And-Or Tableaux for Fixpoint Logics with Converse: LTL, CTL, PDL and CPDL

Rajeev Goré

Logic and Computation Group
Research School of Computer Science
The Australian National University
`rajeev.gore@anu.edu.au`

Abstract. Over the last forty years, computer scientists have invented or borrowed numerous logics for reasoning about digital systems. Here, I would like to concentrate on three of them: Linear Time Temporal Logic (LTL), branching time Computation Tree temporal Logic (CTL), and Propositional Dynamic Logic (PDL), with and without converse. More specifically, I would like to present results and techniques on how to solve the satisfiability problem in these logics, with global assumptions, using the tableau method. The issues that arise are the typical tensions between computational complexity, practicality and scalability. This is joint work with Linh Anh Nguyen, Pietro Abate, Linda Postniece, Florian Widmann and Jimmy Thomson.

1 Introduction and Credits

Over the last forty years, computer scientists have invented or borrowed numerous logics for reasoning about digital systems [1]. Here, I would like to concentrate on three of them: Linear Time Temporal Logic (LTL), branching time Computation Tree temporal Logic (CTL), and Propositional Dynamic Logic (PDL). More specifically, I would like to present results and techniques on how to solve the satisfiability problem in these logics, with global assumptions, using the tableau method. The issues that arise are the typical tensions between computational time-complexity, space-complexity, practicality and scalability. This overview is based on joint work with Linh Anh Nguyen [2, 3], Linda Postniece [4] and Florian Widmann [5–7]. Some of the implementations have been refined by Jimmy Thomson. The current best account with full algorithmic details and proofs is Widmann's doctoral dissertation [8].

I have deliberately concentrated on tableaux methods, but the satisfiability problem for some of these fixpoint logics can also be solved using resolution methods and automata methods. These are beyond my expertise.

I assume that the reader is familiar with the syntax and semantics of propositional modal, description and fixpoint logics, the notion of global logical consequence in these logics, the associated notions of being satisfiable with respect to a set of global assumptions (TBox) and with basic tableau methods for classical propositional logic. I assume that all formulae are in negation normal form since this reduces the number of rules. It is well-known that, in all the logics I consider, a formula can be put into negation normal form with only a polynomial increase in size, while preserving validity. I also assume that we are given a finite set $\mathcal{T}$ of "global assumptions" (TBox) and asked to solve the problem of whether ϕ is satisfiable with respect to the global assumptions $\mathcal{T}$ in the logic under consideration. Thus a formula ϕ is a global logical consequence of $\mathcal{T}$ iff the formula $\texttt{nnf}\ (\neg\phi)$ is unsatisfiable with respect to $\mathcal{T}$, where $\texttt{nnf}\ (.)$ is the function that returns the negation normal form of its argument.

The tableau method is a very general method for automated reasoning and has been widely applied for modal logics [9] and description logics [10]. Tableau methods usually come in two flavours as we explain shortly. Both methods build a rooted tree with some leaves duplicating ancestors, thereby giving cycles. Because the same node may be explored on multiple branches, tableau algorithms are typically suboptimal w.r.t. the known theoretical bounds for many logics. For example, the traditional tableau method for $\mathcal{ALC}$ can require double-exponential time even though the decision problem is known to be EXPTIME-complete.

For fixpoint logics like LTL, CTL and PDL, optimal tableau methods are possible if we proceed in stages with the first stage building a cyclic graph, and subsequent passes pruning nodes from the graph until no further pruning is possible or until the root node is pruned [11]. Optimality can also be obtained if we construct the set of all subsets of the Fischer-Ladner closure of the given initial formula [12]. But these methods can easily require exponential time even when it is not necessary. Indeed, the method of Fischer and Ladner will always require exponential time since it must first construct the set of all subsets of a set whose size is usually linear in the size of the given formula.

Thus a long-standing open problem in tableau methods for modal, description, and fixpoint logics has been to find a complexity-optimal and "on the fly" method for checking satisfiability which only requires exponential time when it is really necessary. We describe such tableau methods for each of the logics K, Kt (*i.e.* K with converse) and PDL. The resulting methods necessarily build graphs rather than trees. The various components can be combined non-trivially to give an on-the-fly and complexity-optimal tableau method for CPDL (*i.e.* PDL with converse) but we omit details. We also describe sub-optimal tableaux methods for these logics which build one single tree tableau and determine satisfiability in one pass by exploring this tree one branch at a time, reclaiming the space of previous branches. We describe such a method for the logic CTL, and give pointers to how to adapt such one-pass methods to LTL and PDL.

2 Traditional Modal and Description Logic Tree Tableaux

A tableau is a tree of nodes where the children of a node are created by applying a tableau rule to the parent and where each node contains a finite set of formulae. We refer to these formulae as the "contents" of a node, noting that the term "label" is also used to mean the same thing. Thus a label is *not* a name for a

$$(id)\frac{\Gamma\,;\,\neg p\,;\,p}{} \qquad (\wedge)\frac{\Gamma\,;\,\varphi\wedge\psi}{\Gamma\,;\,\varphi\,;\,\psi} \qquad (\vee)\frac{\Gamma\,;\,\varphi\vee\psi}{\Gamma\,;\,\varphi \mid \Gamma\,;\,\psi}$$

$$(\exists)\frac{\Delta\,;\;[]\,\Gamma\,;\;\langle\rangle\,\varphi_1\,;\,\cdots\,;\;\langle\rangle\,\varphi_n}{\Gamma\,;\,\varphi_1\,;\,\mathcal{T}\;\;||\;\;\cdots\;\;||\;\;\Gamma\,;\,\varphi_n\,;\,\mathcal{T}}$$

Δ contain only atoms and negated atoms

Fig. 1. AND/OR Tableaux Rules for Modal Logic with Global Assumptions $\mathcal{T}$

Kripke world as in some formulations of "labelled tableaux". The ancestors of a node are simply the nodes on the unique path from the root to that node.

A leaf node is "closed" when it can be deemed to be unsatisfiable, usually because it contains an obvious contradiction like p and $\neg p$. A leaf is "open" when it can be deemed to be satisfiable, usually when no rule is applicable to it, but also when further rule applications are guaranteed to give an infinite (satisfiable) branch. A branch is closed/open if its leaf is closed/open. The aim of course is to use these classifications to determine whether the root node is satisfiable or unsatisfiable. But the tableau used in modal logics and those used in description logics are dual in a sense which is explained next.

Traditional modal tableaux a là Beth [13] are or-trees in that branches are caused by disjunctions only. Each "diamond" formula in a node causes the creation of a "successor world", fulfilling that formula. But such successors of a given node are created and explored one at a time, using backtracking, until one of them is closed, meaning that there is no explicit trace of previously explored "open" successors in any single tableau.

Traditional description logic tableaux are usually and-trees in that branches are caused by existential/diamond formulae only. Each disjunctive formula causes the creation of a child, one at a time, using backtracking, until one child is open, meaning that there is no explicit trace of previously explored "closed" or-children in any single tableau.

Thus, in both types of tableaux, the overall search space is really an and-or tree: traditional modal (Beth) tableaux display only the or-related branches and explore the and-related branches using backtracking while description logic tableaux do the reverse.

In all such methods, termination is obtained by "blocking" a node from expansion if the node that would be created already exists. For a detailed discussion of the various blocking methods, and the various notions of "caching" see [2].

3 And-Or graph and tree tableaux for K

We unify these two views by taking a global view which considers tableaux as And-Or trees or And-Or graphs rather than as or-trees or and-trees. In particular, since the non-determinism in both traditional tableaux methods is determinised in And-Or tableaux, we need to build one and only one And-Or tableau!

Thus the And-Or tableau rules for modal logic K can be written as shown in Figure 1 where Γ and Δ are finite sets of formulae in negation normal form and $\Gamma\,;\,\varphi$ stands for the set $\Gamma \cup \{\varphi\}$.

