedure based on and-or graphs with sound global caching for checking satisfiability of a concept w.r.t. a TBox in $\mathcal{ALC}$. Our algorithm is easy to implement and provides a foundation for ExpTime (complexity-optimal) tableau-based decision procedures for many modal and description logics, to which various optimisation techniques can be applied.

Keywords Description logic · Modal logic · Tableau-based decision procedures · Global caching complexity-optimal tableaux

1 Introduction

Description logics are (multi-modal) logics that represent the domain of interest in terms of concepts, objects, and roles. They are useful for modelling and reasoning about structured knowledge. The main deduction problems are whether a concept is satisfiable; whether an ABox is consistent; whether a concept is satisfiable w.r.t. a TBox; and whether an ABox is consistent w.r.t. a TBox. In the sequel, we concentrate only on the problem of determining whether a concept is satisfiable w.r.t. a TBox. See [32] for an extension of our method to the problem of checking consistency of an ABox w.r.t. a TBox. The other deduction problems are usually reducible to this latter problem (see [3] for details).

R. Goré (✉)
Logic and Computation Group, Research School of Information Sciences and Engineering, College of Engineering and Computer Science, The Australian National University, Canberra, ACT 0200, Australia
e-mail: rajeev.gore@anu.edu.au

L. A. Nguyen
Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
e-mail: nguyen@mimuw.edu.pl

The tableau method is a very general method for automated reasoning and has been widely applied for modal logics [10] and description logics [2]. Tableau methods usually come in two flavours as we explain shortly. Both methods build a rooted tree with some leaves duplicating ancestors, thereby giving cycles. Because the same node may be explored on multiple branches, tableau algorithms are typically complexity-suboptimal w.r.t. the known theoretical bounds for many logics. For example, the traditional tableau method for $\mathcal{ALC}$ requires double-exponential time even though the decision problem is known to be ExpTime-complete [40].

A tableau is usually defined to be a tree of nodes where the children of a node are created by applying a tableau rule to the parent and where each node contains a finite set of formulae. We refer to this set as the "label" of a node. Thus a label is *not* a name for a Kripke world as in some formulations of "labelled tableaux". The ancestors of a node are simply the nodes on the unique path from the root to that node.

A leaf node is "closed" when it can be deemed to be unsatisfiable, usually because it contains an obvious contradiction like p and $\neg p$. A leaf is "open" when it can be deemed to be satisfiable, usually when no rule is applicable to it, but also when further rule applications are guaranteed to give an infinite branch. A branch is closed (resp. open) if its leaf is closed (resp. open). A tableau is closed if every branch is closed, and it is open if some branch is open. The aim of course is to use these classifications to determine whether the root node is satisfiable or unsatisfiable w.r.t. a TBox. But the tableaux used in modal logics and those used in description logics are dual in a sense which is explained next. We ask the expert reader to bear with us while we digress to elaborate this point.

Traditional Beth Tableaux are Or-trees Traditional modal tableaux a là Beth [4] are **or-trees** in that branches are caused by disjunctions only. Each "existential/diamond" formula in a node causes the creation of a "successor world", fulfilling that formula. But such successors of a given node are created and explored one at a time, using backtracking, until one of them is closed, meaning that there is no explicit trace of previously explored "open" successors in any single tableau. This or-tree perspective makes sense when the goal is to find a closed tableau since we must close both disjuncts of a disjunction but need to close only one "existential/diamond"-successor. A closed tableau implies that the root node is unsatisfiable, but an open tableau does not imply satisfiability since a different existential/diamond choice may give a closed tableau. It is only after all such and-choices have been shown to be open that we can assert satisfiability. We can summarise this view by writing the associated rules as below where we use | for or-branching and use "," for set union:

$$(\vee)\ \frac{X\,,\ C_1 \vee C_2}{X\,,\ C_1 \mid X\,,\ C_2} \qquad\qquad (\exists)\ \frac{X\,,\ \exists R.C}{C\,,\ \{D : \forall R.D \in X\}}$$

Traditional Description Logic Tableaux are And-trees The tableaux used in description logics are usually **and-trees** in that branches are caused by existential/diamond formulae only. Each disjunctive formula causes the creation of a child, one at a time, using backtracking, until one child is open, meaning that there is no explicit trace of previously explored "closed" or-children in any single tableau. This and-tree perspective makes sense when the goal is to find a Kripke model that satisfies the given formula set since we must satisfy every existential/diamond formula in a node but need to satisfy only one disjunct of a disjunction. A particular and-tree is usually

called a "run" and the different or-choices give rise to multiple runs. The task is to find one single run in which all (and-)branches are open since this implies that the root is satisfiable. But a run in which some (and-)branch is closed does not imply unsatisfiability since a different or-choice may give an open run. It is only after all or-choices (i.e. runs) have been shown to be closed that we can assert unsatisfiability. We can summarise this view by writing the associated rules as below where we deliberately use || to flag and-branching and put $X_i = \{D : \forall R_i.D \in X\}$ for $1 \leq i \leq n$, to save space:

$$(\vee)\ \frac{X\ ,\ C_1 \vee C_2}{X\ ,\ C_i}\ i \in \{1,2\} \qquad\qquad (\exists)\ \frac{X\ ,\ \exists R_1.C_1\ ,\ \cdots\ ,\ \exists R_n.C_n}{C_1\ ,\ X_1\ \ ||\ \cdots\ ||\ \ C_n\ ,\ X_n}$$

Summary Thus, in both types of tableaux, the overall search space is really an and-or tree: traditional modal (Beth) tableaux display only the or-related branches and explore the and-related branches using backtracking while description logic tableaux do the reverse. In this work we unify these two views by taking a global view which considers tableaux as and-or graphs rather than as or-trees or and-trees. We can summarise this view by writing the associated rules as below using both or-branching and and-branching:

$$(\vee)\ \frac{X\ ,\ C_1 \vee C_2}{X\ ,\ C_1 \mid X\ ,\ C_2} \qquad\qquad (\exists)\ \frac{X\ ,\ \exists R_1.C_1\ ,\ \cdots\ ,\ \exists R_n.C_n}{C_1\ ,\ X_1\ \ ||\ \cdots\ ||\ \ C_n\ ,\ X_n}$$

Termination via Blocking/Loop-checking Under certain circumstances, both types of tableau can contain infinite branches because the same node appears again and again on the same branch. To obtain termination, most tableau methods employ a "loop check" or "blocking technique" [2, 21]. The simplest is called "ancestor equality blocking" where we stop expansion of a branch when a node duplicates an ancestor (on the same branch). A variation called "ancestor subset blocking" is to stop expansion if a node is a subset of an ancestor (on the same branch). Note that blocking is merely a device for termination: the *a priori* logical status of the blocked node is "unknown" rather than satisfiable or unsatisfiable. Nevertheless, in certain logics, we can classify such nodes as satisfiable, as explained next.

For most (non fix-point) modal and description logics, ancestor cycles are "good" in that a branch ending with a node blocked by an ancestor can be soundly deemed to be satisfiable. In modal tableau, such a branch will cause backtracking to a higher node where a different and-choice can be made in the hope of finding a choice that closes its branch. In description logic tableau, such a branch will cause no backtracking. In description logic tableaux, where branches are all and-branches, a further refinement called "anywhere blocking" [1] is also possible where a node can be blocked by a node which lies on a different (previously created) and-branch of the same run, although extra conditions are usually required to ensure soundness. Thus in both types of tableaux for simple logics, a blocked node is immediately classified as open even though its logical status is "unknown" rather than satisfiable.

Caching Even with all of these blocking refinements, the naive (description logic or modal) tableau method for checking whether a concept C is satisfiable w.r.t. a TBox Γ in ExpTime description logics like $\mathcal{ALC}$ or $\mathcal{SHIQ}$ leads to a double-exponential time (2ExpTime) algorithm because each tableau branch may have an exponential length, meaning that the method may explore a double-exponential number of nodes.

To counter this, some authors have investigated the idea of remembering ("caching") the satisfiable or unsatisfiable status of previously seen nodes in a look-up table (see, e.g., [6, 7, 22]). Then, when a new tableau node is created, we first check whether this node already has a status of satisfiable or unsatisfiable in the cache, and attach that same status to the new node. For most logics we can refine this to the following, assuming that the current node has label X: if the cache contains a node with label $Y \supseteq X$ (respectively $X \subseteq Y$) and Y has status satisfiable (unsatisfiable) then the current node is given the status satisfiable (unsatisfiable).

Differences Between Blocking and Caching Note the difference between blocking and caching: the status of a blocking node must be either satisfiable or unknown/open, but cannot be unsatisfiable/closed, while the status of a cached node must be known as either unsatisfiable or as satisfiable, but cannot be unknown/open. Moreover, a blocking node must be in the same and-tree (*i.e.* run) while a cached node is stored in an external data structure which sits outside the tableau under construction. As a consequence, it is much easier to prove the soundness of blocking than of caching.

Global Caching By global caching we mean that for each possible set of concepts/formulae, the search space contains at most one node with that set as label and that this node is processed (expanded) at most once.[1] The notion of global caching is an immediate foundation for an ExpTime procedure if the search space contains at most an exponential number of different nodes and as long as the "processing" at each node requires at most exponential time, both with respect to the size of the given problem. Moreover, it can replace all of the previously mentioned notions of equality-blocking and caching simultaneously because it does not rely on knowing the satisfiability or unsatisfiability status of the cached nodes viz:

- ancestor equality blocking occurs automatically as a cache-hit;
- anywhere equality blocking occurs automatically as a cache-hit;
- caching occurs automatically since a previously seen node with a status known as unsatisfiable or satisfiable must have been processed, so it will never be processed again.

For simple logics without converse like $\mathcal{ALC}$, when depth-first search is used, global caching can be simulated by the combination of anywhere equality blocking and systematic caching of both satisfiable and unsatisfiable labels which never discards labels with known status: a cache-hit via global caching means that either the label has previously appeared in the current run or it has been processed completely and its status is known, and thus, either anywhere equality blocking or caching can be applied. Note that anywhere equality blocking is restricted to nodes in the same and-tree (*i.e.* run), whereas global caching allows blocking across or-branches (*i.e.* across runs). Therefore, when a non-depth-first search strategy is used, the equivalence does not hold even for $\mathcal{ALC}$. Furthermore, for the description logic $\mathcal{SHI}$, while global caching (with appropriate cut rules) guarantees optimal complexity [13], the exact

[1] A node with status satisfiable/unsatisfiable can be deleted from the graph after propagating its status to its parents, provided that its label and status are recorded in some structure for later loop-checking [28].

complexity of the combination of dynamic ancestor/anywhere equality blocking and caching is not discussed in [23, 24, 26].[2]

Previous Work As shown in [40], $\mathcal{ALC}$ is a sub-logic of propositional dynamic logic (PDL). By "internalizing" TBoxes, decision procedures for PDL, like those of Pratt [38] and Fischer and Ladner [9], can be used to check $\mathcal{ALC}$-satisfiability w.r.t. a TBox.

