Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 278 (2011) 145–158

www.elsevier.com/locate/entcs

A Rooted Tableau for BCTL*

John Christopher M^cCabe-Dansted

Computer Science and Software Engineering
University of Western Australia
Perth, Austrlia

Abstract

The existing pure tableau technique for satisfiability checking BCTL* [7] begins by constructing all possible colours. Traditional tableaux begin with a single root node, and only construct formulae that are derived from that root. These rooted tableaux provide much better performance on most real world formulae, as they only need to construct a fraction of the possible nodes. We present a rooted variant of this tableau for BCTL*, together with an implementation demonstrating the performance of our rooted variant is superior to the original; this implementation is made available as a Java applet. We discuss further possible optimisations. This research will be useful in finding an optimised rooted tableau for CTL*.

Keywords: Full Computation Tree Logic, BCTL*, Tableau, Bundled.

1 Introduction

There has been recent renewed interest in decision procedures for Full Computation Tree Logic (CTL*). It has long been known that CTL* is decidable and is 2EXPTIME complete, [1,2] provides a doubly exponential automaton based satisfiability checker, and [9] gives a lower-bound. The automaton based satisfiability checkers are expected to have poor performance on average and have not been implemented [3]. Recently, tableau based decision procedures have been proposed that have greater potential for reasonable real world performance, and that have publicly accessible implementations [8,3]. The CTL* tableau of [8] builds upon a previous tableau for Bundled Full Computation Tree Logic (BCTL*) [7].

BCTL* is a variant of CTL* which has the same syntax but where only paths in a certain bundle are considered. As the bundle can contain all paths, if a formula is satisfiable in CTL* it is also satisfiable in BCTL*. Similarly if it is a theorem of BCTL* it is also a theorem of CTL*, as such it is reasonable to prove that a formula is a theorem of BCTL* where possible so we can use the theorem without

[1] Email: john@csse.uwa.edu.au

doi:10.1016/j.entcs.2011.10.012

consideration of the difference between BCTL* and CTL*. Also, we note that when only finite periods of time are considered the bundled and unbundled semantics are equivalent (see for example [5]). See Section 2.4 for a comparison of CTL* and BCTL*.

A traditional tableau begins with a single formula and then builds a tree, with the label rooted with that formula. The tableau defined in [7] instead begins with a maximal tableau, and then prunes nodes. This is sufficient to give optimum worst-case performance results; however, it means that we always have to construct all possible nodes. So, with the tableau of [7], it would take longer to check the satisfiability of $\phi \wedge p \wedge \neg p$ than ϕ. By contrast, a rooted tableau would typically find the contradiction quickly and terminate.

Although the hybrid satisfiability checker for CTL* of [3] appears to be faster overall than [8], we investigate finding a rooted tableau for three reasons. Firstly, the tableau of [8] can be quick, for example, using the default settings, the tableau of [8] shows that $(p \to (qUr)) \wedge (qUp)) \to (qUr)$ is satisfiable in 0.16s, while constructing the automata for [3] alone requires 395 seconds on the same Core2Duo. Thus, it makes sense to investigate improvements to tableaux of [7] and [8] to determine whether they can outperform [3]. Secondly, as [3] is based on a rooted tableau, it makes more sense to compare [3] to a rooted variant of [8]. Thirdly, the approach of [7] and [8] works by constructing a finite tableau built from subformulae of the formula inputted by the user, with each step being based upon the semantics of (B)CTL*. The workings of these tableaux are more meaningful to the user than those of [3], which reasons about the existence of an infinite tableau using a parity game.

In this paper we propose a rooted variant [2] of the tableau for BCTL* in [7]. Although, BCTL* has some advantages over CTL* the primary reason we chose to implement a BCTL* tableau is that the CTL* tableau of [8] is built from the much simpler BCTL* tableau of [7]. As such it makes sense to study and optimise the simpler BCTL* tableaux before trying to optimise the corresponding CTL* tableau.

2 BCTL* Logic

2.1 BCTL Syntax*

As with CTL*, BCTL* has a set $\mathcal{V}$ of atomic propositions that we call atoms. Where p varies over $\mathcal{V}$, we define BCTL* (and CTL*) formulae according to the following abstract syntax

$$\phi := \top \mid p \mid \neg\phi \mid (\phi \wedge \phi) \mid (\phi U \phi) \mid X\phi \mid A\phi \ .$$

The $\top$, $\neg$, $\wedge$, X, U and A are the familiar "true", "not", "and", "next", "until" and "all paths" operators from CTL and CTL*. The abbreviations $\bot$, $\vee$, F, G, W, E $\to$ and $\leftrightarrow$ are defined as in CTL* logic, that is: $\bot \equiv (p \wedge \neg p)$, $\top \equiv \neg\bot$,

[2] This is available as a Java Applet at: http://www.csse.uwa.edu.au/~john/BCTL2/, and includes the tableaux of [7] and [8] for comparison.

$\alpha \vee \beta \equiv \neg(\neg\alpha \vee \neg\beta)$, "Finally" $F\alpha = \top U \alpha$, "Globally/Always" $G\alpha \equiv \neg F \neg \alpha$, "Weak Until" $\alpha W \beta \equiv \alpha U \beta \vee G\alpha$, "Exists a Path" $E\alpha \equiv \neg A \neg \alpha$, $\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$ and $\alpha \leftrightarrow \beta \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$.

2.2 CTL-Structures

Definition 2.1 We say that a binary relation R on S is serial (total) if for every a in S there exists b in S such that aRb. We will sometimes write aRb as $(a,b) \in R$.

Definition 2.2 A transition frame is a pair $(W, \rightarrowtail)$, where W is a non-empty set of states and $\rightarrowtail$ is a serial relation on W.

Definition 2.3 We let $\mathcal{V}$ be our set of atoms. A valuation g is a map from a set of states W to the power set of the atoms. Informally, the statement $p \in g(w)$ means "the atom p is true at state w".

