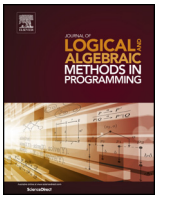
Journal of Logical and Algebraic Methods in Programming 130 (2023) 100828

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



Tableaux and sequent calculi for CTL and ECTL: Satisfiability test with certifying proofs and models

Alex Abuin [a,*], Alexander Bolotov [b,1], Montserrat Hermo [c,1], Paqui Lucio [c,1]

[a] *Ikerlan Technology Research Centre (BRTA), Arrasate-Mondragón, Gipuzkoa, Spain*
[b] *University of Westminster, W1W 6UW, London, UK*
[c] *Computer Languages and Systems, University of the Basque Country, San Sebastián, Spain*

ARTICLE INFO

Article history:
Received 18 January 2022
Received in revised form 10 October 2022
Accepted 15 October 2022
Available online 21 October 2022

Keywords:
Temporal logic
Fairness
Branching-time
Certified model checking

ABSTRACT

Certifying proofs are automated deductive proofs obtained as outcomes of a formal verification of temporal properties, where model checking is one of the most prominent approaches. The satisfiability problem for the Computation Tree Logic (CTL) cannot be reduced to the CTL model checking problem. Hence model checking algorithms for CTL cannot be adapted for testing CTL satisfiability. However, any decision procedure of CTL satisfiability can perform model checking tasks. Our context-based tableau approach to CTL satisfiability introduces a tree-style one-pass tableau that does not require auxiliary constructions or extra-logical rules for branch pruning. As a consequence this method brings the classical duality between tableaux and sequent calculi in temporal logic. For any input formula, a closed tableau represents a formal sequent proof that certifies the unsatisfiability of the input, whereas an open tableau provides at least a model certifying the satisfiability of the input formula. Hence, in this framework the satisfiability test can be performed and complemented with certifying proofs and models. This is also true in relation to more expressive branching-time logic, Extended CTL (ECTL), which enriches CTL with simple fairness formulae. This paper continues the development of dual systems of tableau method and sequent calculi, introducing these techniques for CTL and ECTL. We prove the soundness and completeness of both methods and define algorithms for obtaining systematic tableaux which produce models and formal proofs (as certificates) depending on whether the input formulae are satisfiable or not. We also describe the implementation of this technique and provide experimental results.

1. Introduction

Temporal logic is nowadays essential for the specification and verification of concurrent and reactive systems. There are two types of temporal logics in relation to the treatment of the future of a given moment of time. When each moment of time has a unique possible future we have timelines as linear sequences of these moments (or states). Thus linear temporal logic extends classical propositional logic by future time temporal operators such as $\bigcirc$ - 'at the next moment of time', $\Diamond$ - 'eventually in the future', $\Box$ - 'always in the future' $\mathscr{U}$ - 'from now until', and $\mathscr{R}$ - 'releases'. When each moment of time

* Corresponding author.
E-mail address: aabuin@ikerlan.es (A. Abuin).

[1] These authors have been supported by the European Union (ERDF funds) under the grant PID2020-112581GB-C22, European COST Action CA20111 EuroProofNet (European Research Network on Formal Proofs), and also by the University of the Basque Country under the LoRea GIU21/044 Project.

https://doi.org/10.1016/j.jlamp.2022.100828
2352-2208/

is allowed to have several possible futures (or paths) we have branching temporal logics. The language of branching-time logics extends the language of linear temporal logic with paths quantifiers A - 'for all paths' quantifier, and E - 'there exists a path' quantifier. Thus, in linear temporal logics, the underlying models of time are discrete, linear sequences of states, finite in the past and infinite in the future. For branching temporal logics the underlying models of time are trees (or computation trees) where each branch is a discrete, linear sequence of states, finite in the past and infinite in the future. Recall that the formulation of temporal logic with the future-time temporal operators only is based on the remarkable Gabbay's *separation result* [17,18].

The method of semantic tableaux, invented in the 1950's by Beth and Hintikka and later perfected by Smullyan [35] and Fitting [16], is nowadays well established in the field of Automated Deduction. It brings together the proof-theoretical and the semantic approaches to the presentation of a logical system as a set of inference rules. A tableau is a tree-like structure designed to make semantic reasoning fully systematic to decide whether a set of formulae is satisfiable or not. The construction of a tableau is guided by a set of rules that decompose formulae according to the semantics of its logical operators. Temporal tableaux, first introduced in [36], tackling the problem that formulae must be analysed in a infinite sequences of states, introduce a mechanism which controls repeated appearances of formulae and identify periodic situations in finite time. Later, we explain the two options to cope with this problem. For the survey of the tableau method for temporal logic we refer an interested reader to [24].

The hierarchy of CTL-type family of Branching-time logics (BTL) is defined by releasing restrictions on the concatenations of temporal operators and paths quantifiers which define classes of admissible state formulae distinguished for these logics. As in Computation Tree Logic (CTL) [10] every temporal operator must be preceded by a path quantifier, this logic cannot express fairness which requires at least the concatenation of $\Box$ and $\Diamond$. These are tackled by Extended Computation Tree Logic (ECTL) [14] which enables simple fairness constraints but not their Boolean combinations. ECTL$^+$ [15] further extends the expressiveness of ECTL allowing Boolean combinations of temporal operators and ECTL fairness constraints (but not permitting their nesting). The logic ECTL$^\#$ [7] extends ECTL$^+$ by allowing the combinations $\Box(A\mathscr{U}B)$ or $A\mathscr{U}\Box B$, referred to as modalities $\Box\mathscr{U}$ and $\mathscr{U}\Box$. The logic CTL*, often considered as the full branching-time logic' overcomes all these restrictions on syntax allowing any arbitrary combinations of temporal operators and path quantifiers.