The method of Fischer and Ladner first constructs the set of all subsets of the Fischer-Ladner closure of the given initial formula [9]. The size of the Fischer-Ladner closure is linear in the size of the given formula, so this method always requires exponential time.

Pratt's method [38] is formulated in a very indirect way via a labeled tableau calculus, tree-like labeled tableaux, tree-like traditional ("lean") tableaux, and "and-or" graphs. Pratt proves the soundness of his "lean" tableaux [38, Lemma 4.8] via the statement *"The lean procedure leaves the root of a tableau for r unmarked if and only if r is satisfiable"*. Since Pratt's "lean" tableaux are tree-tableaux they do not use our notion of global caching. He does, however, discuss this notion informally as follows [38, page 253]: *"This suggests that we* filter *the tableau (the term used by modal logicians for the process used in the proof in* [6] *of the finite model theorem). That is, we identify equivalent vertices to yield a directed graph, instead of a tree, having at most* 2^n *vertices. Such a graph can be effectively constructed by a machine"*. Thus Pratt [38] informally mentions the idea of global caching but does not prove its soundness. Similar informal comments about reusing previously seen nodes are made by Bucheit et al. [25, pages 127–128].

Pratt's method proceeds in two stages, with the first stage requiring exponential time and space even in simple cases. De Giacomo et al. [5] therefore sketched a direct (single-stage) tableau calculus for checking consistency of a concept w.r.t. a TBox in $\mathcal{ALC}$. Donini and Massacci [7] extended this into a full ExpTime algorithm because *"the transformation of a tableau calculus into an* ExpTime *algorithm is only sketched"* [7, page 89] by De Giacomo et al. Donini and Massacci also showed that many different optimisation techniques can be applied to their algorithm.

Donini and Massacci [7] state that the optimisation of caching both satisfiable and unsatisfiable sets *"prunes heavily the search space but its unrestricted usage may lead to unsoundness* [37]. *It is conjectured that 'caching' leads to* ExpTime*-bounds but this has not been formally proved so far, nor the correctness of caching has been shown"*. [7, page 89]. Later they explain that this is because *"the [global] caching optimisations are left out of the formal descriptions"* [7, page 126]. Their algorithm does not use global caching since it permanently caches *"all and only unsatisfiable sets of concepts"*, and temporarily caches visited nodes on the current branch, even though this means that *"many potentially satisfiable sets of concepts are discarded when passing from a branch to another branch"* [7].

[2] In our opinion, dynamic anywhere equality blocking and systematic caching still requires NExpTime for $\mathcal{SHI}$, regardless of whether depth-first search is used, because of the presence of inverse roles and the fact that dynamic anywhere equality blocking is restricted to nodes in the same and-tree (*i.e.* run). Note, a refinement of global caching called "global state caching" guarantees optimal complexity for inverse roles [18] without cut rules.

Although the algorithms of De Giacomo et al. [5] and Donini and Massacci [7] differ in many respects, they both globally cache only unsatisfiable sets ($\bot$-sets). In contrast, our algorithm *soundly* caches both satisfiable and unsatisfiable sets.

Despite these ExpTime (complexity-optimal) algorithms, the implemented description logic tableau provers like DLP and FaCT [22] usually have non-optimal complexity (2ExpTime). In their overview [3], Baader and Sattler explain that: *"The point in designing these [non-optimal] algorithms was not to prove worst-case complexity results, but ... to obtain 'practical' algorithms ... that are easy to implement and optimise, and which behave well on realistic knowledge bases. Nevertheless, the fact that 'natural' tableau algorithms for such* ExpTime*-complete logics are usually* NExpTime*-algorithms is an unpleasant phenomenon. ... Attempts to design* ExpTime*-tableaux for such logics* (*De Giacomo and Massacci, 1996; Donini and Massacci, 1999*) *usually lead to rather complicated* (*and thus not easy to implement*) *algorithms, which* (*to the best of our knowledge*) *have not been implemented yet"*. [3, page 26].

In summary: the existing ExpTime (complexity-optimal) method of Fischer-Ladner always uses exponential time; the existing ExpTime (complexity-optimal) method of Pratt is based upon and-or graphs and global caching but lacks a proof of soundness of global caching; the existing ExpTime (complexity-optimal) method of Donini and Massacci does not use global caching and is considered to be "rather complicated (and thus not easy to implement)"; and finally, the existing implementations like DLP and FaCT deliberately use complexity-suboptimal methods precisely because they are easy to optimise and easy to implement. One possible explanation is that since some of the optimisations themselves are very complicated, it is much easier to incorporate them into a simple complexity-suboptimal base method than into a complicated ExpTime (complexity-optimal) one.

Our Contributions We present a simple, easy to implement, and easy to optimise ExpTime (complexity-optimal) tableau decision procedure for checking satisfiability of a concept w.r.t. a TBox in $\mathcal{ALC}$ using sound global caching. Our method can be applied to many logics to convert complexity-suboptimal (tree) tableau procedures into ExpTime (complexity-optimal) (and-or graph) ones when complexity-suboptimality is caused by the exploration of the same node on multiple branches. Various optimisation techniques can be incorporated into our procedure to increase efficiency [28]. For example, as we show in Section 6, a basic kind of "on-the-fly" propagation of satisfiability and unsatisfiability can be incorporated in a sound way into our decision procedure.

Our algorithm is based on the and-or viewpoint explained previously so it builds an and-or graph where an or-node reflects the application of a traditional Beth or-branching rule while an and-node reflects the application of a traditional description logic and-branching rule. We build global caching into the construction of the and-or graph by ensuring that no two nodes of the graph have the same label. The status of a non-end node is computed from the status of its children using its kind (and-node/or-node) and treating satisfiability w.r.t. the TBox (i.e. sat) as true and unsatisfiability w.r.t. the TBox (i.e. unsat) as false. When a node gets status sat or unsat, the status can be propagated to its parents in a way appropriate to the graph's and-or structure if desired.

Finally, our framework does not depend on depth-first search and the basic version of our ExpTime algorithm can accept any systematic search strategy.

The Structure of This Paper In Section 2, we recall the notation and semantics of $\mathcal{ALC}$. In Section 3, we present our tableau calculus for $\mathcal{ALC}$, formulated with global caching. In Section 4, we prove soundness and completeness of the calculus. In Section 5, we present a simple ExpTime decision procedure for $\mathcal{ALC}$ that is based on the calculus. In Section 6, we give an improved algorithm that incorporates on-the-fly propagation of satisfiability and unsatisfiability, giving an ExpTime (complexity-optimal) decision procedure containing global caching whose best-case behaviour is not always its worst-case-behaviour. In Section 7, we discuss further optimisations for the procedure. In Section 8 we present comparisons with related work, and in Section 9 we conclude.

Remark 1.1 This paper is an extension of our original workshop paper [12]. The algorithm given in the workshop paper contains an error. The correction is to delete the occurrences of "*propagate*(G, v)" in items (h) and (i), and add the following line immediately after line (j) at the same indentation level:

(k) if $v.status \in \{\mathsf{sat}, \mathsf{unsat}\}$ then *propagate*(G, v)

2 Notation and Semantics of $\mathcal{ALC}$

We use A and B for *concept names* and use R and S for *role names*. We refer to A and B also as *atomic concepts*, and refer to R and S also as *roles*. We use C and D to denote arbitrary concepts.

Concepts in $\mathcal{ALC}$ are formed using the following BNF grammar:

$$C, D ::= \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C$$

A *TBox* is a finite set of axioms of the form $C \sqsubseteq D$ or $C = D$.

An *interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ consists of a non-empty set $\Delta^{\mathcal{I}}$, the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$, the *interpretation function* of $\mathcal{I}$, that maps every concept name A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and maps every role name R to a binary relation $R^{\mathcal{I}}$ on $\Delta^{\mathcal{I}}$. The interpretation function is extended to complex concepts as follows.

$$\begin{array}{lcl}
\top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} & = & \emptyset \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} & = & \left\{x \in \Delta^{\mathcal{I}} \mid \forall y \left[(x, y) \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\right]\right\} \\
(\exists R.C)^{\mathcal{I}} & = & \left\{x \in \Delta^{\mathcal{I}} \mid \exists y \left[(x, y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\right]\right\} .
\end{array}$$

An interpretation $\mathcal{I}$ *satisfies* a concept C if $C^{\mathcal{I}} \neq \emptyset$, and *validates* C if $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$. Clearly, $\mathcal{I}$ *validates* C iff it does not *satisfy* $\neg C$.

An interpretation $\mathcal{I}$ is a *model of a TBox* $\mathcal{T}$ if for every axiom $C \sqsubseteq D$ (resp. $C = D$) of $\mathcal{T}$, we have that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (resp. $C^{\mathcal{I}} = D^{\mathcal{I}}$).

We say that an interpretation $\mathcal{I}$ *satisfies* a set X of concepts if there exists $x \in \Delta^{\mathcal{I}}$ such that $x \in C^{\mathcal{I}}$ for all $C \in X$. We say that a set X of concepts is *satisfiable w.r.t. a TBox* $\mathcal{T}$ if there exists a model of $\mathcal{T}$ that satisfies X.

3 A Tableau Calculus for $\mathcal{ALC}$

Let $\mathcal{T}$ be a TBox and X be a finite set of concepts. In this section, we present a tableau calculus for the problem of checking whether X is satisfiable w.r.t. $\mathcal{T}$.

We assume that concepts are in negation normal form (NNF), where $\neg$ occurs only directly before atomic concepts.[3] We denote the NNF of $\neg C$ by $\overline{C}$. For simplicity, we treat axioms of $\mathcal{T}$ as concepts representing global assumptions: an axiom $C \sqsubseteq D$ is treated as $\overline{C} \sqcup D$, while an axiom $C = D$ is treated as $(\overline{C} \sqcup D) \sqcap (\overline{D} \sqcup C)$. That is, we assume that $\mathcal{T}$ consists of concepts in NNF and call such a $\mathcal{T}$ a TBox in NNF. Thus, an interpretation $\mathcal{I}$ is a model of $\mathcal{T}$ iff $\mathcal{I}$ validates every concept $C \in \mathcal{T}$. As this way of handling TBoxes is not efficient in practice, in Section 7 we will discuss how the "absorption" optimization techniques can be used to improve the performance of our algorithm.

Tableau Rules Tableau rules are written downwards, with a set of concepts above the line as the *premise* and a number of sets of concepts below the line as the *conclusions*. A k-ary tableau rule has k conclusions and some rules have a side-condition which must be true for their application. Each tableau rule is either an *or-rule* or an *and-rule*. The conclusions of an or-rule are separated by |, while conclusions of an and-rule are separated by ||. An or-rule has the meaning that, if the premise is satisfiable w.r.t. the TBox $\mathcal{T}$ then some of the conclusions are also satisfiable w.r.t. $\mathcal{T}$. On the other hand, an and-rule has the meaning that, if the premise is satisfiable w.r.t. $\mathcal{T}$ then all of the conclusions are also satisfiable w.r.t. $\mathcal{T}$.