Definition 2.4 We call an ω-sequence $\sigma = \langle w_0, w_1, \ldots \rangle$ of states a fullpath iff for all non-negative integers i we have $w_i \rightarrowtail w_{i+1}$. For all i in $\mathbb{N}$ we define $\sigma_{\geq i}$ to be the fullpath $\langle w_i, w_{i+1}, \ldots \rangle$, we define σ_i to be w_i and we define $\sigma_{\leq i}$ to be the sequence $\langle w_0, w_1, \ldots, w_i \rangle$. We say that a set of fullpaths B is fusion closed iff for all non-negative integers i, j and $\sigma, \pi \in B$ we have $\langle \sigma_0, \sigma_1, \ldots, \sigma_i, \pi_j, \pi_{j+1}, \ldots \rangle \in B$ if $\sigma_{i+1} = \pi_j$. We say that a set of fullpaths B is suffix closed iff for all integers i and $\sigma \in B$ we have $\sigma_{\geq i} \in B$. We say a set of fullpaths is a bundle if it is non-empty, suffix closed and fusion closed.

Definition 2.5 A BCTL-structure $M = (W, \rightarrowtail, g, B)$ is a 4-tuple containing a set of states W, a serial binary relation $\rightarrowtail$, a valuation g on the set of states W, and B is a bundle on $(W, \rightarrowtail)$.

2.3 BCTL Semantics*

We define truth of a BCTL* formula ϕ on a fullpath $\sigma = \langle w_0, w_1, \ldots \rangle$ in a BCTL-structure M recursively as follows:

$$\begin{aligned} M, \sigma \vDash X\phi &\text{ iff } M, \sigma_{\geq 1} \vDash \phi \\ M, \sigma \vDash \phi U \psi &\text{ iff } \exists_{i \in \mathbb{N}} \text{ s.t. } M, \sigma_{\geq i} \vDash \psi \text{ and } \forall_{j \in \mathbb{N}} j < i \implies M, \sigma_{\geq j} \vDash \psi \\ M, \sigma \vDash A\phi &\text{ iff } \forall_{\pi \in B \text{ s.t. } \pi_0 = \sigma_0} M, \pi \vDash \phi \end{aligned}$$

The definitions for p, $\neg$ and $\wedge$ are as we would expect from classical logic:

$$\begin{aligned} M, \sigma \vDash p &\text{ iff } p \in g(p_0) \\ M, \sigma \vDash \neg\phi &\text{ iff } M, \sigma \nvDash \phi \\ M, \sigma \vDash \phi \wedge \psi &\text{ iff } M, \sigma \vDash \phi \wedge M, \sigma \vDash \psi \ , \end{aligned}$$

We say that a formula ϕ is satisfiable in BCTL* iff there exists a BCTL structure $M = (W, \rightarrowtail, g, B)$ such that there exists a path σ in B such that we have $M, \sigma \vDash \phi$. A formula is valid iff its negation is not satisfiable. CTL* is similar, but B must be the set of all possible paths through $\rightarrowtail$.

2.4 CTL*

In this paper we will not formally consider CTL*; however, we will briefly discuss it here so we can compare it to BCTL*. We see that if a formula is satisfiable in CTL* then it must be satisfiable in BCTL*; equivalently if it is valid in BCTL* then it must be valid in CTL*. The axiomatisation of CTL* [6] includes a limit closure (LC) axiom, which is not valid in BCTL:

$$\text{LC: } AG(E\alpha \to EX((E\beta U E\alpha)) \to (E\alpha \to EG((E\beta U E\alpha)))$$

Consider a system where we process non-empty input of finite but unbounded size. Since the size of input is unbounded, if we have just read an input symbol, then it is possible that we will read another input symbol at the next step, which we may want to represent as $r \to EXr$; this will always be true since no matter how many input symbols we have read, it is always possible there will be one more, which we can represent as $AG\,(r \to EXr)$, under CTL* we can deduce that $r \to EGr$. Since the input is non-empty we have r and thus EGr; indicating that there is a computation path where we read new input symbols forever. However, we have stated that the input is always finite. We see that the semantics of CTL* are not appropriate to this specification. It is reasonable to first attempt to prove that a formula is valid in BCTL*, and only worry about whether the semantics of BCTL* or CTL* are more appropriate if it turns out not to be valid in BCTL*.

3 A Tableau for BCTL*

Here we define a tableau BCTL*-RTAB for checking the satisfiability of BCTL* formulae. This tableau is derived from Reynolds' [7] tableau for BCTL*.

Definition 3.1 For any pair of formulae (ϕ, ψ), we say that $\phi \leq \psi$ iff ϕ is a subformula of ψ.

Definition 3.2 The closure $\mathbf{cl}\phi$ of the formula ϕ is defined as the smallest set that satisfies the four following requirements:

(i) $\phi \in \mathbf{cl}\phi$

(ii) For all $\psi \in \mathbf{cl}\phi$, if $\delta \leq \psi$ then $\delta \in \mathbf{cl}\phi$.

(iii) For all $\psi \in \mathbf{cl}\phi$, $\neg\psi \in \mathbf{cl}\phi$ or there exists δ such that $\psi = \neg\delta$ and $\delta \in \mathbf{cl}\phi$.

Definition 3.3 We say that $a \subseteq \mathbf{cl}\phi$ is Partially Propositionally Consistent (PPC) iff for all $\alpha, \beta \in \mathbf{cl}\phi$:

(M1) if $\neg\neg\alpha \in a$ then $\alpha \in a$,

(M2) if $(\alpha \wedge \beta) \in a$ then $\alpha \in a$ and $\beta \in a$.

