Artificial Intelligence 124 (2000) 87–138

Artificial Intelligence

www.elsevier.com/locate/artint

EXPTIME tableaux for $\mathcal{ALC}$

Francesco M. Donini [a,c], Fabio Massacci [b,c,*]

[a] *Dip. di Elettrotecnica ed Elettronica—Politecnico di Bari, Via G. Re David, 200, 70125 Bari, Italy*
[b] *Dip. Ingegneria dell'Informazione—Università degli Studi di Siena, Via Roma 56, 53100 Siena, Italy*
[c] *Dip. di Informatica e Sistemistica—Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy*

Received 10 December 1999

Abstract

The last years have seen two major advances in Knowledge Representation and Reasoning. First, many interesting problems (ranging from Semi-structured Data to Linguistics) were shown to be expressible in logics whose main deductive problems are EXPTIME-complete. Second, experiments in automated reasoning have substantially broadened the meaning of "practical tractability". Instances of realistic size for PSPACE-complete problems are now within reach for implemented systems.

Still, there is a gap between the reasoning services needed by the expressive logics mentioned above and those provided by the current systems. Indeed, the algorithms based on tree-automata, which are used to prove EXPTIME-completeness, require exponential time and space even in simple cases. On the other hand, current algorithms based on tableau methods can take advantage of such cases, but require double exponential time in the worst case.

We propose a tableau calculus for the description logic $\mathcal{ALC}$ for checking the satisfiability of a concept with respect to a TBox with general axioms, and transform it into the first simple tableau-based decision procedure working in single exponential time.

To guarantee the ease of implementation, we also discuss the effects that optimizations (propositional backjumping, simplification, semantic branching, etc.) might have on our complexity result, and introduce a few optimizations ourselves. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Automated reasoning; Tableaux; Description logics; Modal logics; Algorithms; Computational complexity

* Corresponding author.
E-mail addresses: donini@poliba.it (F.M. Donini), massacci@dis.uniroma1.it (F. Massacci).

1. Introduction and motivations

Automated Reasoning has been an active research area in the mainstream of Artificial Intelligence in the last decades and has received special attention as a tool to provide "reasoning services" for Knowledge Representation and Reasoning (KR&R) [53]. That is, when knowledge about a problem is coded as formulae in a particular logic, *automated reasoning services* about such knowledge can be offered to a more complex system—in more or less the same way implemented data structures offer methods to update the structure itself [52]. Of course, to make reliable such a service, some bounds on the resources needed to solve the reasoning problem associated to the service should be given. This implies that the reasoning problem should be at least decidable; but more pragmatically, the service should be accomplished within the resource bounds available to the current technology—which is sometimes mentioned as "tractability".

Traditionally, "tractability" has been largely understood as polynomial-time solvability of the reasoning problems. This resulted in a tremendous effort on isolating the so-called *tractable fragments* of various logics for knowledge representation [21,53,60,61,67].

The last years have seen a shift from this paradigm, at least for two reasons.

First, logics with polynomial-time deductive problems have been criticized for their too limited expressive power [23]. Although systems with a limited-but-reliable KR&R component have been successfully used in complex applications [7,19,79], it is now recognized that many interesting KR&R problems can be expressed only in logics whose main deductive problems—satisfiability and logical implication—are EXPTIME-complete. This is true especially for Description Logics, in which many problems, like reasoning in conceptual data models, schema integration and semi-structured data [9,10], are expressible in EXPTIME-complete Description Logics. The need is also true for logics very similar to Description Logics, such as Modal Logics [25,29], Propositional Dynamic Logics [27,49], Temporal Logics for Computer Aided Verification [24,78], Hybrid Logics for Linguistics [6], Security Modal Logics [1,35].

Second, a number of recent experimental advances in description and modal logics theorem proving and satisfiability checking have substantially broadened the meaning of "tractability" for practical purposes. Better algorithms, better coding, better computer technology have brought forward a number of novel and effective systems for modal and description logics based on on different calculi and implementations, ranging from tableau and constraint systems [41,43,44] to DPLL-based implementations [32,34] and first order theorem provers [48]. Problems such as as concept satisfiability or modal provability, which are PSPACE-complete problems in the worst case, are now within reach for instances of realistic size [58]. Potentially EXPTIME-complete problems stemming from real applications can be also be reasonably solved [30,41,45].

1.1. The problem

Notwithstanding the impressive amount of theoretical and experimental work, there is still a gap between the reasoning services for the expressive logics mentioned above and those provided by the currently implemented systems. Indeed, although many systems such as FaCT and DLP [41,43,64] provide reasoning services for problems in PSPACE (e.g., concept satisfiability) and in EXPTIME (e.g., subsumption with respect to TBoxes, validity for propositional dynamic logic), it is only for problems in PSPACE that the offered reasoning services are proved to guarantee the computational complexity upper bound [45].

For sake of concreteness, in this paper we concentrate on a reasoning technique for a particular Description Logic $\mathcal{ALC}$ [22,70], which is a subset of nearly every expressive Description Logic. However, we remark that $\mathcal{ALC}$ is merely a notational variant of the multi-modal logic K [38], and, with simple adaptations, one obtains Propositional Dynamic Logics (without converse) and Temporal Logics.

From correspondences with Propositional Dynamic Logic (PDL) it is known that the satisfiability of an $\mathcal{ALC}$ concept C with respect to a TBox KB is in EXPTIME [65,77]. However, the algorithms directly derived from the tree-automata methods, which have been used to prove such a result, require exponential time and space even in simple cases, e.g., when a simple model satisfying both KB and C can be easily found. Loosely speaking, this happens because one first constructs an automaton which accepts the tree models of *KB* and C, and whose size is exponential in the size of the input. Then, one checks its emptiness, i.e., whether the automaton accepts no model [13,24,31]. The construction of efficient methods for testing emptiness—i.e., unsatisfiability—on-the-fly while constructing a tree automata, is still an active area of research. The problem has been solved only for PSPACE problems such as satisfiability of linear temporal logic [13,31].

In contrast, proposed tableau methods [8], which explore a space of candidate models for *KB* and C starting from simple ones, can take advantage of such cases. However, there can be an exponential number of possibly exponential-size candidate models. Hence, a straightforward implementation based on tableaux requires doubly exponential time in the worst case.

As a solution to this problem, many implemented $\mathcal{ALC}$ satisfiability solvers “cache” the (un)satisfiability status of sets of concepts for later use [37,43]. This optimization 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.

Indeed, a practical and easily implementable algorithm, whose correctness and complexity have been formally proven, has not been given in the literature. This topic is only quickly discussed in [8] and the transformation of a tableau calculus into an EXPTIME algorithm is only sketched in [16,20].

1.2. The contribution of this paper

We devise a refined tableau calculus that neatly formalizes and integrates the techniques used by $\mathcal{ALC}$ solvers with the theoretical work on PDL with tableaux, thus achieving the first simple tableau-based procedure working in single exponential time for the satisfiability of an $\mathcal{ALC}$ concept C with respect to a TBox *KB* containing general axioms of the form $C \sqsubseteq D$.

This result is particularly important as it shows that it is possible to exploit a search-based technique, with the possibility of finding rapid solutions for “easy problems”, without sacrificing worst-case complexity.

In a nutshell, traditional tableau methods close a branch only by "first principles" (atomic clashes), whereas our enhanced tableaux exploit previously proved inconsistencies as additional lemmata to decide that a branch can be closed, without delving into the same formulae only to find the same atomic clashes again and again.

We remark that, in a realistic setting, several satisfiability tests are posed to the same TBox [3]. Even if most concepts usually turn out to be satisfiable, the reasoning process may analyze many inconsistent sets of concepts. These lemmata are related to the TBox; they are independent on the particular concept being tested, and can therefore be exploited for answering subsequent tests. Hence, their use not only speeds up a single deduction, but the cost of memorizing them can be amortized over several tests.

Once correctly formalized, our tableau calculus is very simple—even though the complexity proof is far from simple—and could be easily implemented in existing top-down tableau-based systems, such as FaCT [41,42], KSAT [32] or DLP [43,64]. Indeed, we can see a further contribution of this paper in the actual distillation and formalization of many intuitions present in current implementations.

Our techniques could also be extended to translation-based approaches *à la* Ohlbach [62], using the simulation techniques developed by Hustadt and Schmidt [47] which makes it possible to simulate the actual proof search with prefixed tableaux *à la* Massacci [54,57] in a first-order resolution framework.

To guarantee the ease of implementation, we have taken particular care in discussing the effects that the various optimizations (propositional backjumping, simplification, semantic branching, etc.) might have on our complexity results, and how our results can be lifted to more expressive logics beyond $\mathcal{ALC}$. Moreover, we introduced a few optimizations ourselves (e.g., modal backjumping).

1.3. Plan of the paper

In the next section we introduce the syntax and the semantics of the description logic $\mathcal{ALC}$. Then, we present the general principles of an innovative tableau calculus for $\mathcal{ALC}$ (Section 3) and sketch how to lift it to more expressive logics. We show how to transform it into an EXPTIME efficient algorithm (Section 4) and discuss the incorporation of many state-of-the-art optimizations into the basic algorithm.

We also prove the correctness of the tableau calculus (Section 5) and the required exponential complexity bounds on the algorithm (Section 6). Finally we review some related approaches (Section 7) and conclude (Section 8).

The reader interested in the practical working of the algorithm can find in the appendix a worked example.

2. Notation and semantics

Let A denote a concept name, C and D arbitrary concepts, and R a role name. Concepts in $\mathcal{ALC}$ are formed with the following syntax:

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

In the sequel we denote sets of concepts by the calligraphic letters $\mathcal{C}$ and $\mathcal{D}$.

α	α_1	α_2
$C \sqcap D$	C	D
$\neg(C \sqcup D)$	$\neg C$	$\neg D$

β	β_1	β_2
$\neg(C \sqcap D)$	$\neg C$	$\neg D$
$C \sqcup D$	C	D

neg	*pos*
$\neg\neg C$	C

$\nu(R)$	ν_0
$\forall R.C$	C
$\neg\exists R.C$	$\neg C$

$\pi(R)$	π_0
$\neg\forall R.C$	$\neg C$
$\exists R.C$	C

Fig. 1. Uniform notation of concept expressions.

A TBox KB is a set containing *inclusions* of the form $C \sqsubseteq D$. We do not impose any constraint on the form of these inclusions.

For sake of modularity, we classify concepts with the types α, β introduced by Smullyan [73] in the 1960s, and extended to modal logic by Fitting [28]. This classification is recalled in Fig. 1.

Definition 2.1. If A is a concept name, we refer to formulae of the form A, $\neg A$, $\nu(R)$ and $\pi(R)$ as *modal atoms*.

The intuition is that modal atoms cannot be reduced further by simple propositional rules.

2.1. Semantics

An *interpretation* $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ consists of a non-empty set Δ, the *domain* of $\mathcal{I}$—whose members are called *elements*—and a function $\cdot^{\mathcal{I}}$, the *interpretation function* of $\mathcal{I}$, that maps every concept to a subset of Δ and every role to a subset of $\Delta \times \Delta$ such that

$$\begin{aligned}
&\top^{\mathcal{I}} = \Delta,\\
&\bot^{\mathcal{I}} = \emptyset,\\
&(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}},\\
&(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}},\\
&(\neg C)^{\mathcal{I}} = \Delta \setminus C^{\mathcal{I}},\\
&(\forall R.C)^{\mathcal{I}} = \bigl\{a \in \Delta \mid \forall b.\, (a,b) \in R^{\mathcal{I}} \text{ implies } b \in C^{\mathcal{I}}\bigr\},\\
&(\exists R.C)^{\mathcal{I}} = \bigl\{a \in \Delta \mid \exists b.\, (a,b) \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\bigr\}.
\end{aligned}$$

According to this definition, an interpretation function is completely determined by the way it interprets atomic concepts and roles.

An interpretation $\mathcal{I}$ *satisfies a concept* C if there exists an element $d \in \Delta$ such that $d \in C^{\mathcal{I}}$, i.e., if $C^{\mathcal{I}} \neq \emptyset$. An interpretation *validates a concept* C if for every element $d \in \Delta$ one has $d \in C^{\mathcal{I}}$, that is, if $C^{\mathcal{I}} = \Delta$.

An interpretation $\mathcal{I}$ *validates* the inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. In other words, this interpretation validates the concept $\neg C \sqcup D$.

A TBox is a finite set of inclusions. An interpretation $\mathcal{I}$ is a *model* for a TBox KB if $\mathcal{I}$ validates all inclusions in KB. We say that a TBox KB *entails* the inclusion $C \sqsubseteq D$, written $KB \models C \sqsubseteq D$, if every model of KB validates the inclusion $C \sqsubseteq D$. A concept C is satisfiable with respect to a TBox KB if there is a model $\mathcal{I}$ of KB such that $\mathcal{I}$ satisfies C.

Observe that satisfiability of a concept with respect to a TBox can solve also other problems such as entailment. For instance, KB entails $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable with respect to *KB*.

3. A tableau calculus for $\mathcal{ALC}$

Our calculus combines features of Gentzen-type tableaux, prefixed tableaux, and constraint systems. Below we give (to the reader who is familiar with these calculi) some intuitive reasons why we need a calculus which is more general than each one of them.

- Gentzen-type tableaux [28,36] provide too rough a level of granularity, since they implicitly work with sets of concepts, but concentrate on one element at the time, and there is no way to distinguish different elements linked to the same set of concepts.
- Prefixed tableaux [28,36,54,57] and constraint systems [8] provide too fine a level of granularity. They give names to elements, but they do not immediately identify the whole set of concepts linked to an element.
- Graph-based systems, from Kripke's original proposal [50], to more recent incarnations in modal logics [11,26] and description logics [44,45] allow to identify the sets of concepts linked to a node. However, since they represent only a single tentative model in a tableau, and employ nondeterministic rules, they shift the problem of obtaining a deterministic algorithm into the implementation. Therefore they are suited only for proving decidability [26,44,45], or PSPACE-completeness, by exploiting Savitch's Theorem that PSPACE is closed under nondeterminism [45].
- Loop checking [18,36,39,54,57], also known as blocking [8,44,45] and filtering [36], is only useful for "reusing" satisfiable concepts, namely the concepts that can be used to build a (partial) model.
- Standard algorithms do not learn from local proofs of unsatisfiability. Indeed, even if caching of unsatisfiable concepts is often claimed by many implementors [37, 41,43,64], this feature is left out of the formal description of the calculus and the algorithm. Therefore, it is impossible to prove its correctness or its complexity. In contrast, we must *formally re-use parts of the computations* both for satisfiability and unsatisfiability.

In a nutshell, the intuitions behind our system are close to those commonly described in the recent modal and description logic literature. What we have done is to actually distil and formalize the different fragments into a coherent and effective whole.

3.1. Tableau rules

The basic component of our calculus is a pair $e : \mathcal{C}$, composed by a *prefix* e and a *set of concepts* $\mathcal{C}$. We call this pair a *prefixed set*. A prefix is an alternating sequence of integers and role names, starting with 1. Formally, if R is a role name and n an integer we have that the syntax of prefixes is $e ::= 1 \mid e.R.n$. For example, if P and Q are roles, $1.P.5.Q.11.P.12$ is a prefix, and $1.Q.8.R.9 : \{A, B \sqcup \forall R.C\}$ is a prefixed set.

Intuitively, a prefix e in a pair $e : \mathcal{C}$ is a name for an element in a domain of a model the calculus tries to build. If a model is actually built, e satisfies all concepts contained in the set $\mathcal{C}$.

To define a tableau, we adapt from Smullyan's book [72, pp. 24 and 29] and use the uniform notation defined in Fig. 1 to avoid any preliminary reduction to negation normal form.

A *tableau for a concept C* is an ordered dyadic tree, whose points are (occurrences of) prefixed sets, which is constructed as follows. We start by placing $1 : \{C\}$ at the root. Now suppose $\mathcal{T}$ is a tableau for C which has already been constructed; let $\mathcal{B}$ be a *branch* in $\mathcal{T}$, i.e., a path from the root to a leaf. Then we may extend $\mathcal{T}$ by using the rules in Fig. 2 as follows: if the antecedents of a rule appear along $\mathcal{B}$, we add to $\mathcal{B}$ the consequent(s) of the rule. For the β rule, we simultaneously adjoin the two consequents as the left successor and the right successor of the leaf. Notice that the antecedent of a rule needs not be a leaf in the tree as in [41].

We call the rules in Fig. 2 *Prefixed Set rules* (PS-rules for short).

We say that *a rule is applied to a prefixed set* $e : \mathcal{C}$ if $e : \mathcal{C}$ is the antecedent of the rule being applied. For the $\nu(R)$ rule, which has two antecedents, we say that the rule is applied to the first antecedent (the prefixed set $e : \mathcal{C} \cup \{\nu(R)\}$).

$$\frac{e : \mathcal{C} \cup \{\alpha\}}{e : \mathcal{C} \cup \{\alpha_1, \alpha_2\}}(\alpha) \qquad \frac{e : \mathcal{C} \cup \{neg\}}{e : \mathcal{C} \cup \{pos\}}(dneg)$$

$$\frac{e : \mathcal{C} \cup \{\beta\}}{e : \mathcal{C} \cup \{\beta_1\} \qquad e : \mathcal{C} \cup \{\beta_2\}}(\beta)$$

$$\frac{e : \mathcal{C}}{e : \mathcal{C} \cup \{\neg C \sqcup D\}}(KB) \quad \text{where } C \sqsubseteq D \in KB$$

$$\frac{e : \mathcal{C} \cup \{\pi(R)\}}{e.R.n : \{\pi_0\}}\pi(R) \quad \text{where } e.R.n \text{ is new in the tableau } \mathcal{T}$$

$$\frac{e : \mathcal{C} \cup \{\nu(R)\} \qquad e.R.n : \mathcal{D}}{e.R.n : \mathcal{D} \cup \{\nu_0\}}\nu(R)$$

Fig. 2. Prefixed Set rules.

Regarding rule $\pi(R)$, a prefix is *present* in a tableau $\mathcal{T}$, if there is a prefixed set in $\mathcal{T}$ with that prefix, and it is *new* if it is not present.

The same rule might be applicable to different prefixed sets, due to the presence in the sets of the very same concept. For example, starting the tableau from $1 : \{A \sqcap B, C \sqcup D\}$, and applying the α rule, the prefixed set $1 : \{A, B, C \sqcup D\}$ is added to the tableau as a successor of the root. Now the β rule is applicable both to the root and to its successor; but obviously, there is no point in applying the rule to the root, since it contains a concept already decomposed. To prevent this ambiguity, we impose the following restriction: *we cannot apply a rule to a prefixed set $e : \mathcal{C}$ if below in the branch there is already a prefixed set $e : \mathcal{C}'$ (with the same prefix).*

Remark 3.1. The condition in $\pi(R)$ that $e.R.n$ is new in $\mathcal{T}$ ensures that each prefix is uniquely identified in $\mathcal{T}$. Moreover, it is easy to see that if an prefix $e.R.n$ is present in $\mathcal{T}$, so is e.

3.2. Reusing computations for satisfiability

We need the preliminary notion of reduced concept:

Definition 3.2. A *concept $C \in \mathcal{C}$ is PS-reduced* for the prefixed set $e : \mathcal{C}$ in the branch $\mathcal{B}$ iff one of the following conditions holds:

(1) if C is of type α, and both α_1 and α_2 are in $\mathcal{C}$;
(2) if C is of type β, and either β_1 or β_2 is in $\mathcal{C}$;
(3) if C is of type *neg*, and *pos* is in $\mathcal{C}$;
(4) if C is of type $\pi(R)$, and there is another prefixed set $e.R.n : \mathcal{D}$ in $\mathcal{B}$ such that $\pi_0 \in \mathcal{D}$;
(5) if C is of type $\nu(R)$, and for all prefixed sets $e.R.n : \mathcal{D}$ present in $\mathcal{B}$ it is $\nu_0 \in \mathcal{D}$.

Moreover, an *inclusion $C \sqsubseteq D \in KB$ is PS-reduced* for the prefixed set $e : \mathcal{C}$ in a branch $\mathcal{B}$ iff $\neg C \sqcup D \in \mathcal{C}$.