We use letters like Y and Z to denote sets of concepts and write Y, C or C, Y to denote the set $Y \cup \{C\}$. We define the tableau calculus $\mathcal{CALC}$ w.r.t. a TBox $\mathcal{T}$ to be the set of the tableau rules given in Table 1. The rule ($\exists$) is the only and-rule and is also the only *transitional rule* (since it "realizes" concepts of the form $\exists R.C$). The other rules of $\mathcal{CALC}$ are or-rules, and are also called *static rules*. For each rule of $\mathcal{CALC}$, the distinguished concepts of the premise are called the *principal concepts* of the rule.

A rule ρ is applicable to a node if the node's label is an instance of the premise of ρ. The same instantiation then gives the instances of the conclusions of ρ that correspond to this rule application. As is standard, we assume that the principal concept is not a member of the set Y which appears in the rule descriptions, meaning that no static rule carries its principal concepts into its conclusion.

Example 3.1 Let $D_1, \cdots, D_7$ be arbitrary concepts and let $\mathcal{T} = \{D_7\}$. Instantiating the premise $Y, \exists R_1.C_1, \ldots, \exists R_k.C_k$ of the rule ($\exists$) to the label $\{\exists R.D_1, \exists R.D_2, \exists S.D_3, \forall R.D_4, \forall R.D_5, \forall S.D_6\}$ gives three ||-separated conclusions: $\{D_1, D_4, D_5, D_7\}$, $\{D_2, D_4, D_5, D_7\}$, and $\{D_3, D_6, D_7\}$.

Rule Application Strategy We assume the following order for applying the rules of $\mathcal{CALC}$:

$$(\bot_0), (\bot), (\sqcap), (\sqcup), (\exists).$$

[3] Every concept can be transformed in linear time to an equivalent concept in NNF.

Table 1 Rules of the tableau calculus $\mathcal{CALC}$

$$(\bot_0)\ \frac{Y,\ \bot}{\bot} \qquad (\bot)\ \frac{Y,\ A,\ \neg A}{\bot} \qquad (\sqcap)\ \frac{Y,\ C \sqcap D}{Y,\ C,\ D} \qquad (\sqcup)\ \frac{Y,\ C \sqcup D}{Y,\ C \mid Y,\ D}$$

$$(\exists)\ \frac{Y,\ \exists R_1.C_1,\ \ldots,\ \exists R_k.C_k}{C_1,\ Y_1,\ \mathcal{T}\ \mid\mid\ \ldots\ \mid\mid\ C_k,\ Y_k,\ \mathcal{T}}\ \text{if (*)}$$

(*): Y contains no concepts of the form $\exists R.C$ and $Y_i = \{D : \forall R_i.D \in Y\}$

Tableau for $(\mathcal{T}, X)$ We now give a non-algorithmic description of the procedure to create an and-or tableau. We have chosen this format over the more algorithmic description in [12] to highlight its simplicity.

A graph (V, E) is an and-or graph if each node in V is classified as either an and-node or an or-node. An *and-or graph for* $(\mathcal{T}, X)$, also called a *tableau for* $(\mathcal{T}, X)$, is an and-or graph obtained by using our strategy to apply the rules of $\mathcal{CALC}$ repeatedly to the nodes of a graph whose root node has the label $\mathcal{T} \cup X$ as follows:

- For each node v of the graph, if at least one rule is applicable to its label and no rules have been applied to it previously, then choose the applicable rule with the highest priority and apply it to the label of the node to obtain the k conclusions. Remove duplicates, leaving $j \leq k$ distinct sets $Z_1, \ldots, Z_j$.
- If the graph already contains a node w_i with label Z_i then make w_i a child of v by adding the edge (v, w_i), else create a new node w_i with label Z_i and add it to the current graph as a child of v by adding the edge (v, w_i).
- If the applied rule is $(\exists)$ then:
  - if the edge (v, w_i) has not been adorned yet, adorn it by the empty set (of labels)
  - add the principal concept $\exists R_i.C_i$ that corresponds to the child w_i to the set of labels adorning the edge (v, w_i).
- If the rule applied to v is an or-rule then v is an *or-node*, else v is an *and-node*. This information about which rule is applied to v is recorded for later use.
- If no rule is applicable to v then v is an *end node* as well as an and-node.

Remark 3.2 Note that:

- Each non-end node is "expanded" exactly once, using one rule, so expansion continues until no further node expansion is possible.
- The strategy for choosing which node to expand next is totally arbitrary.
- The graph is constructed using global caching and its nodes have unique labels.
- The strategy for rule applications (via the "Rule Application Strategy" given above) is the standard one whereby we "saturate" a node whose label is free of obvious contradictions by applying the $(\sqcap)$ and $(\sqcup)$ rules as much as possible, and then apply the transitional $(\exists)$ rule to realise existential concepts (but not necessarily in a depth-first manner).
- Nodes expanded by applying a non-branching static rule can be treated either as or-nodes or as and-nodes so we choose to treat them as or-nodes (and this

assumption is used in the proofs). Thus applying the $(\sqcap)$ rule to a node causes the node to become an or-node (which might seem counter-intuitive).
– An edge can be labelled with a set of concepts.

A *marking* of an and-or graph G is a subgraph G^c of G such that:

– the root of G is the root of G^c
– if a node v of G^c is an or-node of G then some edge (v, w) of G is an edge of G^c
– if a node v of G^c is an and-node of G then every edge (v, w) of G is an edge of G^c
– if (v, w) is an edge of G^c then v and w are nodes of G^c.

A marking G^c of an and-or graph G for $(\mathcal{T}, X)$ is *consistent* if it does not contain any node with label $\{\bot\}$. Informally, the existence of a consistent marking flags that the and-or tableau G is "open" while the absence of a consistent marking flags that the and-or tableau G is "closed". The most important point is that we need to explore only one and-or tableau.

Example 3.3 In Fig. 1 we present an and-or graph for $(\mathcal{T}, X)$, where

$$\mathcal{T} = \{A \sqsubseteq B \sqcap C\} \equiv \{\neg A \sqcup (B \sqcap C)\}$$
$$X = \{(\exists R.A \sqcap \exists R.(A \sqcap \neg B)) \sqcup (\exists R.A \sqcap \exists R.(A \sqcap \neg C))\}.$$

The node (7) is used by 4 parents; the nodes (8) and (18) are used by 2 parents. As the graph does not have any consistent marking, by Theorem 3.5 given and proved later in this paper, X is unsatisfiable w.r.t. $\mathcal{T}$. Note that the $(\bot_0)$ rule can be applied to node (11) with label $\{\bot\}$ in Fig. 1, which is why there is a dashed edge from (11) to itself.

Example 3.4 Figures 2 and 3 show an application of our method to the example used by Haarslev and Möller in [20] to point out that caching in their framework must be done carefully. Our method requires no such care.

Haarslev and Möller [20, page 59] consider the satisfiability of the concept E wrt the TBox $\{C \sqsubseteq (\exists R.D) \sqcap (\exists S.F) \sqcap (\forall S.(\neg F \sqcap A)), D \sqsubseteq (\exists R.C), E \sqsubseteq (\exists R.C) \sqcup (\exists R.D)\}$. Their tableau derivation involves the steps

1. $\{i_0 : E\}$
2. $\{i_0 : E, i_0 : \exists R.C\}$ or $\{i_0 : E, i_0 : \exists R.D\}$
3. $\{i_1 : C\}$
4. $\{i_1 : C, i_1 : \exists R.D, i_1 : \exists S.F, i_1 : \forall S.(\neg F \sqcap A)\}$
5. $\{i_2 : D\}$
6. $\{i_2 : D, i_2 : \exists R.C\}$
7. $\{i_3 : C\}$
8. $\{i_4 : F, i_4 : \neg F, i_4 : A\}$
9. $\{i_5 : D\}$

At line 7, further expansion of $i_3 : C$ is blocked by the presence of $i_1 : C$ at line 3 and the concept D in $\{i_2 : D\}$ is cached incorrectly as a satisfiable concept. We therefore return to line 4 and expand the $\exists S.F$ to give line 8 above. Since line 8 is

ExpTime Tableaux for $\mathcal{ALC}$ Using Sound Global Caching

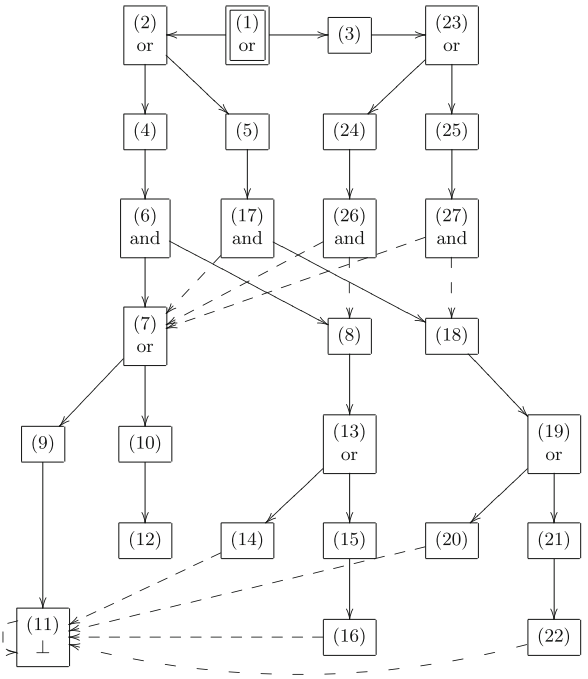
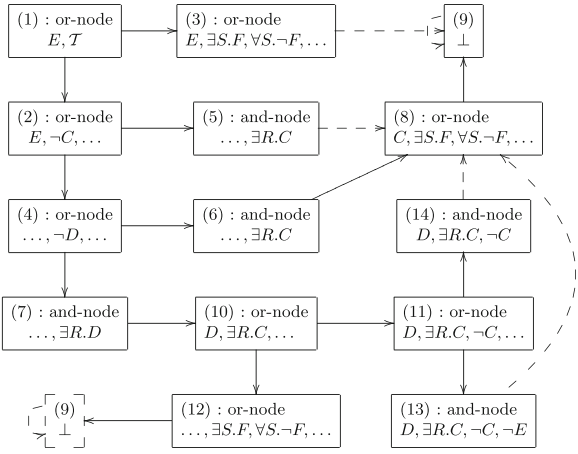


Fig. 1 An and-or graph for $(\mathcal{T}, X)$, where $\mathcal{T} = \{A \sqsubseteq B \sqcap C\} \equiv \{\neg A \sqcup (B \sqcap C)\}$ and X consists of the only concept $(\exists R.A \sqcap \exists R.(A \sqcap \neg B)) \sqcup (\exists R.A \sqcap \exists R.(A \sqcap \neg C))$. Principal concepts are marked with superscript $*$. Nodes are numbered when created but expanded using depth-first search. *Dashed arrows* are cache hits. The graph has no consistent marking