PPC sets are quite similar to Maximally Propositionally Consistent (MPC) sets, but with MPC sets M1 is stronger: if $\beta = \neg\alpha$ then $\beta \in a$ iff $\alpha \notin a$. With an MPC set a, for every formula α in the closure, a either has α or its negation. By contrast,

the intuition behind PPC is that a PPC set a can exclude both α and $\neg\alpha$ unless one of these formulae is a direct consequence of other formulae in a.

A hue is roughly speaking a set of formulae that could hold along a single fullpath.

Definition 3.4 [Hue] A set $a \subseteq \mathbf{cl}\phi$ is a hue for ϕ if a satisfies H1,H3 and H4 (below). A set $a \subseteq \mathbf{cl}\phi$ is a finished hue for ϕ iff it satisfies all of:

(H1) a is PPC;

(H2) if $\alpha U \beta \in a$ then $\alpha \in a$ or $\beta \in a$;

(H3) if $\neg(\alpha U \beta) \in a$ then $\neg\beta \in a$; and

(H4) if $A\alpha \in a$ then $\alpha \in a$.

The hues of [7] are similar to finished hues, exception that they must be MPC (not just PPC). We do not include H2 in our definition of hues so that the following definition is unique:

Definition 3.5 For a set a of formulae we let $\mathrm{hue}(a)$ be the minimal superset of a that is also a hue.

Definition 3.6 Where h is a finished hue we define $\mathrm{succ}(a)$ as the minimal set such that

(i) if $X\alpha \in a$ then $\alpha \in \mathrm{succ}(a)$; and

(ii) if $\neg\beta \in a$ and $\alpha U \beta \in a$ then $\alpha U \beta \in \mathrm{succ}(a)$; and

(iii) if $\neg(\alpha U \beta) \in a$ and $\alpha \in a$; then $\neg(\alpha U \beta) \in \mathrm{succ}(a)$.

Definition 3.7 [r_A] For hues a, b, we put (a, b) in r_A iff the following conditions hold:

(A1) $A\alpha \in a$ iff $A\alpha \in b$

(A2) For all $p \in \mathcal{V}$, we have $p \in a$ iff $p \in b$

Definition 3.8 We say a set C of hues is a colour if it satisfies C1. A set of hues C is a finished colour of ϕ iff it satisfies both of:

(C1) for all $a, b \in C$ we have $(a, b) \in r_A$; and

(C2) if $a \in C$ and $\neg A\alpha \in a$ then there is $b \in C$ such that $\neg\alpha \in b$; and

We now define a function Col; given a set of sets of formulae the set $\mathrm{Col}(C)$ is intuitively the minimal colour that is a superset of C; however, we have not defined minimal sets of sets, so we formally define Col as follows:

Definition 3.9 For a set of sets of formulae C, the set $(C)^\star$ is the set that results when we iteratively:

(i) replace each member a of C with $\mathrm{hue}(a)$;

(ii) replace each member a of C with $a \cup \{\alpha\}$ where $\alpha \in b \in C$ and α is of the form $A\beta$, $\neg A\beta$, p or $\neg p$.

(iii) repeat 1 and 2 until it is not possible to modify C any further.

All of the tableau rules use the above definition to ensure that every node is a colour. This increases the similarity of this tableau to that of [7] (where all nodes are colours).

As with [7], a hue a represents the formula $\bigwedge a$, and $C = \{h_1, h_2, \ldots, h_n\}$ represents the formula $A(\bigvee_i(\bigwedge h_i)) \wedge \bigwedge_i E(\bigwedge h_i)$.

In the tableau we allow the hues to contain a special marker e. This indicates that we have chosen this hue as the hue that we will try to find temporal successors for. We will always have e in exactly one hue of each colour. For a colour C we let C_e be the hue in C that contains e. The hue containing e is similar to the first hue h_0 in the paper [8], incidentally this is also how we implemented e in our sample implementation.

Definition 3.10 For each formula ψ of one of the forms below, we define a left choice $L(\psi)$ and right choice $R(\psi)$ as follows:

ψ	$L(\psi)$	$R(\psi)$
$\neg(\alpha \wedge \beta)$	$\neg\alpha$	$\neg\beta$
$(\alpha U \beta)$	β	$\neg\beta$
$\neg(\alpha U \beta)$	$\neg\alpha$	α

Note that the bottom two rules are required because (as with [7]) we have not added $X(\alpha U\beta)$ to the closure set so we use the pair of formulae $\neg\beta$ and $(\alpha U\beta)$ to indicate that the temporal successor must also contain $\alpha U\beta$.

A rule node is a tuple $(C, \text{“}R\text{”})$, where C is a colour, and a temporal node is a tuple $(C, \text{“}T\text{”})$. A node is either a rule node or a temporal node. Each node is associated with a branch. Each branch has a letter to indicate the type of branch, and a node is a rule node unless it is the child of a H-branch (defined below).

We begin the tableau with a single node $((\{\{\phi, e\}, \emptyset\})^\star, \text{“}R\text{”})$. We need to include the empty hue $\emptyset$, as otherwise the initial colour would represent the case where all paths satisfy ϕ, which may not be the case. We then define the tableau iteratively as follows: for each rule node $(C, \text{“}R\text{”})$, where $C = \{h_1, h_2, \ldots, h_{|C|}\}$, we associate a branch as defined as follows,

(i) If there exists $\psi \in h \in C$ for some leaf such that $L(\psi)$ is defined, $L(\psi) \notin h$, $R(\psi) \notin h$ then we let the branch of $(C, \text{“}R\text{”})$ be a disjunctive B-branch with the following four children:
  (a) A child $((C_1)^\star, \text{“}R\text{”})$ where C_1 results from adding $L(\psi)$ to h in C.
  (b) A child $((C_2)^\star, \text{“}R\text{”})$ where C_2 results from adding $R(\psi)$ to h in C.
  (c) A child $((C_3)^\star, \text{“}R\text{”})$ where $C_3 = C_1 \cup (C_2 - \{e\})$.
  (d) A child $((C_4)^\star, \text{“}R\text{”})$ where $C_4 = (C_1 - \{e\}) \cup (C_2)$.