The ($\vee$)-rule creates or-branching, indicated by "|" while the $\exists$-rule creates and-branching, indicated by "||". These are dual in the following senses:

($\vee$): if the set $\Gamma; \varphi \vee \psi$ is satisfiable w.r.t. $\mathcal{T}$ then the set $\Gamma; \varphi$ is satisfiable w.r.t. $\mathcal{T}$ *or* the set $\Gamma; \psi$ is satisfiable w.r.t. $\mathcal{T}$

($\vee$): if both sets $\Gamma; \varphi$ and $\Gamma; \psi$ are unsatisfiable w.r.t. $\mathcal{T}$ then so is $\Gamma; \varphi \vee \psi$

($\exists$): if the set $\Delta;\ []\,\Gamma;\ \langle\rangle\,\varphi_1; \cdots;\ \langle\rangle\,\varphi_n$ is satisfiable w.r.t. $\mathcal{T}$ then the set $\Gamma; \varphi_1; \mathcal{T}$ is satisfiable w.r.t. $\mathcal{T}$ *and* the set $\Gamma; \varphi_2; \mathcal{T}$ is satisfiable w.r.t. $\mathcal{T}$ *and ... and* the set $\Gamma; \varphi_n; \mathcal{T}$ is satisfiable w.r.t. $\mathcal{T}$.

($\exists$): if there is some integer $1 \le i \le n$, such that the set $\Gamma; \varphi_i; \mathcal{T}$ is unsatisfiable w.r.t. $\mathcal{T}$ then the set $\Delta;\ []\,\Gamma;\ \langle\rangle\,\varphi_1; \cdots;\ \langle\rangle\,\varphi_n$ is unsatisfiable w.r.t. $\mathcal{T}$.

We now give a non-algorithmic description of the procedure to create an and-or tableau. We have chosen this format over the more algorithmic description in [14, 2] to highlight its simplicity.

1. start with a root node and repeatedly try to apply exactly one of the rules in the order (id), ($\wedge$), ($\vee$), ($\exists$) to each node but if a rule application to node x will create a copy y' of an existing node y then make y the child of x instead
2. if we apply the ($\vee$)-rule to x then x is an or-node and if we apply the rule ($\wedge$) or ($\exists$) to x then it is an and-node
3. whenever we apply the (id) rule to x then set the status of x to `unsat`, and if we cannot apply any rule to x then set its status to `sat`, and propagate this status through the current graph as follows:
   or-node: `unsat` if all its children have status `unsat` and `sat` if some child has status `sat`
   and-node: `sat` if all its children have status `sat` and `unsat` if some child has status `unsat`
4. when every node has been expanded in this way then set the status of all nodes with undefined status to `sat` and propagate as above.

A little more formally but still non-algorithmically. Given a TBox $\mathcal{T}$ and a formula ϕ, both in negation normal form, our method searches for a model which satisfies ϕ w.r.t. $\mathcal{T}$ by building an and-or graph G with root node τ containing $\mathcal{T} \cup \{\phi\}$. A node in the constructed and-or graph is a record with three attributes:

content: the set of formulae carried by the node
status: {`unexpanded`, `expanded`, `sat`, `unsat`}
kind: {`and-node`, `or-node`, `leaf-node`}

The root node has initial status `unexpanded` and our method constructs the and-or graph using a traditional strategy explained shortly. But we interleave this generation strategy with a propagation phase which propagates the status of a node throughout the graph. We explain each in turn.

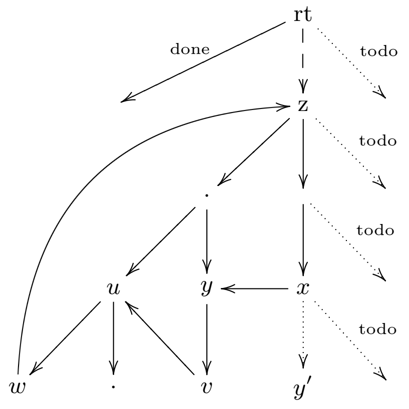

Fig. 2. Graph constructed by our algorithm for K using global caching

Our strategy for building the and-or graph applies the rules for decomposing $\wedge$ and $\vee$ repeatedly until they are no longer applicable to give a "saturated" node x, and then applies the $\exists$-rule which creates a child node for x containing $\mathcal{T} \cup \{\varphi\} \cup \{\psi \mid [\,]\,\psi \in x\}$ for each $\langle\rangle\,\varphi \in x$. The addition of the TBox $\mathcal{T}$ to such a child is a naive way to handle global assumption (TBoxes) but suffices for our needs. We now saturate any such child to obtain a saturated node y, then apply the $\exists$-rule to y, and so on, until we find a contradiction, or find a repeated node, or find a saturated node which contains no $\langle\rangle$-formulae. For uniformity with our method for the extensions to converse and PDL, we explore/expand children in a left to right depth-first manner, although any search strategy can be used for K (or *ALC*) [2]. All nodes are initially given a status of `unexpanded`.

An application of ($\vee$) to a node v causes v to be an `or-node`, while an application of ($\wedge$) or ($\exists$) to a node v causes v to be an `and-node`. Notice that our method uses the ($\vee$) and ($\exists$) rules which use both or-branching and and-branching as summarised in Section 2. The crucial difference from traditional tableau methods is that we create an and-or graph rather than an and-tree or an and-tree, and we create the required child in the graph G only if it does not yet exist in the graph: this step therefore uses global caching [2]. Notice that the required child need not be an ancestor but can exist on any previous branch of the tableau. For example, as shown in Figure 2, suppose the current node is x and that the rule applied to x generates a node y' which duplicates y. The node y' is not put into G, but y becomes the child of x instead. Thus, G is really a rooted and-or tree with cross-branch edges to nodes on previously created branches like that from x to y or from v to u, or edges to ancestors like that from w to z. The problem of course is to show that this remains sound.

The propagation phase begins whenever we determine the status of a node as either `unsat` or `sat` as explained next.

A generated node that contains both p and $\neg p$ for some atomic formula p becomes a `leaf-node` with status `unsat` (i.e. unsatisfiable w.r.t. $\mathcal{T}$). A generated node to which no tableau rule is applicable becomes a `leaf-node` with status `sat` (i.e. satisfiable w.r.t. $\mathcal{T}$). Both conclusions are **irrevocable** because each relies only on classical propositional principles and not on modal principles. We therefore propagate this information to the parent node v using the kind (`or-node`/`and-node`) of v and the status of the children of v, treating `unsat` as irrevocably **f** and `sat` as irrevocably **t**. That is, an `or-node` gets status `sat` as soon as one of its children gets status `sat`, and gets status `unsat` when all of its children get status `unsat`. Dually for `and-nodes`. In particular, it does not matter whether the parent-child edge is a cross-branch edge or whether it is a traditional top-down edge. If these steps cannot determine the status as `sat` or `unsat`, then the rule application sets the status to `expanded` and we return to the generation phase.

The main loop ends when the status of the initial node τ becomes `sat` or `unsat`, or when no node has status `unexpanded`. In the last case, all nodes with status $\neq$ `unsat` are given status `sat` (effectively giving the status "open" to tableau branches which loop to an ancestor) and this status is propagated through the graph to obtain the status of the root node as either `unsat` or `sat`.

Theorem 1 (Soundness and Completeness). *The root node of the And-Or graph for $\mathcal{T} \cup \{\phi\}$ has status* `sat` *iff ϕ is* K*-satisfiable with respect to $\mathcal{T}$.*

Theorem 2 (Complexity of And-Or graph tableaux). *If the sum of the sizes of the formulae in $\mathcal{T} \cup \{\phi\}$ is n then the algorithm requires $O(2^n)$ space and $O(2^n)$ time.*

This algorithms thus uses both caching and propagation techniques and runs in EXPTIME [2].

3.1 And-Or Tree Tableaux

The method described above creates an And-Or graph as shown in Figure 2 which means that we have to keep previous branches in memory. An alternative is to only allow "loops" to ancestors. Using this strategy gives an And-Or tree which can be explored one branch at a time, and there is no need to keep previous branches in memory. We use the term And-Or tree tableaux for the resulting tableau method.

The soundness and completeness is not affected by this change, but the ability to reclaim previous branches saves memory but leads to sub-optimality.