Node	Label	Node	Label
(1)	$\neg A \sqcup^* (B \sqcap C), (\exists R.A \sqcap \exists R.(A \sqcap \neg B)) \sqcup (\exists R.A \sqcap \exists R.(A \sqcap \neg C))$		
(2)	$\neg A, (\exists R.A \sqcap \exists R.(A \sqcap \neg B)) \sqcup^* (\exists R.A \sqcap \exists R.(A \sqcap \neg C))$		
(3)	$B \sqcap^* C, (\exists R.A \sqcap \exists R.(A \sqcap \neg B)) \sqcup (\exists R.A \sqcap \exists R.(A \sqcap \neg C))$		
(23)	$B, C, (\exists R.A \sqcap \exists R.(A \sqcap \neg B)) \sqcup^* (\exists R.A \sqcap \exists R.(A \sqcap \neg C))$		
(4)	$\neg A, \exists R.A \sqcap^* \exists R.(A \sqcap \neg B)$	(5)	$\neg A, \exists R.A \sqcap^* \exists R.(A \sqcap \neg C)$
(6)	$\neg A, \exists^* R.A, \exists^* R.(A \sqcap \neg B)$	(7)	$A, \neg A \sqcup^* (B \sqcap C)$
(8)	$A \sqcap^* \neg B, \neg A \sqcup (B \sqcap C)$	(9)	$A, \neg A$
(10)	$A, B \sqcap^* C$	(12)	A, B, C
(13)	$A, \neg B, \neg A \sqcup^* (B \sqcap C)$	(14)	$A, \neg B, \neg A$
(15)	$A, \neg B, B \sqcap^* C$	(16)	$A, \neg B, B, C$
(17)	$\neg A, \exists^* R.A, \exists^* R.(A \sqcap \neg C)$	(18)	$A \sqcap^* \neg C, \neg A \sqcup (B \sqcap C)$
(19)	$A, \neg C, \neg A \sqcup^* (B \sqcap C)$	(20)	$A, \neg C, \neg A$
(21)	$A, \neg C, B \sqcap^* C$	(22)	$A, \neg C, B, C$
(24)	$B, C, \exists R.A \sqcap^* \exists R.(A \sqcap \neg B)$	(25)	$B, C, \exists R.A \sqcap^* \exists R.(A \sqcap \neg C)$
(26)	$B, C, \exists^* R.A, \exists^* R.(A \sqcap \neg B)$	(27)	$B, C, \exists^* R.A, \exists^* R.(A \sqcap \neg C)$

clearly a clash, we then backtrack to line 2 and explore the alternative of the "or" by expanding $\exists R.D$ to obtain line 9 above. Since D has been cached as satisfiable, this leads to an incorrect classification of E as a satisfiable concept.

To reduce the number of nodes to fit Fig. 2 onto one page, we assume that conjunctions are flattened automatically, that $\forall$ is distributed over $\sqcap$ automatically, that an implicit modus ponens rule is used to obtain $(\exists R.C) \sqcup (\exists R.D)$ from $\{E, E \sqsubseteq (\exists R.C) \sqcup (\exists R.D)\}$, that we can derive $\bot$ from $\exists S.F$ and $\forall S.\neg F$, and that disjunctions are subsumed by either of their disjuncts and hence deleted. The resulting and-or graph has no consistent marking, and so our tableau method gives the correct answer "unsatisfiable".



$\mathcal{T} = \{C \sqsubseteq (\exists R.D) \sqcap (\exists S.F) \sqcap \forall S.(\neg F \sqcap A),\ D \sqsubseteq \exists R.C,\ E \sqsubseteq (\exists R.C) \sqcup (\exists R.D)\}$

Node	Label
(1)	$E, C \sqsubseteq^* (\exists R.D) \sqcap (\exists S.F) \sqcap \forall S.(\neg F \sqcap A), D \sqsubseteq \exists R.C, (\exists R.C) \sqcup (\exists R.D)$
(2)	$E, \neg C, D \sqsubseteq^* \exists R.C, (\exists R.C) \sqcup (\exists R.D)$
(3)	$E, \exists R.D, \exists S.F, \forall S.\neg F, \forall S.A, D \sqsubseteq \exists R.C$
(4)	$E, \neg C, \neg D, (\exists R.C) \sqcup^* (\exists R.D)$
(5)	$E, \neg C, \exists^* R.C$
(6)	$E, \neg C, \neg D, \exists^* R.C$
(7)	$E, \neg C, \neg D, \exists^* R.D$
(8)	$C, \exists R.D, \exists S.F, \forall S.\neg F, \forall S.A, D \sqsubseteq \exists R.C, E \sqsubseteq (\exists R.C) \sqcup (\exists R.D)$
(9)	$\bot$
(10)	$D, \exists R.C, C \sqsubseteq^* (\exists R.D) \sqcap (\exists S.F) \sqcap \forall S.(\neg F \sqcap A), E \sqsubseteq (\exists R.C) \sqcup (\exists R.D)$
(11)	$D, \exists R.C, \neg C, E \sqsubseteq^* (\exists R.C) \sqcup (\exists R.D)$
(12)	$D, \exists R.C, \exists R.D, \exists S.F, \forall S.\neg F, \forall S.A, E \sqsubseteq (\exists R.C) \sqcup (\exists R.D)$
(13)	$D, \exists^* R.C, \neg C, \neg E$
(14)	$D, \exists^* R.C, \neg C$

Fig. 2 An and-or graph for $(\mathcal{T}, \{E\})$. Principal concepts are marked with superscript $*$. Nodes are numbered when created but expanded using depth-first search: 1:(2,3), 2:(4,5), 4:(6,7), 6:8, 8:9, 9, 7:10, 10:(11,12), 11:(13,14), 13:8, 14:8, 12:9, 5:8, 3:9. *Dashed arrows* are cache hits. The graph does not have any consistent marking. See the main text for an explanation of implicit optimisations used to reduce the number of nodes to make the example fit on one page

Although technically correct, Fig. 2 does not adequately highlight the nub of the issue since it is difficult to compare it directly with the process followed by Haarslev and Möller [20, page 59]. We therefore present another view of the same example in Fig. 3 in which we use absorption and lazy-unfolding, as also used by Haarslev and Möller [20], to apply modus ponens on only the relevant parts of the TBox. We also decompose a conjunction into its conjuncts in one step and mimic their derivation as much as possible.

Thus, for example, node (4) in Fig. 3 is obtained from node (3) in one step by using a modus ponens (unfolding) step on C and $C \sqsubseteq (\exists R.D) \sqcap (\exists S.F) \sqcap (\forall S.(\neg F \sqcap A))$, and then decomposing the right hand side of the inclusion into its three conjuncts $(\exists R.D)$, $(\exists S.F)$ and $(\forall S.(\neg F \sqcap A))$. The crucial step is the one where our procedure finds node (3) as a cache hit proxy for the R-child of node (6), but our procedure does not mistakenly mark (5) as satisfiable. Consequently, when node (2') also hits the cached node (5), it does not mistakenly mark itself as satisfiable. Ultimately this leads to the correct answer "unsatisfiable" since the resulting and-or graph has no consistent marking.

For readers who may not be interested in the detailed proofs, we now state the soundness and completeness theorem for $\mathcal{CALC}$.

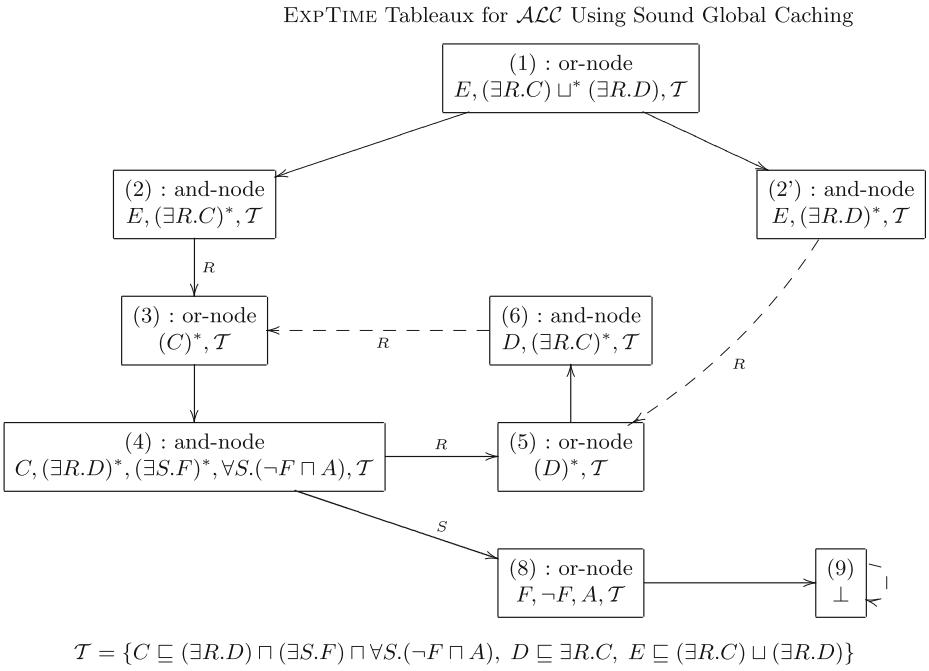


Fig. 3 An and-or graph for $(\mathcal{T}, \{E\})$. Principal concepts are marked with superscript $*$ and *dashed arrows* are cache hits. Nodes are numbered to match our explanation of Haarslev and Möller [20] in the main text so there is no node (7) since our procedure finds it as the existing node (3). As explained in the text, we use absorption and lazy unfolding to apply modus ponens in one step on only the relevant parts of the TBox to make the example fit on one page. The graph does not have any consistent marking

Theorem 3.5 (Soundness and Completeness of $\mathcal{CALC}$) *Let $\mathcal{T}$ be a TBox in NNF, X be a finite set of concepts in NNF, and G be an and-or graph for $(\mathcal{T}, X)$. Then X is satisfiable w.r.t. $\mathcal{T}$ iff G has a consistent marking.*

The proofs of this theorem are in the next section.

4 Soundness and Completeness of $\mathcal{CALC}$

In this section we prove the soundness and completeness of $\mathcal{CALC}$ with respect to the Kripke semantics of $\mathcal{ALC}$. Intuitively, soundness of a calculus states that the calculus gives correct positive answers while completeness states that it gives correct negative answers. One of the most confusing differences between the modal and description logic tableau viewpoints is that these notions are interchanged when moving from one viewpoint to the other. We therefore first explain the differences that give rise to this confusion.

In modal tableaux, a closed or-tableau for $\{\neg C\}$ w.r.t. global assumptions $\mathcal{T}$ is viewed as a proof of C from global assumptions $\mathcal{T}$. So soundness can be stated as: if there is a closed or-tableau for X w.r.t. global-assumptions $\mathcal{T}$ then X is

unsatisfiable w.r.t. global-assumptions $\mathcal{T}$. To compare it with the notion of soundness from description logic tableaux we can rewrite it as:

Modal Tableaux Soundness If X is satisfiable w.r.t. the TBox $\mathcal{T}$ then every or-tableau for X w.r.t. $\mathcal{T}$ is open.

In description logic tableaux, an open and-tableau for X w.r.t. a TBox $\mathcal{T}$ is viewed as a model of $\mathcal{T}$ which satisfies X. Thus soundness becomes:

Description Logic Tableaux Soundness If some and-tableau for X w.r.t. $\mathcal{T}$ is open then X is satisfiable w.r.t. TBox $\mathcal{T}$.