Model checking (MC) [5,12] is one of the most developed and applied approaches to formal verification of temporal properties. In this paper we continue our investigation of deductive techniques for temporal logic targeting their potential application in MC. Our particular interest is in so called *certified model checking (CMC)* [30]. This type of model checking aims to

- generate proofs as certificates of the properties that are verified,
- produce counterexamples for those properties that are invalidated.

Certification of proofs is the process of obtaining automated deductive proofs as outcomes of a formal verification process. Traditional model checkers integrate some deductive methods for generating proofs, however, typically for invariant properties only. This idea has been recently extended to the certification of general linear-time properties [25]. It is known that for linear temporal logic both satisfiability and MC are PSPACE-complete [34] and can be reduced to each other. However, the MC problem for CTL is known to be P-complete [11], while the satisfiability problem for CTL is EXPTIME-complete [14]. As a consequence, an MC algorithm for CTL properties (for example implemented in NuSMV [9]) cannot be adapted for testing CTL satisfiability. However, any decision procedure of CTL satisfiability can be used to perform MC tasks and if based on a deductive system, such procedure enables certifying proofs. More precisely, if $\varphi_{\mathscr{S}}$ is the formula that characterises the computation tree of a system $\mathscr{S}$,[2] then the verification problem $\mathscr{S} \models \psi$ for some property ψ reduces to the (un)satisfiability problem of the formula $\varphi_{\mathscr{S}} \wedge \neg\psi$, where a model of the formula $\varphi_{\mathscr{S}} \wedge \neg\psi$ is a counterexample of the former verification problem. At the same time the proof of the unsatisfiability of $\varphi_{\mathscr{S}} \wedge \neg\psi$ (i.e. its refutation proof) certifies that $\mathscr{S} \models \psi$.

There are two known ways to build tableau constructions for temporal logic formulae. Two-pass constructions check the validity of the given tableaux input in two passes - in the first pass a tableau graph is obtained and the second 'pass' checks the fulfillment of all eventualities. The one-pass tableau method [33] checks the fulfillment of eventualities on-the-fly along the construction on the branches using an extra-logical mechanism. This mechanism prevents the formulation of a sequent calculus dual to the tableau method. More recently, [19,7,3] introduced the context-based tableau method as a one-pass method that does not require any extra-logical auxiliary construction and therefore keeps the mentioned classical duality. Therefore it enables not only to certify satisfiability results by means of a model of the input set, but also the construction of certifying sequent proofs of the unsatisfiability of the input. The core context-based tableau construction is based on the concept of *a context of an eventuality*, which is a set of formulae that 'accompanies' the eventuality in the label of a node of a tableaux graph. Our specific tableau rules that involve context force the earliest fulfilment of eventualities.

This paper continues the development of (one-pass) context-based tableau method for temporal logics [19,7,2,3]. We aim at the formulation of a 'generic' approach to CMC for a variety of branching-time logics based on context-based tableaux method and dual sequent calculi. Here the crucial aspect of the developments of deductive techniques is that the same reasoning mechanism applies for both tasks - providing certificates and generating counterexamples. In previous works

[2] In the sense that the set of Kripke models of $\varphi_{\mathscr{S}}$ exactly represents the set of all possible computations of the system $\mathscr{S}$ (see e.g. [8]).

this context-based tableaux approach has been developed for propositional linear-time temporal logic, PLTL [19], and for the branching-time logic ECTL$^{\#}$ [7]. The latter formalism specifically allows to reason about a richer than ECTL^{+} class of branching-time fairness constraints utilising the 'until' temporal operator. It has also been shown how, in the linear-time case, the method, being mingled with a SAT solver, can be invoked as part of the certified model checking for PLTL [2]. For the branching-time case, the developed context-based tableaux for ECTL$^{\#}$ has quite complex rules. We also note that the distinguished (and unavoidable) feature of context-based technique for ECTL$^{\#}$ is the utilisation of two types of contexts: the so-called 'outer' (similar to PLTL) context which is a collection of state formulae, is complemented by so called 'inner' context, a collection of path formulae. The context-based tableau methods for CTL and ECTL, similarly to the one for PLTL, only need the "outer" context, yet, similarly to ECTL$^{\#}$, the generated tableaux are AND-OR trees. Subsequently, the application of a model checking procedure for CTL simply based on the existing technique for ECTL$^{\#}$ would become far too 'non-intuitive'.

Hence, the development of a simpler context-based method for CTL and ECTL is an important task. This paper extends our earlier work [3] where context-based tableaux methods for CTL and ECTL were introduced. Our results provide a more intuitive tableau algorithm for CTL and ECTL, along with the dual sequent calculi and the extension of the tableau algorithm that constructs both the proof certificates and also the countermodels. We prove soundness and completeness of the method, and illustrate its practical implementation that produces certificates and counterexamples. All these bring us one step closer to the formulation of a 'generic' approach to certified model-checking applicable to a variety of branching-time logics based on context-based tableaux method and dual sequence calculi –the same reasoning mechanism applies for both tasks– providing certificates and generating counterexamples.

A useful comparison of five satisfiability procedures for CTL can be found in [23] (two of these procedures are tableau-based). The first method [6] is a two-pass tableau which, in the first pass creates a cyclic graph. In this graph, a 'bad loop' is a loop containing some eventuality that is not fulfilled along it. Therefore, in the second pass, 'bad loops' are pruned. The second method [1,22] is a single-pass tableaux decision procedure based on Schwendimann's one-pass procedure for PLTL [33]. This tableau method uses an additional mechanism for collecting information on the set of formulae in the nodes and passing it to subsequent nodes along branches. The information on previously generated nodes helps detecting 'bad loops' without constructing the whole graph.