Theorem 3 (Complexity of And-Or tree tableaux). *If the sum of the sizes of the formula in $\mathcal{T} \cup \{\phi\}$ is n then the tree-tableaux algorithm requires $O(2^n)$ space and $O(2^{2^n})$ time.*

There are n subformulae of $\mathcal{T} \cup \{\phi\}$ and hence 2^n subsets which might appear on a branch before a node repeats, hence a branch can require $O(2^n)$ space. We explore the And-Or tree one branch at a time, so we require at most this much space. An and-or tree of depth $O(2^n)$ may have $O(2^{2^n})$ or-branches. In the worse

(a)

$$\begin{tikzcd}
x \arrow[d, "\langle\rangle\,\varphi"] \\
ps_0 \arrow[d, dotted] \\
ps_k \arrow[d] \\
z \\
\texttt{toosmall} \\
(\{\xi \mid \blacksquare\,\xi \in z\} \setminus x) \in \mathrm{alt}_x
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
x \arrow[d, "\langle\rangle\,\varphi"] \\
ps_0 \arrow[d, dotted] \\
ps_k \arrow[d] \\
z \arrow[d] \\
y
\end{tikzcd}$$

(c)

$$\begin{tikzcd}
x \arrow[d, "\langle\rangle\,\varphi"] & & & \\
ps_0 \arrow[d, dotted] & & & \\
ps_k \arrow[d] & & & \\
z \arrow[d] \arrow[dr] \arrow[drrr] & & & \\
y & y_1^+/z_1 & \cdots & y_n^+/z_n
\end{tikzcd}$$

Fig. 3. The use of special node z to handle in/compatibility between states x and y. Scenario (a) occurs when x and y are incompatible. Scenario (b) occurs when x and y are compatible. Scenario (c) occurs when x and y are compatible, but y is/becomes `toosmall`.

case, we have to close each branch, hence we may require $O(2^{2^n})$ time. Thus And-Or tree tableaux are sub-optimal: the satisfiability problem for K is known to be EXPTIME-complete, but our algorithm has worst-case complexity of 2EXPTIME.

The soundness shows that an ancestor loop *always* represents a "good loop" in which every node is *satisfiable*. It is this property that fails for fixpoint logics.

4 And-Or Graph Tableaux for adding Converse

Recall that the standard strategy for rule applications in tableau algorithms is to apply the rules for decomposing $\wedge$ and $\vee$ repeatedly until they are no longer applicable, giving a "saturated" node which contains only atoms, negated atoms, $[]$-formulae and $\langle\rangle$-formulae. Let us call such a "saturated" node a *state* and call the other nodes *prestates*. Thus the only rule applicable to a state x is the $\exists$-rule which creates a node containing $\mathcal{T} \cup \{\varphi\} \cup \{\psi \mid [] \, \psi \in x\}$ for each $\langle\rangle\,\varphi \in x$. The standard strategy will now saturate any such child to obtain a state y, then apply the $\exists$-rule to y, and so on, until we find a contradiction, or find a repeated node, or find a state which contains no $\exists$-formulae. Let us call x the parent state of y since all intervening nodes are not states.

When converse modalities $\blacklozenge / \blacksquare$ (inverse roles) present, we require that $\{\xi \mid \blacksquare\,\xi \in y\} \subseteq x$, since y is then compatible with being an R-successor of x in the putative model under construction. If some $\blacksquare\,\xi \in y$ has $\xi \notin x$ then x is "too small", and must be enlarged into an alternative node x^+ by adding all such ξ. If any such ξ is a complex formula then the alternative node x^+ is not "saturated", and hence not a state. So we must saturate it using the $\wedge/\vee$-rules until we reach a state. That is, a state x may conceptually be "replaced" by an alternative prestate x^+ which is an enlargement of x, and which may have to be saturated further in order to reach a state.

Our algorithm handles these "alternatives" by introducing a new type of node called a *special node*, introducing a new type of status called `toosmall`, allowing states to contain a field alt for storing these alternatives, and ensuring that a state always has a special node as its parent. When we need to replace a state x by its alternatives, the special node above x extracts these alternatives from the alt_x field and creates the required alternative nodes as explained next.

Referring to Fig. 3, suppose state x has an R-successor prestate ps_0, and further saturation of ps_0 leads to prestate ps_k, and an application of an $\wedge/\vee$-rule to p_k will give a state y. Instead of directly creating y, we create a special node z which carries the same set of formulae as would y, and make z a child of ps_k. We now check whether z is compatible with its parent state x by checking whether $\{\xi \mid \blacksquare\xi \in z\} \subseteq x$. If z is not compatible then we mark z as `toosmall`, and add $\{\xi \mid \blacksquare\xi \in z\} \setminus x$ to the set of alternative sets contained in alt_x, without creating y, as shown in Fig. 3(a). If z is compatible with x, we create a state y if it does not already exist, and make the new/old y a child of z, as in Fig. 3(b).

Suppose that y is compatible with x and that either y is already `toosmall` or becomes so later because of some descendant state w of y. In either case, the attribute alt_y then contains a number of sets $y_1, y_2, \ldots, y_n$ (say), and the `toosmall` status of y is propagated to the special node z. In response, z will create the alternatives $y_1^+, y_2^+, \ldots, y_n^+$ for y with $y_i^+ := y \cup y_i$. If y_i^+ is a state then our algorithm will create a special node z_i below z, and if z_i is compatible with x then y_i^+ will be created or retrieved and will become the child of z_i as in (b) else y_i^+ will not be created and z_i will be marked as `toosmall` as in (a). If y_i^+ is not a state then it will be created as a direct prestate child of z. Figure 3(c) captures this by using y_i^+/z_i to stand for either y_i^+ or z_i. Each of these new non-special nodes will eventually be expanded by our algorithm but now the "lapsed" special node z will be treated as a $\vee$-node.

Global State Caching. The complexities introduced by alternative nodes makes it difficult to use global caching so instead we use "global state caching": that is, the saturation phase is allowed to re-create prestates that occur on previous branches, but states cannot be duplicated so we must use cross-branch edges to their previous incarnations. The resulting algorithm runs in EXPTIME [6].

5 Traditional Tableaux Methods for Fixpoint Logics

As we have seen, modal and description logic tableaux require some form of "loop check" to guarantee termination, but fix-point logics require a further test to distinguish a "good loop" that represents a path in a model from a "bad loop" that represents an infinite branch with no hope of ever giving a model.

Most tableau-based methods for fix-point logics solve this problem using a multi-pass graph procedure [11, 15–17]. The first pass applies the tableau rules to construct a finite rooted cyclic graph. The subsequent passes prune node that are unsatisfiable because they contain contradictions like $\{p, \neg p\}$, and also remove nodes which give rise to "bad loops". The main practical disadvantage of such multi-pass methods is that the cyclic graph built in the first pass has a size which is *always* exponential in the size of the initial formula. So the very act of building this graph immediately causes EXPTIME behaviour even in the average case.

6 One-pass And-Or Tree Tableaux for Fixpoint Logics

One-pass And-Or tableaux avoid this bottle-neck by building a rooted cyclic *tree* (where all cyclic edges loop back to ancestors) one branch at a time, using backtracking. The experience from one-pass tableaux for very expressive description logics [18] of similar worst-case complexity shows that their average case behaviour is often much better since the given formulae may not contain the full complexity inherent in the decision problem, particularly if the formula arises from real-world applications. Of course, there is no free lunch, since in the worst case, these one-pass methods may have significantly worse behaviour than the known optimal behaviour: 2EXPTIME than EXPTIME in the case of CTL for example. Moreover, the method for separating "good loops" from "bad loops" becomes significantly more complicated since it cannot utilise the global view offered by a graph built during a previous pass. Ideally, we want to evaluate each branch on its own during construction, or during backtracking, using only information which is "local" to this branch since this allows us to explore these branches in parallel using multiple processors.