At first sight, these two statements of soundness are not actually exact converses of each other since the first contains "every" while the second contains "some". They are however exact converses once we take into account the differences between or-tableaux and and-tableaux as explained in Section 1. The primary reason why so much confusion arises is that each community uses "tableau" in stating soundness, without adding whether they mean or-tableau or and-tableau.

Which viewpoint should we take with and-or graphs? As we shall see, our procedure tests whether a given set X is satisfiable w.r.t. a TBox $\mathcal{T}$, so in keeping with the principle that soundness should state that the procedure gives correct positive answers, we could use the description logic viewpoint. On the other hand, as we shall see, the fundamental principle is that unsatisfiability ultimately arises from some node label containing a pair $A, \neg A$ or $\bot$, while satisfiability can result from the "failure to be unsatisfiable". Thus we could also take the modal tableau viewpoint, which is exactly what we do with one proviso: since we only ever have to consider one and-or tableau, the notions of "every tableau" (and "some tableau") are replaced by any one "and-or graph G".

Lemma 4.1 (Soundness of $\mathcal{CALC}$) *Let $\mathcal{T}$ be a TBox in NNF, X be a finite set of concepts in NNF, and G be an and-or graph for $(\mathcal{T}, X)$. If X is satisfiable w.r.t. $\mathcal{T}$ then G has a consistent marking.*

Proof Let the root r of G be the root of G^c. Clearly, as X is assumed to be satisfiable w.r.t. $\mathcal{T}$ and the label of the root r of G is $X \cup \mathcal{T}$, the label of the root r of G^c cannot contain $\bot$, nor a contradictory pair $\{A, \neg A\}$. Moreover, it is easy to check that each of our or-rules creates at least one satisfiable child from a satisfiable parent, and all children of the and-rule are satisfiable if the parent is satisfiable. Thus, starting with r, a node $w \in G$ not containing $\bot$ is always available to us as required to construct G^c. □

We prove completeness of $\mathcal{CALC}$ via model graphs. The technique has previously been used in [10, 27, 39] for other traditional modal tableau calculi. A *model graph* is a tuple $\langle \Delta, \mathcal{C}, \mathcal{E} \rangle$, where:

- Δ is a finite set
- $\mathcal{C}$ is a function that maps each element of Δ to a set of concepts
- $\mathcal{E}$ is a function that maps each role name to a binary relation on Δ.

A model graph $\langle \Delta, \mathcal{C}, \mathcal{E} \rangle$ is *saturated* if every $x \in \Delta$ satisfies:

(1) if $C \sqcap D \in \mathcal{C}(x)$ then $\{C, D\} \subseteq \mathcal{C}(x)$
(2) if $C \sqcup D \in \mathcal{C}(x)$ then $C \in \mathcal{C}(x)$ or $D \in \mathcal{C}(x)$
(3) if $\forall R.C \in \mathcal{C}(x)$ and $(x, y) \in \mathcal{E}(R)$ then $C \in \mathcal{C}(y)$
(4) if $\exists R.C \in \mathcal{C}(x)$ then there exists $y \in \Delta$ s.t. $(x, y) \in \mathcal{E}(R)$ and $C \in \mathcal{C}(y)$.

A saturated model graph $\langle \Delta, \mathcal{C}, \mathcal{E} \rangle$ is *consistent* if no $x \in \Delta$ has a $\mathcal{C}(x)$ containing $\bot$ or containing a pair $A, \neg A$ for some atomic concept A.

Given a model graph $M = \langle \Delta, \mathcal{C}, \mathcal{E} \rangle$, the *interpretation corresponding to* M is the interpretation $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ where $A^{\mathcal{I}} = \{x \in \Delta \mid A \in \mathcal{C}(x)\}$ for every concept name A, and $R^{\mathcal{I}} = \mathcal{E}(R)$ for every role name R.

Lemma 4.2 *If $\mathcal{I}$ is the interpretation corresponding to a consistent saturated model graph $\langle \Delta, \mathcal{C}, \mathcal{E} \rangle$, then for every $x \in \Delta$ and $C \in \mathcal{C}(x)$ we have $x \in C^{\mathcal{I}}$.*

Proof By induction on the structure of C. □

Let G be an and-or graph for $(\mathcal{T}, X)$ with a consistent marking G^c and let v be a node of G^c. A *saturation path* of v w.r.t. G^c is a finite sequence $v_0 = v, v_1, \ldots, v_k$ of nodes of G^c, with $k \geq 0$, such that, for every $0 \leq i < k$, v_i is an or-node and (v_i, v_{i+1}) is an edge of G^c, and v_k is an and-node.

Under this definition, if $k = 0$ then there are no i such that $0 \leq i < k$, and v_0 must be an and-node. That is, every and-node has exactly one saturation path of length 1 with $k = 0$. Moreover, every or-node of G^c has at least one saturation path because the rules ($\sqcap$) and ($\sqcup$) reduce a concept to *simpler* ones so that "saturation" eventually leads to a node v_k whose label contains no concept with a top-level constructor of $\sqcap$ or $\sqcup$. Thus v_k in G is "completed" in that no static rules are applicable to its label, and if the label contains no existential concepts, then no rule is applicable at all. In both case, v_k is an and-node as required.

Lemma 4.3 *Let $v_0, v_1, \ldots, v_k$ be a saturation path of v_0 w.r.t. a consistent marking G^c of G. Then*

1. *all concepts of the form $A, \neg A, \forall R.C$ and $\exists R.C$ of the label of each v_i are in the label of v_k;*
2. *the label of each v_i, $0 \leq i \leq k$, cannot contain $\bot$, nor contain a pair $A, \neg A$;*
3. *the set formed by taking the union of the labels of $v_0, v_1, \cdots, v_k$ cannot contain a contradictory pair $A, \neg A$, nor contain $\bot$.*

Proof Part 1 holds because the static rules do not affect concepts of these forms. Part 2 holds because each node v of G^c comes from G, and in G, a node v whose label contains a complimentary pair $A, \neg A$, or contains $\bot$, has to be an or-node via the rules ($\bot_0$) or ($\bot$), whose only child is labelled with $\{\bot\}$, meaning that v cannot be part of a consistent marking G^c of G. For part 3, to appear in the union, each offending concept $A, \neg A$ or $\bot$ has to appear in the label of some $v_i, 0 \leq i \leq k$, which rules out $\bot$ by Part 2. For $A, \neg A$, they must then both appear in v_k by Part 1, which contradicts Part 2. □

Lemma 4.4 (Completeness of $\mathcal{CALC}$) *Let $\mathcal{T}$ be a TBox in NNF, X be a finite set of concepts in NNF, and G be an and-or graph for $(\mathcal{T}, X)$. If G has a consistent marking G^c then X is satisfiable w.r.t. $\mathcal{T}$.*

Proof We construct a model graph $M = \langle \Delta, \mathcal{C}, \mathcal{E} \rangle$ from G^c as given below. During construction, each node of Δ is marked either as *unresolved* or as *resolved* and f is constructed to map each node of M to an and-node of G^c. Nodes of M are denoted by x, y, τ, nodes of G are denoted by u, v, w, and $\mathcal{C}(x)$ denotes the label of x (in M), while $\mathcal{L}(u)$ denotes the label of u (in G):

1. Let v_0 be the root of G^c and let $v_0, \ldots, v_k$ be a saturation path of v_0 w.r.t. G^c. Create a new node (name) τ, set $\Delta := \{\tau\}$, set $\mathcal{C}(\tau) := \bigcup_{i=0}^{k} \mathcal{L}(v_i)$, mark τ as unresolved, and set $f(\tau) := v_k$. For each role name R, set $\mathcal{E}(R) := \emptyset$.
2. While Δ contains unresolved nodes, pick an unresolved node $x \in \Delta$ and do:
   (a) For every concept $\exists R.C \in \mathcal{C}(x)$ do:
      i. Let $u := f(x)$.
      ii. Let w_0 be the node such that (u, w_0) is an edge of G^c with label containing $\exists R.C$.
      iii. Let $w_0, \ldots, w_h$ be a saturation path of w_0 w.r.t. G^c, and let $Y := \bigcup_{i=0}^{h} \mathcal{L}(w_i)$.
      iv. If no $y \in \Delta$ has $\mathcal{C}(y) = Y$ then add a new node y to Δ, set $\mathcal{C}(y) := Y$, mark y as unresolved, and set $f(y) := w_h$.
      v. Add the pair (x, y) to $\mathcal{E}(R)$.
   (b) Mark x as resolved.

Remark 4.5 Note that:

1. At Step 2(a)i, $\exists R.C \in \mathcal{C}(x)$ belongs to $\mathcal{L}(u)$:
   - for the case $x = \tau$ and $u = v_k$, this follows from Lemma 4.3(1)
   - for the other case, see item 3 of this remark.
2. At Step 2(a)ii, w_0 is unique and $C \in \mathcal{L}(w_0)$.
3. At Step 2(a)iv, intuitively, y is the result of sticking together the nodes $w_0, \ldots, w_h$ of a saturation path of G^c. Once again, every $\exists R.C \in \mathcal{C}(y)$ belongs to $\mathcal{L}(w_h)$ by Lemma 4.3(1).

The above construction terminates and results in a finite model graph because: for every $x, x' \in \Delta$, $x \neq x'$ implies $\mathcal{C}(x) \neq \mathcal{C}(x')$, and for every $x \in \Delta$, $\mathcal{C}(x)$ is the set of all subconcepts occurring in $(\mathcal{T}, X)$.

We show that M satisfies all Conditions (1)–(4) of being a *saturated* model graph. M satisfies Conditions (1) and (2) because at Step 1 (of the construction of M), the sequence $v_0, \ldots, v_k$ is a saturation path w.r.t. G^c of v_0, and at Step 2a, the sequence $w_0, \ldots, w_h$ is a saturation path w.r.t. G^c of w_0. That is, in each case, the saturation path is from the original G obtained by applying the rules for $(\sqcap)$ and $(\sqcup)$ repeatedly, and these rules ensure that the union of the labels of this saturation path in M satisfies Conditions (1) and (2) of a saturated model graph. M satisfies Condition (4) because at Step 2a, C belongs to the label of w_0 and hence also to $\mathcal{C}(y)$. For Condition (3), assume $x \in \Delta$, $\forall R.D \in \mathcal{C}(x)$, and $(x, y) \in \mathcal{E}(R)$. We show that $D \in \mathcal{C}(y)$. Consider

Step 2a at which the pair (x, y) is added to $\mathcal{E}(R)$. Because $\forall R.D \in \mathcal{C}(x)$ and $\mathcal{C}(x)$ is the union of the labels of nodes of a saturation path that ends at u, by Lemma 4.3(1), we have that $\forall R.D \in \mathcal{L}(u)$. By the tableau rule ($\exists$), it follows that $D \in \mathcal{L}(w_0)$ and hence also in $\mathcal{C}(y) \supseteq \mathcal{L}(w_0)$.