To evaluate the feasibility of our approach we have implemented a prototype, called *MomoCTL*, of our tableau method for CTL that returns sequent proofs for unsatisfiable inputs, as well as models for satisfiable ones. *MomoCTL* has been tested against the benchmarks used in [23].[3] We also compare our results with another, unique, one-pass tableau for CTL which was introduced in [1] and, in turn, compared in [23] with other methods. Finally, we note that, to the best of our knowledge, there has been no explicit formulation of a tableau (one or two pass) method for ECTL.

Outline of the paper. In §2 we define the syntax and the semantics of CTL and ECTL as sublogics of CTL*. The formulation of the tableau method is presented in §3, where we first give some preliminaries and then overview the tableau construction as an AND-OR tree and provide examples. A *systematic* tableau construction and illustrative examples are presented in §4. Soundness and completeness of the tableau method for CTL are proved in §5. In §6 we introduce the dual sequent calculus for CTL. In §7, we describe our implementation that tests the satisfiability of a set of CTL formulae, and returns a formal proof (refutation certificate) for an unsatisfiable input and, otherwise, a model. In §8 we introduce further extension of the method for the logic ECTL. In §9 we draw the conclusions and prospects of future work that the presented results open.

2. Syntax and semantics of CTL and ECTL as sublogics of CTL*

As all logics we are interested in are subsumed by CTL*, for the sake of generality, we first present the CTL* syntax and then, by restricting it, derive the syntax for each of ECTL$^{\#}$, ECTL^{+}, ECTL and CTL. In §2.2.1 we introduce the syntax and semantics of CTL*. In §2.2.2 we discuss a hierarchy of sublogics of CTL*. In §2.2.3 and §2.2.4, we respectively define the grammars and the sets of modalities for CTL and ECTL formulae in negation normal form.

2.1. The logic CTL*

Definition 2.1 *(Syntax of* CTL**).* Given Prop is a fixed set of propositions, and $p \in$ Prop, we define sets of state (σ) and path (π) CTL* formulae over Prop as follows:

$$\begin{array}{ll} \sigma & ::= \mathbf{T} \mid p \mid \neg\sigma \mid \sigma_1 \wedge \sigma_2 \mid \mathsf{E}\pi \\ \pi_{\mathsf{CTL}^{\star}} & ::= \sigma \mid \neg\pi \mid \pi_1 \wedge \pi_2 \mid \bigcirc\pi \mid \pi_1 \mathscr{U} \pi_2 \end{array} \quad \blacksquare$$

Two observations are needed here. Definition 2.1 above introduces the minimal CTL* grammar: we only use one path quantifier - E, and two temporal modalities - $\bigcirc$ and $\mathscr{U}$. From this combination we can derive a richer syntax, which is often more appropriate to use when we speak about the intuitive interpretation of formal specification of targeted systems: the

[3] Available in http://users.cecs.anu.edu.au/~rpg/CTLComparisonBenchmarks/.

other path quantifier - A and the remaining temporal operators ($\Diamond$, $\Box$ and $\mathscr{R}$). In particular, the 'falsehood' constant $\mathbf{F} \equiv \neg\mathbf{T}$ and the classical disjunction operator is defined as follows: $\varphi_1 \vee \varphi_2 \equiv_{def} \neg(\neg\varphi_1 \wedge \neg\varphi_2)$ (here and below we will use the notation $\varphi_1 \equiv_{def} \varphi_2$ to abbreviate that the structure φ_1 is defined as φ_2). The 'for all paths' quantifier $\mathsf{A}\varphi \equiv_{def} \neg\mathsf{E}\neg\varphi$ and the remaining temporal operators are defined as follows: $\Diamond\varphi \equiv_{def} \mathbf{T}\mathscr{U}\varphi$, $\Box\varphi \equiv_{def} \neg\Diamond\neg\varphi$, and $\varphi_1\mathscr{R}\varphi_2 \equiv_{def} \neg(\neg\varphi_1\mathscr{U}\neg\varphi_2)$.

Second, observe that in Definition 2.1 for the set of path formulae, $\pi_{\mathsf{CTL}^\star}$, we deliberately used an index $\mathsf{CTL}^\star$ to indicate that this grammar introduces a set of path formulae specifically for $\mathsf{CTL}^\star$. At the same time we did not use any index for the set of state formulae. This reflects the tradition in defining the grammar for BTL logics in a way that the grammar for the path formulae determines relevant changes in the grammar for the set of state formulae. For $\mathsf{CTL}^\star$, 'no restrictions' on the construction of path formulae determine 'no restriction' on the construction of the arguments of the 'path' quantifiers. For each of $\mathsf{CTL}^\star$ sublogics that we will define later, CTL and ECTL, their specific restrictions on the construction of path formulae will determine relevant classes of their state formulae (however, the grammar scheme to generate state formulae remains as presented here for $\mathsf{CTL}^\star$ formulae).

For interpreting $\mathsf{CTL}^\star$ formulae, we invoke Kripke structures that are labelled directed graphs corresponding to Emerson's R-generable structures, i.e. the transition relation R is suffix, fusion and limit closed [13].

Definition 2.2 *(Labelled Kripke structure).* A Kripke structure, $\mathscr{K}$, is given by a quadruple (S, R, I, L) where $S \neq \emptyset$ is a set of *states*, $R \subseteq S \times S$ is a total binary relation, called *the transition relation*, I is a finite non-empty set of *initial states*, and $L : S \to 2^{\mathsf{Prop}}$ is a *labelling function*. ■

A *path* x through a Kripke structure $\mathscr{K}$ is an infinite sequence of states $s_i, s_{i+1}, s_{i+2} \ldots$ $(i \geq 0)$ such that $(s_j, s_{j+1}) \in R$ for any $j \geq i$. Note that the totality of the transition relation R causes the infinity of paths. It is possible that a path in a Kripke structure, from some point onward, contains a repetitive sequence of states - a cycle. Later, in Definition 2.7, we introduce the notion of a cyclic Kripke structure that we will essentially use in this paper. A *fullpath* x through a Kripke structure $\mathscr{K}$ is an infinite sequence of states $s_0, s_1, s_2 \ldots$, where $s_0 \in I$. By $\mathsf{fullpaths}(\mathscr{K})$ we denote the set of all fullpaths in $\mathscr{K}$. Given a fullpath $x \in \mathsf{fullpaths}(\mathscr{K})$ such that $x = s_0, s_1 \ldots$, the state s_i $(0 \leq i)$ is denoted by $x(i)$.