Implemented one-pass [19, 20] and multi-pass [21] tableau provers already exist for LTL. A comparison between them [22] shows that the median running time for Janssen's highly optimised multi-pass prover for LTL is greater than the median running time for Schwendimann's not-so-optimised one-pass prover for LTL [20] *for problems which are deliberately constructed to be easy for tableau provers*, indicating that the multi-pass prover spends most of its time in the first pass building the cyclic graph. There is also a one-pass "tableau" method for propositional dynamic logic (PDL) [23] which constructs a rooted cyclic tree and uses a finite collection of automata, pre-computed from the initial formula, to distinguish "good loops" from "bad loops".

7 One-pass And-Or Tree Tableaux for CTL

For simplicity, we ignore global assumptions (TBoxes) and concentrate on only the satisfiability problem since global assumptions can be added by a simple modification of the rules that create modal successors.

A *tableau algorithm* is a systematic search for model for a formula ϕ. The algorithm stores additional information with each node of the tableau using *histories* and *variables* [20]. A history is a mechanism for collecting extra information during proof search and passing it from parents to children. A variable is a mechanism to propagate information from children to parents.

In the following, we restrict ourselves to the tableau algorithm for CTL.

Definition 1. *A* tableau node x *is of the form* $(\Gamma :: \mathrm{HCr} :: \mathrm{mrk}, \mathrm{uev})$ *where:*

Γ *is a set of formulae;*
HCr *is a list of the formula sets of some designated ancestors of* x*;*
mrk *is a Boolean valued variable indicating whether the node is marked; and*
uev *is a partial function from formulae to* $\mathbb{N}_{>0}$.

The list HCr is the only history since its value in a node is determined by the parent node, whereas mrk and uev are variables since their values in a node are determined by the children. In the following we call tableau nodes just nodes when the meaning is clear.

Informally, the value of mrk at node x is **true** if x is "closed". Since repeated nodes cause "cycles" or "loops", a node that is not "closed" is not necessarily "open" as in traditional tableaux. That is, although we have enough information to detect that further expansion of the node will cause an infinite branch, we may not yet have enough information to determine the status of the node. Informally, if a node x lies on such a "loop" in the tableau, and an "eventuality" *EU*- or *AU*-formula φ appears on this loop but remains unfulfilled, then uev of x is defined for φ by setting $\mathrm{uev}(\varphi) = n$, where n is the height of the highest ancestor of x which is part of the loop.

We postpone the definition of a rule for a moment and proceed with the definition of a tableau.

Definition 2. *A* tableau *for a formula set* Γ *and a list of formula sets* HCr *is a tree of tableau nodes with root* $(\Gamma :: \mathrm{HCr} :: \mathrm{mrk}, \mathrm{uev})$ *where the children of a node* x *are obtained by a single application of a rule to* x *(*i.e. *only one rule can be applied to a node). A tableau is* expanded *if no rules can be applied to any of its leaves.*

Note that mrk and uev in the definition are not given but are part of the result as they are determined by the children of the root.

Definition 3. *The partial function* $\mathrm{uev}_\perp : \mathrm{Fml} \rightharpoonup \mathbb{N}_{>0}$ *is the constant function that is undefined for all formulae (*i.e. $\mathrm{uev}_\perp(\psi) = \perp$ *for all* ψ*).*

Note 1. In the following, we use Λ to denote a set containing only propositional variables or their negations (*i.e.* $\varphi \in \Lambda \;\Rightarrow\; \varphi = p$ or $\varphi = \neg p$ for some atom p). To focus on the "important" parts of the rule, we use "$\cdots$" for the "unimportant" parts which are passed from node to node unchanged (*e.g.* $(\Gamma :: \cdots :: \cdots)$). We define $\sim\varphi := \texttt{nnf}\ (\neg\varphi)$.

7.1 The Rules

Terminal Rule.

$$(id)\ \ \frac{(\Gamma :: \cdots :: \mathrm{mrk}, \mathrm{uev})}{}\ \ \{p, \neg p\} \subseteq \Gamma \text{ for some atomic formula } p$$

with mrk := **true** and uev := $\mathrm{uev}_\perp$. The intuition is that the node is "closed" so we pass this information up to the parent by putting mrk to **true**, and putting uev as undefined for all formulae.

Linear (α) Rules.

$$(\wedge)\ \frac{(\varphi \wedge \psi\ ;\ \Gamma :: \cdots :: \cdots)}{(\varphi\ ;\ \psi\ ;\ \Gamma :: \cdots :: \cdots)} \qquad (D)\ \frac{(AX\Delta\ ;\ \Lambda :: \cdots :: \cdots)}{(EX(p_0 \vee \neg p_0)\ ;\ AX\Delta\ ;\ \Lambda :: \cdots :: \cdots)}$$

$$(EB)\ \frac{(E(\varphi\, B\, \psi)\ ;\ \Gamma :: \cdots :: \cdots)}{(\sim\psi\ ;\ \varphi \vee EXE(\varphi\, B\, \psi)\ ;\ \Gamma :: \cdots :: \cdots)}$$

$$(AB)\ \frac{(A(\varphi\, B\, \psi)\ ;\ \Gamma :: \cdots :: \cdots)}{(\sim\psi\ ;\ \varphi \vee AXA(\varphi\, B\, \psi)\ ;\ \Gamma :: \cdots :: \cdots)}$$

The $\wedge$-rule is standard and the D-rule captures the fact that the binary relation of a model is total by ensuring that every potential dead-end contains at least one EX-formula. The EB- and AB-rules capture the fix-point nature of the corresponding formulae according to the valid formulae $E(\varphi\, B\, \psi) \leftrightarrow \neg\psi \wedge (\varphi \vee EXE(\varphi\, B\, \psi))$ and $A(\varphi\, B\, \psi) \leftrightarrow \neg\psi \wedge (\varphi \vee AXA(\varphi\, B\, \psi))$.

These rules do not modify the histories or variables at all.

Universal Branching (β) Rules.

$$(\vee)\ \frac{(\varphi \vee \psi\ ;\ \Gamma :: \cdots :: \mathrm{mrk}, \mathrm{uev})}{(\varphi\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_1, \mathrm{uev}_1) \mid (\psi\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_2, \mathrm{uev}_2)}$$

$$(EU)\ \frac{(E(\varphi\, U\, \psi)\ ;\ \Gamma :: \cdots :: \mathrm{mrk}, \mathrm{uev})}{(\psi\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_1, \mathrm{uev}_1) \mid (\varphi\ ;\ EXE(\varphi\, U\, \psi)\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_2, \mathrm{uev}_2)}$$

$$(AU)\ \frac{(A(\varphi\, U\, \psi))\ ;\ \Gamma :: \cdots :: \mathrm{mrk}, \mathrm{uev})}{(\psi\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_1, \mathrm{uev}_1) \mid (\varphi\ ;\ AXA(\varphi\, U\, \psi)\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_2, \mathrm{uev}_2)}$$

with:

$$\begin{aligned}
\mathrm{mrk} &:= \mathrm{mrk}_1 \ \&\ \mathrm{mrk}_2 \\
\mathrm{excl}_\phi(f)(\chi) &:= \begin{cases} \bot & \text{if } \chi = \phi \\ f(\chi) & \text{otherwise} \end{cases} \\
\mathrm{uev}_1' &:= \begin{cases} \mathrm{uev}_1 & \text{for the } \vee\text{-rule} \\ \mathrm{excl}_{E(\varphi\, U\, \psi)}(\mathrm{uev}_1) & \text{for the } EU\text{-rule} \\ \mathrm{excl}_{A(\varphi\, U\, \psi)}(\mathrm{uev}_1) & \text{for the } AU\text{-rule} \end{cases} \\
\min{}_\bot(f,g)(\chi) &:= \begin{cases} \bot & \text{if } f(\chi) = \bot \text{ or } g(\chi) = \bot \\ \min(f(\chi), g(\chi)) & \text{otherwise} \end{cases} \\
\mathrm{uev} &:= \begin{cases} \mathrm{uev}_\bot & \text{if } \mathrm{mrk}_1 \ \&\ \mathrm{mrk}_2 \\ \mathrm{uev}_1' & \text{if } \mathrm{mrk}_2 \ \&\ \text{not } \mathrm{mrk}_1 \\ \mathrm{uev}_2 & \text{if } \mathrm{mrk}_1 \ \&\ \text{not } \mathrm{mrk}_2 \\ \min{}_\bot(\mathrm{uev}_1', \mathrm{uev}_2) & \text{otherwise} \end{cases}
\end{aligned}$$