If the concept C is reduced according the first three rules, we say that it is *propositionally reduced*. Clearly, modal atoms are propositionally reduced.

Definition 3.3. A *prefixed set $e : \mathcal{C}$* is *PS-reduced* in branch $\mathcal{B}$ if every concept $C \in \mathcal{C}$ is PS-reduced for $e : \mathcal{C}$ and every inclusion $C \sqsubseteq D \in KB$ is PS-reduced for $e : \mathcal{C}$.

Of course we do not need to apply a rule to a concept which is already PS-reduced, nor to consider PS-reduced prefixed sets.

Although this may be sufficient to avoid some useless computations, it is still not enough to provide termination. We need to introduce the notion of implicitly reduced concept and, to this extent, the standard lexicographic ordering for prefixes. This is obtained by taking the transitive (but not reflexive) closure of the following relations:

$$\begin{array}{ll} e \prec e.R.n & \\ e.R.n \prec e.R'.n' & \text{provided } n < n' \\ e.R.e_1 \prec e'.R'.e_2 & \text{provided } e \prec e'. \end{array}$$

For example, $1.R.6 \prec 1.Q.11$, and $1.R.12.Q.20 \prec 1.Q.15.R.16$. For simplicity, we keep the same symbol $\prec$ for the transitive closure of the above relations.

To identify loops in a branch, we define the following notion of *witness*.

Definition 3.4. A prefixed set $e : \mathcal{C}$ is a *witness in a branch* $\mathcal{B}$ for a prefixed set $e' : \mathcal{C}$, if $e \prec e'$, $e : \mathcal{C}$ is PS-reduced, and there is no other prefixed set $e'' : \mathcal{C}$ in $\mathcal{B}$ such that $e'' \prec e$.

In the above definition $\mathcal{C}$ is the same in both prefixed sets $e : \mathcal{C}$ and $e' : \mathcal{C}$. Note also that if $e : \mathcal{C}$ has a witness, then it is reduced with respect to points (1)–(3) in the above Definition 3.2. Since the $\prec$ relation is well-founded, there is at most one witness for a given set $\mathcal{C}$. Clearly, there is no point in reducing further a prefixed set which has a witness.

Definition 3.5. A prefixed set $e : \mathcal{C}$ is *implicitly PS-reduced* in a branch $\mathcal{B}$ iff either it is PS-reduced or it has a witness $e' : \mathcal{C}$ in $\mathcal{B}$.

3.3. Reusing computations for unsatisfiability

We start by recalling the usual definition of explicit inconsistency.

Definition 3.6. A prefixed set $e : \mathcal{C}$ is *inconsistent* if there is a concept C such that both $C \in \mathcal{C}$ and $\neg C \in \mathcal{C}$. For a given concept C, we call *clash* the set $\{C, \neg C\}$.

To re-use unsatisfiable sets of concepts we introduce a special kind of prefixed sets called *inconsistent sets* ($\perp$-sets for short), denoted as $e : (\mathcal{C})^{\perp}$.

The semantic interpretation of $e : (\mathcal{C})^{\perp}$is the logical implication $KB \models \sqcap\mathcal{C} \sqsubseteq \perp$, where if $\mathcal{C} = \{C_1, \ldots, C_n\}$ then $\sqcap\mathcal{C} = C_1 \sqcap \cdots \sqcap C_n$. In other words, a $\perp$-set contains a set of concepts whose conjunction has already been proven inconsistent for the given knowledge base KB, during the expansion of the tableau.

The intuition is that $\perp$ is an *extra label* that we may propagate along the current labels of the nodes of the tableau. So we have a series of rules for relabeling, shown in Fig. 3. The rules are applied as follows: if the antecedent prefixed sets occur in the tableau *and* the consequent $e : \mathcal{D}$ occurs also in the tableau, then we relabel the consequent as $e : (\mathcal{D})^{\perp}$.

Without the condition that both antecedents and consequent already appear in the tableau, the rules for the $\perp$-sets would be a parallel calculus to derive the inconsistency of a set of concepts bottom-up rather than top-down, and many (irrelevant) $\perp$-sets could be generated. On the contrary, in this way we can have at most as many $\perp$-sets as “normal” prefixed sets.

For all rules, the consequent appears (unlabelled) in the tableau.

$$\frac{C, \neg C \in \mathcal{C}}{e : (\mathcal{C})^{\perp}}(\perp) \qquad \frac{e' : (\mathcal{C})^{\perp} \qquad e' \prec e}{e : (\mathcal{C})^{\perp}}(\perp\text{-}\mathit{witness})$$

$$\frac{e : (\mathcal{C} \cup \{\alpha_1, \alpha_2\})^{\perp}}{e : (\mathcal{C} \cup \{\alpha\})^{\perp}}(\perp\text{-}\alpha) \qquad \frac{e : (\mathcal{C} \cup \{\mathit{pos}\})^{\perp}}{e : (\mathcal{C} \cup \{\mathit{neg}\})^{\perp}}(\perp\text{-}\mathit{dneg})$$

$$\frac{e : (\mathcal{C} \cup \{\beta_1\})^{\perp} \qquad e : (\mathcal{C} \cup \{\beta_2\})^{\perp}}{e : (\mathcal{C} \cup \{\beta\})^{\perp}}(\perp\text{-}\beta)$$

$$\frac{e : (\mathcal{C} \cup \{\neg C \sqcup D\})^{\perp} \qquad C \sqsubseteq D \in \mathit{KB}}{e : (\mathcal{C})^{\perp}}(\perp\text{-}\mathit{KB})$$

$$\frac{\mathcal{D} \subseteq \{\nu_0 \mid \nu(R) \in \mathcal{C}\} \qquad e.R.n : (\mathcal{D} \cup \{\pi_0\})^{\perp}}{e : (\mathcal{C} \cup \{\pi(R)\})^{\perp}}(\perp\text{-}\pi(R))$$

Fig. 3. Generation of $\perp$-sets.

The $\perp$-rules are the (almost) dual version of the PS-rules. The major difference is that PS-rules are applied within a branch, whereas rules of Fig. 3 can be applied across branches, i.e., antecedents and consequents may appear in different branches of the tableau—cf. in particular the rule $\perp$-*witness*.

Then, we can broaden the definition of an inconsistent prefixed set:

Definition 3.7. A prefixed set $e : \mathcal{C}$ is *implicitly inconsistent* if it is a $\perp$-set.

Later, in Lemma 5.2 we prove that this name is faithful to its meaning, namely, that an implicitly inconsistent set is indeed inconsistent.

Since we have more rules than simply PS-rules, we must also add the notion of $\perp$-reduced prefixed set.

Definition 3.8. A prefixed set $e : (\mathcal{C})^{\perp}$ is $\perp$-*reduced* if every rule of Fig. 3 which can be applied to it does not introduce a new $\perp$-set.

3.4. Tableau proof search

The presence of witnesses and the notion of implicitly inconsistent, PS-reduced, and $\perp$-reduced sets require a novel definition of open and closed branches with respect to the standard tableau definitions.

Definition 3.9. A branch $\mathcal{B}$ is *implicitly closed* if there is an implicitly inconsistent prefixed set in $\mathcal{B}$. A tableau is implicitly closed if all its branches are implicitly closed.

Definition 3.10. A branch $\mathcal{B}$ is *open* if every prefixed set in $\mathcal{B}$ is both implicitly PS-reduced and $\perp$-reduced, and $\mathcal{B}$ is not implicitly closed. A tableau $\mathcal{T}$ is open if at least one branch of $\mathcal{T}$ is open.

Now we have defined all the notions needed to state the first two results (proofs in Section 5).

Theorem 3.11. *If there is an implicitly closed tableau for the concept C with a TBox KB then C is unsatisfiable for KB.*

Theorem 3.12. *If there is an open tableau for the concept C with a TBox KB then C is satisfiable for KB.*

The key problem is how do we find an open or implicitly closed tableau, possibly using single exponential time in the size of KB and C. To this extent we need to apply the following high-level search techniques. For each technique, we give some intuitive rationale.

Technique 1. Never apply a rule to an implicitly PS-reduced prefixed set, nor to a $\perp$-reduced prefixed set.

Regarding implicitly PS-reduced prefixed sets, either they are PS-reduced (and then, no PS-rule can be applied to them), or they have a witness. Observe that following this technique, only the first encountered witness can be properly PS-reduced. In fact, the prefixed set having a witness will not be PS-reduced with respect to conditions (4)–(5) in Definition 3.2.

A $\perp$-reduced prefixed set is inconsistent (we show this formally in Section 5), so there is no point in further expanding it.

We now impose that we do not expand prefixed sets which we have already "proved" to be inconsistent:

Technique 2. Never apply a PS-rule to an implicitly inconsistent prefixed set.

In order to apply Technique 2 as often as possible, saving useless expansions, we need to "discover" $\perp$-sets as soon as possible. Then, we give precedence to the generation of $\perp$-sets:

Technique 3. Apply the rules for the addition of new $\perp$-sets before other rules.

These techniques preserve the soundness and the completeness of the search strategies that apply them, but we need more constraints [1] for proving that the proof search requires single exponential time.

[1] We do not know whether the constraints we impose are minimal or there are alternative ones. For instance, see the work of Demri on non-uniform strategies [18] or Fariñas and Gasquet [26] for a different strategy which guarantees termination.

Technique 4. For every prefixed set $e : \mathcal{C}$, apply rules α, *pos*, *KB* and $\nu(R)$ before other rules, and apply rule β before rule $\pi(R)$.

Technique 5. The new prefix $e.R.n$ generated by rule $\pi(R)$ must be such that $n > m$ for every other integer m already present in the tableau.

Intuitively, this technique simply says that we use a global counter for generating the successors of e: first generate $e.R'.1$, then $e.R''.2$, then $e.R'''.3$ and so on, where R', R'', and R''' may be the same role.

Technique 6. Apply a rule to a prefixed set $e : \mathcal{C}$ only if there is no prefixed set $e' : \mathcal{D}$, with $e' \prec e$, to which a rule can be applied.

Technique 7. Use a depth-first strategy for the traversal of the branches of the tableau.

The combination of Techniques 4 and 6 force the application of a $\nu(R)$ rule just after an application of a $\pi(R)$ rule. That is, just after rule $\pi(R)$ introduces a new prefix, all additional concepts ν_0 imposed by universal formulae $\nu(R)$ are transferred to the newly generated prefix by the application of $\nu(R)$ rule.

We now have the machinery to prove that the search process terminates, using single exponential time in the worst case.

Theorem 3.13. *Any search strategy respecting Techniques* 1–7 *terminates using single deterministic exponential time in the size of* C *and the TBox KB.*

The proof is carried out in Section 6, after making the above calculus and techniques more concrete with the help of a set of algorithms (Section 4).

3.5. Extension to logics beyond $\mathcal{ALC}$

The calculus and the corresponding correctness and complexity results can be easily extended to cope with reflexive and transitive roles (i.e., with modal logics such as T_n, $\mathsf{K4}_n$ and $\mathsf{S4}_n$). We simply need to incorporate the (T) and (4) rule for Single Step Prefixed Tableaux [36,54,57]. In a nutshell we simply need to augment the calculus with the rules:

$$\frac{e : \mathcal{C} \cup \{\nu(R)\}}{e : \mathcal{C} \cup \{\nu(R), \nu_0\}}\nu^T(R) \quad \text{when } R \text{ is a reflexive role,}$$

$$\frac{e : \mathcal{C} \cup \{\nu(R)\} \qquad e.R.n : \mathcal{D}}{e.R.n : \mathcal{D} \cup \{\nu(R)\}}\nu^4(R) \quad \text{when } R \text{ is a transitive role.}$$

The $\nu^T(R)$ rule has the same priority of α and *dneg* rules whereas the $\nu^4(R)$ rule have the same priority of the classical $\nu(R)$ rule.

Role hierarchies, in absence of transitive roles, in the form $P \sqsubseteq Q$ for atomic roles P and Q can also be easily accommodated by using the $\Rightarrow$ rule given in [56]. The combination of role hierarchies and transitive roles requires major modifications to the calculus, because of the interaction between transitive roles and role hierarchies (see [44] for a graph-based calculi).

Our results easily carry over to any extended calculus such that:

- it has the finite superformula property [36], i.e., the reduction of a formula only requires the introduction of formulae which are in a bounded[2] superset of the subformulae of the TBox and the initial concept;
- its rules can be casted as prefixed tableau rules *à la* Fitting [28] (see also [36,54,57]), where each rule involves formulae with either the same prefix (e.g., α, β, *dneg* rules, and $\nu^T(R)$ rules for reflexive roles) or with a longer one (e.g., $\pi(R)$, $\nu(R)$ rules, and $\nu^4(R)$ rules for transitive roles).

The use of converse is more problematic: it requires to go back and forth prefixes. Obviously, we could simply adapt the rules for converse proposed by De Giacomo and Massacci [16]. This would give us a calculus which is sound and complete for $\mathcal{ALC}$ with converse, and in which one only visits a number of prefixed sets which is exponentially bounded by the size of the TBox and the initial concept. However, visiting only an exponential number of concepts is not sufficient when designing a depth-first-search algorithm, because we could visit the same concepts again and again up to a doubly exponential number of recursive calls. What is missing is an easy way to transform the generation of $\perp$-sets sketched in [16] into a simple depth-first-search-style algorithm.

4. An EXPTIME efficient algorithm

We first give a simple version of the algorithm (without any of the standard optimizations adopted by $\mathcal{ALC}$ satisfiability testers), and then progressively explain how it can be enhanced without affecting correctness and complexity.

4.1. *A simple* EXPTIME-*efficient algorithm*

Regarding the data structures, we assume that two auxiliary functions for working with sets are available.

```
methods boolean member(set of concepts, set of (set of concepts))
        void insert(set of concepts,set of (set of concepts))
```

Their meaning is obvious from the names themselves: the first function returns true if a set of concepts is a member of a set of sets of concepts; the second one inserts the set into the collection.

Remark 4.1. To obtain our single exponential upper bound, it is essential that such procedures require at most single exponential time in the size of the first argument (the set of concepts) and polynomial time in the size of the second argument (the set of set of concepts).

[2] To guarantee the single exponential time is necessary that such bound is linear in the size of the TBox and the concept.

This is not a restriction, as many algorithms with much better bounds for equality testing of sets are known [74] and efficiently implemented [59]. From the viewpoint of practical implementations, methods requiring logarithmic or sublogarithmic time and space are much better. The set passed as first argument can be as big as the whole KB, and the second argument can contain all possible subsets of the subconcepts of KB.

These functions are repeatedly applied to two sets of sets of concepts:

- Visited which contains the sets of concepts which are PS-reduced;
- NoGoods which contains the sets of concepts which have been shown to be $\perp$-sets.

For what regards the control structure, we assume two auxiliary procedures for selecting concepts and prefixed sets of concepts to be reduced according our calculus:

- **chooseConcept** selects the concept C to be reduced next in a given prefixed set $e : \mathcal{C}$;
- **chooseSet** selects the prefixed set $e : \mathcal{C}$ to be reduced next in a given set of prefixed sets concept $\mathcal{B}$ (which is just a branch of a tableau).

We have no restriction on these procedures, beside the obvious one that they should work in polynomial time in the size of the input, and they should respect the search techniques set forth in Section 3. We assume that

- **chooseConcept** works accordingly to Technique 4;
- **chooseSet** works accordingly to Technique 6.

In the remaining algorithms, we use the symbol "=" for assignment, and "==" for equality testing.

The main algorithm SATISFIABLE is just a shell, which initializes the search strategy, and calls the proper search procedure DFS (depth-first search). It is shown in Fig. 4.

The core algorithm DFS, which is directly derived from the calculus and the various techniques we have listed, is shown in Fig. 5. It takes a branch $\mathcal{B}$ of the tableau and extends it, according the rules of the calculus.

A tricky part of the algorithm is the value which is returned by DFS. It is the constant value *sat* if the branch $\mathcal{B}$ on which DFS is called can be extended to an open branch (i.e., if

```
Algorithm SATISFIABLE;
input a TBox KB, and a set of concepts $\mathcal{C}$;
output sat if $\mathcal{C}$ is satisfiable with respect to KB,
          unsat otherwise;
variables set of (set of concepts) NoGoods;
              set of (set of concepts) Visited;
              integer $n$;
methods boolean member(set of concepts, set of (set of concepts))
             void insert(set of concepts, set of (set of concepts))
begin
    NoGoods= $\emptyset$;
    Visited= $\emptyset$;
    $n = 1$;
    return DFS($\{1 : \mathcal{C}\}$)== sat
end
```

Fig. 4. The algorithm for deciding satisfiability.

```
Algorithm DFS;
input a branch $\mathcal{B}$
output sat if $\mathcal{B}$ is satisfiable, a prefix otherwise;
variable set of (set of concepts) OldVisited;
begin
    if chooseSet a prefixed set $e:\mathcal{C}\in\mathcal{B}$ s.t. for some $C\in\mathcal{C}$ it is $\neg C\in\mathcal{C}$ $\boxed{A1}$
    then return $e$;
    else if chooseSet a prefixed set $e:\mathcal{C}\in\mathcal{B}$ s.t. member($\mathcal{C}$, NoGoods) $\boxed{A2}$
    then return $e$;
    else chooseSet a prefixed set $e:\mathcal{C}\in\mathcal{B}$ s.t. $e:\mathcal{C}$ is not reduced and
                 not member($\mathcal{C}$, Visited); $\boxed{A3}$
        if chooseSet fails
        then return sat
        else chooseConcept a concept $C\in\mathcal{C}$ which is not reduced
            Apply the appropriate tableau rule to $C$
                in the prefixed set $e:\mathcal{C}$ in the branch $\mathcal{B}$
end
```

Fig. 5. The DFS algorithm for deciding satisfiability.

the the initial concept is satisfiable), and a prefix e otherwise. The prefix e indicates which prefixed set of $\mathcal{B}$ has been shown to be inconsistent with respect to the TBox, i.e., (at least) the set prefixed by e is a $\perp$-set.

To keep the set NoGoods up to date and consistent with the value returned by DFS, we make sure that each time DFS($\mathcal{B}\cup\{e:\mathcal{C}\}$) returns the prefix e, the set $\mathcal{C}$ has been inserted in NoGoods.

We also use the returned prefix to implement an optimization that we call *modal backjumping* and which we discuss later in this section together with other optimizations.

Then, the DFS algorithm checks that the branch contains neither atomic clashes (point A1), nor previously seen $\perp$-sets (point A2), then selects some prefixed sets which is neither reduced, nor identical to some other set which is reduced, and applies the appropriate rule. For sake of readability, we replaced the actual "code" corresponding to the application of our tableau rules with an English sentence.

Fig. 6 shows how rules are applied by the DFS algorithm. As a reminder, the order in which rules are listed corresponds to the priority in which they should be applied by the selection function **chooseConcept**.

The rules are directly derived from the calculus we presented in Section 3 and, as usual, alternative choices due to the presence of β-formulae are considered by recursive calls.