The label of each node of Δ is formed by taking the union of the labels of some saturation path w.r.t. a consistent marking G^c. By Lemma 4.3(3), such a union cannot contain a complimentary pair $A, \neg A$, nor contain $\bot$. Therefore M is a consistent saturated model graph.

By the definition of and-or graphs for $(\mathcal{T}, X)$ and the construction of M, we have that $X \subseteq \mathcal{C}(\tau)$, and for all $x \in \Delta$ we have $\mathcal{T} \subseteq \mathcal{C}(x)$. Hence, by Lemma 4.2, the interpretation corresponding to M is a model of $\mathcal{T}$ that satisfies X. □

5 A Simple ExpTime Decision Procedure for $\mathcal{ALC}$

Let $\mathcal{T}$ be a TBox in NNF and X be a finite set of concepts in NNF. We claim that Algorithm 1 given below is an ExpTime (complexity-optimal) algorithm for checking satisfiability of X w.r.t. $\mathcal{T}$. In the algorithm, a node u is a parent of v and v is a child of u iff the edge (u, v) is in G. Optimizations for the algorithm will be discussed in the next section. To prove our claim we need some definitions and two lemmata.

Algorithm 1 for checking satisfiability in $\mathcal{ALC}$.

Input: a TBox $\mathcal{T}$ in NNF and a finite set X of concepts in NNF.
Output: *true* if X is satisfiable w.r.t. $\mathcal{T}$, and *false* otherwise.

```
1  construct an and-or graph G with root v0 for (T, X);
2  UnsatNodes := ∅, U := ∅;
3  if G contains a node v⊥ with label {⊥} then
4      U := {v⊥}, UnsatNodes := {v⊥};
5      while U is not empty do
6          remove a node v from U ;
7          for every parent u of v do
8              if u ∉ UnsatNodes
9                 and u is an and-node
10                or u is an or-node and every child of u is in UnsatNodes then
11                add u to both UnsatNodes and U
12 return if v0 ∈ UnsatNodes then false else true.
```

By $sc(C)$ we denote the set of all subconcepts of C, including C. For a set X of concepts, define

$$sc(X) = \{D \mid D \in sc(C) \text{ for some } C \in X\} \ .$$

Define the *length* of a concept to be the number of its symbols, and the *size* of a finite set of concepts to be the sum of the lengths of its elements.

Lemma 5.1 *Let $\mathcal{T}$ be a TBox in NNF, X be a finite set of concepts in NNF, n be the size of $\mathcal{T} \cup X$, and G be an and-or graph for $(\mathcal{T}, X)$. Then G has $2^{O(n)}$ nodes and the label of each node of G is a subset of $sc(\mathcal{T} \cup X)$, which consists of at most $O(n)$ concepts.*

Proof The label of each node of G is a subset of $sc(\mathcal{T} \cup X)$ and therefore consists of at most $O(n)$ concepts. Since the labels of nodes are unique, G has $2^{O(n)}$ nodes. □

Lemma 5.2 *Algorithm* 1 *terminates and computes the set* UnsatNodes *in* $2^{O(n)}$ *steps where n is the size of* $\mathcal{T} \cup X$.

Proof Lemma 5.1 guarantees that the and-or graph G can be built in $2^{O(n)}$ steps since it contains $2^{O(n)}$ nodes. Every node put into U is also put into UnsatNodes, but once a node is in UnsatNodes, it never leaves UnsatNodes and cannot be put back into U. Each iteration of the "while" removes one member of U. Since the number of nodes in G is $2^{O(n)}$, this means that after at most $2^{O(n)}$ iterations, U must become empty. Each iteration is done in $2^{O(n)}$ steps. Hence the algorithm terminates after $2^{O(n)}$ steps. □

Theorem 5.3 *Algorithm* 1 *is an* ExpTime *(complexity-optimal) decision procedure for checking satisfiability of* X *w.r.t.* $\mathcal{T}$.

Proof Let G be the and-or graph with root v_0 constructed by Algorithm 1 for $(\mathcal{T}, X)$ and let UnsatNodes_f be the final value of the set UnsatNodes.

Suppose that Algorithm 1 returns *true* for $(\mathcal{T}, X)$. Then we must have $v_0 \notin \text{UnsatNodes}_f$. We show that G has a consistent marking, which, by Theorem 3.5, implies that X is satisfiable w.r.t. $\mathcal{T}$.

We construct a consistent marking G^c of G as follows. We initialize G^c with the node v_0. Repeatedly, for every node $v \in G^c$, we add $w \in G$ and the edge (v, w) from G to G^c if w is a child of v in G and $w \notin \text{UnsatNodes}_f$.

Observe that, for every node $u \notin \text{UnsatNodes}_f$ of G:

- If u is an and-node then no child of u belongs to UnsatNodes_f. For a contradiction, suppose that a child v of u belongs to UnsatNodes_f. When v was put into UnsatNodes, it was put into U too. The main "while" loop terminates only when U is empty so consider the moment when v was removed from U (at step 6 of Algorithm 1). At that time: u was a parent of v since G was built at step 1 of Algorithm 1; $u \notin$ UnsatNodes since UnsatNodes never shrinks and we already have $u \notin \text{UnsatNodes}_f$; and u was an and-node. Hence u was added to UnsatNodes (at step 11 of Algorithm 1) and must end up in UnsatNodes_f since UnsatNodes never shrinks. This contradicts the assumption that $u \notin \text{UnsatNodes}_f$.
- If u is an or-node then at least one child of u does not belong to UnsatNodes_f. For a contradiction, suppose that all children of u belong to UnsatNodes_f. Since u is an or-node, it must have at least one child. Since UnsatNodes never shrinks, there is a moment when all children of u belong to UnsatNodes. The main "while" loop terminates only when U is empty so consider the moment when a child v of u was removed from U (at step 6 of Algorithm 1) and all children of u had been added to UnsatNodes. At that time: u was a parent of v since G was built at step 1 of Algorithm 1; $u \notin$ UnsatNodes since UnsatNodes never shrinks and we already have $u \notin \text{UnsatNodes}_f$; and u was an or-node. Hence u was added to UnsatNodes (at step 11 of Algorithm 1) and ends up in UnsatNodes_f. This contradicts the assumption that $u \notin \text{UnsatNodes}_f$.

Hence G^c is a marking of G. It is a consistent marking since the only possible node with a label $\{\bot\}$ is $v_\bot \in \text{UnsatNodes}_f$ and hence $v_\bot$ is not in G^c.

Next, suppose that Algorithm 1 returns *false* for $(\mathcal{T}, X)$. We have that $v_0 \in \text{UnsatNodes}_f$. We show that X is unsatisfiable w.r.t. $\mathcal{T}$.

For each $u \in \text{UnsatNodes}_f$, let unsat-timestamp$(v)$ be the iteration number of the main "while" loop at which u was added to UnsatNodes. Observe that, by steps 8–11 of Algorithm 1, for every $u \in \text{UnsatNodes}_f$:

- if u is an or-node and $u \neq v_\bot$ then every child v of u belongs to UnsatNodes_f and has unsat-timestamp$(v) <$ unsat-timestamp(u)
- if u is an and-node then u has a child $v \in \text{UnsatNodes}_f$ with unsat-timestamp$(v) <$ unsat-timestamp(u).

It follows that if G has a consistent marking G^c then starting from v_0 we can construct an infinite path of nodes from G^c consisting of nodes in UnsatNodes_f such that every node has a greater unsat-timestamp than the next node on the path. This is impossible because values of unsat-timestamp are natural numbers. Therefore G does not have any consistent marking and, by Theorem 3.5, it follows that X is unsatisfiable w.r.t. $\mathcal{T}$.

We have proved that Algorithm 1 is a decision procedure for checking satisfiability of X w.r.t. $\mathcal{T}$. By Lemma 5.1, the algorithm runs in $2^{O(n)}$ steps. □

Algorithm 1 is essentially a sound, complete and ExpTime (complexity-optimal) Pratt-like method for testing $\mathcal{ALC}$-satisfiability which incorporates global caching directly into its definition rather than leaving it as an afterthought with no proof. But it suffers from the problem that we first construct a full and-or graph and then make multiple passes to decide satisfiability. That is, its worst-case behaviour is also its best-case behaviour. In the next section we give an ExpTime (complexity-optimal) algorithm which also incorporates global caching in its definition, and which can often avoid this undesirable behaviour.

6 On-the-Fly Propagation of Satisfiability and Unsatisfiability

In this section we optimize Algorithm 1 by incorporating a basic kind of on-the-fly backward propagation of satisfiability of unsatisfiability through the and-or graph. Observe that Algorithm 1 first constructs an and-or graph and then checks whether the graph contains a consistent marking. To speed up the performance these two tasks can be done concurrently. For this we update the set UnsatNodes mentioned in the algorithm and check the condition $v_0 \in$ UnsatNodes "on-the-fly" during the construction of G. This is the basic kind of backward propagation of unsat (unsatisfiability w.r.t. the TBox). We can propagate sat (satisfiability w.r.t. the TBox) in the dual way: end nodes with a label different from $\{\bot\}$ receive status sat; if a child of an or-node has status sat then the node receives status sat; if all children of an and-node have status sat then the node receives status sat.

Algorithm 2 (given on page 21) realizes the above mentioned idea. It builds an and-or graph using a set V of nodes and a set E of ordered pairs of

edges. Each node of V carries a finite set of concepts as a label, a status from {unexpanded, expanded, sat, unsat}, and an and-or type from {and-node, or-node} in the case the status differs from unexpanded.

Algorithm 2 for checking satisfiability in $\mathcal{ALC}$.

Input: a TBox $\mathcal{T}$ in NNF and a finite set X of concepts in NNF.
Output: *true* if X is satisfiable w.r.t. $\mathcal{T}$, and *false* otherwise.

```
1  create the root node v0 with label T ∪ X;
2  initialise status(v0) := unexpanded; V := {v0}; E := ∅
3  while status(v0) ∉ {sat, unsat} and ∃v ∈ V.status(v) = unexpanded do
4      select a node v with status(v) = unexpanded;
5      if no rule is applicable to v then
6          status(v) := sat;
7          make v an and-node;
8          propagate-status (v)
9      else
10         choose the applicable rule with the highest priority and apply it to the label of v
           to obtain k conclusions Z1, ..., Zk, respectively
11         for i := 1 to k do
12             if V already contains a node wi with label Zi then
13                 make wi a child of v by adding the edge (v, wi) to E
14             else
15                 create a new node wi with label Zi;
16                 status(wi) := unexpanded;
17                 add wi to V and add the edge (v, wi) to E
18             if the applied rule is (∃) then
19                 if the edge (v, wi) has not been adorned yet then
20                     adorn it by the empty set
21                 add the principal concept ∃Ri.Ci of rule (∃) to the set of labels of the
                   edge (v, wi)
22         if the applied rule is an or-rule then
23             make v an or-node
24         else
25             make v an and-node
26         status(v) := expanded;
27         update-status (v);
28         if status(v) ∈ {sat, unsat} then propagate-status (v)
29 if status(v0) = unsat then return false else return true
```

Procedure `propagate-status(`v`)`

Input: a node v of an and-or graph with $status(v) \in \{\mathsf{sat}, \mathsf{unsat}\}$
Global Data: the data structures of Algorithm 2
Purpose: propagating the status of v through the and-or graph

```
1  foreach parent u of v do
2      if status(u) = expanded then
3          update-status (u) ;
4          if status(u) ∈ {sat, unsat} then
5              propagate-status (u)
```

Procedure `update-status`(v)

Input: a node v of an and-or graph with $status(v) =$ expanded
Global Data: the data structures of Algorithm 2
Purpose: updating the status of v

```
1  if the label of v is {⊥} then
2  |  status(v) := unsat
3  else
4  |  if v is an or-node then
5  |  |  if all children of v have status unsat then
6  |  |  |  status(v) := unsat
7  |  |  else if a child of v has status sat then
8  |  |  |  status(v) := sat
9  |  else // v is an and-node
10 |  |  if all children of v have status sat then
11 |  |  |  status(v) := sat
12 |  |  else if a child of v has status unsat then
13 |  |  |  status(v) := unsat
14 |
```

Lemma 6.1 *Algorithm* 2 *terminates in* $2^{O(n)}$ *steps where n is the size of* $\mathcal{T} \cup X$.