The $\vee$-rule is standard except for the computation of uev. The EU- and AU-rules capture the fix-point nature of the EU- and AU-formulae, respectively, according to the valid formula $E(\varphi\, U\, \psi) \leftrightarrow \psi \vee (\varphi \wedge EXE(\varphi\, U\, \psi))$ and $A(\varphi\, U\, \psi) \leftrightarrow \psi \vee (\varphi \wedge AXA(\varphi\, U\, \psi))$. The intuitions of the definitions of the histories and variables are:

mrk: the value of the variable mrk is **true** if the node is “closed”, so the definition of mrk just captures the “universal” nature of these rules whereby the parent node is closed if both children are closed.

excl: the definition of $\mathrm{excl}_\phi(f)(\psi)$ just ensures that $\mathrm{excl}_\phi(f)(\phi)$ is undefined.

uev'_1: the definition of uev'_1 ensures that its value is undefined for the principal formulae of the *EU*- and *AU*-rules.

$\min_\perp$: the definition of $\min_\perp$ ensures that we take the minimum of $f(\chi)$ and $g(\chi)$ only when both functions are defined for χ.

uev: if both children are “closed” then the parent is also closed via mrk so we ensure that uev is undefined in this case. If only the right child is closed, we take uev'_1, which is just uev_1 modified to ensure that it is undefined for the principal *EU*- or *AU*-formula. Similarly if only the left child is closed. Finally, if both children are unmarked, we define uev for all formulae that are defined in the uev of both children but map them to the minimum of their values in the children, and undefine the value for the principal formula.

Existential Branching Rule.

$$(EX)\ \frac{\begin{array}{l} EX\varphi_1\ ; \ldots ; EX\varphi_n\ ;\ EX\varphi_{n+1}\ ; \ldots ; EX\varphi_{n+m}\ ;\ AX\Delta\ ;\ \Lambda \\ ::\ \mathrm{HCr}\ ::\ \mathrm{mrk}, \mathrm{uev} \end{array}}{\begin{array}{l} \varphi_1\ ;\ \Delta \\ ::\ \mathrm{HCr}_1\ ::\ \mathrm{mrk}_1, \mathrm{uev}_1 \end{array} \Big| \cdots \Big| \begin{array}{l} \varphi_n\ ;\ \Delta \\ ::\ \mathrm{HCr}_n\ ::\ \mathrm{mrk}_n, \mathrm{uev}_n \end{array}}$$

where:

(1) $\{p, \neg p\} \not\subseteq \Lambda$
(2) $n + m \geq 1$
(3) $\forall i \in \{1, \ldots, n\}.\ \forall j \in \{1, \ldots, \mathrm{len}(\mathrm{HCr})\}.\ \{\varphi_i\} \cup \Delta \neq \mathrm{HCr}[j]$
(4) $\forall k \in \{n+1, \ldots, n+m\}.\ \exists j \in \{1, \ldots, \mathrm{len}(\mathrm{HCr})\}.\ \{\varphi_k\} \cup \Delta = \mathrm{HCr}[j]$

with:

$$\mathrm{HCr}_i := \mathrm{HCr}\ @\ [\{\varphi_i\} \cup \Delta] \text{ for } i = 1, \ldots, n$$

$$\begin{aligned} \mathrm{mrk} := {} & \textstyle\bigvee_{i=1}^{n} \mathrm{mrk}_i \text{ or} \\ & \exists i \in \{1, \ldots, n\}.\ \exists \psi \in \{\varphi_i\} \cup \Delta.\ \perp \neq \mathrm{uev}_i(\psi) > \mathrm{len}(\mathrm{HCr}) \end{aligned}$$

$$\begin{aligned} \mathrm{uev}_k(\cdot) := {} & j \in \{1, \ldots, len(\mathrm{HCr})\} \text{ such that } \{\varphi_k\} \cup \Delta = \mathrm{HCr}[j] \\ & \text{for } k = n+1, \ldots, n+m \end{aligned}$$

$$\mathrm{uev}(\psi) := \begin{cases} \mathrm{uev}_j(\psi) & \text{if } \psi \in \mathrm{FmlEU}\ \&\ \psi = \varphi_j \quad (j \in \{1, \ldots, n+m\}) \\ l & \text{if } \psi \in \mathrm{FmlAU} \cap \Delta\ \& \\ & \quad l = \max\{\mathrm{uev}_j(\psi) \neq \perp \mid j \in \{1, \ldots, n+m\}\} \\ \perp & \text{otherwise} \end{cases}$$
$$(\text{where } \max(\emptyset) := \perp)$$

Some intuitions are in order:

(1) The EX-rule is applicable if the parent node contains no α- or β-formulae and Λ, which contains propositional variables and their negations only, contains no contradictions.
(2) Both n and m can be zero, but not together.
(3) If $n > 0$, then each $EX\varphi_i$ for $1 \leq i \leq n$ is not "blocked" by an ancestor, and has a child containing $\varphi_i; \Delta$, thereby generating the required EX-successor;
(4) If $m > 0$, then each $EX\varphi_k$ for $n+1 \leq k \leq m$ is "blocked" from creating its required child $\varphi_k; \Delta$ because some ancestor does the job;

HCr_i: is just the HCr of the parent but with an extra entry to extend the "history" of nodes on the path from the root down to the i^{th} child.

mrk: captures the "existential" nature of this rule whereby the parent is marked if some child is closed or if some child contains a formula whose uev is defined and "loops" lower than the parent. Moreover, if n is zero, then mrk is set to **false** to indicate that this branch is not "closed".

uev_k: for $n+1 \leq k \leq n+m$ the k^{th} child is blocked by a proxy child higher in the branch. For every such k we set uev_k to be the *constant* function which maps every formula to the level of this proxy child. Note that this is just a temporary function used to define uev as explained next.

$\mathrm{uev}(\psi)$: for an EU-formula $\psi = E(\psi_1 \, U \, \psi_2)$ such that there is a principal formula $EX\varphi_i$ with $\varphi_i = \psi$, we take uev of ψ from the child if $EX\psi$ is "unblocked", or set it to be the *level* of the proxy child higher in the branch if it is "blocked". For an AU-formula $\psi = A(\psi_1 \, U \, \psi_2) \in \Delta$, we put uev to be the maximum of the defined values from the real children and the levels of the proxy children. For all other formulae, we put uev to be undefined. The intuition is that a defined $\mathrm{uev}(\psi)$ tells us that there is a "loop" which starts at the parent and eventually "loops" up to some blocking node higher up on the current branch. The actual value of $\mathrm{uev}(\psi)$ tells us the level of the proxy because we cannot distinguish whether this "loop" is "good" or "bad" until we backtrack up to that level.

Note that the EX-rule and the *id*-rule are mutually exclusive since their side-conditions cannot be simultaneously true.

Proposition 1 (Termination). *Let $\phi \in \mathrm{Fml}$ be a formula in negation normal form. Any tableau T for a node $(\{\phi\} :: \cdots :: \cdots)$ is a finite tree, hence the procedure that builds a tableau always terminates [24].*

Let $\phi \in \mathrm{Fml}$ be a formula in negation normal form and T an expanded tableau with root $r = (\{\phi\} :: [] :: \mathrm{mrk}, \mathrm{uev})$: that is, the initial formula set is $\{\phi\}$ and the initial HCr is the empty list.