Remark 4.2. In contrast to standard trace-based techniques—used in all current implementations [32,34,37,41,43,64]—we do *not* reduce $\pi(R)$ formulae by making "parallel" recursive calls to the satisfiability testing procedure for each modal successor, and by back-tracking (locally) as soon as one of these returns unsatisfiable.

```
Apply the appropriate tableau rule to $C$ in the prefixed set $e:\mathcal{C}$ in the branch $\mathcal{B}$
begin
   case type $(C)$ of
   $\alpha$ :
       $E = \mathrm{DFS}(\mathcal{B} \cup e : (\mathcal{C} \cup \{\alpha_1, \alpha_2\}))$;
       if $E == e$ then insert($\mathcal{C}$, NoGoods);
       return $E$
   $neg$:
       $E = \mathrm{DFS}(\mathcal{B} \cup e : (\mathcal{C} \cup \{pos\}))$;
       if $E == e$ then insert($\mathcal{C}$, NoGoods);
       return $E$
   $KB$ :
       select $D_1 \sqsubseteq D_2 \in KB$ such that $\neg D_1 \sqcup D_2 \notin \mathcal{C}$;
       $E = \mathrm{DFS}(\mathcal{B} \cup e : (\mathcal{C} \cup \{\neg D_1 \sqcup D_2\}))$;
       if $E == e$ then insert($\mathcal{C}$, NoGoods);
       return $E$
   $\beta$ :
       OldVisited= Visited;
       $E = \mathrm{DFS}(\mathcal{B} \cup e : (\mathcal{C} \cup \{\beta_1\}))$;
       if $E == e$ and not member($\mathcal{C}$, NoGoods) $\boxed{A4}$
       then Visited= OldVisited;
           $E = \mathrm{DFS}(\mathcal{B} \cup e : (\mathcal{C} \cup \{\beta_2\}))$;
           if $E == e$ then insert($\mathcal{C}$, NoGoods);
       return $E$
   $\nu(R)$ :
       chooseSet another node $e.R.n : \mathcal{D} \in \mathcal{B}$ such that $\nu_0 \notin \mathcal{D}$
       return $\mathrm{DFS}(\mathcal{B} \cup e.R.n : (\mathcal{D} \cup \{\nu_0\}))$
   $\pi(R)$ :
       if all other $\pi'(R')$ are reduced
       then insert($\mathcal{C}$,Visited); $\boxed{A5}$
       $n = n + 1$;
       $E = \mathrm{DFS}(\mathcal{B} \cup e.R.n : \{\pi_0\})$;
       if $E == e.R.n$ then insert($\mathcal{C}$, NoGoods); $\boxed{A6}$
                             $E = e$;
       return $E$
   endcase
end
```

Fig. 6. The application of tableau rules in DFS.

It is easy to recast the algorithm according to this "schema" and retain correctness, but it is open whether our complexity result will transfer. A substantially different proof would be needed.

For sake of example, we recall the intuitions behind the reduction of β-formulae. Since we selected a β-formula we continue the search by visiting the left branch of the tableau, obtained by adding the prefixed node $e : (\mathcal{C} \cup \{\beta_1\})$. When DFS returns from the left call with a value E, we have three possibilities:

- DFS returns *sat* and then clearly we have no need to check the right branch;
- DFS returns a prefix E different from e and then we know that there is a $\perp$-set in $\mathcal{B}$, but this set has nothing to do with $e : \mathcal{C}$, so there is no point in continuing case analysis on the concepts in $\mathcal{C}$;
- DFS returns e and therefore $\mathcal{C} \cup \{\beta_1\}$ is a $\perp$-set, so we must continue on the right branch with $e : (\mathcal{C} \cup \{\beta_2\})$.

Then, if a right call is issued, and DFS returns a prefix from the case analysis on β_2, we have two more cases:

- DFS returns e. Then we know that also $\mathcal{C} \cup \{\beta_2\}$ is a $\perp$-set, so we conclude that $\mathcal{C}$ too is a $\perp$-set and we insert it into NoGoods.
- DFS returns a prefix E different from e. Then, we have found an alternative $\perp$-set in $\mathcal{B}$ (maybe because $\mathcal{C} \cup \{\beta_2\}$ is satisfiable with respect to the TBox), so we do nothing and return the new prefix for continuing the search.

Remark 4.3. A tricky point here is the check “and not member ($\mathcal{C}$, NoGoods) (A4)” which would seem to be unnecessary, because of the preventive check A2 (Fig. 5). Instead, check A4 is essential to prove our complexity result. The intuition is that at the time we met $\mathcal{C}$ for the first time and applied the rule β we had not yet shown $\mathcal{C}$ to be a $\perp$-set. However, when returning from the left branch, we might have already met $\mathcal{C}$ when visiting a modal successor of e, and found a different proof of unsatisfiability for $\mathcal{C}$.

The techniques we presented in Section 3 are implemented as follows:

Technique 1 is realized through condition A3 in the DFS algorithm for what concerns PS-rules. For $\perp$-rules, the technique is cast into the DFS algorithm by exiting each DFS call whenever a $\perp$-set is inserted.

Technique 2 is realized through conditions A1 and A2 of the DFS algorithm.

Technique 3 is realized indirectly through conditions A1, A2, A4, since they prevent the application of PS-rules to an implicitly inconsistent prefixed set.

Technique 4 must be implemented through the selection function **chooseConcept**, called in the DFS algorithm, to select the concept to be reduced next.

Technique 5 is implemented through the global variable n in the main algorithm, which is incremented at each generation of a new prefix.

Technique 6 must be implemented through the selection function **chooseSet**, called in the DFS algorithm, to select the prefixed set to be reduced next.

Technique 7 is the DFS algorithm itself.

From the above correspondences, it is straightforward to prove that the collection of algorithms is a correct implementation of the calculus, since each rule is one-one with a case in the algorithm DFS.

4.2. Adding optimizations to DFS

The presence of optimizations is a standard feature of any efficient tableau-based implementation [34,43,48,63,71]. We discuss the most important and widespread ones.

Some obvious preprocessing steps such as *lexical normalization and boolean reduction* can be added without further ado. They are immaterial for the results presented here.

The first substantial optimization is the use of *shallow reduction rules*, sometimes called *simplification rules*, such as unit resolution, modus ponens and modus tollens. For instance, a typical rule has the form "if $C \sqcup D$ is present in a set of concepts $\mathcal{C}$ and also $\neg C$ is present in $\mathcal{C}$ then add D to $\mathcal{C}$". Any efficient implementation applies them in an eager way. See, for instance, the classical paper by Oppacher and Suen [63], or the papers on the comparison of modal provers [32,43,48].

These rules do not change the set of concepts visited by the algorithm and make it possible to avoid branches of shallow depth. Hence, their addition to the algorithm does not change our correctness and complexity proofs.

The second optimization is the use of *deep reduction/simplification rules*, introduced in [55], which allow to simplify the structure of concepts at any level of nesting of connectives. An instance of the rule might have the form "if $\forall R.C$ is present in a set of concepts and $\forall R.(C \sqcap D)$ is also present in $\mathcal{C}$ then replace the second concept with $\forall R.D$".

These rules eliminate further branches and can speed up the search exponentially. Their addition does not change the correctness proof, but the complexity proof must be modified since they change the set of concepts visited by the algorithm. The important property to guarantee is that the number of concepts that are introduced by these rules is linearly (or polynomially) bounded by the number of subconcepts of the original input. Thus, this must be proved for each simplification rule.

The second substantial optimization is the use of *limited forms of analytic cut*, sometimes called *semantic branching* [43,45], asymmetric beta rule, split, principle of bivalence etc. The rule has only one form: "whenever β is present in a set of concepts $\mathcal{C}$ then add two branches and add β_1 to $\mathcal{C}$ in the first branch and $\neg\beta_1$ to $\mathcal{C}$ in the second branch". Then β_2 might be derived by shallow reduction rules in the second branch. The way in which β_1 is chosen might vary or the "presentation of the rule" might be different but the resulting calculus is essentially the same. For instance this rule is at the basis of the KSAT calculi of Giunchiglia et al. [32,34].

From a complexity-theoretic point of view, this rule strengthens the classical tree-like tableau calculus in propositional logic. Indeed, there are unsatisfiable sets of propositional concepts whose shortest proof using "semantic branching" rules is exponentially shorter than the shortest proof using traditional rules—see the survey of Urqhart [76] for further details.

The exponential slow down due to the absence of semantic branching is not true for our algorithm. Although we do not use analytic cut (i.e., semantic branching), we store $\perp$-sets to cut the search (see again check A2 and A4 in the DFS algorithm). This means that, for what regard unsatisfiability proofs in propositional logic, our calculus corresponds to a directed acyclic graph (DAG) variant of a tableau calculus. DAG tableaux have the same power as tree-like tableaux with full analytic cut (and not just the restricted version above). It is known [2] that in propositional logic we have exponentially shorter proofs with respect to calculi using the restricted cuts currently employed by semantic branching implementations.

Thus, semantic branching is neither necessary nor useful to speed up the search in our case. Indeed, in our full worked example—see Appendix A—we show an example of a simple concept for which our method substantially prunes more branches than analytic cut.

The next batch of optimizations does not regard properly the calculus but rather the search process. They do not affect the calculus because they just say that, under certain conditions, the application of some rules can be skipped without loosing completeness. Thus, they do not introduce new formulae among the set of formulae which can be potentially visited by the algorithm, but can make this set of potentially visited formulae smaller. Therefore they do not change nor the complexity nor the soundness proof and must be definitely added in an efficient implementation.

The first optimization in this batch which is often employed is *unit subsumption*: "if $C \sqcup D$ is present in a set of concepts $\mathcal{C}$ and also C is present in $\mathcal{C}$ then delete $C \sqcup D$". This rule is generalized by our selection rule (Technique 1): if C is present then $C \sqcup D$ is PS-reduced and so it will be never be considered for further reductions.

The next one is *absorption*, the lazy unfolding of definitions in the TBox. The basic intuition is that whenever we have two inclusions of the form $A \sqsubseteq C$ and $C \sqsubseteq A$, for an atomic concept A, we can see it as a definition for the concept $A \doteq C$, provided that C does not depends on A directly or via other inclusions.

Then, we can reformulate the KB-rule for this particular case: we add C to a prefixed set $e : C$ if $A \in \mathcal{C}$, we add $\neg C$ if $\neg A \in \mathcal{C}$, and skip the axioms $A \sqsubseteq C$ and $C \sqsubseteq A$ otherwise. In other words this corresponds to apply the traditional KB-rule introducing $\neg A \sqcup C$ (or $A \sqcup \neg C$) only when it can be followed by a shallow simplification rule (unit resolution step). Since this optimization has been recently proven complete by Horrocks and Tobies [46], it can be applied without ado.

Indeed, we can state a stronger result:

Remark 4.4. Any optimization which only restricts the application of one or more PS-rules and which has been proven complete for any fair strategy can be added without changing the complexity result.

We do not know whether this result can be extended to $\perp$-rules. Likely this is not possible as the the continuous introduction of new $\perp$-sets as the search proceeds along different branches is one of the backbone of our complexity proof.

The last optimization makes it possible to skip branching points in the search. First proposed by Shanin et al. in [71], it has been re-discovered many times under different names such as *proof condensation* [63], *conflict directed backjumping* [17], level cut [5], etc. A good discussion on backjumping in description logics implementations can be found in [43,45].

The idea behind this form of optimization can be explained by looking at the check A4 in our algorithm: we could also check whether β_1 actually contributed to the unsatisfiability of the branch. If not, then we just skip over this branching point, without going to the right branch with β_2. The intuition is that we can get a closed tableau without performing case analysis on β.

Notice that this is not precisely an optimization rule of those mentioned above, because it does not prevent entirely the application of the β-rule, it just says that we can do one-half of it.

This traditional form of proof condensation, backjumping, etc., which we call *propositional backjumping*, can be added by a slight modification of the value returned by the DFS algorithm without changing its correctness.

Instead of a prefix, the algorithm now returns a pair $\langle e, \mathcal{D}\rangle$. The intuition is that $\mathcal{D}$ contains only a minimal set of concepts responsible for the unsatisfiability of the set of concepts labeled by the prefix e. Then, when the DFS algorithm exits a call, we examine the concept in $e : \mathcal{C}$ and see whether the subconcepts we have added to $\mathcal{C}$ can be found in $\mathcal{D}$. If this is indeed the case, we delete the subconcepts and add the parent concept to $\mathcal{D}$, otherwise we just skip the parent concept.

Remark 4.5. So, one may be tempted to conclude that "propositional backjumping can be added without changing the complexity results", but this is *not* the case. Indeed, one of the landmark of our complexity proof is that we never do an exponential amount of work on unsatisfiable branches without storing a new $\perp$-set. When backjumping over a disjunction we might have done precisely that. So it is essential to specify precisely the format of the backjumping rule, and how it affects the insertion of $\perp$-sets.

For instance, we could further extend the optimization by storing only the minimal set $\mathcal{D}$ among the NoGoods. It is open whether an algorithm implementing would terminate in single exponential time, when *membership checking* of a new set of concepts in NoGoods is used.

Our check A4 already offers a form of backjumping, which we call *modal backjumping*. Indeed, by checking that the returned prefix is equal to the current prefix we check whether the inferences performed on the concepts of e on the whole actually contributed to the proof of unsatisfiability. If the returned prefix is different from the current prefix, this means that no rule actually contributed; then, we skip *all* branching points linked to that "useless" prefix.

This is different from the traditional (trace-based) technique employed by current systems: avoiding the exploration of further modal successor once a modal successor has been found unsatisfiable. Modal backjumping allows to skip chronological branching points of modal siblings visited by DFS prior to the actual prefix that has been proven unsatisfiable. For instance, suppose that we apply a β-rule to a prefixed set $e : \mathcal{C}$ and find that the left branch (with β_1) is unsatisfiable with the DFS returning e. Then we must go in the right branch. This subtree may turn out to be unsatisfiable as well, but the DFS may return a prefix e' different from e. In a nutshell, the heuristics of DFS have found out a shorter proof somewhere else. In standard trace-based techniques, where backtracking is local, we should anyway consider the previous branching points related to e. In contrast, when DFS returns up in the execution stack, all previous branching points related to e, its modal predecessors, and its modal successors will be skipped until a rule applied to e' is found.

4.3. From membership testing to subset checking

One could therefore wonder whether our algorithm can be extended to work with subset-checking, as common in some recent description logics and modal logics literature [11,26, 44,45].

At first glance, there is an optimization that significantly shrinks the set to be tested, it is easy (and even desirable) to implement, and affects neither the correctness, nor the complexity proof, while keeping the membership test based on equality of sets.

The optimization consists in *restricting insertions and membership tests only to sets of modal atoms and propositionally unreduced concepts*. Thus, whenever we call either member($\mathcal{C}$, Visited) or insert($\mathcal{C}$, NoGoods), we delete from the set to be inserted or checked all propositionally reduced concepts and all axioms $\neg C \sqcup D$ from the knowledge base KB; we just keep modal atoms and propositionally unreduced concepts which are not , then we insert the reduced $\mathcal{C}^*$ in Visited or NoGoods, or test its presence.

We use this technique in the example in Appendix A. Then, when presenting the proofs of correctness and complexity of the algorithm in subsequent sections, we discuss the modifications needed to take into account the above optimization.

Our complexity results can be extended to full subset checking if one has efficient *subset checking procedure* for NoGoods. This means that we have a data structure for storing NoGoods such that given a set of concepts $\mathcal{C}$ and a set of sets of concepts NoGoods, we can test in polynomial time in the size of NoGoods whether NoGoods contains a subset of the input set $\mathcal{C}$.

As we have remarked, efficient algorithms for equality testing of sets are known [74] and efficiently implemented [59], but we do not know of any such implementation for efficient (polynomial time) subset checking algorithms [80]. An implementation for insertion and subset checking has been recently presented in [40], but its complexity properties are not studied enough.

If subset checking can be efficiently implemented, our algorithm can be optimized for subset checking. For the optimized algorithm DFS-with-subset-checking, we assume three procedures for working with sets are available.

```
methods set of concepts subset(set of concepts, set of (set of concepts))
        set of concepts superset(set of concepts, set of (set of concepts))
        void insert(set of concepts, set of (set of concepts))
```

The meaning of each procedure or function is obvious from the name itself. For instance the function "subset($\mathcal{C}, \mathcal{SC}$)", where $\mathcal{C}$ is a set of concepts and $\mathcal{SC}$ a set of sets of concepts, returns a set of concepts $\mathcal{D}$ such that $\mathcal{D} \subseteq \mathcal{C}$ and $\mathcal{D} \in \mathcal{SC}$ if one exists, otherwise, if no such $\mathcal{D}$ exists in $\mathcal{SC}$, it returns a distinguished value *no*. The function "superset" finds a set $\mathcal{D} \supseteq \mathcal{C}$ if one exists and *no* if no such set can be found in $\mathcal{SC}$.

For the convenience of implementors, Figs. 7 and 8 show the optimized algorithm for subset checking. We have restructured it, so that we only have one **return** instruction at the end of the procedure.

Notice that it directly incorporates propositional and modal backjumping. Indeed, for propositional backjumping it is not necessary to add the condition "**and** $\beta_1 \in \mathcal{D}_1$" at A4

```
Algorithm DFS-with-subset-checking-and-backjumping;
input a branch $\mathcal{B}$
output a pair $\langle sat, ?\rangle$ if $\mathcal{B}$ is satisfiable, $\langle$prefix, set of concepts$\rangle$ otherwise;
variable set of (set of concepts) OldVisited;
begin
   if chooseSet a pref. set $e : \mathcal{C} \in \mathcal{B}$ s.t. for some $C \in \mathcal{C}$ it is $\neg C \in \mathcal{C}$ $\boxed{A1}$
   then $\langle E, \mathcal{D}\rangle = \langle e, \{C, \neg C\}\rangle$;
   else if chooseSet a pref. set $e : \mathcal{C} \in \mathcal{B}$ s.t. not subset($\mathcal{C}$, NoGoods) == $no$ $\boxed{A2}$
   then $\langle E, \mathcal{D}\rangle = \langle e, \text{subset}(\mathcal{C}, \text{NoGoods})\rangle$;
   else chooseSet a pref. set $e : \mathcal{C} \in \mathcal{B}$ s.t. $e : \mathcal{C}$ is not reduced and
                 superset($\mathcal{C}$, Visited)==$no$; $\boxed{A3}$
      if chooseSet fails
      then $\langle E, \mathcal{D}\rangle = \langle sat, ?\rangle$;
      else chooseConcept a concept $C \in \mathcal{C}$ which is not reduced
         Apply the appropriate tableau rule to $C$
               in the pref. set $e : \mathcal{C}$ in the branch $\mathcal{B}$
               yielding $\langle E, \mathcal{D}\rangle$

   return $\langle E, \mathcal{D}\rangle$;
end
```

Fig. 7. DFS algorithm with subset checking.

because it is subsumed by "subset($\mathcal{C}$, NoGoods) == no". The optimization of deleting reduced concepts and KB concepts that we mentioned at the beginning of the section can also be easily applied here.

The correctness and completeness proofs of the DFS algorithm can be extended to the subset-checking version in the standard way [26,36,45].

The extension of our complexity proof based on equality-testing to the optimized algorithm for subset-checking is possible, but it is a result of its own. The basic intuitions behind the extension of the complexity proof are sketched in the following points.