Given a path $x = s_i, s_{i+1}, \ldots$ and $k \geq 0$, we denote by $x^{<k}$ a finite prefix of x of length k and by $x^{\geq k}$ we denote its infinite suffix starting at state s_{i+k}. Hence $x^{<k} = s_i, s_{i+1} \ldots, s_{i+k-1}$ and $x^{\geq k} = s_{i+k}, s_{i+k+1}, \ldots$. Note that, $x = x^{<k}, x^{\geq k}$ (which means that this path x starts with prefix $x^{<k}$ to the k-th state and then, from this k-th state, continues with the suffix $x^{\geq k}$).

Given a Kripke structure $\mathscr{K} = (S, R, I, L)$ and a state $s \in S$, let $\mathscr{K}' = (S', R, \{s\}, L)$ be a Kripke structure obtained from $\mathscr{K}$ by restricting S to S' such that S' is the set of all states of S that are R-reachable from s, we will denote $\mathscr{K}'$ by $\mathscr{K} \restriction s$. Note that if $I \neq \{s\}$ then $\mathscr{K} \restriction s$ is a proper substructure of $\mathscr{K}$ with the unique initial state s. Intuitively, $\mathscr{K} \restriction s$ represents the behaviour of a system from state s 'forward'. The set I helps modelling systems whose initial state is not definitely determined, for each $s \in I$, $\mathscr{K} \restriction s$ represents the behaviour of the system for the initial state s.

For a given $x \in \mathsf{fullpaths}(\mathscr{K})$ such that $x = s_0, s_1 \ldots$, and given $i \geq 0$, $\mathscr{K} \restriction x(i)$ is the Kripke structure that allows us to express path's *fusion closure*: if $y \in \mathsf{fullpaths}(\mathscr{K} \restriction x(i))$ then $x^{\leq i-1}, y \in \mathsf{fullpaths}(\mathscr{K})$.

For the convenience of the subsequent presentation (as we will be presenting the context-based tableaux and the dual sequent calculi for the sublogics of $\mathsf{CTL}^\star$), in Definition 2.3 below we introduce the evaluation of $\mathsf{CTL}^\star$ state and path formulae constructed with the extended grammar - with both classical $\mathbf{F}$ and $\vee$, with both path quantifiers, and the full set of temporal operators - $\bigcirc$, $\Diamond$, $\Box$, $\mathscr{U}$ and $\mathscr{R}$. In our formulation of the $\mathsf{CTL}^\star$ semantics below we will also label the conditions related to the evaluation of state $\mathsf{CTL}^\star$ formulae by 's' followed by the reference to the relevant constraint (for example, '$(s_\neg)$' labels the condition evaluating a state formula $\neg\sigma$ in some state). Similarly, we label the conditions related to the evaluation of path $\mathsf{CTL}^\star$ formulae by 'p' followed by the reference to the relevant constraint (for example, '$(p_\vee)$' labels the condition evaluating a path formula $\pi_1 \vee \pi_2$ along some path). Recall that any $\mathsf{CTL}^\star$ state formula is also a path formula and that any rule (p_σ) applies when the path formula is really an state formula.

Definition 2.3 *(Models).* Given the structure $\mathscr{K} = (S, R, I, L)$, the relation $\models$, which evaluates path formulae in a given path x and state formulae at the state index i of the given path x, is inductively defined as follows.

$(s_{\mathbf{T}})$	$\mathscr{K}, x, i \models \mathbf{T}$		
$(s_{\mathbf{F}})$	$\mathscr{K}, x, i \not\models \mathbf{F}$		
(s_{prop})	$\mathscr{K}, x, i \models p$	iff	$p \in L(x(i))$.
$(s_\neg)$	$\mathscr{K}, x, i \models \neg\sigma$	iff	$\mathscr{K}, x, i \models \sigma$ does not hold.
$(s_\vee)$	$\mathscr{K}, x, i \models \sigma_1 \vee \sigma_2$	iff	$\mathscr{K}, x, i \models \sigma_1$ or $\mathscr{K}, x, i \models \sigma_2$.
$(s_\wedge)$	$\mathscr{K}, x, i \models \sigma_1 \wedge \sigma_2$	iff	$\mathscr{K}, x, i \models \sigma_1$ and $\mathscr{K}, x, i \models \sigma_2$.
(s_E)	$\mathscr{K}, x, i \models \mathsf{E}\pi$	iff	there exists $y \in \mathsf{fullpaths}(\mathscr{K} \restriction x(i))$ such that $\mathscr{K}, y \models \pi$.
(s_A)	$\mathscr{K}, x, i \models \mathsf{A}\pi$	iff	$\mathscr{K}, y \models \pi$ holds for all $y \in \mathsf{fullpaths}(\mathscr{K} \restriction x(i))$.
(p_σ)	$\mathscr{K}, x \models \sigma$	iff	$\mathscr{K}, x, 0 \models \sigma$.
$(p_\neg)$	$\mathscr{K}, x \models \neg\pi$	iff	$\mathscr{K}, x \models \pi$ does not hold.
$(p_\vee)$	$\mathscr{K}, x \models \pi_1 \vee \pi_2$	iff	$\mathscr{K}, x \models \pi_1$ or $\mathscr{K}, x \models \pi_2$.