Theorem 4 (Soundness and Completeness). *The root r is marked iff ϕ is not satisfiable [24].*

Theorem 5 (Complexity). *The tableau algorithm runs in double exponential deterministic time and needs exponential space [24]..*

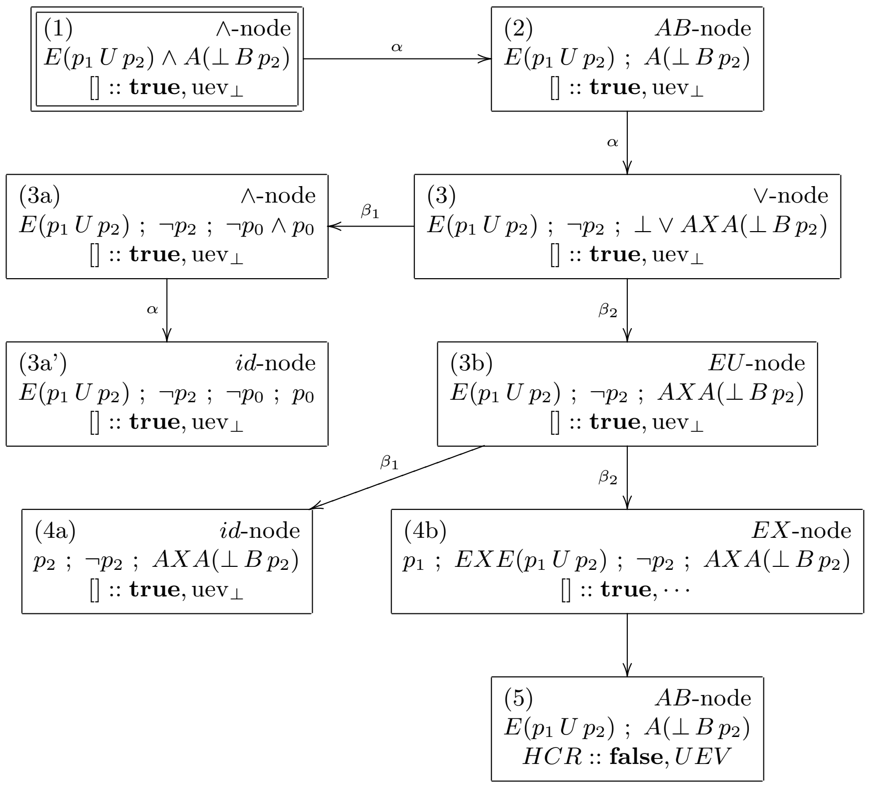
(1) $\wedge$-node
$E(p_1\,U\,p_2) \wedge A(\bot\,B\,p_2)$
$[] :: \mathbf{true}, \mathrm{uev}_\bot$
α
(2) AB-node
$E(p_1\,U\,p_2)$; $A(\bot\,B\,p_2)$
$[] :: \mathbf{true}, \mathrm{uev}_\bot$
α
(3) $\vee$-node
$E(p_1\,U\,p_2)$; $\neg p_2$; $\bot \vee AX\,A(\bot\,B\,p_2)$
$[] :: \mathbf{true}, \mathrm{uev}_\bot$
β_1
(3a) $\wedge$-node
$E(p_1\,U\,p_2)$; $\neg p_2$; $\neg p_0 \wedge p_0$
$[] :: \mathbf{true}, \mathrm{uev}_\bot$
α
(3a') id-node
$E(p_1\,U\,p_2)$; $\neg p_2$; $\neg p_0$; p_0
$[] :: \mathbf{true}, \mathrm{uev}_\bot$
β_2
(3b) EU-node
$E(p_1\,U\,p_2)$; $\neg p_2$; $AX\,A(\bot\,B\,p_2)$
$[] :: \mathbf{true}, \mathrm{uev}_\bot$
β_1
(4a) id-node
p_2 ; $\neg p_2$; $AX\,A(\bot\,B\,p_2)$
$[] :: \mathbf{true}, \mathrm{uev}_\bot$
β_2
(4b) EX-node
p_1 ; $EX\,E(p_1\,U\,p_2)$; $\neg p_2$; $AX\,A(\bot\,B\,p_2)$
$[] :: \mathbf{true}, \cdots$
(5) AB-node
$E(p_1\,U\,p_2)$; $A(\bot\,B\,p_2)$
$HCR :: \mathbf{false}, UEV$

Fig. 4. An example: a tableau for $E(p_1\,U\,p_2) \wedge A(\bot\,B\,p_2)$

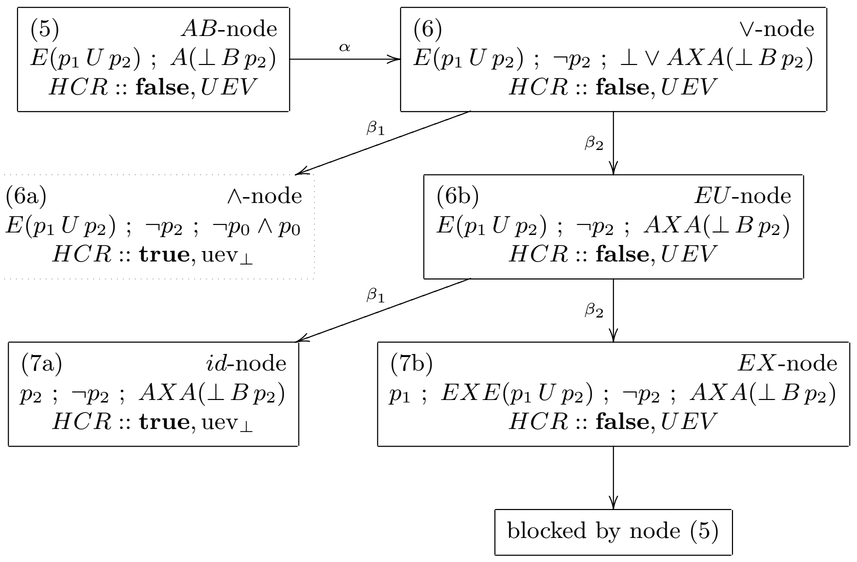
(5) AB-node
$E(p_1\,U\,p_2)$; $A(\bot\,B\,p_2)$
HCR :: **false**, UEV
α
(6) $\vee$-node
$E(p_1\,U\,p_2)$; $\neg p_2$; $\bot \vee AXA(\bot\,B\,p_2)$
HCR :: **false**, UEV
β_1
β_2
(6a) $\wedge$-node
$E(p_1\,U\,p_2)$; $\neg p_2$; $\neg p_0 \wedge p_0$
HCR :: **true**, $\mathrm{uev}_\bot$
(6b) EU-node
$E(p_1\,U\,p_2)$; $\neg p_2$; $AXA(\bot\,B\,p_2)$
HCR :: **false**, UEV
β_1
β_2
(7a) id-node
p_2 ; $\neg p_2$; $AXA(\bot\,B\,p_2)$
HCR :: **true**, $\mathrm{uev}_\bot$
(7b) EX-node
p_1 ; $EXE(p_1\,U\,p_2)$; $\neg p_2$; $AXA(\bot\,B\,p_2)$
HCR :: **false**, UEV
blocked by node (5)

Fig. 5. An example: a tableau for $E(p_1\,U\,p_2) \wedge A(\bot\,B\,p_2)$ (continued)

7.2 A Fully Worked Example

As an example, consider the formula $E(p_1\,U\,p_2) \wedge \neg E(\top\,U\,p_2)$ which is obviously not satisfiable. Converting the formula into negation normal form gives us $E(p_1\,U\,p_2) \wedge A(\bot\,B\,p_2)$. Hence, any expanded tableau with root $E(p_1\,U\,p_2) \wedge A(\bot\,B\,p_2)$ should be marked.

Figure 4 and Fig. 5 show such a tableau where the root node is node (1) in Fig. 4 and where Fig. 5 shows the sub-tableau rooted at node (5). Each node is classified as a ρ-node if rule ρ is applied to that node in the tableau. The unlabelled edges go from states to pre-states. Dotted frames indicate that the sub-tableaux at these nodes are not shown because they are very similar to sub-tableaux of other nodes: that is node (6a) behaves the same way as node (3a). Dots "$\cdots$" indicate that the corresponding values are not important because they are not needed to calculate the value of any other history or variable. The partial function UEV maps the formula $E(p_1\,U\,p_2)$ to 1 and is undefined otherwise as explained below. The history HCR is defined as $HCR := [\{E(p_1\,U\,p_2), A(\bot\,B\,p_2)\}]$.