(ii) If there exists $\neg A\psi \in a \in C$ such that $\forall b \in C$ we have $\neg\psi \notin b$, we then let the branch of $(C, \text{“}R\text{”})$ be a disjunctive E-branch, with $|C|$ children:

(a) The i^{th} child is $((C_i)^\star, \text{“}R\text{”})$, where $C_i = C \cup \{h_i \cup \{\neg\psi\} - \{e\}\}$.

(iii) If we cannot apply rules 1 or 2, then we let the branch of $(C, \text{“}R\text{”})$ be a disjunctive H-branch with $|C|$ children:

(a) the i^{th} child is $(C_i, \text{“}R\text{”})$, where C_i is the same as C except that e has been moved into the i^{th} hue (given some arbitrary ordering of the hues). We let the branch of $(C_i, \text{“}R\text{”})$ be a disjunctive X-branch with $2^{|C_i|-1}$ children. Each child of $(C_i, \text{“}R\text{”})$ is of the form $(C', \text{“}T\text{”})$ where C' is formed by picking some subset S of C such that $C_e \in S$, and replacing each a in S with $\text{succ}(a)$, finally we let C' be $(S)^\star$.

Intuitively, (i) is about finishing hues, (ii) is about finishing colours, and (iii) is for taking a temporal step, moving to the next world on the fullpath.

Definition 3.11 We define the set of eventualities in a colour to be the set of formulae of the form $\alpha U \beta$ such that $\alpha U \beta \in C_e$. We define the set of eventuality-children (eChildren) of an H-branch for a node $(C, \text{“}R\text{”})$ to be the singleton set $\{(C, \text{“}T\text{”})\}$ (that is, the eChild of an H-branch is the child that “moves” the e marker back to its original position); for all other branches we define the set of eChildren as being the same as the set of children. We define the eventuality descendants (eDescendants) as being the transitive closure of the eChildren. We say an eventuality $\alpha U \beta$ of C is directly satisfied if $\beta \in C_e$. We say an eventuality is satisfied at a node if there exists an eDescendant of the node where $\beta \in C_e$.

We say a disjunctive branch is covered if any of its children are not pruned. We say a conjunctive branch is covered if all its children are not pruned. We mark a branch as pruned if:

(i) the colour C of the branch is contradictory, i.e. $\{\psi, \neg\psi\} \subseteq h \in C$; or

(ii) the branch is not covered; or

(iii) there is an unsatisfied eventuality.

We say the tableau succeeds if it halts and the root is not pruned.

3.1 Soundness and completeness

BCTL*-RTAB is sound, that is, if it halts and succeeds on ϕ then ϕ is satisfiable in BCTL*. We sketch a proof of this here, for more details see [4].

Say that BCTL*-RTAB finishes with the set S' of colours. Then we define a BCTL-structure $M = (W, \rightarrowtail, g, B)$ as follows: the set of worlds W are the set of colours used in H-branches. We put a pair of colours (C, D) in $\rightarrowtail$ iff we can reach the node $(D, \text{“}R\text{”})$ from $(C, \text{“}R\text{”})$ in the tableau by crossing precisely one X-branch); the valuation $g(C)$ of a world/colour C contains an atom p iff $p \in C$. We now define the set of bundled paths B.

Definition 3.12 [r_X] The temporal successor r_X relation on hues below is defined as in [7]; for all hues a, b put (a, b) in r_X iff the following conditions are satisfied:

(R1) $\quad X\alpha \in a$ implies $\alpha \in b$

(R2) $\neg X\alpha \in a$ implies $\neg\alpha \in b$

(R3) $\alpha U\beta \in a$ and $\beta \notin a$ implies $\alpha U\beta \in b$

(R4) $\neg(\alpha U\beta) \in a$ and $\alpha \in a$ implies $\neg(\alpha U\beta) \in b$

Lemma 3.13 *For every $(C, D) \in \rightarrowtail$ we have $C_e r_X D_e$ and for every $b \in D$ we have $a \in C$ such that $a r_X b$.*

Note that the X-branch replaces every hue a with $\mathrm{succ}(a)$; that a is a finished hue and so $(a, \mathrm{succ}(a)) \in r_X$; that X-branches are the only branches that remove formulae from hues, and we only cross one X-branch when travelling from C to D for $(C, D) \in \rightarrowtail$. From this is easy to prove this Lemma 3.13.

Definition 3.14 We call an ω-sequence $\langle (c_0, h_0), (c_1, h_1), \ldots\rangle$ a thread through W iff for all $i \geq 0$: each $c_i \in S'$, each $h_i \in c_i$, each $(c_i, c_{i+1}) \in \rightarrowtail$, each $(h_i, h_{i+1}) \in r_X$. We say that this is a fulfilling thread iff for all $i \geq 0$

(i) For all formulae of the form $(\alpha U\beta)$ in h_i, there exists $j \geq i$ such that $\beta \in h_j$

We include a fullpath $\sigma = \langle c_0, c_1, \ldots\rangle$ in B iff there exists a fulfilling thread $\langle (c_0, h_0), (c_1, h_1), \ldots\rangle$, and we say that this thread justifies σ being in B.

Proposition 3.15 *If $\mu = \langle (c_0, h_0), (c_1, h_1), \ldots\rangle$ justifies $\sigma \in B$ then for all $j \geq 0$, $\mu_{\geq j}$ justifies $\sigma_{\geq j} \in B$.*

Lemma 3.16 *B is fusion closed*

Say that σ, π are in B and $\sigma_0 = \pi_1$. We will show below that

$$\langle \pi_0, \sigma_0, \sigma_1, \ldots\rangle \in B\ .$$

The general case where $\sigma_0 = \pi_j$ follows from prefix closure and induction.