$(p_\wedge)$ $\mathscr{K}, x \models \pi_1 \wedge \pi_2$ iff $\mathscr{K}, x \models \pi_1$ and $\mathscr{K}, x \models \pi_2$.
$(p_\bigcirc)$ $\mathscr{K}, x \models \bigcirc \pi$ iff $\mathscr{K}, x^{\geq 1} \models \pi$.
$(p_\Diamond)$ $\mathscr{K}, x \models \Diamond \pi$ iff there exists $j \geq 0$ such that $\mathscr{K}, x^{\geq j} \models \pi$.
$(p_\mathscr{U})$ $\mathscr{K}, x \models \pi_1 \mathscr{U} \pi_2$ iff there exists $k \geq 0$ such that $\mathscr{K}, x^{\geq k} \models \pi_2$ and $\mathscr{K}, x^{\geq j} \models \pi_1$ for all $0 \leq j < k$.
$(p_\Box)$ $\mathscr{K}, x \models \Box \pi$ iff $\mathscr{K}, x^{\geq j} \models \pi$ holds for all $j \geq 0$.
$(p_\mathscr{R})$ $\mathscr{K}, x \models \pi_1 \mathscr{R} \pi_2$ iff either $\mathscr{K}, x^{\geq k} \models \pi_2$ holds for all $k \geq 0$, or there exists some $k \geq 0$ such that $\mathscr{K}, x^{\geq k} \models \pi_1 \wedge \pi_2$ and $\mathscr{K}, x^{\geq j} \models \pi_2$ for all $0 \leq j \leq k$.

Given a Kripke structure $\mathscr{K} = (S, R, I, L)$. For a state formula φ, we say that $\mathscr{K} \models \varphi$ (in words, $\mathscr{K}$ models φ) if and only if $\mathscr{K} \restriction s_0 \models \varphi$ for all $s_0 \in I$. For a set of state formulae Σ, $\mathscr{K} \models \Sigma$ if and only if $\mathscr{K} \models \varphi$ holds for every $\varphi \in \Sigma$. $\mathsf{Mod}(\varphi)$ (resp. $\mathsf{Mod}(\Sigma)$) denotes the set of all models of φ (resp. Σ). ■

To illustrate the semantics, we will consider the following CTL* formula

$$\mathsf{A}\Diamond(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p) \tag{1}$$

To show that this formula does not have a model, let us reason by contradiction assuming that there exists a model, say $\mathscr{K}$, for this formula. We will show that this assumption leads us to a contradiction when trying to build such a model. To start with, we pick a state s_0 from the set of the initial states of $\mathscr{K}$ and assume that formula (1) is true at $\mathscr{K} \restriction s_0$. Note that $\mathsf{fullpaths}(\mathscr{K} \restriction s_0) \subseteq \mathsf{fullpaths}(\mathscr{K})$. This means that for any fullpath, $x \in \mathsf{fullpaths}(\mathscr{K})$, $\mathscr{K}, x \models \Diamond(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p)$. Now pick a path from $\mathsf{fullpaths}(\mathscr{K})$, say x_1. Among the states of x_1, let $x_1(i)$ $(i \geq 0)$ be the first state satisfying the condition $\mathscr{K}, x_1^{\geq i} \models \bigcirc p \wedge \mathsf{E}\bigcirc\neg p$. This state must exist following the CTL* semantics. Therefore, both $\bigcirc p$ and $\mathsf{E}\bigcirc\neg p$ are satisfied along $x_1^{\geq i}$. This means that p itself is satisfied at the state $x_1(i+1)$, i.e. at the successor of $x_1(i)$ on the path x_1. Since $\mathscr{K}, x_1^{\geq i} \models \mathsf{E}\bigcirc\neg p$ there should be a path starting at state $x_1(i)$ such that $\bigcirc\neg p$ is satisfied along this path. As $\neg p$ can not be satisfied at the state $x_1(i+1) \in x_1^{\geq i}$ where p has been already satisfied, this new path, call it x_2, to satisfy $\bigcirc\neg p$ should differ from $x_1^{\geq i}$. Now we invoke the fusion closure property which we have already discussed in this section (after Definition 2.2). Due to the fusion closure property there is a fullpath in $\mathscr{K}$ with the prefix $x_1^{<i}$ and the suffix x_2, namely $y_1 = x_1^{<i}, x_2$. Since in the given formula (1), $\Diamond(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p)$, is in the scope of the A path quantifier, any fullpath, hence, also y_1, must satisfy $\Diamond(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p)$. Hence, we must have a state $x_2(j)$ such that $j > i$ and $\mathscr{K}, y_1^{\geq j} \models \bigcirc p \wedge \mathsf{E}\bigcirc\neg p$. Considering this state $x_2(j)$, we invoke the same reasoning as we applied to the state $x_1(i)$ evaluating $\mathsf{E}(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p)$ on the path x_1. This leads us to the analogous conclusion, that there must be another path x_3 starting at $x_2(j)$ such that $\bigcirc\neg p$ is true along it. Again, the path $y_2 = x_1^{<i}, x_2^{<j}, x_3$ should satisfy the property $\mathsf{E}(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p)$, hence we must have a state $x_3(k)$ such that $k > j$ and $\mathscr{K}, y_1^{\geq k} \models \bigcirc p \wedge \mathsf{E}\bigcirc\neg p$. Therefore (due to limit closure), there exists $y \in \mathsf{fullpaths}(\mathscr{K})$ formed by the finite prefixes $x_1^{<i}, x_2^{<j}, x_3^{<k} \ldots$, such that $\mathscr{K}, y \not\models \Diamond(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p)$. So, our assumption on the satisfiability of (1) was wrong. Note that limit closure of the underlying Kripke structures was important for the above proof. Without the limit closure, for example, for so called bundled structures [21], the situation would have been different as we would not be able to assemble this new path y.