The marking of the nodes (1) to (4a) in Fig. 4 with **true** is straightforward. Note that $\bot$ is just an abbreviation for $\neg p_0 \wedge p_0$ to save some space and make things easier for the reader; the tableau procedure as described in this paper does not know about the symbol $\bot$. It is, however, not a problem to adapt the rules so that the tableau procedure can handle $\top$ and $\bot$ directly. For node (5), our procedure constructs the tableau shown in Fig. 5. The leaf (7b) is an EX-node, but it is "blocked" from creating the desired successor containing $\{E(p_1\,U\,p_2), A(\bot\,B\,p_2)\}$ because there is a $j \in \mathbb{N}$ such that $\mathrm{HCr}_{7\mathrm{b}}[j] =$

$HCR[j] = \{E(p_1 \, U \, p_2), A(\bot \, B \, p_2)\}$: namely $j = 1$. Thus the EX-rule computes $UEV(E(p_1 \, U \, p_2)) = 1$ as stated above and also puts $\mathrm{mrk}_{7\mathrm{b}} := \mathbf{false}$. As the nodes (7a) and (6a) are marked, the function UEV is passed on to the nodes (6b), (6), and (5) according to the corresponding β- and α-rules.

The crux of our procedure happens at node (4b) which is an EX-node with $\mathrm{HCr}_{4\mathrm{b}} = []$ and hence $len(\mathrm{HCr}_{4\mathrm{b}}) = 0$. The EX-rule therefore finds a child node (5) and a formula $E(p_1 \, U \, p_2)$ in it such that $1 = UEV(E(p_1 \, U \, p_2)) = \mathrm{uev}_5(E(p_1 \, U \, p_2)) > len(\mathrm{HCr}_{4\mathrm{b}}) = 0$. That is, node (4b) "sees" a child (5) that "loops lower", meaning that node (5) is the root of an "isolated" subtree which does not fulfil its eventuality $E(p_1 \, U \, p_2)$. Thus the EX-rule sets $\mathrm{mrk}_{4\mathrm{b}} = \mathbf{true}$, marking (4b) as "closed". The propagation of **true** to the root is then just via simple β- and α-rule applications.

7.3 One-pass And-Or Tree Tableaux for other fixpoint logics

One-pass And-Or tree tableaux were first given by Schwendimann [20] for LTL. There is a slight bug in the original formulation but a correct version can be obtained from our method for CTL by using the appropriate α/β-rules for LTL instead of CTL in our description and by changing the (EX)-rule to be linear since the premise of this rule becomes $\bigcirc\varphi; \bigcirc\Delta; \Lambda$ and the conclusion just becomes $\varphi; \Delta$. A correct implementation can be found here: `http://users.cecs.anu.edu.au/~rpg/PLTLProvers/`. A recent experimental comparison of it also exists [25].

One-pass And-Or tree tableaux for PDL also exist [6] and a correct implementation can be found here: `http://users.cecs.anu.edu.au/~rpg/PDLProvers/`

The method has been extended to the logic of common knowledge (LCK) [26].

8 On-the-fly And-Or Graph Tableaux for PDL

The one-pass tableau given in the previous section are complexity-suboptimal: 2EXPTIME rather than EXPTIME. Next we show how to regain complexity optimality. Again, we ignore global assumptions (TBoxes) for simplicity.

Our algorithm starts at a root containing a given formula ϕ and builds an and-or tree in a depth-first and left to right manner to try to build a model for ϕ. The rules are based on the semantics of PDL and either add formulae to the current world using Smullyan's α/β rules from Table 1, or create a new world in the underlying model and add the appropriate formulae to it. For a node x, the attribute Γ_x carries this set of formulae.

The strategy for rule applications is the usual one where we "saturate" a node using the α/β-rules until they are no longer applicable, giving a "state" node s, and then, for each $\langle a \rangle \xi$ in s, we create an a-successor node containing $\{\xi\} \cup \Delta$, where $\Delta = \{\psi \mid [a]\psi \in s\}$. These successors are saturated to produce new states using the α/β-rules, and we create the successors of these new states, and so on.

Our strategy can produce infinite branches as the same node can be created repeatedly on the same branch. We therefore "block" a node from being created

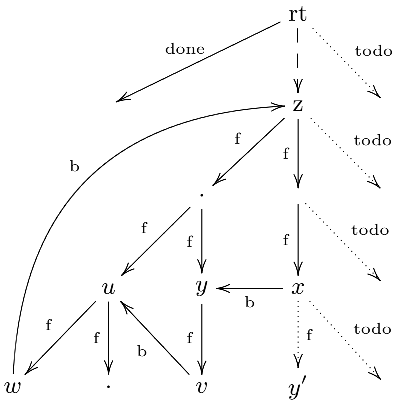

Fig. 6. Graph constructed by our algorithm using forward (f) and backward edges (b)

if this node exists already on any previous branch, thereby using global caching again, but now nodes are required to contain "focused sets of formulae" [6]. For example, in Fig. 6, if the node y' already exists in the tree, say as node y, then we create a "backward" edge from x to y (as shown) and do not create y'. If y' does not duplicate an existing node then we create y' and add a "forward" edge from x to y'. The distinction between "forward" and "backward" edges is important for the proofs. Thus our tableau is a tree of forward edges, with backward edges that either point upwards from a node to a "forward-ancestor", or point leftwards from one branch to another. Cycles can arise only via backward edges to a forward-ancestor.

Our tableau must "fulfil" every formula of the form $\langle\delta\rangle\varphi$ in a node but only eventualities, defined as those where δ contains $*$-connectives, cause problems. If $\langle\delta\rangle\varphi$ is not an eventuality, the α/β-rules reduce the size of the principal formula, ensuring fulfilment. If $\langle\delta\rangle\varphi$ is an eventuality, the main problem is the β-rule for formulae of the form $\langle\gamma*\rangle\varphi$. Its left child reduces $\langle\gamma*\rangle\varphi$ to a strict subformula φ, but the right child "reduces" it to $\langle\gamma\rangle\langle\gamma*\rangle\varphi$. If the left child is always inconsistent, this rule can "procrastinate" an eventuality $\langle\gamma*\rangle\varphi$ indefinitely and never find a world which makes φ true. This non-local property must be checked globally by tracking eventualities.

Table 1. Smullyan's α- and β-notation to classify formulae

α	$\varphi \wedge \psi$	$[\gamma \cup \delta]\varphi$	$[\gamma*]\varphi$	$\langle\psi?\rangle\varphi$	$\langle\gamma;\delta\rangle\varphi$	$[\gamma;\delta]\varphi$
α_1	φ	$[\gamma]\varphi$	φ	φ	$\langle\gamma\rangle\langle\delta\rangle\varphi$	$[\gamma][\delta]\varphi$
α_2	ψ	$[\delta]\varphi$	$[\gamma][\gamma*]\varphi$	ψ		

β	$\varphi \vee \psi$	$\langle\gamma \cup \delta\rangle\varphi$	$\langle\gamma*\rangle\varphi$	$[\psi?]\varphi$
β_1	φ	$\langle\gamma\rangle\varphi$	φ	φ
β_2	ψ	$\langle\delta\rangle\varphi$	$\langle\gamma\rangle\langle\gamma*\rangle\varphi$	$\sim\psi$

Consider Fig. 6, and suppose the current node x contains an eventuality e_x. We distinguish three cases. The first is that some path from x fulfils e_x in the existing tree. Else, the second case is that some path from x always procrastinates the fulfilment of e_x and hits a forward-ancestor of x on the current branch: *e.g.* the path x, y, v, u, w, z. The forward-ancestor z contains some "reduction" e_z of e_x. The path from the root to the current node x contains the only currently existing nodes which may need further expansion, and may allow z to fulfil e_z at a later stage, and hence fulfil e_x. We call the pair (z, e_z) a "potential rescuer" of e_x in Γ_x. The only remaining case is that $e_x \in \Gamma_x$ is unfulfilled, has no potential rescuers, and hence can never become fulfilled later, so x can be "closed". The machinery to distinguish these three cases and compute, if needed, all currently existing potential rescuers of every eventuality in Γ_x is described next.