As $\sigma \in B$, there is a fulfilling thread $\mu = \langle (\sigma_0, h_1), (\sigma_1, h_2), \ldots\rangle$. As $(\pi_0, \pi_1) \in \rightarrowtail$, we can choose h_0 from π_0 such that $(h_0, h_1) \in r_X$ (see Lemma 3.13).

If $\alpha U\beta \in h_0$, then $\beta \in h_0$ or $\alpha U\beta \in h_1$. As μ is fulfilling, if $\alpha U\beta \in h_1$ then there exists $j \geq 1$ such that $\beta \in h_j$.

Lemma 3.17 *If $h \in c \in S'$ then there is a fulfilling thread*

$$\mu = \langle (c_0, h_0), (c_1, h_1), \ldots\rangle$$

such that $h_0 = h$ and $c_0 = c$. Thus $\sigma = \langle c_0, c_1, c_2, \ldots\rangle \in B$.

As with Reynolds [7] we iteratively satisfy the oldest eventuality first. Hence every eventuality is eventually fulfilled.

Say we have chosen the first n elements of μ and $0 \leq i \leq n$. We say that an eventuality $\alpha U\beta \in h_i$ is unfulfilled iff for all $j \leq i \leq n$ the formula $\beta \notin h_j$.

For the lowest i such that there exists an unfulfilled eventuality in h_i we fulfil this eventuality as follows:

Case 1: If no such i exists, we choose (c_{n+1}, h_{n+1}) such that $(c_n, c_{n+1}) \in \rightarrowtail$ and $(h_n, h_{n+1}) \in r_X$.

Case 2: If the eventuality is of the form $\alpha U \beta$, then there must exist $\alpha U \beta \in h_n$. Due to the pruning rule that removes unfulfilled eventualities, for some j there must exist a sequence of instances

$$(c_n, h_n, \alpha U \beta), (c_{n+1}, h_{n+1}, \alpha U \beta), \ldots, (c_j, h_j, \alpha U \beta)$$

such that the final instance is directly fulfilled ($\beta \in h_j$), and each other instance is fulfilled by the next instance in the chain. Having now chosen μ up to (c_j, h_j) with $\beta \in h_j$, the eventuality $\alpha U \beta \in h_i$ is now fulfilled.

Lemma 3.18 *For all α in $\mathbf{cl}\phi$, for all threads $\mu = \langle (c_0, h_0), (c_1, h_1), \ldots \rangle$ justifying $\sigma = \langle c_0, c_1, \ldots \rangle$ we have*

$$M, \sigma \vDash \alpha \text{ if } \alpha \in h_0$$

We can prove this using induction. Let L_ψ be the statement: for all threads $\mu = \langle (c_0, h_0), (c_1, h_1), \ldots \rangle$ justifying $\sigma = \langle c_0, c_1, \ldots \rangle$ we have $(W, \rightarrowtail, g, B), \sigma \vDash \psi$ iff $\psi \in h_0$. First note that L_ψ holds by definition when ψ is an atom. Assume that L_ψ holds for all ψ in $\mathbf{cl}\phi$ where $|\psi| \leq n$. We can show [4] that L_ψ holds for any ψ in $\mathbf{cl}\phi$ where $|\psi| \leq n+1$. For example, say that ψ is of the form $\alpha \wedge \beta$. Then since the hue is PPC, we see that α and β are also in h_0. Then $M, \sigma \vDash \alpha$ and $M, \sigma \vDash \beta$. Hence $M, \sigma \vDash \alpha \wedge \beta$.

Soundness follows from Lemmas 3.17 and 3.18.

Completeness

Lemma 3.19 *The tableau will halt, in an amount of time at worst doubly exponential in the length of the input formula.*

The closure set is linear in the length of the input formula. We see that the number of hues is singly exponential in the size of the closure set, the number of colours are exponential in the number of hues. We only apply rules to the rule nodes, and there is at most one rule node per colour. Each step is at worst polynomial in the number of colours, or singly exponential in the number of hues. Thus BCTL*-RTAB will halt in an amount of time at worst doubly exponential in the length of the formula.

Definition 3.20 For a colour $C = \{h_0, h_1, \ldots, h_n\}$ in the tableau we define $f(C)$ as follows:

$$f(C) = \left(\bigwedge C_e \right) \wedge \left(A(\bigvee_i (\bigwedge h_i)) \wedge \bigwedge_i E \left(\bigwedge h_i \right) \right).$$

Lemma 3.21 *BCTL*-RTAB is complete, that is, if ϕ is satisfiable in BCTL* then BCTL*-RTAB halts and succeeds on ϕ.*

The proof [4] involves showing that a node with colour C will never be pruned if $f(C)$ is satisfiable.

4 Performance

We have constructed a sample implementation of this tableau. This implementation stops constructing a node once it has constructed enough descendants of the node to know whether the node will be pruned.

We present a list of results in Tables 1 and 2. Table 1 is based upon a class of formulae proposed by [3] for testing asymptotic performance; in this table $\alpha_1 = AFGq$, $\beta_1 = AFAGq$ and for each $i \geq 1$ we have $\alpha_{i+1} = AFG\alpha_i$ and $\beta_{i+1} = AFAG\beta_i$. Table 2 gives standard examples used in [7] and [8]. In each column, the first subcolumn is for the our new BCTL* tableau, the second subcolumn is for the old BCTL* tableau of [7], and the third is for the hybrid technique of [3]. If the technique completes successfully "Sat" will be "Y" (or "N") to indicate that the technique reported that the formula was (not) satisfiable; "m" indicates that the procedure exceeded 1GB and failed, "h" indicates that a hardcoded limit in the implementation was exceeded, and "t" indicates that we simply terminated the procedure after an hour. The "CPU Time" column is the number of seconds used by the implementation on a computer with 2620M Core i7 processor and 8GB of ram. The number of colours in the new tableau is the number of unique colours used in rule nodes (note that there tend to be many more rule nodes than temporal nodes, and every temporal node is the direct descendant of a rule node). For the implementation of [3], we let the "Colours" be the number of states in the parity game. We do not define an equivalent to hues in [3], so there is no third hues column. A "+" in any column indicates that the number may have been larger if we had not terminated the implementation before it finished.