The following Definition 2.4 introduces notions of satisfiability, validity and equivalence for CTL* formulae and generalises satisfiability and validity for the sets of CTL* formulae. These are based on the concept of *Mod* introduced in Definition 2.3.

Definition 2.4 *(CTL* satisfiability, validity and logical equivalence).*

- A state formula φ is *satisfiable* (denoted $\mathsf{Sat}(\varphi)$) whenever $\mathsf{Mod}(\varphi) \neq \emptyset$, otherwise φ is *unsatisfiable* (denoted $\mathsf{UnSat}(\varphi)$).
- A state formula φ is *valid* whenever $\mathscr{K} \models \varphi$ for all $\mathscr{K}$.
- A set of state formulae Σ is *satisfiable* (denoted $\mathsf{Sat}(\Sigma)$) if $\mathsf{Mod}(\Sigma) \neq \emptyset$. Otherwise Σ is *unsatisfiable* (denoted $\mathsf{UnSat}(\Sigma)$).
- A set of state formulae Σ is *valid* whenever $\mathscr{K} \models \Sigma$ for all $\mathscr{K}$.
- State formulae φ and φ' are *logically equivalent* if $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\varphi')$ (denoted as $\varphi \equiv \varphi'$). ■

2.2. Sublogics of CTL*

For each of the BTL logics - ECTL$^\#$, ECTL$^+$, ECTL and CTL- which are sublogics of CTL*, we define its syntax over a fixed set of propositions Prop, preserving the definition of state formulae from CTL* (Definition 2.1) and formulating in Definition 2.5 the specific restrictions for these logics on the CTL* grammar that generate corresponding sets for path formulae. Note that similarly to the CTL* grammar, we also utilise here the minimal set of operators, for the sake of consistency of the presentation and its rigour.

Table 1
Classification of context-based tableaux systems for CTL-type logics and relevant difficult cases of concatenations of temporal operators and path quantifiers.

BTL Logic	$\mathsf{E}\Box\Diamond q$	$\mathsf{E}(\Box\Diamond q \wedge \Diamond\Box\neg q)$	$\mathsf{A}((p\,\mathscr{U}\,\Box q) \vee (s\,\mathscr{U}\,\Box\neg r))$	$\mathsf{A}\Diamond(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p)$	Dual T & SC
$\mathscr{B}(\mathscr{U},\bigcirc)$ (CTL)	X	X	X	X	This paper
$\mathscr{B}(\mathscr{U},\bigcirc,\Box\Diamond)$ (ECTL)	$\surd$	X	X	X	This paper
$\mathscr{B}^{+}(\mathscr{U},\bigcirc,\Box\Diamond)$ (ECTL^{+})	$\surd$	$\surd$	X	X	$\surd$
$\mathscr{B}^{+}(\mathscr{U},\bigcirc,\mathscr{U}\Box)$ ($\mathsf{ECTL}^{\#}$)	$\surd$	$\surd$	$\surd$	X	$\surd$
$\mathscr{B}^{\star}(\mathscr{U},\bigcirc)$ ($\mathsf{CTL}^{\star}$)	$\surd$	$\surd$	$\surd$	$\surd$	X

Definition 2.5 *(Path formulae for* $\mathsf{ECTL}^{\#}$, ECTL^{+}, *ECTL and CTL).*

logic		*inherited*	*characteristic*
$\pi_{\mathsf{ECTL}^{\#}}$	$::=$	$\pi_{\mathsf{ECTL}^{+}}$	$\mid \sigma_1\,\mathscr{U}\,(\sigma_2 \wedge (\mathbf{T}\,\mathscr{U}\,\sigma_3)) \mid$ $\neg(\mathbf{T}\,\mathscr{U}\,\neg(\mathbf{T}\,\mathscr{U}\,(\neg\sigma_1 \wedge (\mathbf{T}\,\mathscr{U}\,\neg\sigma_2)))) \mid$ $\sigma_1\,\mathscr{U}\,\neg(\mathbf{T}\,\mathscr{U}\,\neg\sigma_2) \mid \neg(\mathbf{T}\,\mathscr{U}\,\neg(\sigma_1\,\mathscr{U}\,\sigma_2))$
$\pi_{\mathsf{ECTL}^{+}}$	$::=$	π_{ECTL}	$\mid \pi_1 \wedge \pi_2$
π_{ECTL}	$::=$	π_{CTL}	$\mid \neg(\mathbf{T}\,\mathscr{U}\,\neg(\mathbf{T}\,\mathscr{U}\,\neg\sigma)) \mid \mathbf{T}\,\mathscr{U}\,\neg(\mathbf{T}\,\mathscr{U}\,\neg\sigma)$
π_{CTL}	$::=$		$\sigma \mid \neg\pi \mid \bigcirc\sigma \mid \sigma_1\,\mathscr{U}\,\sigma_2$. ■

Our aim is to highlight different kinds of path formulae that are generated in each sublogic of $\mathsf{CTL}^{\star}$ and those characteristic to it. For example, for ECTL, the characteristic path formulae are those expressing linear-time fairness - $\neg(\mathbf{T}\,\mathscr{U}\,\neg(\mathbf{T}\,\mathscr{U}\,\neg\sigma))$ and $\mathbf{T}\,\mathscr{U}\,\neg(\mathbf{T}\,\mathscr{U}\,\neg\sigma)$. Now, if, similar to the case of $\mathsf{CTL}^{\star}$, we extend this minimal grammar by the derivable constraints, these fairness constraints are read as $\Box\Diamond\sigma$ and $\Diamond\Box\sigma$. The characteristic path formulae for ECTL^{+} are $\pi_1 \wedge \pi_2$ - those that allow Boolean combination of linear-time temporal operators and fairness constraints. Finally, the characteristic path formulae for $\mathsf{ECTL}^{\#}$ are $\sigma_1\,\mathscr{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Box(\sigma_1 \vee \Box\sigma_2), \sigma_1\,\mathscr{U}\,(\Box\sigma_2)$ and $\Box(\sigma_1\,\mathscr{U}\,\sigma_2)$.