- With DFS-with-membership-testing, we are sure that each time we apply a rule to a set of concepts $\mathcal{C}$, DFS have not previously met $\mathcal{C}$ and have been previously shown it to be a $\perp$-set. To guarantee this property, we always store each set $\mathcal{C}$ in NoGoods as soon as we found that it is a $\perp$-set.
- With DFS-with-subset-checking, we are sure that each time we apply a rule to a set of concepts $\mathcal{C}$, DFS has not yet encountered a subset of $\mathcal{C}$ which was previously shown to be a $\perp$-set. To guarantee this property, we always store a $\perp$-subset of $\mathcal{C}$ in NoGoods as soon as we found that it is a $\perp$-set.
- With DFS-with-membership testing we are sure that each time DFS($\mathcal{B} \cup \{e : \mathcal{C}\}$) returns the prefix e then $\mathcal{C}$ has been inserted in NoGoods.
- With DFS-with-subset-checking we are also sure that each time DFS($\mathcal{B} \cup \{e : \mathcal{C}\}$) returns the pair $\langle e, \mathcal{D}\rangle$ then $\mathcal{D}$ has been inserted in NoGoods and is a (not necessarily proper) subset of $\mathcal{C}$.

```
**Apply** the appropriate tableau rule to $C$ in the prefixed set $e:\mathcal{C}$ in the branch $\mathcal{B}$
    yielding $\langle E, \mathcal{D}\rangle$
**begin**
    **case type** $(C)$ **of**
    $\alpha$ :
        $\langle E, \mathcal{D}\rangle = \mathrm{DFS}(\mathcal{B} \cup \{e:\mathcal{C} \cup \{\alpha_1, \alpha_2\}\})$;
        **if** $E == e$ **and** $(\alpha_1 \in \mathcal{D}$ **or** $\alpha_2 \in \mathcal{D})$
        **then** $\mathcal{D} = (\mathcal{D} \setminus \{\alpha_1, \alpha_2\}) \cup \{\alpha\}$;
            insert($\mathcal{D}$, NoGoods);
    *neg*:
        $\langle E, \mathcal{D}\rangle = \mathrm{DFS}(\mathcal{B} \cup \{e:\mathcal{C} \cup \{pos\}\})$;
        **if** $E == e$ **and** $pos \in \mathcal{D}$
        **then** $\mathcal{D} = (\mathcal{D} \setminus \{pos\}) \cup \{neg\}$;
            insert($\mathcal{D}$, NoGoods);
    $KB$ :
        **select** $D_1 \sqsubseteq D_2 \in KB$ such that $\neg D_1 \sqcup D_2 \notin \mathcal{C}$;
        $\langle E, \mathcal{D}\rangle = \mathrm{DFS}(\mathcal{B} \cup \{e:\mathcal{C} \cup \{\neg D_1 \sqcup D_2\}\})$;
        **if** $E == e$ **and** $\neg D_1 \sqcup D_2 \in \mathcal{D}$
        **then** $\mathcal{D} = \mathcal{D} \setminus \{\neg D_1 \sqcup D_2\}$;
            insert($\mathcal{D}$, NoGoods);
    $\beta$ :
        OldVisited = Visited;
        $\langle E, \mathcal{D}_1\rangle = \mathrm{DFS}(\mathcal{B} \cup \{e:\mathcal{C} \cup \{\beta_1\}\})$;
        **if** $E == e$ **and** subset($\mathcal{C}$, NoGoods) == *no* $\boxed{A4}$
        **then** Visited = OldVisited;
            $\langle E, \mathcal{D}_2\rangle = \mathrm{DFS}(\mathcal{B} \cup \{e:\mathcal{C} \cup \{\beta_2\}\})$;
            **if** $E == e$ **and** $\beta_2 \in \mathcal{D}_2$
            **then** $\mathcal{D} = (\mathcal{D}_1 \setminus \{\beta_1\}) \cup (\mathcal{D}_2 \setminus \{\beta_2\}) \cup \{\beta\}$;
                insert($\mathcal{D}$, NoGoods);
            **else** $\mathcal{D} = \mathcal{D}_2$;
        **else** $\mathcal{D} = \mathcal{D}_1$
    $\nu(R)$ :
        **chooseSet** another node $e.R.n:\mathcal{D} \in \mathcal{B}$ such that $\nu_0 \notin \mathcal{D}$
        $\langle E, \mathcal{D}\rangle = \mathrm{DFS}(\mathcal{B} \cup \{e.R.n:\mathcal{D} \cup \{\nu_0\}\})$;
    $\pi(R)$ :
        **if** all other $\pi'(R')$ are reduced
        **then** insert($\mathcal{C}$,Visited); $\boxed{A5}$
        $n = n + 1$;
        $\langle E, \mathcal{D}\rangle = \mathrm{DFS}(\mathcal{B} \cup \{e.R.n:\{\pi_0\}\})$;
        **if** $E == e.R.n$ **then** $\mathcal{D} = \{\nu(R) \in \mathcal{C} \mid \nu_0 \in \mathcal{D}\} \cup \{\pi(R)\}$;
                              insert($\mathcal{D}$, NoGoods); $\boxed{A6}$
                              $E = e$;
    **endcase**;
**end**
```

Fig. 8. Rules for the DFS algorithm with subset checking.

- Then, in the worst case subset checking boils down to membership testing and the proof for DFS-with-membership-testing can be carried over to DFS-with-subset-checking.

5. Correctness

We divide in two parts the proof of the correctness of our tableau method. First we prove the correctness for what regards concept unsatisfiability, then we address satisfiability.

5.1. Proving unsatisfiability

In this section we prove that if the tableau for a concept C and a TBox KB can be extended to a closed tableau, then C is unsatisfiable for KB. This is a "classical" proof by contradiction: we assume that the tableau $\mathcal{T}$ is satisfiable, show that satisfiability is preserved by tableau rules and derived a contraction from the fact that a tableau can be (implicitly) closed.

Definition 5.1. A tableau branch $\mathcal{B}$ is PS-satisfiable for a TBox KB if there is an interpretation $\langle \mathcal{I}, \Delta \rangle$ which satisfies KB and a mapping $\iota(e) \mapsto d$ from prefixes to elements of the domain such that

(1) for every prefixed set $e : \mathcal{C}$ in $\mathcal{B}$ and every concept $C \in \mathcal{C}$ one has $\iota(e) \in C^{\mathcal{I}}$.
(2) for every pair of prefixes e, $e.R.n$, appearing in prefixed sets of $\mathcal{B}$, one has $\langle \iota(e), \iota(e.R.n) \rangle \in R^{\mathcal{I}}$.

A tableau is PS-satisfiable for a TBox KB if at least a tableau branch is PS-satisfiable for KB.

For sake of simplicity we say that "a branch is PS-satisfiable" rather than "a branch is PS-satisfiable for a TBox KB", leaving the TBox implicit.

We start with a lemma saying that $\perp$-rules correctly propagate inconsistencies.

Lemma 5.2. *Let $e : (\mathcal{C})^{\perp}$ appear in a tableau $\mathcal{T}$ for a TBox KB. Then $KB \models \sqcap\mathcal{C} \sqsubseteq \perp$.*

Proof. By induction on the application of $\perp$-rules.

Base case. If both C and $\neg C$ are in $\mathcal{C}$, then $\sqcap\mathcal{C}$ is clearly unsatisfiable. Hence, $KB \models \sqcap\mathcal{C} \sqsubseteq \perp$.

Inductive cases. Suppose the claim holds for the antecedent of each $\perp$-rule. We analyze the application of each $\perp$-rule in turn.

($\perp$-witness): If $e' : (\mathcal{C})^{\perp}$ appears in $\mathcal{T}$, then by the inductive hypothesis $KB \models \sqcap\mathcal{C} \sqsubseteq \perp$, which is the claim.

($\perp$-α): If $e : (\mathcal{C} \cup \{\alpha_1, \alpha_2\})^{\perp}$ appears in $\mathcal{T}$, then by inductive hypothesis $KB \models (\sqcap\mathcal{C}) \sqcap \alpha_1 \sqcap \alpha_2 \sqsubseteq \perp$. Since $\alpha_1 \sqcap \alpha_2 = \alpha$, then also $KB \models (\sqcap\mathcal{C}) \sqcap \alpha \sqsubseteq \perp$.

($\perp$-dneg): Similar to the previous case.

($\bot$-β): If both $e : (\mathcal{C} \cup \{\beta_1\})^{\bot}$ and $e : (\mathcal{C} \cup \{\beta_2\})^{\bot}$ then by inductive hypothesis both $KB \models (\sqcap\mathcal{C}) \sqcap \beta_1 \sqsubseteq \bot$ and $KB \models (\sqcap\mathcal{C}) \sqcap \beta_2 \sqsubseteq \bot$ hold. Then $KB \models (\sqcap\mathcal{C}) \sqcap (\beta_1 \sqcup \beta_2) \sqsubseteq \bot$. Since $(\beta_1 \sqcup \beta_2) = \beta$, the claim follows.

($\bot$-KB): If $e : (\mathcal{C} \cup \{\neg C \sqcup D\})^{\bot}$ and $C \sqsubseteq D \in KB$, then by inductive hypothesis $KB \models (\sqcap\mathcal{C}) \sqcap (\neg C \sqcup D) \sqsubseteq \bot$. Since $C \sqsubseteq D \in KB$, in every model of KB the concept $(\neg C \sqcup D)$ is equivalent to $\top$. Hence, $KB \models (\sqcap\mathcal{C}) \sqcap \top \sqsubseteq \bot$, that is $KB \models \sqcap\mathcal{C} \sqsubseteq \bot$.

($\bot$-$\pi(R)$): If $e.R.n : (\mathcal{C} \cup \{\pi_0\})^{\bot}$ and $e : \mathcal{D}$ in $\mathcal{T}$, and $\mathcal{C} \subseteq \{\nu_0 \mid \nu(R) \in \mathcal{D}\}$, then by inductive hypothesis $KB \models (\sqcap\mathcal{C}) \sqcap \pi_0 \sqsubseteq \bot$. We prove the claim by contradiction. Suppose there is a model $\mathcal{I}$ for KB such that an element $a \in \Delta$ is in $(\mathcal{D} \cup \{\pi(R)\})^{\mathcal{I}}$: then there exists another element $b \in \Delta$ such that $(a, b) \in R^{\mathcal{I}}$, $b \in (\pi_0)^{\mathcal{I}}$ and for every $\nu(R) \in \mathcal{D}$, $b \in (\nu_0)^{\mathcal{I}}$. Therefore, $b \in ((\sqcap\{\nu_0 \mid \nu(R) \in \mathcal{D}\}) \sqcap \pi_0)^{\mathcal{I}}$. Since $((\sqcap\{\nu_0 \mid \nu(R) \in \mathcal{D}\}) \sqcap \pi_0)^{\mathcal{I}} \subseteq ((\sqcap\mathcal{C}) \sqcap \pi_0)^{\mathcal{I}}$, this contradicts the hypothesis that the latter concept is interpreted as the empty set in every model of KB. □

If we wish to optimize our algorithm by storing and testing only modal atoms and unreduced formulae, the witness step of the proof is the only one that may fail since the node which is tested among the NoGoods would not be $e' : \mathcal{C}$ but rather some $e' : \mathcal{C}'$ such that $\mathcal{C}$ and $\mathcal{C}'$ only share the same modal atoms and the same unreduced concepts. However it is immediate to prove by induction on the (absent) reduced formulae the following proposition:

Proposition 5.3. *Let $\mathcal{C}$ and $\mathcal{C}'$ be two sets of concepts with the same modal atoms and unreduced concepts. Then, they are logically equivalent, i.e., the concept $KB \models \sqcap\mathcal{C} \sqsubseteq \sqcap\mathcal{C}'$ and $KB \models \sqcap\mathcal{C}' \sqsubseteq \sqcap\mathcal{C}$.*

And this is all we need.

Lemma 5.4. *A PS-satisfiable branch in a tableau for a Tbox KB cannot be implicitly closed.*

Proof. Let $\mathcal{B}$ be the PS-satisfiable branch and let $\langle \mathcal{I}, \Delta \rangle$ and $\iota()$ be the corresponding interpretation and mapping.

By definition of implicitly closed branch, there is a prefixed set $e : (\mathcal{C})^{\bot}$ in the branch. By definition of PS-satisfiable branch it is $\iota(e) \in C^{\mathcal{I}}$ for every $C \in \mathcal{C}$. Thus $\iota(e) \in (\sqcap\mathcal{C})^{\mathcal{I}}$. By Lemma 5.2 it is $KB \models \sqcap\mathcal{C} \sqsubseteq \bot$, contradiction. □

Corollary 5.5. *A PS-satisfiable tableau cannot be implicitly closed.*

Now we continue the proof of Theorem 3.11 by showing that PS-rules preserve the satisfiability of the tableau.

Lemma 5.6. *The application of a PS-rule or a $\bot$-rule to a PS-satisfiable tableau for a TBox KB yields another PS-satisfiable tableau for KB.*

Proof. Let $\mathcal{B}$ be a PS-satisfiable branch in the initial tableau and let $\langle \mathcal{I}, \Delta \rangle$ and $\iota()$ be the corresponding interpretation and mapping. If either a PS-rule or a $\perp$-rule is applied to a prefixed set in a branch different from $\mathcal{B}$, the resulting tableau is obviously still satisfiable. The only interesting case is when a rule is applied to a prefixed set in $\mathcal{B}$.

First, note that if a $\perp$-rule can be applied to a prefixed set in $\mathcal{B}$, then $\mathcal{B}$ is implicitly closed. This is impossible for Lemma 5.4.

So we are only left with PS-rules and the proof proceeds by cases. We show that we can extend the mapping to accommodate the newer prefixed sets added by the applied PS-rule.

dneg Suppose that $e : \mathcal{C} \cup \{neg\}$ has been selected and that $e : \mathcal{C} \cup \{pos\}$ has been added to $\mathcal{B}$. Consider this new branch $\mathcal{B}'$. Observe that for all $C \in \mathcal{C}$ it is $\iota(e) \in C^{\mathcal{I}}$ since $e : \mathcal{C} \cup \{neg\}$ was already in $\mathcal{B}$, and $\mathcal{B}$ was PS-satisfiable. By hypothesis $\iota(e) \in (neg)^{\mathcal{I}}$ and $(neg)^{\mathcal{I}} = (pos)^{\mathcal{I}}$ according the semantics. Hence $\mathcal{B}'$ is still PS-satisfiable.

α Suppose that $e : \mathcal{C} \cup \{\alpha\}$ has been selected and that $e : \mathcal{C} \cup \{\alpha_1, \alpha_2\}$ has been added. By hypothesis $\iota(e) \in (\alpha)^{\mathcal{I}}$ and by definition of interpretation it is $(\alpha)^{\mathcal{I}} = (\alpha_1)^{\mathcal{I}} \cap (\alpha_2)^{\mathcal{I}}$. Thus $\iota(e) \in (\alpha_1)^{\mathcal{I}}$ and $\iota(e) \in (\alpha_2)^{\mathcal{I}}$. The claim follows with the same line of reasoning we used for *dneg*.

β Suppose that $e : \mathcal{C} \cup \{\beta\}$ has been selected and that $e : \mathcal{C} \cup \{\beta_1\}$ has been added as a left leaf and $e : \mathcal{C} \cup \{\beta_2\}$ has been added as a right leaf. By hypothesis $\iota(e) \in (\beta)^{\mathcal{I}}$, and by definition $(\beta)^{\mathcal{I}} = (\beta_1)^{\mathcal{I}} \cup (\beta_2)^{\mathcal{I}}$. Hence either $\iota(()e) \in (\beta_1)^{\mathcal{I}}$ or $\iota(()e) \in (\beta_2)^{\mathcal{I}}$. Suppose that $\iota(e) \in (\beta_1)^{\mathcal{I}}$. Then consider the left branch $\mathcal{B}_1$ extending $\mathcal{B}$, i.e., the branch including $e : \mathcal{C} \cup \{\beta_1\}$. For all $C \in \mathcal{C}$ it is $\iota(e) \in C^{\mathcal{I}}$ since $e : \mathcal{C} \cup \{\beta\}$ was already in $\mathcal{B}$, and $\mathcal{B}$ was PS-satisfiable. So $\mathcal{B}_1$ is PS-satisfiable. A similar argument applies if $\iota(e) \in (\beta_2)^{\mathcal{I}}$, using the right branch.

KB Suppose that $e : \mathcal{C}$ has been selected and that $e : \mathcal{C} \cup \{\neg C \sqcup D\}$ has been added for some $C \sqsubseteq D \in KB$. Since $\mathcal{B}$ is PS-satisfiable for KB, the corresponding interpretation $\langle \mathcal{I}, \Delta \rangle$ is a model for the KB. Thus it validates $\neg C \sqcup D$ and since $\iota(e) \in \Delta$ it follows that $\iota(e) \in (\neg C \sqcup D)^{\mathcal{I}}$. The claim follows with the same line of reasoning we used for *dneg*.

$\pi(R)$ Suppose that $e : \mathcal{C} \cup \{\pi(R)\}$ has been selected and that $e.R.n : \mathcal{C} \cup \{\pi_0\}$ has been added. By hypothesis of PS-satisfiable branch $\iota(e) \in (\pi(R))^{\mathcal{I}}$ and thus there is an element of the domain d such that $\langle \iota(e), d \rangle \in R^{\mathcal{I}}$ and $d \in (\pi(R))^{\mathcal{I}}$, since $\mathcal{B}$ is PS-satisfiable. Then we extend the mapping as follows: $\iota(e') = d$ if e'=$e.R.n$. For all other elements, $\iota()$ is as before.

Since $e.R.n$ is new there is no prefixed set $e.R.n.R'.m : \mathcal{D}$ in the branch. By construction $\langle \iota(e), \iota(e.R.n) \rangle = \langle \iota(e), d \rangle \in R^{\mathcal{I}}$.

For all other pairs of prefixes $\langle \iota(e'), \iota(e'.R'.n') \rangle \in (R')^{\mathcal{I}}$ by hypothesis of PS-satisfiable branch. Then the extended branch is PS-satisfiable with the same interpretation $\langle \mathcal{I}, \Delta \rangle$ and the extended mapping $\iota()$.

$\nu(R)$ Suppose that $e.R.n : \mathcal{C}$ and $e : \mathcal{D} \cup \{\nu(R)\}$ have been selected and that $e.R.n : \mathcal{C} \cup \{\nu_0\}$ has been added. By hypothesis of PS-satisfiable branch $\iota(e) \in \nu(R)^{\mathcal{I}}$ and $\langle \iota(e), \iota(e.R.n) \rangle \in R^{\mathcal{I}}$. Therefore $\iota(e.R.n) \in \nu_0^{\mathcal{I}}$ by the semantics. □

Theorem 3.11. *Let C be a concept and KB be a TBox. If there is a closed tableau for C using KB, then no model of KB satisfies C.*

Proof. The proof is a standard proof by contradiction, using Lemma 5.6 and Corollary 5.5. See, e.g., [28, Chapter 8]. □

5.2. *Proving satisfiability*

Now we prove the correctness of our tableau method for what regards concept satisfiability (Theorem 3.12): if the tableau for the concept C and the TBox KB can be extended to an open tableau then C is satisfiable for KB. At an abstract level, this proof is carried over in four steps:

(1) we define a canonical interpretation from an open branch;
(2) we define a suitable mapping $\iota()$;
(3) we prove by induction that the canonical interpretation is a model of the branch: we have to prove separately the base case, and the induction;
(4) we combine these results into the main theorem.

The first part of the proof is the construction of an interpretation from an open branch $\mathcal{B}$.

Definition 5.7. The *canonical interpretation* $\mathcal{I}$ of an open branch $\mathcal{B}$ is constructed as follows:

$$\Delta = \{e \mid e : \mathcal{C} \in \mathcal{B} \text{ and there is no prefixed set } e : \mathcal{D} \text{ which has a witness in } \mathcal{B}\}, \tag{1}$$

$$A^{\mathcal{I}} = \{e \in \Delta \mid e : \mathcal{C} \in \mathcal{B} \text{ and } A \in \mathcal{C}\}. \tag{2}$$

The interpretation of atomic roles is composed of two sets:

$$R^{\mathcal{I}} = R_d \cup R_w, \tag{3}$$

where R_d and R_w are defined by:

$$R_d = \bigl\{\langle e, e.R.n\rangle \mid \text{ both } e \in \Delta \text{ and } e.R.n \in \Delta\bigr\}, \tag{4}$$

$$R_w = \bigl\{\langle e, e'\rangle \mid \text{ both } e \in \Delta \text{ and } e' \in \Delta \text{ and } e' : \mathcal{C} \text{ is the } \prec\text{-minimal witness in } \mathcal{B} \text{ for } e.R.n : \mathcal{C}\bigr\}. \tag{5}$$

The interpretation is extended to complex concepts following their semantics.

Note that in (3) the first set R_d takes into account the relations which are explicitly represented in the domain and the second set R_w takes into account the presence of witnesses. Note also that in (5) $e.R.n$ is not in Δ, otherwise $e.R.n : \mathcal{C}$ would not have a witness, because of (1).

The second step is to devise a uniform mapping $\iota()$ from elements to members of the domain. The use of a mapping is a device that makes the proof easier. The mapping is set such that $\iota(e) = e$ unless the prefix e appears in a prefixed set $e : \mathcal{C}$ which has a witness $e' : \mathcal{C}$. In this case we set $\iota(e) = \iota(e')$.

We can prove the third step of the main proof: we start with the base case of the structural induction on concept construction. We recall that a branch is a path in a tree, whose nodes are prefixed sets. In what follows, we use the intuitive notion of *descendant* of a prefixed set referring to such a tree. We first highlight two properties about prefixed sets.