Our new rooted BCTL* tableau performs better than the original. In Table 2 we see that every example formula used by Reynolds can be handled by our tableau, while many cannot be handled by the original tableau (both using default settings). In each table we see the new tableau can handle every formula that can be handled by the original, also our new tableau is often much faster and never significantly slower.

The comparison with [3] is preliminary. Note that the [3] technique is for CTL*, while out technique is for BCTL* and so comparing the running times of the implementations may be misleading. If the running time of [3] is marked with a "*" the [3] technique has proven that the formula is satisfiable in CTL* (and hence BCTL*) as well. If the running time is marked with an "L" the formula is an LTL formula, that is it does not contain any path quantifiers (A/E) and so it is satisfiable in CTL* iff it is satisfiable in BCTL*. If it is marked with an "N" then, as we have shown that it is not satisfiable in BCTL*, and hence not satisfiable in CTL* either, we have proven a stronger result. Given that a [8] style variant of this tableau would require duplicating colours, we consider BCTL* easier than CTL*, and so we consider it reasonable to compare running times marked with an "N". The implementation of [3] requires a parity game solver, but does not select one by default. We have chosen the "recursive" parity game solver. In our benchmarks we found that the implementation of [3] spent less than one tenth of the time in the parity game solver, so there is little point in investigating other solvers for these benchmarks. For con-

sistency, we limited the heap size of the pure tableaux to 1GB; we were not able to do so for the implementation provided by [3]; however, we used a 32bit architecture so we doubt that their implementation could have used more than one order of magnitude more memory. In our benchmarks [3] performed significantly worse than in the benchmarks provided by [3]. We believe this is because we used a 32bit architecture.

Our intuition is that our technique will usually be able to prove that a formula is satisfiable much faster than that of [3], but that it would be harder to find a short formula that [3] cannot reason about. Our pure tableau technique can quit as soon as it has found a model of the input formula. This provides our technique with a significant advantage as a satisfiable formula can be satisfied by many different models. By contrast the [3] technique always completes a parity game. However, once the parity game is completed, many well researched parity game solving techniques can be used, and so there is reason to believe that it would be hard to find a relatively short theorem that could not be proven by [3]. While it is too early to rigorously compare these techniques, our benchmarks are at least compatible with this intuition.

There are many possible optimisations.

(i) The implementation of [7] first computed the hues that were consistent with the semantics of LTL. We could similarly require that colours contain only LTL consistent hues. For benchmarks with this optimisation see [4].

(ii) if $\{a, h, b\} \subseteq C$ and $a \subseteq h \subseteq b$ then we can simplify C by removing h from C. To see that this is safe, note that this does not affect $f(C)$.

(iii) The implementation blindly adds eDescendants trying to fulfil eventualities, and then covers them. If would be faster if it only covered the eDescendants on the branch which actually fulfilled the eventuality.

(iv) Like [7], we use $\neg\beta$ and $\alpha U\beta$ to indicate that the temporal successor must contain $\alpha U\beta$. This means that β and its subformulae occur both positively and negatively, potentially effectively doubling the size of the closure set. It is possible that using $X(\alpha U\beta)$ would be more efficient.

(v) X-branches have an exponential number of children. We could change X-branches so that they have only a single child, by including all hues, but marking the hues that do not descend from C_e as optional, we change H-branches so they only chose non-optional hues. When a contradiction is discovered in an optional hue we remove the hue from the colour rather than prune the colour. When an optional hue contains a formula ψ of the form $A\beta$, $\neg A\beta$, p or $\neg p$ that does not exist in some other hue of C then instead of always adding ψ to all the other hues we add a branch along which we remove the optional hue instead. There are a number of reasons to believe this would be more efficient: we might come across a contradiction before we need to branch; the number of branches is at most equal to the number of formulae ψ of the form $A\beta$, $\neg A\beta$, p or $\neg p$; it may be the case that all hues already have ψ in which case we can avoid the branch entirely.

Figure 1. Example Output: Proof that $Xp \to Fp$

>/.	Enter interior node/Enter unsatisfiable leaf
N/Y	Exit node (Not Satisfiable)/ Exit node (Satisfiable)
L	Leaf
U-	B-branch for formula of form $\neg(\alpha U \beta)$
X	Temporal Successor Branch, also: next time operator X
H	Choose Hue Branch (move selected hue to h0)
C:	Colour of node
P:	Parents of node
D:	Children of node
S:	Set of eventualities
o/c/e	Original Node/Node created to cover parent/to resolve eventuality
-/&/|/>	Negation ¬ / And (∧) / or (∨) / Implication (→)