Proof First, note that `update-status`(v) runs in linear time in the number of children of v. As the graph contains at most $2^{O(n)}$ nodes by Lemma 5.1, a call of `update-status` runs in $2^{O(n)}$ steps.

Consider a call `propagate-status`(v). Without counting time spent for recursive calls of `propagate-status` (at line 5), the number of executed steps is linear with respect to the number of steps needed for updating the status of all parents of v, and is thus of rank $2^{O(n)}$. For any node, the procedure `propagate-status` is called at most once when the status of the node is changed from unexpanded or expanded to sat or unsat (at lines 8 or 28 of Algorithm 2 or at line 5 of procedure `propagate-status`). Here, note that once the status of a node is sat or unsat, it will never change. Hence the total number of steps executed for all calls of `propagate-status` in Algorithm 2 is of rank $2^{O(n)} \times 2^{O(n)}$, which is also $2^{O(n)}$ (with another constant for O).

The additional cost of expanding a node v in Algorithm 2 is the cost of possible calls of `update-status`(v) and `propagate-status`(v). Hence, expanding a node can be done in $2^{O(n)}$ steps. As the graph contains only $2^{O(n)}$ nodes, the algorithm terminates in $2^{O(n)} \times 2^{O(n)}$, i.e. $2^{O(n)}$, steps. □

Theorem 6.2 *Algorithm* 2 *is an* ExpTime *(complexity-optimal) decision procedure for checking satisfiability of* X *w.r.t.* $\mathcal{T}$.

Proof Let G be the possibly partial and-or graph constructed by Algorithm 2 for $(\mathcal{T}, X)$ and let the map *status* be the one constructed for G by Algorithm 2. In this proof, "satisfiability" and "unsatisfiability" are understood as w.r.t. $\mathcal{T}$.

Case Algorithm 2 returns false. We have that $status(v_0) =$ unsat. We proceed directly by induction on the temporal order in which the nodes of G get status unsat. The first

such node is always the node with label $\{\bot\}$, at Step 2 of procedure `update-status`, and this label is clearly unsatisfiable.

Observe that for every node v of G with $status(v) = \mathsf{unsat}$:

- if v is an or-node and the label of v is not $\{\bot\}$ then every child u of v gets $status(u) = \mathsf{unsat}$ before v does (by Step 6 of procedure `update-status`)
- if v is an and-node then v has a child u which gets $status(u) = \mathsf{unsat}$ before v does (by step 13 of procedure `update-status`).

Moreover, it is easy to check that each of our or-rules creates at least one satisfiable child (label) from a satisfiable parent (label), and all children (labels) of the and-rule are satisfiable if the parent (label) is satisfiable. Each such child u of a v with $status(v) = \mathsf{unsat}$ falls under the induction hypothesis, meaning that its label must be unsatisfiable. Thus the label of each such v is unsatisfiable. Since v_0 with label $X \cup \mathcal{T}$ is such a v, it follows that X is unsatisfiable w.r.t. $\mathcal{T}$.

Case Algorithm 2 returns true *because* $status(v_0) = \mathsf{sat}$. Observe that, for every node v of G with $status(v) = \mathsf{sat}$, we have that:

- either no rule is applicable to v and v is an and-node (by step 6 of Algorithm 2)
- or v is an or-node and some child u gets the status sat (by step 8 of procedure `update-status`) before v does
- or v is an and-node and each child u of v gets the status sat (by step 11 of procedure `update-status`) before v does.

The (labels of the) nodes to which no rule is applicable are clearly satisfiable since such labels cannot contain $\bot$, and consist only of atomic concepts and negated atomic concepts, with no contradictory pair $\{A, \neg A\}$. These are the base cases so let us exclude these nodes from further consideration and proceed directly by induction on the temporal order in which the remaining nodes of G which get status sat do so. Each of these remaining nodes v must be either an and-node or an or-node as outlined above. Thus each child u of v with $status(u) = \mathsf{sat}$ is either covered by the base case, or falls under the induction hypothesis, meaning that the label of u must be satisfiable.

It is easy to check that for each of our or-rules, the parent (label) is satisfiable if at least one child (label) is satisfiable, and the parent (label) of our and-rule is satisfiable if all its children (labels) are satisfiable. Thus the label of each such v (with $status(v) = \mathsf{sat}$) is satisfiable. Since v_0 with label $X \cup \mathcal{T}$ is such a v, it follows that X is satisfiable w.r.t. $\mathcal{T}$.

Case Algorithm 2 returns true *because* $status(v_0) = \mathsf{expanded}$. Since $status(v_0) \notin \{\mathsf{sat}, \mathsf{unsat}\}$ then G is actually a (full) and-or graph and every node of G has a status expanded, sat or unsat.

We construct a consistent marking G^c of G as follows. We initialize G^c with the node $v_0 \in G$ and repeatedly, for every node v of G^c, we add $w \in G$ and the edge $(v, w) \in G$ to G^c if w is a child of v with $status(w) \neq \mathsf{unsat}$. Observe that, for every node v of G with $status(v) \neq \mathsf{unsat}$:

- If v is an and-node then all children of v also have a status different from unsat. For a contradiction, suppose that a child u of v has status unsat. When the status of u was changed from expanded to unsat by calling `update-status`(u)

(at line 27 of Algorithm 2 or at line 3 of procedure `propagate-status`), the subsequent call to procedure `propagate-status`(u) (at line 28 of Algorithm 2 or at line 5 of procedure `propagate-status`, respectively) would have set *status*(v) to unsat, contradicting the assumption that *status*(v) $\neq$ unsat.

– If v is an or-node then at least one child of v has a status different from unsat. For a contradiction, suppose that all children of v have status unsat. Since v is an or-node, it must have at least one child, and some child u must be the last child to get the status unsat. When the status of u was changed from expanded to unsat by calling `update-status`(u) (at line 27 of Algorithm 2 or at line 3 of procedure `propagate-status`), all other children of v already had status unsat. The subsequent call to `propagate-status`(u) (at line 28 of Algorithm 2 or at line 5 of procedure `propagate-status`, respectively), must have changed *status*(v) to unsat, which contradicts the assumption that *status*(v) $\neq$ unsat.

It follows that G^c is a marking of G. Clearly, the node with label $\{\bot\}$, if it exists in G, has status unsat and does not belong to G^c. Therefore G^c is a consistent marking of G. By Theorem 3.5, it follows that X is satisfiable w.r.t. $\mathcal{T}$.

We have shown that if *status*(v_0) $\in$ {sat, expanded} then X is satisfiable w.r.t. $\mathcal{T}$. That is, if the algorithm returns *true* then X is satisfiable w.r.t. $\mathcal{T}$. We have also shown that if *status*(v_0) = unsat then X is unsatisfiable w.r.t. $\mathcal{T}$. That is, if the algorithm returns *false* then X is unsatisfiable w.r.t. $\mathcal{T}$. Thus, Algorithm 2 is a decision procedure for checking satisfiability of X w.r.t. $\mathcal{T}$. By Lemma 6.1, the algorithm runs in $2^{O(n)}$ steps. □

7 Further Possible Optimisations

We now discuss some further possible optimizations without proving their correctness.

Algorithm 2 adopts a very basic type of backward propagation of sat and unsat through the and-or graph. Other kinds of propagation can also be applied, including global subset-checking of unsat (see, e.g., [7, 14, 28]), local subset-checking of unsat and local superset-checking of sat for parent nodes and sibling nodes [28]. To maximize propagation of unsat, when a node v gets status unsat, one can try to identify a minimal inconsistent subset of the label of v called an unsat-core of v. Computation of unsat-cores can also be done by backward propagation [14, 28]. In our experience, propagation of sat and unsat when used together with cutoffs usually reduces the search space significantly. The general idea of cutoffs is that a node should be expanded only when it may affect the status of the root of the and-or graph [14]. See [28] for a specific heuristic used for doing cutoffs.

Treating axioms of the TBox as concepts representing global assumptions is not efficient because it generates too many expansions with or-branching. A solution for this problem is to use "absorption" techniques. A basic kind of absorption is "lazy unfolding" for acyclic TBoxes.[4] For the case when the TBox is acyclic and

[4] If A is defined by $A = C$ or $A \sqsubseteq C$, and B occurs in C, then A *directly depends* on B. Let "depend" be the transitive closure of "directly depend". If in a TBox $\mathcal{T}$ no atomic concept depends on itself, then $\mathcal{T}$ is acyclic. For simplicity, we assume that each atomic concept is defined at most once.

consists of only concept definitions of the form $A = C$, by using lazy unfolding, A is treated as a reference to C and will be "unfolded" only when necessary. For the case when the TBox is acyclic and also contains concept inclusions of the form $A \sqsubseteq C$, a simple solution can be adopted: treat $A \sqsubseteq C$ as $A = (C \sqcap A')$ for a new atomic concept A'. For the case when the TBox is cyclic, one can try to divide the TBox into two parts $\mathcal{T}_1$ and $\mathcal{T}_2$, where $\mathcal{T}_1$ is a maximal acyclic sub-TBox "not depending" on the concepts defined in $\mathcal{T}_2$, then one can apply the mentioned replacing and lazy unfolding techniques for $\mathcal{T}_1$.

Also note that various search strategies can be used for expanding the and-or graph, including depth-first search and heuristic search [14, 28].

For further optimisation techniques for tableau methods, see [7, 14, 22, 28].

8 Comparisons With Other Work

In [6], Ding and Haarslev studied tableau caching for description logics with inverse and transitive roles. As expected, caching improves the performance for these description logics. The authors gave some sufficient conditions that guarantee soundness of caching. When used for $\mathcal{ALC}$, those conditions are too restrictive, while we do not require any condition for soundness of caching. Our global caching method can be adapted in a sound way for description logics with inverse [18, 30] and transitive roles [13, 31].

Recall that the algorithm given by Donini and Massacci [7] permanently caches "*all and only unsatisfiable sets of concepts*" and temporarily caches visited nodes on the current branch, even though this means that *"many potentially satisfiable sets of concepts are discarded when passing from a branch to another branch"* [7, Page 126]. This is usually known as "mixed caching".