Proposition 5.8. *Let $e : \mathcal{C}$ and $e : \mathcal{D}$ be two prefixed sets occurring in one branch $\mathcal{B}$. If $e : \mathcal{D}$ is a descendant of $e : \mathcal{C}$, then $\mathcal{C} \subseteq \mathcal{D}$.*

Proposition 5.9. *Let $\langle \mathcal{I}, \Delta \rangle$ be the canonical model of an open branch $\mathcal{B}$, let R be an atomic role, and let $e : \mathcal{C}$ and $e.R.n : \mathcal{D}$ be prefixed sets occurring in $\mathcal{B}$. Then, $\langle \iota(e), \iota(e.R.n) \rangle \in R^{\mathcal{I}}$.*

The former property follows from the definition of PS-rules, while the latter is immediate from (4) and (5). We state and prove the base case of the induction.

Lemma 5.10. *Let $\langle \mathcal{I}, \Delta \rangle$ be the canonical model of an open branch $\mathcal{B}$. Let A be an atomic concept and let $e : \mathcal{C}$ be a prefixed set occurring in $\mathcal{B}$. If $A \in \mathcal{C}$ then one has $\iota(e) \in A^{\mathcal{I}}$; otherwise if $\neg A \in \mathcal{C}$ then one has $\iota(e) \notin A^{\mathcal{I}}$.*

Proof. Let $e : \mathcal{D}$ be the deepest descendant of $e : \mathcal{C}$ with the same prefix e. From Proposition 5.8 one has that $A \in \mathcal{C}$ (respectively $\neg A \in \mathcal{C}$) implies $A \in \mathcal{D}$ (respectively $\neg A \in \mathcal{D}$), and thus we restrict our attention to $e : \mathcal{D}$.

First, assume that no prefixed set $e : \mathcal{C}'$ has a witness in $\mathcal{B}$. Then, from (1) $e \in \Delta$. If $A \in \mathcal{C}$ then by definition of assignment $\iota(e) = e$ and by (2), $e \in A^{\mathcal{I}}$. If $\neg A \in \mathcal{C}$ then suppose that $\iota(e) \in A^{\mathcal{I}}$. Again by (2), there is a $\mathcal{C}'$ such that $e : \mathcal{C}'$ is present in the branch and $A \in \mathcal{C}'$. Then both $\neg A \in \mathcal{D}$ and $A \in \mathcal{D}$ hold because of Proposition 5.8 and because $e : \mathcal{D}$ is the deepest descendant of $e : \mathcal{C}$. Then the branch would not be open, contradiction.

Second, suppose that $e : \mathcal{D}$ has a witness $e' : \mathcal{D}$. By Proposition 5.8, if $A \in \mathcal{C}$ (respectively $\neg A \in \mathcal{C}$) then $A \in \mathcal{D}$ (respectively $\neg A \in \mathcal{D}$). Since $e' : \mathcal{D}$ has no witnesses the claim holds for $e' : \mathcal{D}$, therefore $\iota(e') \in A^{\mathcal{I}}$ (respectively $\iota(e') \notin A^{\mathcal{I}}$). By construction we have that $\iota(e') = \iota(e)$. □

The inductive step of the structural induction on concept construction is proved in the next lemma. To prove that for a given concept C in a prefixed set $e : \mathcal{C}$ of $\mathcal{B}$, it holds $\iota(e) \in C^{\mathcal{I}}$, we assume the inductive hypothesis: for every prefix and every concept D which is a syntactic component of C, it holds $\iota(e') \in D^{\mathcal{I}}$. The formal statement follows.

Lemma 5.11. *Let $\langle \mathcal{I}, \Delta \rangle$ be the canonical model of an open branch $\mathcal{B}$. Let C be a concept, let $e : \mathcal{C}$ be a prefixed set occurring in $\mathcal{B}$ and assume that for every sub-concept D of C and every $e' : \mathcal{D}$ such that $D \in \mathcal{D}$ one has $\iota(e') \in D^{\mathcal{I}}$. If $C \in \mathcal{C}$, then $\iota(e) \in C^{\mathcal{I}}$.*

Proof. First, we suppose that the deepest descendant $e : \mathcal{D}$ of $e : \mathcal{C}$ has no witness.

Propositional Connectives. So that C has the form $D_1 \sqcap D_2$, $\neg(D_1 \sqcap D_2)$, $D_1 \sqcup D_2$, $\neg(D_1 \sqcup D_2)$ and $\neg\neg D_1$. We show only the case for $\neg(D_1 \sqcap D_2)$ since the others are similar.

Suppose that $\neg(D_1 \sqcap D_2) \in \mathcal{D}$: then, since the branch is implicitly PS-reduced, either $\neg D_1 \in \mathcal{D}$ or $\neg D_2 \in \mathcal{D}$. By construction we have that $\iota(e) = e$ and then by hypothesis either $\iota(e) \in (\neg D_1)^{\mathcal{I}}$ or $\iota(e) \in (\neg D_2)^{\mathcal{I}}$. Then we have $\iota(e) \in (\neg(D_1 \sqcap D_2))^{\mathcal{I}}$.

Existential Quantifier. Suppose that C has the form $\exists R.D$ and $C \in \mathcal{D}$. Since the branch is implicitly PS-reduced there is a node $e.R.n : \mathcal{D}'$ such that $D \in \mathcal{D}'$. By (4) and by Proposition 5.9, $\langle \iota(e), \iota(e.R.n)\rangle \in R^{\mathcal{I}}$. Since D is a subconcept of C, by hypothesis we have that $D \in \mathcal{D}''$ and $e' : \mathcal{D}''$ imply $\iota(e') \in D^{\mathcal{I}}$. Therefore $\iota(e.R.n) \in D^{\mathcal{I}}$ and by definition of interpretation we have that $\iota(e) \in \exists R.D^{\mathcal{I}}$.

Universal quantifier. Suppose C has the form $\forall R.D$. In this case we have to prove that for all $d \in \Delta$ if $\langle \iota(e), d\rangle \in R^{\mathcal{I}}$ then $d \in D^{\mathcal{I}}$. The interpretation $\mathcal{I}$ in Definition 5.7 is such that only the following sub-cases are possible:

(1) d is equal to $\iota(e.R.n)$ for some n;
(2) d is equal to $\iota(e')$ for some n, where $e' : \mathcal{D}'$ is the witness of a node $e.R.n : \mathcal{D}'$ in the branch;
(3) d is such that there is a node $e' : \mathcal{D}$ which has $e : \mathcal{D}$ for witness and $\langle \iota(e'), d\rangle \in R^{\mathcal{I}}$, for instance with $d = \iota(e'.R.m)$.

For the first case, observe that by definition of implicitly PS-reduced branch there must be a prefixed set $e.R.n : \mathcal{D}'$ such that $D \in \mathcal{D}'$ (rule ν has been applied). By hypothesis, it is $\iota(e.R.n) \in D^{\mathcal{I}}$.

For the second case, if $e' : \mathcal{D}'$ is the witness of a node $e.R.n : \mathcal{D}'$ in the branch then, according the definition of witness and implicitly PS-reduced branch, rule ν must have been applied to $e.R.n : \mathcal{D}'$ and hence $D \in \mathcal{D}'$. So D is also in the prefixed set of the witness and by hypothesis $\iota(e') \in D^{\mathcal{I}}$. By construction $\iota(e') = \iota(e.R.n)$.

For what regards the last case, by Technique 1 if $e' : \mathcal{D}$ has a witness no other prefixed set extending either e' can be part of the branch. So this possibility is ruled out.

In conclusion, $\iota(e) \in (\forall R.D)^{\mathcal{I}}$.

This closes the overall case in which the deepest descendant $e : \mathcal{D}$ of $e : \mathcal{C}$ has no witness.

Finally, suppose now that $e : \mathcal{D}$ has a witness $e' : \mathcal{D}$. By Proposition 5.8, if $C \in \mathcal{C}$ then $C \in \mathcal{D}$. Since $e' : \mathcal{D}$ has no witnesses the claim holds for $e' : \mathcal{D}$ and thus $\iota(e') \in C^{\mathcal{I}}$. By construction, we have $\iota(e') = \iota(e)$. □

Again, by optimizing our algorithm and storing and testing only modal atoms and unreduced formulae, the witness step of the proof may fail since the witness node would not be $e' : \mathcal{D}$ but rather $e' : \mathcal{D}'$ where $\mathcal{D}$ and $\mathcal{D}'$ only share the same propositionally reduced concepts and the same unreduced formulae. Proposition 5.3 does the necessary adaptation.

We can combine these lemmata above as follows:

Theorem 5.12. *Let $e : \mathcal{C}$ be a prefixed set occurring in an open branch $\mathcal{B}$, let $\langle \mathcal{I}, \Delta\rangle$ be the canonical model of $\mathcal{B}$, and let C be a concept such that $C \in \mathcal{C}$. Then $\iota(e) \in C^{\mathcal{I}}$.*

Proof. For the base case we apply Lemma 5.10 for both A and $\neg A$. Notice that for the base case we need both the positive and the negative version of the concept. For the inductive case we apply Lemma 5.11. □

Theorem 3.12. *Let C be a concept, and KB be a TBox. If the tableau for C and KB can be extended to a tableau with an open branch then C is satisfiable for the TBox KB.*

Proof. Suppose that the tableau for C can be extended to a tableau with an open branch. Let $\mathcal{B}$ be this branch. We prove the claim by exhibiting a model for KB which satisfies C.

First, construct the canonical interpretation $\mathcal{I}$ of $\mathcal{B}$ according to Definition 5.7.

The tableau for C starts with $1:\{C\}$ and by construction $\iota(1) \in \Delta$. From Theorem 5.12 we have $\iota(1) \in C^{\mathcal{I}}$. Hence the interpretation $\mathcal{I}$ satisfies C.

We have to show that $\mathcal{I}$ also satisfies the TBox KB. Observe that $\mathcal{B}$ is PS-implicitly reduced and therefore for every inclusion $C \sqsubseteq D \in KB$ and every prefixed set $e : \mathcal{C}$ one has that either $(\neg C \sqcup D) \in \mathcal{C}$ or there is a node such that $e : \mathcal{D}$is present and $(\neg C \sqcup D) \in \mathcal{D}$. By Theorem 5.12 we have that $\iota(e) \in (\neg C \sqcup D) \in \mathcal{D}^{\mathcal{I}}$. By (1), for every $d \in \Delta$ there is at least one prefixed set $e : \mathcal{D}$ in the branch $\mathcal{B}$ such that $\iota(e) = d$. Hence the interpretation $\mathcal{I}$ validates $(\neg C \sqcup D)$. □

6. Complexity analysis

In this section we prove our main result on complexity, namely, that the algorithm DFS takes single exponential time.

From a high-level perspective, the proof is arranged in four parts:

(1) We set the definitions and some nice basic properties of the call tree of the recursive calls of DFS.
(2) We prove that the height of the call tree is bounded: the size of the largest branch is bounded by a single exponential in the size of the TBox and the initial concept. Without further work, this would only yield an unsatisfactory doubly exponential bound on the call tree.
(3) We prove that the width of the call tree is bounded: the number of branches of the call tree (not to be confused with the branches of the tableau) is bounded by a single exponential in the size of the TBox and the initial concept.
(4) Finally, we combine all bits together and get our single exponential time bound.

The third step is the trickiest. The way in which we prove it is to assign to each branch, or to a fraction of them, a distinguished set of concepts that “characterizes” the branch of the call tree. Since the number of distinct sets of concepts is bounded by the size of the TBox, we obtain the desired result.

This is the place where $\perp$-sets play their role. Ideally, we would like to associate to each $\perp$-set, the first branch of the call tree where it is introduced into NoGoods. Even though a $\perp$-set might be introduced many times in NoGoods, and the same $\perp$-set may be used to stop the search in many DFS-calls, the idea would be that when DFS returns from previous calls, sooner or later a new $\perp$-set is inserted in NoGoods.

Unfortunately, it turned out that it is not possible to assign a distinguished $\perp$-set to each branch of the DFS call tree, nor to a constant fraction of them. However, we can assign a different $\perp$-set to each "right" branch of the call tree.

Intuitively, this is accomplished in three steps.

(1) We identify the "right" branches of the call tree as those where the deepest choice is "going to the right". Clearly, for every branch that goes right, the number of "offshoot" branches going "left-only" is at most as big as the size of the right branch itself. This gives a total bound of the size of the tree.

(2) Each "right" branch is identified by a special *key branching point* in the tree. Remarkably, each key branching point is not just the last branching point but can be fairly up in the tree (see Fig. 9). So the identification of key branching points is the less intuitive part of our construction.

(3) To each key branching point we can associate a different $\perp$-set in NoGoods. Intuitively, when leaving a key branching point we know that some new $\perp$-set is introduced in NoGoods, even if we do not know exactly when.

Throughout the section, we always use n to denote the size of the input—that is, the number of symbols in the TBox *KB* and in the concept C.

6.1. *Preliminary definitions and properties of the* DFS-*call tree*

We start by recalling some terminology for recursive programs.

Definition 6.1. We define the *call tree* $T(P)$ of a recursive procedure P as follows:

- each node N is one-to-one with an invocation of P;
- a node N_1 has an immediate successor node N_2 if the invocation of P related to N_1 calls the invocation of P related to N_2.

A *branching point* is a node with more than one immediate successor (i.e., at least two recursive calls are made), and its related invocation is a *branching call*.

A *leaf* is a node without successors (i.e., there are no further recursive calls), and its related invocation is a *leaf call*.

We call *successor* the transitive closure of the immediate successor relation, and *predecessor* the inverse of successor.

Note that when the procedure is DFS, $T(\text{DFS})$ is a binary tree. In this case, a branching point is an invocation of DFS in which the control flow passes through the test A4, and the test is successful. We call *left immediate successor* of the branching point the first call of DFS, *right* immediate successor the second call. A *left successor* is either the left immediate successor, or a successor of the left immediate successor. A right successor is defined similarly.

Remark 6.2. The call tree is not binary for "standard" trace-based techniques which reduce $\pi(R)$ formulae by making "parallel" recursive calls to the satisfiability testing procedure for each modal successor, and backtrack (locally) as soon as one of these returns "unsatisfiable". In that case, the control structure is much more complex as the call tree is a generic one.

We now prove two properties about the calls of DFS, which we use later on.

Lemma 6.3. *If a* DFS *invocation selects* $e:\mathcal{C}$, *applies a rule different from* ν, *and returns* e, *then* $\mathcal{C}$ *has been inserted in* NoGoods *before exiting the invocation.*

Proof. Simply by inspection over all cases of the algorithm DFS. □

Lemma 6.4. *If a* DFS *invocation* N *selects* $e:\mathcal{C}$ *and applies a rule* $\nu(R)$, *then* N *does not return the prefix* e.

Proof. According to Technique 6 in the search strategy, every prefixed set $e:\mathcal{C}$ is selected before any prefixed set with an extended prefix $e.R.n$ is selected. Hence a $\nu(R)$ rule can be applied only after at least one $\pi(R)$ rule has been applied, and a $\pi(R)$ rule is applied only when all α, β, *KB* and *dneg* rules have been applied (Technique 4). Note that the application of either $\pi(R)$ or $\nu(R)$ rules to the prefixed set $e:\mathcal{C}$ does not add a prefixed set $e:\mathcal{C}'$ with the same prefix. Thus, other prefixed sets $e:\mathcal{C}'$ cannot be added in the calls that are successors of N. Therefore, e cannot be returned by N. □

To prove our main result we need to introduce other properties of $T(\text{DFS})$. We start by defining some particular branches and branching points in $T(\text{DFS})$.

Definition 6.5. A DFS-*branch* is the sequence of invocations of DFS related to the nodes of a path from the root to a node in $T(\text{DFS})$. A DFS-branch is *complete* if its path ends in a leaf.

The *deepest branching point* of a complete branch is a branching point which has no successor which is a branching point.

A *complete right* DFS-*branch* is a complete DFS-branch containing the right successor of its deepest branching point.

A *key branching point* is a branching point such that both its left successor and its right successor belong to a complete right DFS-branch.

For a better understanding of these definitions, refer to Fig. 9. It depicts a possible $T(\text{DFS})$. Branches are numbered from left to right and right branches are marked by a capital R in their leaf call. We show some of its deepest branching points (e.g., D1, D5, D8, etc.) and mark *all* key branching points with a black dot (K3, K7, ...).

Remark 6.6. Each complete branch might be characterized by a different deepest branching point and indeed the same branching point might be the deepest one for a branch including its left successor but not for the branch including the right successor and vice-versa.

For instance, D8 is not the deepest branching point for branch 7, but it is the deepest one for branch 8; D1 is the deepest branching point for branch 1 but not for branch 2.

Remark 6.7. A key branching point may not be a deepest branching point.

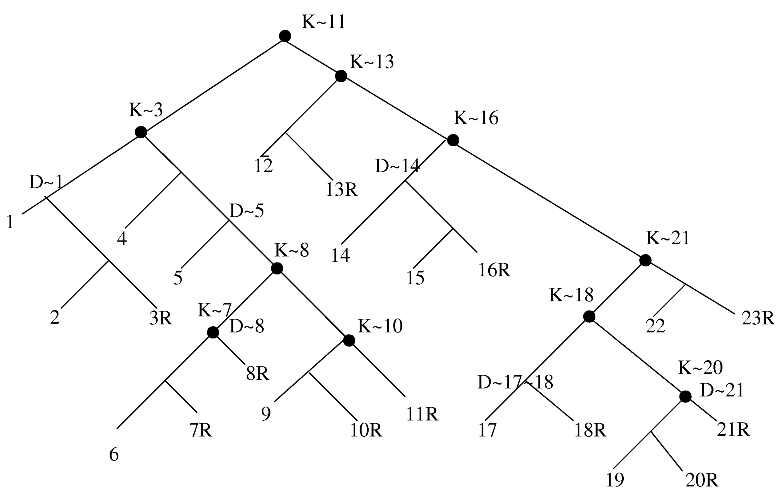


Fig. 9. An instance of $T(\mathrm{DFS})$.

For instance, K3 is a key branching point because its left successor can be extended to a right branch (3R) and also its right successor can be extended to a right branch (e.g., 10R). Yet, K3 is not a deepest branching point for any branch.

Remark 6.8. Still, there is a bijection between each right branch and each key branching point, but for the last right branch of the whole tree (23R). Intuitively, we can detect that a branching point is a key branching point if we go immediately down to the left and then we can go down to the right at least once. The right branch associated to a key branching point can be found by going immediately to the left and then always to the right.

Thus, the key branching point K3 corresponds to the right branch 3R, K8 corresponds to 8R, K11 (the root) corresponds to 11R and so on: Kn corresponds to the the branch nR.

In Fig. 9 we have named each key branching point according the corresponding branch. However, this is not the order in which they are inserted in the stack by DFS. In this respect, we know that a DFS call is a branching point only after the call enters the right branch, because of test A4 in the DFS algorithm. Thus, although K11 is the first key branching point to enter in the stack, the first call which issues right successor call is K3, then K7, then K8, K10, and only after this stage the control returns up to the right successor of K11.

Fig. 10 shows the snapshots of the stack execution of DFS for what regards key branching points: On top of the stack we write down the branches that are visited while these key branching points are in the stack. We underlined a key branching point when the control has returned to the invocation and it is now doing the right branch(es), i.e., when DFS is reducing β_2 after having passed the A4-check.

From Figs. 9 and 10, one may notice that the number of branches visited by the algorithm may vary whereas the key branching points in the stack are not changed. This means that other DFS calls are placed in the stack (we have just not shown them), and the key issue is how many they are.