It is important to note that the nesting of 'pure path formulae', totally unrestricted in $\mathsf{CTL}^{\star}$, is restricted in its sublogics by relevant grammar cases for path formulae. Below, for each of the logics $\mathsf{CTL}^{\star}$, $\mathsf{ECTL}^{\#}$, ECTL^{+}, ECTL and CTL, we will provide an example of a formula which is expressible in this logic but is not expressible in its sublogic. The structures of these formulae reflect the characteristic path formulae for the considered sublogic. Hence, we will refer to these formulae as characteristic formulae for a dedicated logic under consideration. For example, an indicative $\mathsf{CTL}^{\star}$ formula $\mathsf{A}\Diamond(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p)$ mentioned above (1) is not an $\mathsf{ECTL}^{\#}$ formula. Rewriting it as $\mathsf{A}(\mathbf{T}\,\mathscr{U}\,(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p))$ we can see that $\bigcirc p \wedge \mathsf{E}\bigcirc\neg p$, the right-hand side argument of the $\mathscr{U}$ operator, does not meet the $\mathsf{ECTL}^{\#}$ criteria: it is neither a state formula of the form $\sigma_1 \wedge \Diamond\sigma_2$ nor $\Box\sigma$. Recall that the validity of (1) is directly linked to the limit closure property [13]. If we consider an $\mathsf{ECTL}^{\#}$ formula

$$\mathsf{A}((p\,\mathscr{U}\,\Box q) \wedge (s\,\mathscr{U}\,\Box\neg q)) \tag{2}$$

we can see that this is not an ECTL^{+} formula because ECTL^{+} only allows Boolean combinations of the fairness constraints. In this $\mathsf{ECTL}^{\#}$ formula, $p\,\mathscr{U}\,\Box q$ and $s\,\mathscr{U}\,\Box\neg q$, hence their conjunction, are not admissible ECTL^{+} formulae. Further, an ECTL^{+} formula (3) that does not belong to ECTL is

$$\mathsf{E}(\Box\Diamond q \wedge \Diamond\Box\neg q) \tag{3}$$

as $\Box\Diamond q \wedge \Diamond\Box\neg q$ is not an admissible ECTL path formula.

Finally, the fairness constraint (4) which is expressible in ECTL cannot be constructed in CTL syntax as every temporal operator

$$\mathsf{E}\Box\Diamond q \tag{4}$$

in a CTL formula must be preceded by a path quantifier. Obviously, writing, for example, an E quantifier before the $\Diamond q$ we obtain an admissible CTL structure $\mathsf{E}\Box\mathsf{E}\Diamond q$. However, comparing $\mathsf{E}\Box\mathsf{E}\Diamond q$ with $\mathsf{E}\Box\Diamond q$, we can see that the latter requires a model where there exists a path, along which $\Box\Diamond q$ is satisfied, while the former requires a model where there exists a path where each state gives rise for a path along which $\Diamond q$ is satisfied.

Note that it is important to distinguish the problem if a formula of a superlogic belongs to a sublogic and the problem if a formula of a superlogic can be expressed in a sublogic. For example, $\mathsf{E}(\Box\Diamond q \vee \Diamond\Box\neg q)$, similarly to formula (3) does not belong to ECTL but unlike (3), it is expressible in this logic, as $\mathsf{E}(\Box\Diamond q \vee \Diamond\Box\neg q) \equiv \mathsf{E}\Box\Diamond q \vee \mathsf{E}\Diamond\Box\neg q$ which is an ECTL formula if we define $\vee$ via $\wedge$.

In Table 1, following the notation initially proposed in [13] and further tuned in [29], we represent BTL logics (listed in the first column) classified by their expressiveness using '$\mathscr{B}$' for 'Branching', followed by the set of only allowed modalities as parameters; $\mathscr{B}^{+}$ indicates admissible Boolean combinations of the modalities and $\mathscr{B}^{\star}$ reflects 'no restrictions' in either

concatenations of the modalities or Boolean combinations between them. Columns 2-5 of Table 1 illustrate the indicative formulae for the logics under considerations as follows:

- for ECTL, column 2: $\mathsf{E}\Box\Diamond q$, formula (4)
- for ECTL$^+$, column 3: $\mathsf{E}(\Box\Diamond q \wedge \Diamond\Box\neg q)$, formula (3)
- for ECTL$^\#$, column 4: $\mathsf{A}((p\,\mathscr{U}\,\Box q) \wedge (s\,\mathscr{U}\,\Box\neg q))$, formula (2)
- for CTL*, column 5: $\mathsf{A}\Diamond(\bigcirc p \wedge \mathsf{E}\bigcirc\neg p)$, formula (1)

Writing the '$\surd$' in Table 1 against the listed logics we indicate if a logic meets these grammar rules. For example, column 2 now illustrates which of the logics can express the property $\mathsf{E}\Box\Diamond q$: while this property is not expressible in CTL, it becomes expressible in ECTL and any of its extensions. In this respect, ECTL has those minimal grammar requirements enabling to express the property, hence, we can treat $\mathsf{E}\Box\Diamond q$ as ECTL indicative formula.

The last column in this table reflects the development of the dual system of context-based tableaux (T) and sequent calculus (SC) for CTL-type logics. The method has been developed for ECTL$^\#$ [7] where the motivation was to cope with more complex cases of fairness. We note that this method developed for ECTL$^\#$ is obviously applicable to all weaker logics. However, it only tackles one weaker logic - ECTL$^+$- efficiently, introducing unnecessary complications for other ECTL$^\#$ sublogics. This is based upon the fact that ECTL$^+$ and ECTL$^\#$ have similar cases of the Boolean combinations of eventualities in the scope of A and E: disjunctions of the eventualities in the scope of the A quantifier and conjunctions of eventualities in the scope of the E quantifier, see [7] for details. Thus, Table 1 also reflects syntactical cases of concatenations of temporal operators and path quantifiers that are difficult for context-based tableaux.