To illustrate the difference between mixed caching and global caching, consider the TBox $\mathcal{T} = \{A \sqsubseteq C^\top\}$ and suppose $D_1^\perp, \ldots, D_k^\perp, D^\top, C^\top$ are complex concepts not containing A which are independent of each other. Suppose it is easy to show that each of $D_1^\perp, \ldots, D_k^\perp$ is unsatisfiable w.r.t. the TBox $\mathcal{T}$, and also easy to show that $D^\top$ is satisfiable w.r.t. $\mathcal{T}$, but that it is very costly to show that $C^\top$ is satisfiable w.r.t. $\mathcal{T}$. Now suppose we have to check the satisfiability w.r.t. $\mathcal{T}$ of the concept below:

$$(\exists R.A \sqcap \exists R.D_1^\perp) \sqcup \ldots \sqcup (\exists R.A \sqcap \exists R.D_k^\perp) \sqcup D^\top$$

A costly node with label $\{A, C^\top\}$ is explored (and declared to be sat) by our algorithm only once, while it is explored by the algorithm of Donini and Massacci k times.

More generally, take

$$\mathcal{T}' = \{A \sqsubseteq (\exists R.A' \sqcap \exists R.D_1'^\perp) \sqcup \ldots \sqcup (\exists R.A' \sqcap \exists R.D_k'^\perp) \sqcup D'^\top, A' \sqsubseteq C'^\top\}$$

and assume that: $D_1'^\perp, \ldots, D_k'^\perp, D'^\top, C'^\top$ do not contain A and are independent from each other and independent from A'; and it is easy to show that each of $D_1'^\perp, \ldots, D_k'^\perp$ are unsatisfiable and $D'^\top$ is satisfiable w.r.t. $\mathcal{T}'$; but it is very costly to show that $C'^\top$ is satisfiable w.r.t. $\mathcal{T}'$. Now the algorithm of Donini and Massacci explores the costly node k^2 times, while our algorithm explores it only once. Clearly this example can be generalised further to make the difference even worse for mixed caching.

If we compare the DFS algorithm of Donini and Massacci using mixed caching and without any optimisations with a DFS version of our algorithm, then it is easy to see that our algorithm never explores more nodes since global caching subsumes mixed caching.

Of course, there is no such thing as a free lunch: our method may require significantly more memory than traditional methods based upon runs since each new run can reclaim the memory used by the previous runs. Efficient memory management like the one used in [28] is therefore necessary for global caching. More advanced methods of memory management can also be developed.

Recent experimental results by Goré and Postniece also show that global caching is indeed competitive with mixed caching for $\mathcal{ALC}$ [16].

9 Conclusions

We have shown that global caching can indeed be formalised in the description of tableau algorithms in a sound and easy-to-implement manner to give an ExpTime (complexity-optimal) algorithm for checking satisfiability w.r.t. a TBox in $\mathcal{ALC}$. Furthermore, our method is not restricted to depth-first search, can be implemented with various optimisation techniques [28], extends easily to tableau calculi for many other logics [8, 11, 13, 15, 17–19, 29–37], in particular, for dealing with inverse roles or converse modal operators without using cuts [18, 19, 29–31] and for checking consistency of an ABox w.r.t. a TBox [30–32, 35–37]. It gives a general method for obtaining ExpTime (complexity-optimal) tableau algorithms when complexity-suboptimality is caused by exploring the same node on multiple branches.

Acknowledgements We thank three anonymous reviewers for pointing out many improvements to the initial version of this paper.

References

1. Baader, F., Buchheit, M., Hollunder, B.: Cardinality restrictions on concepts. Artif. Intell. **88**(1–2), 195–213 (1996)
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. CUP (2003)
3. Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Stud. Log. **69**, 5–40 (2001)
4. Beth, E.W.: On Padoa's method in the theory of definition. Indag. Math. **15**, 330–339 (1953)
5. De Giacomo, G., Donini, F.M., Massacci, F.: Exptime tableaux for ALC. In: Proc. of Description Logics'1996. AAAI Tech. Report, vol. WS-96-05, pp. 107–110. AAAI Press (1996)
6. Ding, Y., Haarslev, V.: Tableau caching for description logics with inverse and transitive roles. In: Proc. DL'2006, pp. 143–149 (2006)
7. Donini, F., Massacci, F.: EXPTIME tableaux for $\mathcal{ALC}$. Artif. Intell. **124**, 87–138 (2000)
8. Dunin-Kęplicz, B., Nguyen, L.A., Szałas, A.: Converse-PDL with regular inclusion axioms: a framework for MAS logics. J. Appl. Non-Class. Log. **21**(1), 61–81 (2011)
9. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. **18**, 194–211 (1979)
10. Goré, R.: Tableau methods for modal and temporal logics. In: D'Agostino et al. (eds.) Handbook of Tableau Methods, pp. 297–396. Kluwer (1999)
11. Goré, R., Kupke, C., Pattinson, D., Schröder, L.: Global caching for coalgebraic description logics. In: Proc. IJCAR'2010. LNCS, vol. 6173, pp. 46–60. Springer (2010)

12. Goré, R., Nguyen, L.A.: ExpTime tableaux for $\mathcal{ALC}$ using sound global caching. In: Calvanese, D. et al. (eds.) Proc. DL'2007, pp. 299–306 (2007)
13. Goré, R., Nguyen, L.A.: EXPTIME tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In: Olivetti, N. (ed.) Proc. TABLEAUX'2007. LNCS, vol. 4548, pp. 133–148. Springer (2007)
14. Goré, R., Nguyen, L.A.: Optimised ExpTime tableaux for $\mathcal{ALC}$ using sound global caching, propagation and cutoffs. Manuscript, http://www.mimuw.edu.pl/~nguyen/GoreNguyenALC.pdf (2007). Accessed 2 July 2007
15. Goré, R., Nguyen, L.A.: Analytic cut-free tableaux for regular modal logics of agent beliefs. In: Sadri, F., Satoh, K. (eds.) Proc. CLIMA VIII. LNAI, vol. 5056, pp. 268–287. Springer (2008)
16. Goré, R., Postniece, L.: An experimental evaluation of global caching for ALC (system description). In: Baumgartner, P. (ed.) Proc. IJCAR'2008. LNCS, vol. 5195, pp. 299–305. Springer (2008)
17. Goré, R., Widmann, F.: An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability. In: Schmidt, R.A. (ed.) Proc. CADE'2009. LNAI, vol. 5663, pp. 437–452. Springer (2009)
18. Goré, R., Widmann, F.: Sound global state caching for ALC with inverse roles. In: Giese, M., Waaler, A. (eds.) Proc. TABLEAUX'2009. LNAI, vol. 5607, pp. 205–219. Springer (2009)
19. Goré, R., Widmann, F.: Optimal and cut-free tableaux for propositional dynamic logic with converse. In: Proc. IJCAR'2010. LNCS, vol. 6173, pp. 225–239. Springer (2010)
20. Haarslav, V., Möller, R.: Consistency testing: the RACE experience. In: Dyckhoff, R. (ed) Proc. TABLEAUX'2000. LNCS, vol. 1847, pp. 57–61. Springer (2000)
21. Heuerding, A., Seyfried, M., Zimmermann, H.: Efficient loop-check for backward proof search in some non-classical logics. In: Proc. TABLEAUX'1996. LNAI, vol. 1071, pp. 210–225. Springer (1996)
22. Horrocks, I., Patel-Schneider, P.F.: Optimizing description logic subsumption. J. Log. Comput. **9**(3), 267–293 (1999)
23. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. Log. Comput. **9**(3), 385–410 (1999)
24. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for very expressive description logics. Logic Journal of the IGPL **8**(3), 239–263 (2000)
25. Schaerf, A., Buchheit, M., Donini, F.M.: Decidable reasoning in terminological knowledge representation systems. J. Artif. Intell. Res. **1**(1–2), 109–138 (1993)
26. Motik, B., Shearer, R., Horrocks, I.: Optimized reasoning in description logics using hypertableaux. In: Pfenning, F. (ed.) Proc. of CADE-21. LNCS, vol. 4603, pp. 67–83. Springer (2007)
27. Nguyen, L.A.: Analytic tableau systems and interpolation for the modal logics KB, KDB, K5, KD5. Stud. Log. **69**(1), 41–57 (2001)
28. Nguyen, L.A.: An efficient tableau prover using global caching for the description logic ALC. Fundam. Inform. **93**(1–3), 273–288 (2009)
29. Nguyen, L.A.: A cut-free exptime tableau decision procedure for the logic extending converse-PDL with regular inclusion axioms. CoRR, abs/1104.0405 (2011). Accessed 3 April 2011
30. Nguyen, L.A.: Cut-free exptime tableaux for checking satisfiability of a knowledge base in the description logic ALCI. In: Proc. ISMIS'2011. LNCS, vol. 6804, pp. 465–475. Springer (2011)
31. Nguyen, L.A.: A cut-free exptime tableau decision procedure for the description logic SHI. In: Proc. ICCCI'2011. LNAI, vol. 6922, pp. 572–581. Springer (2011). (see http://arxiv.org/abs/1106.2305 for a long version)
32. Nguyen, L.A., Szałas, A.: EXPTIME tableaux for checking satisfiability of a knowledge base in the description logic ALC. In: Nguyen, N.T., Kowalczyk, R., Chen, S.-M. (eds.) Proc. ICCCI'2009. LNAI, vol. 5796, pp. 437–448. Springer (2009)
33. Nguyen, L.A., Szałas, A.: An optimal tableau decision procedure for Converse-PDL. In: Nguyen, N.-T., Bui, T.-D., Szczerbicki, E., Nguyen, N.-B. (eds.) Proc. KSE'2009, pp. 207–214. IEEE Computer Society (2009)
34. Nguyen, L.A., Szałas, A.: A tableau calculus for regular grammar logics with converse. In: Schmidt, R.A. (ed.) Proc. CADE-22. LNAI, vol. 5663, pp. 421–436. Springer-Verlag (2009)
35. Nguyen, L.A., Szałas, A.: Checking consistency of an ABox w.r.t. global assumptions in PDL. Fundam. Inform. **102**(1), 97–113 (2010)
36. Nguyen, L.A., Szałas, A.: Tableaux with global caching for checking satisfiability of a knowledge base in the description logic SH. Transactions on Computational Collective Intelligence **1**, 21–38 (2010)

37. Nguyen, L.A., Szałas, A.: ExpTime tableau decision procedures for regular grammar logics with converse. Stud. Log. **98**(3), 387–428 (2011)
38. Pratt, V.R.: A near-optimal method for reasoning about action. J. Comput. Syst. Sci. **20**(2), 231–254 (1980)
39. Rautenberg, W.: Modal tableau calculi and interpolation. J. Philos. Logic **12**, 403–423 (1983)
40. Schild, K.: A correspondence theory for terminological logics: preliminary report. In: Mylopoulos, J., Reiter, R. (eds.) Proc. IJCAI'1991, pp. 466–471. Morgan Kaufmann (1991)