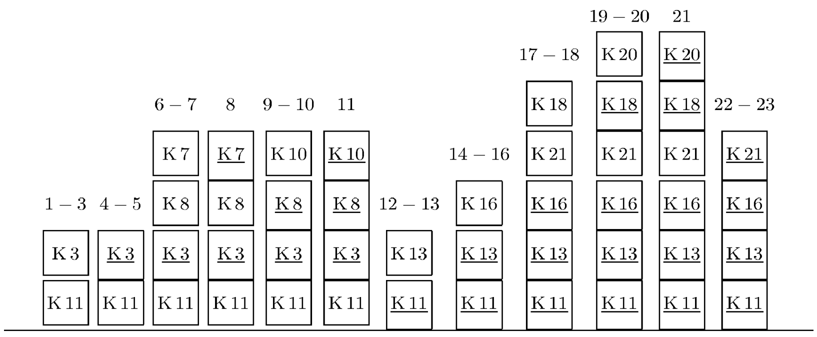


Fig. 10. Snapshots of the key branching points in the stack from Fig. 9.

For instance, we could add many left offshoots branches between the points D5 and K8 in Fig. 9. From the DFS point of view it may simply mean that we keep on closing left branches using the same $\perp$-sets. However, we cannot keep adding offshoots branches *ad libitum*. The maximum number of left-only branches is bounded by the size of the right branch from which they spring. As soon as we add a right branch a new key branching point is added.

As another example, consider the branches below D17 18 in Fig. 9. We might add branches going left off the segment (D17,18R) without changing the number of key branching points. Our only bound is the size that the segment from D17 to 18R can have. However, as soon as we add a branch going right off the segment (D17,17), this transform D17 into a key branching point for that branch (see Remark 6.8).

Note also that a DFS-branch in $T(\text{DFS})$ corresponds to a tableau branch $\mathcal{B}$, where $\mathcal{B}$ is the argument of the last DFS invocation. A complete DFS-branch corresponds to a tableau branch (the argument of the leaf call) which is either implicitly closed (the leaf call returns a prefix) or reduced and open (the leaf call returns *sat*).

When no confusion arises, we say that "$e:\mathcal{C}$ occurs in a DFS-branch" meaning that $e:\mathcal{C}$ occurs in the input branch of an invocation of that DFS-branch.

6.2. *Bounding the height of the* DFS *call tree*

Lemma 6.9. *The recursion depth in a complete* DFS-*branch of* $T(\text{DFS})$ *is* $\mathrm{O}(n^2 \cdot 2^n)$, *where n is the size of the input concept and the TBox.*

Proof. We first give an $\mathrm{O}(n^2)$ bound on the number of calls selecting a prefixed set containing a given prefix, then we give an $\mathrm{O}(2^n)$ bound on how many different prefixes can be selected by successive calls in the recursion stack.

Consider a given prefix e. In a DFS-branch, the calls selecting some prefixed set $e:\mathcal{C}$, are at most $\mathrm{O}(n^2)$. In fact, rule *KB* can add $\mathrm{O}(n)$ concepts to $\mathcal{C}$, and each of the rules α, β, *dneg* can be applied $\mathrm{O}(n)$ times, since they progressively reduce the size of the concepts in $e:\mathcal{C}$ that can be chosen for further reduction. Rule $\pi(R)$ can be applied $\mathrm{O}(n)$ times (number of $\pi(R)$ subconcepts), and for each application of $\pi(R)$, rule $\nu(R)$ can be applied $\mathrm{O}(n)$ times (number of $\nu(R)$ subconcepts).

Regarding the number of different prefixes, recall that once the rule $\pi(R)$ has been applied as many times as possible, a new set $\mathcal{C}$ is added to Visited (point A5 in Fig. 6). From condition A3 in the algorithm, no call up in the recursion stack can then choose a prefixed set $e' : \mathcal{C}$ with the same set of concepts $\mathcal{C}$. Hence in the stack the calls can choose as many different prefixed sets as the number of possible sets of concepts, that is $\mathrm{O}(2^n)$. □

Of course, the above bound is still insufficient to prove that the total number of calls (the size of $T(\mathrm{DFS})$) is exponential. Hence, we prove other properties of $T(\mathrm{DFS})$, which will be useful to reach our goal.

6.3. *Bounding the width of the* DFS *call tree*

Proposition 6.10. *If an invocation N of* DFS *selects $e : \mathcal{C}$, and inserts $\mathcal{C}$ in Visited, then $e : \mathcal{C}$ is a witness for every other node $e' : \mathcal{C}$ occurring in every* DFS*-branch containing N.*

Lemma 6.11. *For every invocation* $\mathrm{DFS}(\mathcal{B})$ *corresponding to a node N in $T(\mathrm{DFS})$, if $e : \mathcal{C} \in \mathcal{B}$ has a witness in $\mathcal{B}$ then the prefix e will never be returned by any (subsequent)* DFS *invocation in any* DFS*-branch containing N.*

Intuitively, this lemma says that prefixed sets responsible for the insertion of a $\perp$-set in NoGoods (i.e., prefixed sets whose prefix is returned) are always “originals” and never “copies”.

We remark that we could prove a stronger lemma, namely, that e will never be returned by any DFS call in the *entire* $T(\mathrm{DFS})$. However, in the following we need just the above weaker version of the lemma.

Lemma 6.12. *Let N be a call in $T(\mathrm{DFS})$, and let N select a prefixed set with prefix e. Then no call successor of N can select a prefixed set with prefix $e' \prec e$.*

Proof. The claim follows from Technique 6, and the observation that every rule, applied to $e : \mathcal{C}$, introduces a new prefixed set with either the same prefix e, or with a longer prefix $e.R.n$, for some n. □

Remark 6.13. The above claim would not hold if the description logic included inverse roles.

Lemma 6.14. *Suppose N is a branching point, selecting $e : \mathcal{C} \cup \{\beta\}$, and let N_1 be the* DFS *call N_1 which is the left immediate successor of N. Then N_1 selects $e : \mathcal{C} \cup \{\beta\} \cup \{\beta_1\}$.*

Proof. Observe that if N is a branching point then $\mathcal{C} \cup \{\beta\} \cup \{\beta_1\}$ is not reduced. Indeed, suppose N_1 selects a prefixed set with a different prefix e'. Since the selection function applies a lexicographic ordering for selecting prefixed sets, then by Lemma 6.12, no successor call of N_1 selects a prefixed set with prefix e. Hence N_1 does not return e, and N is not a branching point from condition A4 in DFS, contradicting the hypothesis. □

Now we prove the key property of the DFS algorithm: every key branching point is one-to-one with a different set of concepts inserted in NoGoods.

Theorem 6.15. *Let N be a key branching point, and let $e : \mathcal{C} \cup \{\beta\}$ be the prefixed set selected by N. Let N_0 the left immediate successor of N, and let $N_1, \ldots, N_k$ be all successor calls of N_0 returning e, in the rightmost* DFS*-branch passing through N_0. Then, there is at least one set of concepts $\mathcal{D}$ such that*:

(1) *$\mathcal{D}$ was not in* NoGoods *when N_0 started*;
(2) *$\mathcal{D}$ is not inserted in* NoGoods *by any* DFS *call successor of N_0, and different from $N_1, \ldots, N_k$*;
(3) *$\mathcal{D}$ is inserted in* NoGoods *by at least one among $N_0, N_1, \ldots, N_k$.*

Proof. For sake of clarity, the DFS calls described in the statement of the theorem are pictorially represented in Fig. 11.

First of all, note that N_0 returns e, otherwise N would not be a branching point because of condition A4 in the algorithm of DFS. Secondly, N_0 cannot be a leaf call, otherwise N would not be a key branching point. Then, N_0 must select a prefixed set, and issue at least one recursive call.

From Lemma 6.14, N_0 selects $e : \mathcal{C} \cup \{\beta\} \cup \{\beta_1\}$. Since it also returns e, Lemma 6.4 implies that N_0 does not apply a $\nu(R)$ rule. Then, from Lemma 6.3 it follows that N_0 inserts $\mathcal{C} \cup \{\beta\} \cup \{\beta_1\}$ in NoGoods. Now we analyze two cases for k:

Suppose $k = 0$. This means that there are no other DFS calls that return e in the rightmost DFS-branch passing through N_0. Then, since N_0 selects $e : \mathcal{C} \cup \{\beta\} \cup \{\beta_1\}$ and returns e, the only rule that N_0 can apply is the π rule. In this case, from point A5 of the DFS algorithm, N_0 inserts $\mathcal{C} \cup \{\beta\} \cup \{\beta_1\}$ in Visited. From Lemma 6.11, no other successor call of N_0 inserts $\mathcal{C} \cup \{\beta\} \cup \{\beta_1\}$ in NoGoods. Hence, the claim holds with $\mathcal{D} = \mathcal{C} \cup \{\beta\} \cup \{\beta_1\}$.

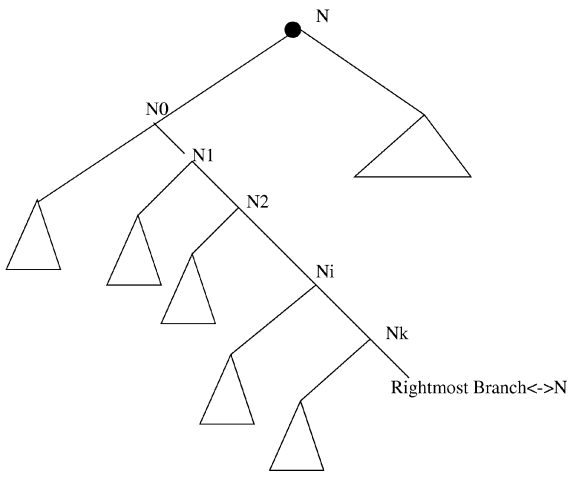


Fig. 11. The key branching point N and its left subtree in Theorem 6.15.

Suppose $k \geqslant 1$. We distinguish two sub cases: first, N_k is not a leaf call; second, it is a leaf call.

(1) Suppose N_k is not a leaf call. Then there exists a call N_{k+1}, successor of N_k, and such that N_{k+1} returns $e' \neq e$. Since N_k returns e, by inspection on the rules in DFS this can only happen if N_{k+1} returns $e' = e.R.n$, and N_k applied a π rule, selecting $e : \mathcal{D}$ and inserting $\mathcal{D}$ in Visited. Then from Lemma 6.11, no other successor call of N_k inserts $\mathcal{D}$ in NoGoods, and N_k inserts $\mathcal{D}$ in NoGoods.

(2) Suppose N_k is a leaf call. Since N is a key branching point, there must be at least one branching point between N_0 and N_k: let us denote N_b the deepest (in $T(\text{DFS})$) branching point between N_0 and N_k, and let N_b select $e : \mathcal{D}_b$. By hypothesis, N_b returns e, and hence it inserts $\mathcal{D}_b$ in NoGoods.
Recall that $N_1, \ldots, N_k$ are nodes in the *rightmost* DFS-branch passing through N_0. Hence, $N_{b+1}, \ldots, N_k$ are all right successors of N_b. From point A2 in the algorithm of DFS, $\mathcal{D}_b$ was not inserted in NoGoods by any call successor of N_0 which is issued before N_b. From point A4, $\mathcal{D}_b$ was not inserted in NoGoods by any call which is a left successor of N_b. Moreover, every right successor $N_{b+1}, \ldots, N_k$ of N_b adds one or more subconcepts to the prefixed set $e : \mathcal{D}_b$. Hence, each of them inserts in NoGoods a set of concepts which is different from $\mathcal{D}_b$. Therefore, the claim of the theorem holds for $\mathcal{D}_b$. □

Theorem 6.16. *The number of key branching points in* $T(\text{DFS})$ *is bounded by* 2^n.

Proof. From the previous theorem, a new set of concepts is inserted in NoGoods for every immediate left successor of a key branching point. Since there are at most 2^n different sets of concepts this is also a bound for the number of key branching points. □

6.4. EXPTIME-*efficiency of the* DFS *algorithm*

We can now put all results together.

Theorem 6.17. *The total number of* DFS *calls in* $T(\text{DFS})$ *is* $O(2^{cn})$, *for a suitable constant* $c > 1$.

Proof. Observe that, starting DFS with a concept that is $C \sqcup \bot$, the number of key branching points equals the number of complete right branches minus one. From the previous Theorem 6.16, the number of right branches is $O(2^n)$.

From Lemma 6.9 the recursion depth of a DFS-branch is $O(n^2 \cdot 2^n)$. This means that for every right branch we can have at most $O(n^2 \cdot 2^n)$ left branches which do not introduce also a new right branch (at worst one for each invocation of DFS in the right branch).

Hence the total number of branches is $O(n^2 \cdot 2^n \cdot 2^n)$ and the total number of calls is $O(n^2 \cdot 2^n \cdot 2^n \cdot 2^n)$, that is, $O(2^{cn})$ for a suitable c. □

As a straightforward corollary of Theorem 6.17 we have the main result:

Theorem 3.13. *Any search strategy respecting Techniques* 1–7 *terminates using single deterministic exponential time in the size of C and the TBox KB.*

Now suppose that we wish to enhance our algorithm by storing and testing for membership only sets composed by modal atoms and unreduced concepts. The above proofs could be adapted to the new algorithm based on the following argument. Each time a new $\perp$-set is inserted in NoGoods, what makes it new is the fact that it contains a new unreduced formula. There are only two exceptions to this observation: the $\nu(R)$ rule (this was the case already without the optimization) and the KB rule. But Lemma 6.9 implies that both $\nu(R)$ and KB rules can only be applied a polynomial number of times for a given prefix.

7. Overview of related methods

Although a large body of results exists on the EXPTIME-completeness of many description logics (see, e.g., [14,15]), most of these results are of theoretical nature: they do not offer a direct decision procedure but just exhibit polynomial translation of satisfiability of extensions of $\mathcal{ALC}$ into Propositional Dynamic Logic, following the ground breaking work by Schild [68].

The classical procedures that match the exact EXPTIME-bound (such as automata on infinite trees [77] or model graph [38,66]) have the unfortunate property of being also "best case" exponential, because they construct tableau structures bottom-up. As we have said, this happens because one first constructs an automaton which accepts the tree models of KB and C, and whose size is exponential in the size of the input. Then, one checks its emptiness, i.e., whether the automaton does not accept any model (see [13,24,31]).

From the proper automated reasoning perspective, a large number of calculi for modal and description logics have been proposed. However, only a limited number of these works concentrate on the aspects of algorithmic complexity.

In the realm of description logics, the complexity analysis of tableau-based algorithms for $\mathcal{ALC}$ was pioneered in [70], where only the PSPACE-satisfiability case without global axioms is considered. More recent works have extended the calculus to deal with additional constructs such as inverse or transitive roles (see, e.g., the works of Horrocks, Sattler, and Tobies [41,44,45,75]) but have not dealt with the algorithmic complexity of reasoning with global axioms. For instance, in [45] a calculus for an EXPTIME-complete logic including $\mathcal{ALC}$ is given. However, complexity results are only shown for a PSPACE-fragment of the logic. For EXPTIME-completeness results they refer to De Giacomo and Massacci on dynamic logic [16].

The only work where the complexity of decision procedures with global axioms has been investigated is the work by Buchheit, Donini, and Schaerf [8], where a calculus working in nondeterministic exponential time is given, and a modification of it working in single exponential time is just foreseen. A preliminary announcement of the results showed in this paper was also reported in [20].

The classical studies on the complexity of (variants of) the sequent or tableau calculus for multi-modal logic K (a notational variant of $\mathcal{ALC}$), such as those by Ladner [51] or by Halpern and Moses [38] only focus on satisfiability without global axioms.

Recent works on prefixed tableaux such as those by Massacci [54,57] or extended sequent calculi such as Hürding et al. [39], Demri [18], or Basin et al. [4] again only give bounds for satisfiability without global axioms, although they discuss a larger set of logics than simply modal logic K.

The idea of using a set of "no-goods" for transforming NEXPTIME-tableau calculi into EXPTIME-algorithms has been given by De Giacomo and Massacci [16] for Propositional Dynamic Logic but an explicit algorithm has not been given there, and that calculus is complicated by the need to accommodate both the iteration and converse operator.

Studies based on translation into first order logic *à la* Ohlbach [62] have mainly dealt with the problem of decidability [47,69], and only recently there has been an attempt to discuss the actual proof complexity of the resolution method [12,47]. Also, these approaches do not study the complexity of satisfiability with global axioms. For instance, Hustadt and Schmidt have just shown that it is possible to use resolution as a decision procedure for $\mathcal{ALC}$ with TBoxes [47] by means of a step-by-step simulation of prefixed tableau proof search.

This latter result is particularly interesting as it allows to transfer our complexity result to translation-based methods. It is enough to impose that the simulated tableau proof respects our proof search techniques. However, as $\perp$-sets corresponds to derived clauses in the resolution framework, it might be that features such as clause subsumption or clause deletion by resolution theorem provers may affect adversely the key result which requires $\perp$-sets to be used as much and as soon as possible.

There has been also a substantial work on the implementation of efficient theorem provers for Description Logics including $\mathcal{ALC}$—among others, FaCT [41,43,44], DLP [43,64], HAM-ALC (now RACE) [37], KRIS [3], KSAT [32–34]. These implementations are usually based on the standard trace based technique, with a different emphasis on the various optimizations employed (see again Section 4 for a presentation of some of them).

Setting optimizations aside, usual tableau strategies (which explore one disjunctive branch at a time) are applied, and in exploring a branch there is no use of inconsistent sets of concepts already discovered in another branch (unless the "caching" optimization is employed). Moreover, existential concepts are reduced by making "parallel" recursive calls to the satisfiability testing procedure for each modal successor, and by backtracking (locally) as soon as one of these returns unsatisfiable. These characteristics are not present in our algorithm.

So, an interesting question is whether our algorithm combined with modal backjumping could be recasted into the standard trace-based technique. The reason that our algorithm does not employ that technique is precisely that we could *not* prove the worst-case complexity result with it. Indeed, the standard trace-based techniques has been widely present in the literature, but only for proving PSPACE, NEXPTIME, or decidability results. In contrast, up to now only bottom-up techniques were used for proving EXPTIME bounds. The odds seem against such equivalence.

For what regard the re-use of concepts during proof search, this is typically captured by the so called "caching" optimization. For DLP, "caching" of the satisfiability status of all encountered sets of concepts is claimed [43,64]. Other systems such as RACE and FaCT cache different information using pseudo-models of satisfiable concepts (see, e.g., [37, 42]), KSAT most recent incarnation uses different bounded caches [33], FaCT even disables caching of satisfiability status, claiming that it adversely affects the performance on a the particular knowledge base one is interested in classifying [42].

The most difficult problem is that the caching optimizations are left out of the formal descriptions, and this makes it difficult even to ascertain whether two different implementations mean the same thing for "caching". As more important consequence of the lack of a formal treatment, it has never been formally proved whether the different caching optimizations are sound (see for instance [37] for an example of unsound caching) and whether they can provide the desired EXPTIME-upper bound.

For what regard other optimizations, even though most systems employ propositional backjumping (see [43,45] for a discussion of the problems relative to its implementations), there is no evidence that the optimization that we call "modal backjumping" has been implemented. As we have already remarked, this is mostly due to the different structure of our algorithm which does not employ "parallel" recursive calls for analyzing modal successors.

Our work distils and formalizes many of the intuitions and techniques present in modal and description logic literature and provides the logical and algorithmic rationale of a method that works in single exponential time, which might have been implemented (or might be easily implementable) in DLP or in similar systems.

Notice that we have *not* proved that "caching" all sets of concepts (including potentially satisfiable concepts) is a sound procedure. Indeed, in our calculus we "permanently cache" all and *only* unsatisfiable sets of concepts; many potentially satisfiable sets of concepts are discarded when passing from a branch to another branch. Storing all sets of concepts might lead to an unsound calculus (see [37] for examples).

8. Conclusions

In this paper we have presented the first tableau-based algorithm for satisfiability of a concept with respect to a TBox (and hence also for subsumption in a TBox) which works in worst-case single exponential time. In fact, we do not need to change substantially the "normal" construction used by tableaux which has proven to be reasonably effective in practice [41,43]. The key point is to forfait in part the standard trace-based technique and make use of an auxiliary data structure which is used to store sets of concepts whose conjunction was already proved to be inconsistent. Nevertheless, as it can be seen from the machinery of Sections 5 and 6, the proofs that the new reasoning method is correct, and that it indeed requires single exponential time, were neither simple, nor short.