Table 1
Asymptotic Benchmarks

Formula	Sat			CPU Time			Colours			Hues	
$q \to q$	Y	Y	Y	0.10	0.11	0.10L	3	2	8	5	2
$\beta_1 \to AFGq$	Y	Y	Y	0.11	0.34	0.11*	13	76	258	16	28
$\beta_2 \to \alpha_2$	Y	m	Y	0.10	108.50+	0.58*	31	67,096,162	10,730	38	508
$\beta_3 \to \alpha_3$	Y		Y	0.12	?.?	113.44*	72	?	636,517	87	?
$\beta_4 \to \alpha_4$	Y		m	0.12	?.?	492.72+*	169	?	?	201	?
$\beta_5 \to \alpha_5$	Y			0.14	?.?	?.?*	398	?	?	467	?
$\beta_6 \to \alpha_6$	Y			0.17	?.?	?.?*	931	?	?	1,080	?
$\beta_7 \to \alpha_7$	Y			0.23	?.?	?.?*	2,152	?	?	2,474	?
$\beta_8 \to \alpha_8$	Y			0.38	?.?	?.?*	4,909	?	?	5,604	?
$\beta_9 \to \alpha_9$	Y			0.80	?.?	?.?*	11,058	?	?	12,553	?
$\beta_{10} \to \alpha_{10}$	Y			1.95	?.?	?.?*	24,631	?	?	27,835	?
$\beta_{11} \to \alpha_{11}$	Y			5.56	?.?	?.?*	54,332	?	?	61,173	?
$\beta_{12} \to \alpha_{12}$	Y			17.90	?.?	?.?*	118,849	?	?	133,402	?
$\beta_{13} \to \alpha_{13}$	Y			70.39	?.?	?.?*	258,118	?	?	288,972	?
$\beta_{14} \to \alpha_{14}$	Y			262.81	?.?	?.?*	557,131	?	?	622,342	?
$\beta_{15} \to \alpha_{15}$	m			2,962.13+	?.?	?.?*	855,000+	?	?	956,321+	?
$\neg(q \to q)$	N	N	N	0.12	0.11	0.11L	1	2	7	2	2
$\neg(AFGq \to \beta_1)$	Y	Y	Y	0.10	0.35	0.13*	42	76	650	22	28
$\neg(\alpha_2 \to \beta_2)$	Y	m	Y	0.16	109.00+	5.44*	622	67,096,162	73,812	95	508
$\neg(\alpha_3 \to \beta_3)$	Y		m	2.30	?.?	489.97+*	21,361	?	?	751	?
$\neg(\alpha_4 \to \beta_4)$	m			3,600.20+	?.?	?.?*	1,345,060+	?	?	6,515+	?
$\neg(q \to q)$	N	N	N	0.12	0.11	0.11L	1	2	7	2	2
$\neg(\beta_1 \to AFGq)$	N	N	N	0.13	0.40	0.12N	89	76	241	43	28
$\neg(\beta_2 \to \alpha_2)$	N	m	N	713.46	109.32+	0.33N	163,068	67,096,162	6,003	1,747	508
$\neg(\beta_3 \to \alpha_3)$	t		N	3,600.08+	?.?	16.83*	128,604+	?	163,256	1,982+	?
$\neg(\beta_4 \to \alpha_4)$			m	?.?	?.?	478.40+*	?	?	?	?	?

(a) If $\{a, h\} \subseteq C$ and $a \subseteq h$ and h is optional, we can simplify C by removing h. (Similar to (1) above).

(b) if "$\neg A$" does not occur anywhere in a colour we can simplify C by removing all optional hues.

(c) Instead of always constructing the optional hues, we could only try to construct them only when they are required because we have come across a formula of the form $\neg A\alpha$.

Table 2
Benchmarks

Formula	Sat			CPU Time			Colours			Hues	
$G(p \to q) \to (Gp \to Gq)$	Y	Y	Y	0.10	2.70	0.11L	4	276	335	7	16
$\neg(G(p \to q) \to (Gp \to Gq))$	N	N	N	0.09	2.71	0.11L	4	276	79	6	16
$Gp \to (p \wedge Xp \wedge XGp)$	Y	Y	Y	0.09	0.27	0.39L	4	270	404	6	12
$\neg(Gp \to (p \wedge Xp \wedge XGp))$	N	N	N	0.09	0.26	0.10L	31	270	33	8	12
$(pUq) \leftrightarrow (q \vee (p \wedge X(pUq)))$	Y	Y	Y	0.11	0.10	0.14L	4	24	1,483	7	10
$\neg((pUq) \leftrightarrow (q \vee (p \wedge X(pUq))))$	N	N	N	0.09	0.10	0.10L	11	24	182	7	10
$(pUq) \to Fq$	Y	Y	Y	0.09	0.10	0.10L	4	20	166	7	8
$\neg((pUq) \to Fq)$	N	N	N	0.09	0.15	0.36L	15	20	17	8	8
$p \to AEp$	Y	Y	Y	0.09	0.09	0.10*	3	2	13	5	5
$\neg(p \to AEp)$	N	N	N	0.09	0.09	0.11N	2	2	11	4	5
$Ap \to AAp$	Y	Y	Y	0.09	0.09	0.15*	4	2	14	6	4
$\neg(Ap \to AAp)$	N	N	N	0.09	0.09	0.10N	2	2	13	4	4
$AXp \to XAp$	Y	Y	Y	0.09	0.10	0.10*	5	30	15	6	18
$\neg(AXp \to XAp)$	N	N	N	0.09	0.10	0.10N	7	30	12	6	18
$p \to Ap$	Y	Y	Y	0.09	0.16	0.10*	3	2	10	5	3
$\neg(p \to Ap)$	N	N	N	0.09	0.08	0.10N	2	2	9	4	3
$E(pU(E(pUq))) \to E(pUq)$	Y	Y	Y	0.09	0.10	0.10*	4	8	77	7	10
$\neg(E(pU(E(pUq))) \to E(pUq))$	N	N	N	0.10	0.09	0.10N	32	8	53	18	10
$(AG(p \to (qUr)) \wedge (qUp)) \to (qUr)$	Y	Y	Y	0.20	3.88	0.20*	7	290	4,476	14	35
$\neg((AG(p \to (qUr)) \wedge (qUp)) \to (qUr))$	N	N	N	0.12	3.15	0.10N	25	290	301	17	35
$G(EFp \to XFEFp) \to (EFp \to GFEFp)$	Y	Y	Y	0.10	137.68	12.60*	9	8,190	159,144	11	30
$\neg(G(EFp \to XFEFp) \to (EFp \to GFEFp))$	N	N	N	0.13	137.25	1.49N	107	8,190	17,867	35	30
$AG(p \to EXp) \to (p \to EGp)$	Y	Y	Y	0.09	0.35	0.10*	6	258	31	11	39
$\neg(AG(p \to EXp) \to (p \to EGp))$	Y	Y	N	0.09	0.36	0.28*	11	258	38	15	39
$AG(Ep \to EX((Eq)U(Ep))) \to (Ep \to EG((Eq)U(Ep)))$	Y	Y	Y	0.09	63.29	0.10*	8	4,596	154	13	234
$\neg(AG(Ep \to EX((Eq)U(Ep))) \to (Ep \to EG((Eq)U(Ep))))$	Y	Y	N	0.13	62.93	0.21*	167	4,596	3,484	52	234
$(AG(p \to EXr) \wedge AG(r \to EXp)) \to (p \to EG(Fp \wedge Fr))$	Y	h	Y	0.10	6.07+	0.12*	7	?	537	12	1,506
$\neg((AG(p \to EXr) \wedge AG(r \to EXp)) \to (p \to EG(Fp \wedge Fr)))$	Y	h	N	206.04	6.16+	34.75*	17,554	?	303,671	314	1,506
p	Y	Y	Y	0.09	0.09	0.10L	2	2	5	3	2
$\neg(p)$	Y	Y	Y	0.09	0.09	0.10L	2	2	5	3	2
$p \wedge Xp \wedge F\neg p$	Y	Y	Y	0.09	0.10	0.10L	5	18	37	7	6
$\neg(p \wedge Xp \wedge F\neg p)$	Y	Y	Y	0.09	0.10	0.10L	4	18	45	6	6
$AG((p \wedge X\neg p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge Xp \wedge q \wedge \neg r) \vee (\neg p \wedge Xp \wedge \neg q \wedge r)) \wedge E(Fq \wedge Fr$))	Y	m	Y	0.11	53.93+	0.16*	91	16,851,480	2,629	54	164
$\neg(AG((p \wedge X\neg p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge Xp \wedge q \wedge \neg r) \vee \ldots$	Y	m	Y	0.09	53.97+	0.11*	12	16,851,480	407	14	164
$AG(EXp \wedge EX\neg p) \wedge AG(Gp \vee ((\neg r)U(r \wedge \neg p)))$	Y	t	Y	0.12	3,600.08+	0.38*	72	?	7,454	47	357
$\neg(AG(EXp \wedge EX\neg p) \wedge AG(Gp \vee ((\neg r)U(r \wedge \neg p))))$	Y	t	Y	0.09	3,600.08+	0.10*	7	?	151	10	357