A tableau node x also contains a status sts_x. The value of sts_x is the constant `closed` if the node x is closed. Otherwise, the node is "open" and sts_x contains a function prs which maps each eventuality $e_x \in \Gamma_x$ to $\bot$ or to a set of pairs (v, e) where v is a forward-ancestor of x and e is an eventuality. The status of a node is determined from those of its children once they have all been processed. A closed child's status is propagated as usual, but the propagation of the function prs from open children is more complicated. The intuition is that we must preserve the following invariant for each eventuality $e_x \in \Gamma_x$:

> if e_x is fulfilled in the tree to the left of the path from the root to the node x then $\mathrm{prs}_x(e_x) := \bot$, else $\mathrm{prs}_x(e_x)$ is exactly the set of all potential rescuers of e_x in the current tableau.

An eventuality $e_x \in \Gamma_x$ whose $\mathrm{prs}_x(e_x)$ becomes the empty set can never become fulfilled later, so $\mathrm{sts}_x :=$ `closed`, thus covering the three cases as desired.

Whenever a node n gets a status `closed`, we interrupt the depth-first and left-to-right traversal and invoke a separate procedure which explicitly propagates this status transitively throughout the and-or graph rooted at n. For example, if z gets closed then so will its backward-parent w, which may also close u and so on. This propagation (update) may break the invariant for some eventuality e in this subgraph by interrupting the path from e to a node that fulfils e or to a potential rescuer of e. We must therefore ensure that the propagation (update) procedure re-establishes the invariant in these cases by changing the appropriate prs entries. At the end of the propagation (update) procedure, we resume the usual depth-first and left-to-right traversal of the tree by returning the status of n to its forward-parent. This "on-the-fly" nature guarantees that unfulfilled eventualities are detected as early as possible.

Our algorithm terminates, runs in EXPTIME, and formula ϕ is satisfiable iff the root is open [6].

9 On-the-fly And-Or Graph Tableaux for CPDL

The methods described in the previous sections can be combined to give a complexity-optimal on-the-fly And-Or graph tableau method for CPDL but the extension is non-trivial and cannot really be described without giving an actual algorithm [7]. An implementation by Florian Widmann can be found here: `http://users.cecs.anu.edu.au/~rpg/CPDLTabProver/`

10 Further Work

All methods described here have been implemented (`http://users.cecs.anu.edu.au/~rpg/software.html`) but further work is required to add optimisations to make these methods practical on large examples and to extend them to more expressive logics like SHOIQ: but see [27, 28]. There is also the possibility to marry these methods with advances from SAT and SMT [29].

Our original aim in this endeavour was to obtain tableaux algorithms for the modal mu-calculus but we have been unable to extend our one-pass or on-the-fly methods to this logic. Similarly, we have been unable to extend our methods to handle full computation tree logic CTL*. Finally, we have been unable to find a complexity-optimal and-or graph tableaux method for CTL using our approach.

Tableaux-like methods for both CTL* and the modal mu-calculus have been given and implemented, and are an exciting avenue for future work [30–33].

References

1. Vardi, M.Y.: From philosophical to industrial logics. In: ICLA. (2009) 89–115
2. Goré, R., Nguyen, L.A.: EXPTIME tableaux for ALC using sound global caching. In: Proc. of the International Workshop on Description Logics (DL-07). (2007)
3. Goré, R., Nguyen, L.: EXPTIME tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In Olivetti, N., ed.: Proc. TABLEAUX'2007. Volume 4548 of LNCS., Springer (2007) 133–148
4. Goré, R., Postniece, L.: An experimental evaluation of global caching for ALC (system description). In: Proc. IJCAR-08. Volume 5195 of LNCS., Springer (2008) 299–305
5. Goré, R., Widmann, F.: Sound global state caching for ALC with inverse roles. In: Proc. TABLEAUX-09. Volume 5607 of LNCS., Springer (2009) 205–219
6. Goré, R., Widmann, F.: An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability. In: Proc. CADE-09. Volume 5663 of LNCS. (2009) 437–452
7. Goré, R., Widmann, F.: Optimal and cut-free tableaux for propositional dynamic logic with converse. In: IJCAR. (2010) 225–239
8. Widmann, F.: Tableaux-based Decision Procedures for Fixpoint Logics. PhD thesis, The Australian National University, Australia (2010)
9. Goré, R.: Tableau methods for modal and temporal logics. In D'Agostino, M., Gabbay, D.M., Hähnle, R., Posegga, J., eds.: Handbook of Tableau Methods. Kluwer (1999) 297–396
10. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge, England (2003)
11. Pratt, V.R.: A near-optimal method for reasoning about action. Journal of Computer and System Sciences **20**(2) (1980) 231–254

12. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. Journal of Computer and Systems Science **18** (1979) 194–211
13. Beth, E.: On Padoa's method in the theory of definition. Indag. Math. **15** (1953) 330–339
14. Goré, R., Nguyen, L.: ExpTime tableaux for $\mathcal{ALC}$ using sound global caching. In et al., D.C., ed.: Proc. DL'2007. (2007) 299–306
15. Wolper, P.: Temporal logic can be more expressive. Information and Control **56** (1983) 72–99
16. Ben-Ari, M., Manna, Z., Pnueli, A.: The temporal logic of branching time. In: Proceedings of Principle of Programming Langauages. (1981)
17. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. Journal of Computer and System Sciences **30**(1) (1985) 1–24
18. Horrocks, I., Sattler, U.: A tableaux decision procedure for shoiq. In: IJCAI. (2005) 448–453
19. Kesten, Y., Manna, Z., McGuire, H., Pnueli, A.: A decision algorithm for full propositional temporal logic. In: Computer Aided Verification. (1993) 97–109
20. Schwendimann, S.: A new one-pass tableau calculus for PLTL. In de Swart, H., ed.: Proc. TABLEAUX-98. Volume 1397 of LNAI., Springer (1998) 277–291
21. Janssen, G.: Logics for Digital Circuit Verication: Theory, Algorithms, and Applications. PhD thesis, Eindhoven University of Technology, The Netherlands (1999)
22. Hustadt, U., Konev, B.: TRP++: A temporal resolution prover. In Baaz, M., Makowsky, J., Voronkov, A., eds.: Collegium Logicum. Kurt Gödel Society (2004) 65–79
23. Baader, F.: Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In: Proc. IJCAI-91. (1991) 446–451
24. Abate, P., Goré, R., Widmann, F.: One-pass tableaux for computation tree logic. In: Proc. LPAR-07. Volume 4790 of LNCS., Springer (2007) 32–46
25. Schuppan, V., Darmawan, L.: Evaluating ltl satisfiability solvers. In: ATVA. (2011) 397–413
26. Abate, P., Goré, R., Widmann, F.: Cut-free single-pass tableaux for the logic of common knowledge. In: Workshop on Agents and Deduction at TABLEAUX-07. (2007)
27. Nguyen, L.A., Golinska-Pilarek, J.: An exptime tableau method for dealing with nominals and quantified number restrictions in deciding the description logic SHOQ. In: CS&P. (2013) 296–308
28. Nguyen, L.A.: A tableau method with optimal complexity for deciding the description logic SHIQ. In: Advanced Computational Methods for Knowledge Engineering. (2013) 331–342
29. Suda, M., Weidenbach, C.: A pltl-prover based on labelled superposition with partial model guidance. In: IJCAR. (2012) 537–543
30. Jungteerapanich, N.: A tableau system for the modal μ-calculus. In: TABLEAUX. (2009) 220–234
31. Friedmann, O., Latte, M., Lange, M.: A decision procedure for CTL* based on tableaux and automata. In: IJCAR. (2010) 331–345
32. Friedmann, O., Lange, M.: A solver for modal fixpoint logics. Electr. Notes Theor. Comput. Sci. **262** (2010) 99–111
33. Reynolds, M.: A faster tableau for CTL*. In: GandALF. (2013) 50–63