The main ideas behind our algorithms can be used to devise EXPTIME-tableau implementations for various extension of $\mathcal{ALC}$. In particular, since our calculus is tableau-based, it can be easily modified to deal with an ABox as well. Moreover, it is possible to export our complexity result to translation-based methods, by using the step-by-step simulation of the proof search between prefixed tableaux and ordered resolution by Hustadt and Schmidt [47].

This work can be extended in many directions: on the deductive side one may work for extending the logic, and in particular to accommodate individuals or the converse and the star (transitive closure) operators. Regarding complexity of optimizations (see Sections 4.2, 4.3), one may prove that the storage of minimal inconsistent subsets and the use of subset checking, which seems to be more promising in practice, also yields a single exponential decision procedure.

Another promising avenue of research is the generalization of our complexity analysis of the DFS-algorithm to AND/OR graphs and its transformation into algorithms for checking on-the-fly the emptiness of accepting automata for EXPTIME-complete logics.

Acknowledgements

We started this research with "Guiseppe" De Giacomo, whom we are indebted with for inspiration and exhortation. We thank E. Giunchiglia, I. Horrocks, U. Hustadt, P. Patel-Schneider, R. Schmidt, and R. Sebastiani for many useful discussions on modal and description logics theorem proving, M. Vardi for pointing us the subtle relations between our result and automata-theoretic works, and F. d'Amore for useful discussions about data structures for dictionaries. Comments from the anonymous referees were helpful in improving the presentation and clarifying the difficult steps.

Francesco Donini acknowledges the Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", in particular the research project CNR-LAICO, and the CNR research projects "DeMAnD" and "Metodi di Ragionamento Automatico per l'Analisi di Dominio" for supporting this research. Fabio Massacci acknowledges the support of the CNR scholarship 201-15-9.

Appendix A. A full worked example

We test the satisfiability of the concept $A \sqcup \exists P.A$ against the knowledge base $KB = \{A \sqsubseteq \exists Q.B \sqcap \exists R.C_\perp, B \sqsubseteq \exists P.A\}$, where $C_\perp$ denotes an unsatisfiable concept which is not trivially unsatisfiable (e.g., by normalization and simplification).

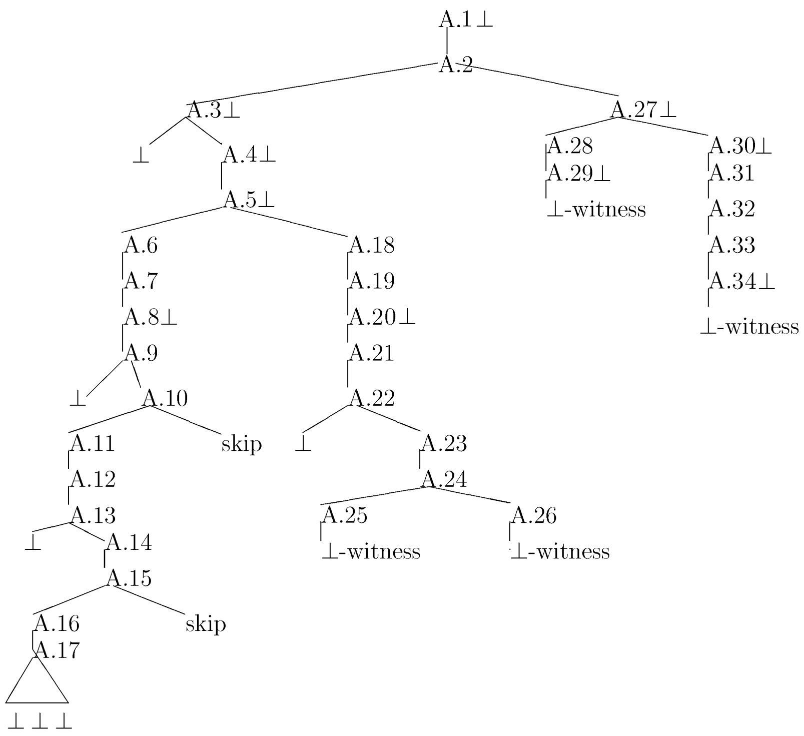
This simple knowledge base is difficult enough to make it interesting as case study. Indeed, the knowledge base is cyclical and the length of the cycle is longer than one and goes through different roles. Therefore simple rewriting techniques do not help, and termination is not guaranteed without loop checking.

To improve the readability, we do not present the whole tableau as a tree. Rather we present the deduction steps as the algorithm performs them and recapitulate the overall structure of the proof tree in Fig. A.1. The tree corresponds to the call tree of the DFS algorithm.

We summarize in Table A.1 and in Table A.2, respectively, the sets which are inserted into NoGoods and Visited during the search process.

We skip uninteresting rules such as the addition of assumptions from the *KB*. We also skip the trivial reduction of a β rule when the extension of the left branch with β_1 generates a clash.

In the proof fragments below we do *not* show reduced concepts although they are to be considered present. Their existence is reminded by ellipsis $(\ldots)$. Moreover we box the concepts that are added. We apply here the simple optimization of storing only modal atoms and unreduced concepts in NoGoods and in Visited concepts.



Numbers in the picture denote the corresponding invocation of the DFS algorithm. The remaining symbols are interpreted as follows:

- "⊥" in a leaf means that a clash has been found;
- "⊥" in an internal node means that a new set of concepts has been inserted in NoGoods when the DFS algorithm returned to that call;
- "⊥-witness" in a leaf denotes an application of step A2 of the DFS algorithm;
- "skip" denotes an application of modal backjumping.

Fig. A.1. The call tree of the DFS-algorithm.

A.1. A step-by-step execution trace of DFS

The tableau starts by adding the concept we want to test for satisfiability:

$$1:\left\{\boxed{A\sqcup\exists P.A}\right\}\tag{A.1}$$

We can either reduce the β concept or add the axioms from the knowledge base. According our strategy, we apply twice the *KB* rule:

$$1:\left\{A\sqcup\exists P.A,\boxed{\neg A\sqcup(\exists Q.B\sqcap\exists R.C_\perp)},\boxed{\neg B\sqcup\exists P.A}\right\}\tag{A.2}$$

Table A.1
Insertion and usage of NoGoods in the search

NoGoods	Added at call	Used at call
$\{C_\bot\}$	(A.17)	(A.26)
$\{A, \neg B, \exists R.B, \exists R.C_\bot\}$	(A.8)	(A.25)
$\{A, \exists Q.B, \exists R.C_\bot, \exists P.A\}$	(A.20)	
$\{A, \exists Q.B, \exists R.C_\bot\}$	(A.5)	
$\{A, \exists Q.B \sqcap \exists R.C_\bot\}$	(A.4)	
$\{A\}$	(A.3)	(A.29, A.34)
$\{\exists P.A, \neg A\}$	(A.29)	
$\{\exists P.A, \exists Q.B, \exists R.C_\bot\}$	(A.34)	
$\{\exists P.A, \exists Q.B \sqcap \exists R.C_\bot\}$	(A.30)	
$\{\exists P.A\}$	(A.27)	
$\{A \sqcup \exists P.A\}$	(A.1)	

Table A.2
Insertion and usage of Visited in the search

Visited	Added at call	Used at call	Discarded at call
$\{A, \exists Q.B, \exists R.C_\bot, \neg B\}$	(A.8)	(A.16)	(A.18)
$\{B, \neg A, \exists P.A\}$	(A.12)		(A.18)
$\{A, \exists Q.B, \exists R.C_\bot, \exists P.A\}$	(A.21)	(A.26)	(A.27)

Now we can apply the β-rule to two different concepts. Without loss of generality, we choose the first concept of the set $A \sqcup \exists P.A$. Then we generate two branches and the left one, with β_1, continues as follows:

$$1 : \left\{ \boxed{A}, \ldots, \neg A \sqcup (\exists Q.B \sqcap \exists R.C_\bot), \neg B \sqcup \exists P.A \right\} \tag{A.3}$$

Recall that the reduced concept $A \sqcup \exists P.A$ is replaced by ellipsis.

A further β-reduction of $\neg A \sqcup (\exists Q.B \sqcap \exists R.C_\bot)$ yields another branch. The resulting node $1 : \{A, \ldots, \boxed{\neg A}, \ldots, \neg B \sqcup \exists P.A\}$ contains a clash, and the DFS algorithm returns the element 1.

Then we come back at point (A.3) above and start the right branch, i.e., reduce β_2 out of $\neg A \sqcup (\exists Q.B \sqcap \exists R.C_\bot)$. We apply an α-rule and we get:

$$1 : \left\{ A, \ldots, \boxed{\exists Q.B \sqcap \exists R.C_\bot}, \ldots, \neg B \sqcup \exists P.A \right\} \tag{A.4}$$

$$1 : \left\{ A, \ldots, \boxed{\exists Q.B}, \boxed{\exists R.C_\bot}, \ldots, \ldots, \neg B \sqcup \exists P.A \right\} \tag{A.5}$$

Then we apply again the β-rule to $\neg B \sqcup \exists P.A$ and insert β_1.

$$1 : \left\{ A, \ldots, \exists Q.B, \exists R.C_\bot, \ldots, \ldots, \boxed{\neg B}, \ldots \right\} \tag{A.6}$$

Now we can choose to expand either $\exists Q.B$ or $\exists R.C_\perp$. Suppose that our heuristic function is unlucky and we proceed by applying the π-rule to $\exists Q.B$ and $\exists R.C_\perp$.

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \\ \boxed{1.Q.2} &: \left\{\boxed{B}\right\} \end{aligned} \tag{A.7}$$

$$\begin{aligned} 1 &: \{A, \exists Q.B, \exists R.C_\perp, \neg B\} \\ 1.Q.2 &: \{B\} \\ \boxed{1.R.3} &: \left\{\boxed{C_\perp}\right\} \end{aligned} \tag{A.8}$$

At step (A.8) the DFS algorithm inserts the set $\{A, \exists Q.B, \exists R.C_\perp, \neg B\}$ in Visited.

Then, we add the axioms from the KB and we arrive at the stage

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \\ 1.Q.2 &: \left\{B, \boxed{\neg A \sqcup (\exists Q.B \sqcap \exists R.C_\perp)}, \boxed{\neg B \sqcup \exists P.A}\right\} \\ 1.R.3 &: \{C_\perp\} \end{aligned} \tag{A.9}$$

Again we must apply a β-rule. This time we are luckier (or want just to cut it short), and select the second disjunction $\neg B \sqcup \exists P.A$ for branching.

Once we add $\beta_1 = \neg B$, the node prefixed by $1.Q.2$ contains a clash and the DFS algorithm returns $1.Q.2$ and the search proceeds in the right branch.

$$\begin{aligned} 1 &: \{A, \ldots \exists Q.B, \exists R.C_\perp, \ldots, \ldots \neg B, \ldots\} \\ 1.Q.2 &: \left\{B, \neg A \sqcup (\exists Q.B \sqcap \exists R.C_\perp), \boxed{\exists P.A}, \ldots\right\} \\ 1.R.3 &: \{C_\perp\} \end{aligned} \tag{A.10}$$

Again we have a β concept to reduce in the set prefixed by $1.Q.2$ and the algorithm continues on the left by adding $\beta_1 = \neg A$, as follows:

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \\ 1.Q.2 &: \left\{B, \boxed{\neg A}, \ldots, \exists P.A, \ldots\right\} \\ 1.R.3 &: \{C_\perp\} \end{aligned} \tag{A.11}$$

No more propositional rules are possible. So we apply a $\pi(P)$ rule to $1.Q.2$ and add the new prefix $1.Q.2.P.4$ to the branch.

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \\ 1.Q.2 &: \{B, \neg A, \ldots, \exists P.A, \ldots\} \\ 1.R.3 &: \{C_\perp\} \\ \boxed{1.Q.2.P.4} &: \left\{\boxed{A}\right\} \end{aligned} \tag{A.12}$$

At step (A.12) we insert in Visited the set of concepts prefixed by $1.Q.2$ (Table A.2).

After two applications of the *KB*-rule we get the following branch:

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \\ 1.Q.2 &: \{B, \neg A, \ldots, \exists P.A, \ldots\} \\ 1.R.3 &: \{C_\perp\} \\ 1.Q.2.P.4 &: \left\{A, \boxed{\neg A \sqcup (\exists Q.B \sqcap \exists R.C_\perp)}, \boxed{\neg B \sqcup \exists P.A}\right\} \end{aligned} \tag{A.13}$$

Again we have to apply a β-rule to the prefix $1.Q.2.P.4$ and the algorithm proceeds by branching on the left and adding the $\beta_1 = \neg A$ subconcept of $\neg A \sqcup (\exists Q.B \sqcap \exists R.C_\perp)$. The DFS algorithm detects the clash and returns $1.Q.2.P.4$, so that the proof search continues on the right branch.

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \\ 1.Q.2 &: \{B, \neg A, \ldots, \exists P.A, \ldots\} \\ 1.R.3 &: \{C_\perp\} \\ 1.Q.2.P.4 &: \left\{A, \boxed{\exists Q.B \sqcap \exists R.C_\perp}, \ldots, \neg B \sqcup \exists P.A\right\} \end{aligned} \tag{A.14}$$

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \\ 1.Q.2 &: \{B, \neg A, \ldots, \exists P.A, \ldots\} \\ 1.R.3 &: \{C_\perp\} \\ 1.Q.2.P.4 &: \left\{A, \boxed{\exists Q.B}, \boxed{\exists R.C_\perp}, \ldots, \ldots, \neg B \sqcup \exists P.A\right\} \end{aligned} \tag{A.15}$$

A β-rule to reduce $\neg B \sqcup \exists P.A$ prefixed by $1.Q.2.P.4$ starts the next DFS-call:

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \\ 1.Q.2 &: \{B, \neg A, \ldots, \exists P.A, \ldots\} \\ 1.R.3 &: \{C_\perp\} \\ 1.Q.2.P.4 &: \left\{A, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \boxed{\neg B}, \ldots\right\} \end{aligned} \tag{A.16}$$

DFS notices that the set of concepts prefixed by 1 that we have inserted in Visited (Table A.2) is a witness for $1.Q.2.P.4$. This prefixed set is not reduced further.

We can focus our attention on the prefixed set $1.R.3 : \{C_\perp\}$ because the next prefix in the lexicographic order is $1.R.3$.

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \\ 1.Q.2 &: \{B, \neg A, \ldots, \exists P.A, \ldots\} \\ 1.R.3 &: \left\{\boxed{C_\perp}\right\} \\ 1.Q.2.P.4 &: \{A, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \end{aligned} \tag{A.17}$$

For simplicity, we assume that after a suitable number of steps the proof search terminates, $\{C_\perp\}$ is added to the NoGoods, and the DFS call returns $1.R.3$.

At this stage the DFS algorithms returns over the potential branching point (A.15). The check prevents the potential branch because $1.Q.2.P.4$ is different from the returned prefix $1.R.3$.

Then we go at call (A.13) after the right branch has been visited. Because the returned prefix $1.R.3$ is different from the current prefix $1.Q.2.P.4$ we do not insert anything in the NoGoods and continue. We skip also the potential branching point on the right at step (A.10) because the returned prefix $1.R.3$ is different from $1.Q.2$.

We exit DFS-calls upward without doing any work until we resume call A.8. Then the instruction A6 of the DFS algorithm inserts the $\bot$-set $\{A, \neg B, \exists R.B, \exists R.C_\bot\}$ among the NoGoods. The new returned prefix is now 1.

The algorithm backtracks directly to call (A.5). At this point the condition A4 is true and the search continues in the right branch. Before going to the right, we discard the set $\{B, \neg A, \exists P.A\}$ from Visited.

$$1 : \{A, \ldots, \exists Q.B, \exists R.C_\bot, \ldots, \ldots, \exists P.A, \ldots\} \tag{A.18}$$

It is clear that this second branch is useless, since the disjunction that we are currently analyzing does not contribute to the contradiction. If we used propositional backjumping we could have avoided that branch too. We have already discussed how to add this optimization to the DFS algorithm (see Sections 4.2 and 4.3).

Here we have again to choose a π concept to reduce. For sake of simplicity assume that we again reduce the concepts in the same order which we have chosen in the previous branches: first A, then $C_\bot$, and now we have also B. We get:

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\bot, \ldots, \ldots, \exists P.A, \ldots\} \\ \boxed{1.P.5} &: \left\{\boxed{A}\right\} \end{aligned} \tag{A.19}$$

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\bot, \ldots, \ldots, \exists P.A, \ldots\} \\ 1.P.5 &: \{A\} \\ \boxed{1.R.6} &: \left\{\boxed{C_\bot}\right\} \end{aligned} \tag{A.20}$$

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\bot, \ldots, \ldots, \exists P.A, \ldots\} \\ 1.P.5 &: \{A\} \\ 1.R.6 &: \{C_\bot\} \\ \boxed{1.Q.7} &: \left\{\boxed{B}\right\} \end{aligned} \tag{A.21}$$

Notice that in call A.21 we introduce $\{A, \exists Q.B, \exists R.C_\bot, \exists P.A\}$ in Visited.

The search now continues on the next prefix in the lexicographic order, that is $1.P.5$. As usual, we add *KB*-axioms first.

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\bot, \ldots, \ldots, \exists P.A, \ldots\} \\ 1.P.5 &: \left\{A, \boxed{\neg A \sqcup (\exists Q.B \sqcap \exists R.C_\bot)}, \boxed{\neg B \sqcup \exists P.A}\right\} \\ 1.R.6 &: \{C_\bot\} \\ 1.Q.7 &: \{B\} \end{aligned} \tag{A.22}$$

Then we must apply a β-rule. Again we choose the first disjunction and the branch on the left immediately closes. We are left with the following (right) branch:

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \exists P.A, \ldots\} \\ 1.P.5 &: \left\{A, \boxed{\exists Q.B \sqcap \exists R.C_\perp}, \ldots, \neg B \sqcup \exists P.A\right\} \\ 1.R.6 &: \{C_\perp\} \\ 1.Q.7 &: \{B\} \end{aligned} \tag{A.23}$$

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \exists P.A, \ldots\} \\ 1.P.5 &: \left\{A, \boxed{\exists Q.B}, \boxed{\exists R.C_\perp}, \ldots, \ldots, \neg B \sqcup \exists P.A\right\} \\ 1.R.6 &: \{C_\perp\} \\ 1.Q.7 &: \{B\} \end{aligned} \tag{A.24}$$

We are still left with a β-concept to reduce; we go to the left.

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \exists P.A, \ldots\} \\ 1.P.5 &: \left\{A, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \boxed{\neg B}, \ldots\right\} \\ 1.R.6 &: \{C_\perp\} \\ 1.Q.7 &: \{B\} \end{aligned} \tag{A.25}$$

This is where our use of NoGoods shows its usefulness: since $\{A, \exists Q.B, \exists R.C_\perp, \neg B\}$ is in NoGoods, we do not need to expand this branch any further. We stop the search closing the branch and DFS returns $1.P.5$.

This result *cannot be obtained with semantic branching*. Indeed, if we had used semantic branching in call A.5 we would have obtained the following branch:

$$\begin{aligned} 1 &: \left\{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \exists P.A, \boxed{B}, \ldots\right\} \\ 1.P.5 &: \{A, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \neg B, \ldots\} \\ 1.R.6 &: \{C_\perp\} \\ 1.Q.7 &: \{B\} \end{aligned}$$

Unfortunately, the knowledge that the concept B is satisfied by the prefix 1 is of no avail to close the search at $1.P.5$.

The DFS algorithm returns $1.P.5$ at the call (A.24). We continue with the right branch.

$$\begin{aligned} 1 &: \{A, \ldots, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \exists P.A, B, \ldots\} \\ 1.P.5 &: \{A, \exists Q.B, \exists R.C_\perp, \ldots, \ldots, \exists P.A, \ldots\} \\ 1.R.6 &: \{C_\perp\} \\ 1.Q.7 &: \{B\} \end{aligned} \tag{A.26}$$

Since the set $\{A, \exists Q.B, \exists R.C_\perp, \exists P.A\}$ is in Visited, we do not expand it.