5 Conclusion

We have presented a rooted tableau for BCTL*. This was expected to have much better performance than the original tableau in [7], and the benchmarks included in this paper show that this is indeed the case. We have tried to keep this tableau otherwise similar to the original; as such it is more "fair" to compare other rooted tableaux to this tableau than the original. It is often hard to directly compare the benchmarks of this tableau with that of [3], as this paper uses BCTL* while [3] uses CTL*, but we suspect that each technique will outperform the other in some cases. To compare the techniques more conclusively we will investigate the effect of different optimisations, develop a similarly efficient CTL* tableau, and compare the techniques using randomly generated formulae.

The pure tableau techniques have the advantage that they work by creating a finite tableau, with each step being based upon sub-formulae of the formulae provided by the user. This output is likely to be more intelligible to the user. For an example of the output tableau see Figure 1.

Proving that a formula is valid in BCTL* has the advantage that it also demonstrates that the formula is valid in CTL*. The semantics of CTL* may not be applicable in all cases. Never-the-less we considered efficient rooted tableau of BCTL* primarily as a step towards finding efficient CTL* tableaux.

References

[1] Emerson, E. A. and A. P. Sistla, *Deciding branching time logic: A triple exponential decision procedure for CTL**, in: E. M. Clarke and D. Kozen, editors, *Logic of Programs*, Lecture Notes in Computer Science **164** (1983), pp. 176–192.

[2] Emerson, E. A. and A. P. Sistla, *Deciding branching time logic*, in: *Proceedings of the 16th annual ACM symposium on Theory on computing (STOC)* (1984), pp. 14–24.

[3] Friedmann, O., M. Latte and M. Lange, *A decision procedure for CTL* based on tableaux and automata*, in: J. Giesl and R. Hähnle, editors, *5th International Joint Conference on Automated Reasoning (IJCAR)*, LNCS **6173**, Springer, 2010 pp. 331–345.
URL http://dx.doi.org/10.1007/978-3-642-14203-1_28

[4] M^{c}Cabe-Dansted, J. C., *A rooted tableau for BCTL** (2011).
URL http://www.csse.uwa.edu.au/~john/papers/Rooted_BCTL_Tableau.pdf

[5] M^{c}Cabe-Dansted, J. C., "A Temporal Logic of Robustness," Ph.D. thesis, The University of Western Australia (2011).
URL http://dansted.co.cc/papers/Thesis_RoCTL.pdf

[6] Reynolds, M., *An axiomatization of full computation tree logic*, The Journal of Symbolic Logic **66** (2001), pp. 1011–1057.
URL http://www.jstor.org/stable/2695091

[7] Reynolds, M., *A Tableau for Bundled CTL**, J Logic Computation **17** (2007), pp. 117–132.
URL http://logcom.oxfordjournals.org/cgi/content/abstract/17/1/117

[8] Reynolds, M., *A tableau for CTL**, in: A. Cavalcanti and D. Dams, editors, *Proceedings of the 16th International Symposium on Formal Methods (FM)*, Lecture Notes in Computer Science **5850** (2009), pp. 403–418.

[9] Vardi, M. Y. and L. Stockmeyer, *Improved upper and lower bounds for modal logics of programs*, in: *Proceedings of the 17th annual ACM symposium on Theory of computing (STOC)* (1985), pp. 240–251.