The next prefix in the lexicographic order is $1.R.6$. Now condition A2 is true with the $\perp$-set $\{C_\perp\}$ and the DFS algorithm returns $1.R.6$. We return up to call (A.20), where we insert the set $\{A, \exists Q.B, \exists R.C_\perp, \exists P.A\}$ among the NoGoods and return 1 and go back to call (A.5).

Both right and left branch return the same prefix. Thus a new $\perp$-set is introduced: $\{A, \exists Q.B, \exists R.C_\perp, \neg B \sqcup \exists P.A\}$. Since we do not store axioms of the *KB*, we insert the smaller set $\{A, \exists Q.B, \exists R.C_\perp\}$.

Returning through calls (A.4) inserts also another set in NoGoods (namely the conjunction of the existentials) and finally call (A.3) inserts in NoGoods the $\perp$-set $\{A, \neg A \sqcup (\exists R.B \sqcap \exists R.C_\perp), \neg B \sqcup \exists P.A\}$ for which we only store $\{A\}$.

Now we are back at the root of the tableau and we can branch on the right.

$$1 : \left\{\boxed{\exists P.A}, \neg A \sqcup (\exists Q.B \sqcap \exists R.C_\perp), \neg B \sqcup \exists P.A\right\} \tag{A.27}$$

The second disjunct is reduced and thus we do not reduce it. Now we have to apply further a β-rule on the left and then the only available $\pi(P)$-rule.

$$1 : \left\{\exists P.A, \boxed{\neg A}, \ldots\ldots\right\} \tag{A.28}$$

$$\begin{aligned} 1 &: \{\exists P.A, \neg A, \neg B \sqcup \exists P.A\} \\ \boxed{1.P.8} &: \left\{\boxed{A}\right\} \end{aligned} \tag{A.29}$$

Now $\{A\} \in$ NoGoods and therefore this branch can be immediately closed. So call (A.29) returns 1, and the set $\{\exists P.A, \neg A\}$ is also added to NoGoods.

We return to call (A.27) and branch on the right. Again we stubbornly do a pointless branching because clearly the disjunct we have branched on does not contribute to the search. Yet, in contrast with standard tableau method, pointless branching does not result in (exponential) disaster:

$$1 : \left\{\exists P.A, \boxed{\exists Q.B \sqcap \exists R.C_\perp}, \ldots\right\} \tag{A.30}$$

$$1 : \left\{\exists P.A, \boxed{\exists Q.B}, \boxed{\exists R.C_\perp}, \ldots\right\} \tag{A.31}$$

then we start reducing the π-concepts.

$$\begin{aligned} 1 &: \{\exists P.A, \exists Q.B, \exists R.C_\perp, \neg B \sqcup \exists P.A\} \\ \boxed{1.P.9} &: \left\{\boxed{A}\right\} \end{aligned} \tag{A.32}$$

$$\begin{aligned} 1 &: \{\exists P.A, \exists Q.B, \exists R.C_\perp, \neg B \sqcup \exists P.A\} \\ 1.P.9 &: \{A\} \\ \boxed{1.R.10} &: \left\{\boxed{C_\perp}\right\} \end{aligned} \tag{A.33}$$

$$\begin{aligned} 1 &: \{\exists P.A, \exists Q.B, \exists R.C_\perp, \ldots, \ldots\} \\ 1.P.9 &: \{A\} \\ 1.R.10 &: \{C_\perp\} \\ \boxed{1.Q.11} &: \left\{\boxed{B}\right\} \end{aligned} \tag{A.34}$$

Looking into NoGoods we find that the next lexicographic prefix $1.P.9$ prefixes a $\perp$-set. We can close the last branch without any further ado. When we return up the DFS tree new $\perp$-sets are added. Although they are not needd in this example, they might be useful for subsequent deductions.

References

[1] M. Abadi, M. Burrows, B. Lampson, G.D. Plotkin, A calculus for access control in distributed systems, ACM Trans. Program. Lang. Syst. 15 (4) (1993) 706–734.

[2] N. Arai, A proper hierarchy of propositional sequent calculi, Theoret. Comput. Sci. 159 (2) (1996) 343–354.

[3] F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, E. Franconi, An empirical analysis of optimization techniques for terminological representation systems or: "making KRIS get a move on", Appl. Intelligence 4 (2) (1994) 109–132.

[4] D. Basin, S. Matthews, L. Vigano, A new method for bounding the complexity of modal logics, in: G. Gottlob, A. Leitsch, D. Mundici (Eds.), Proc. 5th Kurt Gödel Colloquium, KGC'97, Lecture Notes in Computer Science, Vol. 1289, Springer, Berlin, 1997, pp. 89–102.

[5] P. Baumgartner, U. Furbach, I. Niemelä, Hyper tableaux, in: J. Alferes, L. Pereira, E. Orlowska (Eds.), Proc. 5th European Workshop on Logics in Artificial Intelligence (JELIA'96), Lecture Notes in Artificial Intelligence, Vol. 1126, Springer, Berlin, 1996, pp. 1–17.

[6] P. Blackburn, E. Spaan, A modal perspective on computational complexity of attribute value grammar, J. Logic Language and Inform. 2 (1993) 129–169.

[7] A. Borgida, P. Patel-Schneider, A semantics and complete algorithm for subsumption in the CLASSIC description logic, J. Artificial Intelligence Res. 1 (1994) 277–308.

[8] M. Buchheit, F.M. Donini, A. Schaerf, Decidable reasoning in terminological knowledge representation systems, J. Artificial Intelligence Res. 1 (1993) 109–138.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, What can knowledge representation do for semi-structured data?, in: Proc. AAAI-98, Madison, WI, 1998, pp. 205–210.

[10] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati, Description logic framework for information integration, in: Proc. 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98), Trento, Italy, 1998, pp. 2–13.

[11] M. Castilho, L. Fariñas del Cerro, O. Gasquet, A. Herzig, Modal tableaux with propagation rules and structural rules, Fundamenta Informaticae 32 (3) (1997) 281–297.

[12] S. Cerrito, M. Cialdea Myer, Hintikka multiplicities in matrix decision methods for some propositional modal logics, in: D. Galmiche (Ed.), Proc. International Conference on Analytic Tableaux and Related Methods (TABLEAUX'97), Lecture Notes in Artificial Intelligence, Vol. 1227, Springer, Berlin, 1997, pp. 138–152.

[13] C. Courcoubetis, M. Vardi, P. Wolper, M. Yannakakis, Memory efficient algorithms for the verification of temporal properties, Formal Methods in System Design 1 (1992) 275–288.

[14] G. De Giacomo, M. Lenzerini, Tbox and abox reasoning in expressive description logics, in: Proc. 5th International Conference on the Principles of Knowledge Representation and Reasoning (KR-96), Cambridge, MA, Morgan Kaufmann, Los Altos, CA, 1996, pp. 316–327.

[15] G. De Giacomo, M. Lenzerini, A uniform framework for concept definitions in description logics, J. Artificial Intelligence Res. 6 (1997) 87–110.

[16] G. De Giacomo, F. Massacci, Combining deduction and model checking into tableaux and algorithms for Converse-PDL, to appear in Information and Computation (accepted in 1997). An online version is at http://www.academicpress.com/i&c on IDEALfirst. A preliminary version appeared in: Proc. CADE-96, Lecture Notes in Artificial Intelligence, Vol. 1104, 1996, Springer, Berlin, pp. 613–628.

[17] R. Dechter, Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition, Artificial Intelligence 41 (3) (1990) 273–312.

[18] S. Demri, Uniform and non uniform strategies for tableaux calculi for modal logics, J. Appl. Non-Classical Logics 5 (1) (1995) 77–96.

[19] P. Devambu, R.J. Brachman, P.J. Selfridge, B.W. Ballard, LASSIE: A knowledge-based software information system, Comm. ACM 34 (5) (1991) 36–49.

[20] F.M. Donini, G. De Giacomo, F. Massacci, EXPTIME tableaux for $\mathcal{ALC}$, in: L. Padgham, E. Franconi, M. Gehrke, D. McGuinness, P. Patel-Schneider (Eds.), Proc. 1996 Description Logic Workshop (DL-96), no. WS-96-05 in AAAI Technical Reports Series, AAAI Press/MIT Press, 1996, pp. 107–110.

[21] F.M. Donini, M. Lenzerini, D. Nardi, W. Nutt, Tractable concept languages, in: Proc. IJCAI-91, Sydney, Australia, 1991, pp. 458–463.

[22] F.M. Donini, M. Lenzerini, D. Nardi, W. Nutt, The complexity of concept languages, Inform. and Comput. 134 (1997) 1–58.

[23] J. Doyle, R. Patil, Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services, Artificial Intelligence 48 (1991) 261–297.

[24] A.E. Emerson, Automated temporal reasoning about reactive systems, in: F. Moller, G. Birtwistle (Eds.), Logics for Concurrency (Structure versus Automata), Lecture Notes in Computer Science, Vol. 1043, Springer, Berlin, 1996, pp. 41–101.

[25] R. Fagin, J.Y. Halpern, Y. Moses, M.Y. Vardi, Reasoning about Knowledge, MIT Press, Cambridge, MA, 1995.

[26] L. Fariñas del Cerro, O. Gasquet, Tableaux based decision procedures for modal logics of confluence and density, Fundamenta Informaticae 40 (4) (1999) 317–333.

[27] N.J. Fischer, R.E. Ladner, Propositional dynamic logic of regular programs, J. Comput. System Sci. 18 (1979) 194–211.

[28] M. Fitting, Proof Methods for Modal and Intuitionistic Logics, Reidel, Dordrecht, 1983.

[29] M. Fitting, Basic modal logic, in: D. Gabbay, C. Hogger, J. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 1, Oxford University Press, Oxford, 1993, pp. 365–448.

[30] E. Franconi et al. (Eds.), Proc. 1998 Internat. Workshop on Description Logics, Technical Report ITC-IRST 9805-03, 1998. Available as CEUR Publication at http://SunSite.Informatik.RWTH-Aachen. DE/Publications/CEUR-WS/Vol-11/.

[31] R. Gerth, D. Peled, M. Vardi, P. Wolper, Simple on-the-fly automatic verification of linear temporal logic, in: Protocol Specification Testing and Verification (PSVT-95), Warsaw, Poland, Chapman Hall, 1995, pp. 3–18.

[32] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, A. Tacchella, SAT vs. translation based decision procedures for modal logics: A comparative evaluation, J. Appl. Non-Classical Logics 10 (2) (2000).

[33] E. Giunchiglia, A. Tacchella, A subset-matching size-bounded cache for satisfiability in modal logics, in: R. Dyckhoff (Ed.), Proc. 4th International Conference on Analytic Tableaux and Related Methods (TABLEAUX 2000), Lecture Notes in Artificial Intelligence, Vol. 1847, Springer, Berlin, 2000.

[34] F. Giunchiglia, R. Sebastiani, Building decision procedures for modal logics from propositional decision procedures—The case study of modal K, in: M.A. McRobbie, J.K. Slaney (Eds.), Proc. 13th International Conference on Automated Deduction (CADE'96), Lecture Notes in Artificial Intelligence, Vol. 1104, Springer, Berlin, 1996, pp. 583–597.

[35] J. Glasgow, G. MacEwen, P. Panangaden, A logic for reasoning about security, ACM Trans. Comput. Syst. 10 (3) (1992) 226–264.

[36] R. Goré, Tableau methods for modal and temporal logics, in: M. D'Agostino, D. Gabbay, R. Hähnle (Eds.), Handbook of Tableau Methods, Kluwer Academic, Dordrecht, 1999.

[37] V. Haarslev, R. Möller, Consistency testing: The RACE experience, in: R. Dyckhoff (Ed.), Proc. 4th International Conference on Analytic Tableaux and Related Methods (TABLEAUX 2000), Lecture Notes in Artificial Intelligence, Vol. 1847, Springer, Berlin, 2000.

[38] J.Y. Halpern, Y. Moses, A guide to completeness and complexity for modal logics of knowledge and belief, Artificial Intelligence 54 (1992) 319–379.

[39] A. Heuerding, M. Seyfried, H. Zimmermann, Efficient loop-check for backward proof search in some non-classical logics, in: Proc. 5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods (TABLEAUX'96), Lecture Notes in Artificial Intelligence, Vol. 1071, Springer, Berlin, 1996, pp. 210–225.

[40] J. Hoffman, J. Koehler, A new method to index and query sets, in: Proc. IJCAI-99, Stockholm, Sweden, Morgan Kaufmann, Los Altos, CA, 1999, pp. 462–467.

[41] I. Horrocks, Using an expressive description logic: FaCT or fiction?, in: Proc. 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98), Trento, Italy, Morgan Kaufmann, Los Altos, CA, 1998, pp. 636–647.

[42] I. Horrocks, Benchmark analysis with FaCT, in: R. Dyckhoff (Ed.), Proc. 4th International Conference on Analytic Tableaux and Related Methods (TABLEAUX 2000), Lecture Notes in Artificial Intelligence, Vol. 1847, Springer, Berlin, 2000, pp. 62–66.

[43] I. Horrocks, P.F. Patel-Schneider, Optimizing description logic subsumption, J. Logic Comput. 9 (3) (1999) 267–293.

[44] I. Horrocks, U. Sattler, A description logic with transitive and inverse roles and role hierarchies, J. Logic Comput. 9 (3) (1999) 385–410.

[45] I. Horrocks, U. Sattler, S. Tobies, Practical reasoning for expressive description logics, J. Interest Group in Pure and Applied Logic 8 (3) (2000) 239–263.

[46] I. Horrocks, S. Tobies, Reasoning with axioms: Theory and practice, in: A.G.F. Cohen, B. Selman (Eds.), Proc. 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'2000), Breckenridge, CO, Morgan Kaufmann, Los Altos, CA, 2000, pp. 285–296.

[47] U. Hustadt, R.A. Schmidt, On the relation of resolution and tableaux proof systems for description logics, in: T. Dean (Ed.), Proc. IJCAI-99, Stockholm, Sweden, Morgan Kaufmann, Los Altos, CA, 1999, pp. 110–115.

[48] U. Hustadt, R.A. Schmidt, An empirical analysis of modal theorem provers, J. Appl. Non-Classical Logics 9 (4) (1999) 479–522.

[49] D. Kozen, J. Tiuryn, Logic of programs, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Vol. II, Elsevier Science (North-Holland), Amsterdam, 1990, Chapter 14, pp. 789–840.

[50] S. Kripke, Semantical analysis of modal logic I: Normal propositional calculi, Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik 9 (1963) 67–96.

[51] R.E. Ladner, The computational complexity of provability in systems of modal propositional logic, SIAM J. Comput. 6 (3) (1977) 467–480.

[52] H.J. Levesque, Foundations of a functional approach to knowledge representation, Artificial Intelligence 23 (1984) 155–212.

[53] H.J. Levesque, R.J. Brachman, A fundamental tradeoff in knowledge representation and reasoning, in: R.J. Brachman, H.J. Levesque (Eds.), Readings in Knowledge Representation, Morgan Kaufmann, Los Altos, CA, 1985, pp. 41–70.

[54] F. Massacci, Strongly analytic tableaux for normal modal logics, in: A. Bundy (Ed.), Proc. 12th International Conference on Automated Deduction (CADE'94), Lecture Notes in Artificial Intelligence, Vol. 814, Springer, Berlin, 1994, pp. 723–737.

[55] F. Massacci, Simplification: A general constraint propagation technique for propositional and modal tableaux, in: H. de Swart (Ed.), Proc. 2nd International Conference on Analytic Tableaux and Related Methods (TABLEAUX'98), Lecture Notes in Artificial Intelligence, Vol. 1397, Springer, Berlin, 1998, pp. 217–231.

[56] F. Massacci, Tableaux methods for formal verification in multi-agent distributed systems, J. Logic Comput. 8 (3) (1998) 373–400.

[57] F. Massacci, Single step tableaux for modal logics: Methodology, computations, algorithms, J. Automat. Reason. 24 (3) (2000) 319–364.

[58] F. Massacci, F.M. Donini, Design and results of TANCS-00, in: R. Dyckhoff (Ed.), Proc. 4th International Conference on Analytic Tableaux and Related Methods (TABLEAUX 2000), Lecture Notes in Artificial Intelligence, Vol. 1847, Springer, Berlin, 2000.

[59] K. Mehlhorn, S. Nueher, M. Seel, C. Uhrig, The LEDA User Manual, http://www.mpi-sb.mpg.de/LEDA/, Max-Planck-Institut für Informatik and Algorithmic Solutions Software GmbH, 1998.

[60] B. Nebel, Computational complexity of terminological reasoning in BACK, Artificial Intelligence 34 (3) (1988) 371–383.

[61] B. Nebel, H.-J. Bürckert, Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra, J. ACM 42 (1) (1995) 43–66.

[62] H.J. Ohlbach, Translation methods for non-classical logics—An overview, J. Interest Group in Pure and Applied Logic 1 (1) (1993) 69–89.

[63] F. Oppacher, E. Suen, HARP: A tableau-based theorem prover, J. Automat. Reason. 4 (1988) 69–100.

[64] P. Patel-Schneider, DLP system description, in: E. Franconi et al. (Eds.), Proc. 1998 Internat. Workshop on Description Logics, Technical Report ITC-IRST 9805-03, 1998. Available as CEUR Publication at http://SunSite.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-11/.

[65] V.R. Pratt, A practical decision method for propositional dynamic logic, in: Proc. 10th ACM Symposium on Theory of Computing (STOC'78), San Diego, CA, 1978, pp. 326–337.
[66] V.R. Pratt, Models of program logics, in: Proc. 20th Annual Symposium on the Foundations of Computer Science (FOCS'79), IEEE Computer Society Press, 1979, pp. 115–122.
[67] J. Renz, B. Nebel, On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus, Artificial Intelligence 108 (1–2) (1999) 69–123.
[68] K. Schild, A correspondence theory for terminological logics: Preliminary report, in: Proc. IJCAI-91, Sydney, Australia, Morgan Kaufmann, Los Altos, CA, 1991, pp. 466–471.
[69] R. Schmidt, Decidability by resolution for propositional modal logics, J. Automat. Reason. 22 (4) (1999) 379–396.
[70] M. Schmidt-Schauß, G. Smolka, Attributive concept descriptions with complements, Artificial Intelligence 48 (1991) 1–26.
[71] N. Shanin, G.Yu. Davydov, G.E. Mints, V.P. Orenkov, A.O. Slisenko, An algorithm for machine search of a natural logical deduction in a propositional calculus, in: J. Siekmann, G. Wrightson (Eds.), Automation of Reasoning (Classical Papers on Computational Logic), Vol. 1 (1957–1966), Springer, Berlin, 1983.
[72] R.M. Smullyan, First Order Logic, Springer, Berlin, 1968. Republished by Dover, New York, 1995.
[73] R.M. Smullyan, Uniform Gentzen systems, J. Symbolic Logic 33 (4) (1968) 549–559.
[74] R. Sundar, R. Tarjan, Unique binary-search-tree representations and equality testing of sets and sequences, SIAM J. Comput. 23 (1) (1994) 24–44.
[75] S. Tobies, A PSPACE algorithm for graded modal logic, in: H. Ganzinger (Ed.), Proc. 16th International Conference on Automated Deduction (CADE'99), Lecture Notes in Artificial Intelligence, Vol. 1632, Springer, Berlin, 1999, pp. 52–66.
[76] A. Urquhart, The complexity of propositional proofs, Bull. Symbolic Logic 1 (4) (1995) 425–467.
[77] M. Vardi, P. Wolper, Automata-theoretic techniques for modal logics of programs, J. Comput. System Sci. 32 (1986) 183–221.
[78] M. Vardi, P. Wolper, Reasoning about infinite computations, Inform. and Comput. 115 (1) (1994) 1–37.
[79] J.R. Wright, E.S. Weixelbaum, G.T. Vesonder, K.E. Brown, S.R. Palmer, J.I. Berman, H.H. Moore, A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems, AI Magazine 14 (3) (1993) 69–80.
[80] D. Yellin, An algorithm for dynamic subset and intersection testing, Theoret. Comput. Sci. 129 (2) (1994) 397–406.