To manage these cases, in addition to α- and β-rules, that are standard to the tableaux, we introduce novel β^+-rules which use the context to force the eventualities to be fulfilled as soon as possible. As ECTL$^\#$ is more expressive than ECTL$^+$ in allowing new type of fairness constraints that use the $\mathscr{U}$ operator, the relevant rules introduced in [7] cover all difficult concatenations of operators in ECTL$^+$. However, simply treating the case of context-based tableaux for the other two sublogics of ECTL$^\#$–ECTL and CTL– as solved by the relevant development for a richer logic ECTL$^\#$, would introduce extra unnecessary complexity in the construction of dual system of relevant tableaux and sequent calculi. Hence, for both ECTL and CTL, simpler context-based tableaux methods are required. In this paper we concentrate on bridging this gap in our roadmap in supplying BTL logics by this technique, and develop the method for CTL and ECTL.

Subsequently, we proceed by defining the CTL and ECTL semantic conditions in Definition 2.6. These are derived from the CTL* semantic evaluation rules given in Definition 2.3.

Definition 2.6 *(CTL and ECTL semantics).*

- The semantics for logic CTL is obtained from the CTL* semantics given in Definition 2.3 by preserving the evaluation conditions (s_T) - (s_A) and $(p_\bigcirc)$ - $(p_\mathscr{R})$ and setting π, π_1 and π_2 to be a state formula.
- The semantics for logic ECTL is obtained from the above CTL* semantics by deleting the evaluation conditions $(p_\vee)$ and $(p_\wedge)$ and also restricting π in the following rules as follows:
  - in the $(p_\bigcirc)$ rule π is a state formula,
  - in the $(p_\Diamond)$ rule π is a state formula or a formula $\Box\sigma$,
  - in the $(p_\Box)$ rule π is a state formula or a formula $\Diamond\sigma$, and
  - in the $(p_\mathscr{U})$ and $(p_\mathscr{R})$ rules π_1, π_2 are state formulae. ■

For technical convenience, we will use the fact that cyclic Kripke structures have the ability to characterise satisfiability in branching temporal logics.

Definition 2.7 *(Cyclic sequence, path and Kripke structure).* Let $\mathscr{K} = (S, R, I, L)$ be a given Kripke structure. Let z be a finite sequence of states $z = s_0, s_1, \ldots, s_j$ in S such that $(s_k, s_{k+1}) \in R$ for every $0 \leq k < j$. Then

- z is a *cyclic sequence* if and only if there exists s_i, $0 \leq i \leq j$ such that $(s_j, s_i) \in R$. In this case, the subsequence $s_i, \ldots, s_j$ of z is called a *loop* denoted as $\langle s_i, \ldots, s_j\rangle^\omega$.
- If z is a cyclic sequence, then the path

$$s_0, s_1, \ldots, s_{i-1}\langle s_i, s_{i+1}, \ldots, s_j\rangle^\omega$$

is denoted by path(z) and is called cyclic.

A Kripke structure $\mathscr{K}$ is *cyclic* if every fullpath is a cyclic path over a cyclic sequence of states. ■

The fact that branching-time satisfiability can be reduced to the interpretation over cyclic models only, is derived from the existence of the finite model property [14], see also [26]. In particular, for any CTL (ECTL) formula φ, such that

$\mathsf{Mod}(\varphi) \neq \emptyset$, there always exists a model $\mathscr{K} \in \mathsf{Mod}(\varphi)$ such that $\mathscr{K}$ is cyclic. Therefore, when speaking about the satisfiability in CTL (hence ECTL) we can consider *cyclic* Kripke structures. Cyclic paths are also known as *ultimately periodic* paths.

2.3. Negation normal form grammars for CTL and ECTL

In this section, we introduce the grammars for CTL and ECTL formulae in *negation normal form* (shortly, nnf). Along the rest of this paper we only deal with formulae in this normal form.

From now on, we abbreviate by Q either of the path quantifiers A or E.

Definition 2.8 *(Syntax of CTL and ECTL in* nnf*).* Let Prop be a fixed set of propositions, then the sets $\mathscr{F}^{\mathsf{CTL}}_{\mathsf{nnf}}$ of CTL formulae and $\mathscr{F}^{\mathsf{ECTL}}_{\mathsf{nnf}}$ of ECTL formulae in nnf (over Prop) are given by the grammar (where the elements of *Lit* are called *literals*):

$$
\begin{array}{ll}
\mathit{Lit} & ::= \mathbf{F} \mid \mathbf{T} \mid p \mid \neg p \text{ where } p \in \mathsf{Prop} \\
\sigma_{\mathsf{CTL}} & ::= \mathit{Lit} \mid \sigma_1 \wedge \sigma_2 \mid \sigma_1 \vee \sigma_2 \mid \mathsf{Q}\bigcirc\sigma \mid \mathsf{Q}\Diamond\sigma \mid \mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2) \mid \mathsf{Q}\Box\sigma \mid \mathsf{Q}(\sigma_1 \mathscr{R} \sigma_2) \\
\sigma_{\mathsf{ECTL}} & ::= \sigma_{\mathsf{CTL}} \mid \mathsf{Q}\Box\Diamond\sigma \mid \mathsf{Q}\Diamond\Box\sigma \quad \blacksquare
\end{array}
$$

Proposition 2.9 *(Closure under negation). For any $\varphi \in \mathscr{F}^{CTL}_{\mathsf{nnf}}$, we also have* $\mathsf{nnf}(\neg\varphi) \in \mathscr{F}^{CTL}_{\mathsf{nnf}}$*. For any $\varphi \in \mathscr{F}^{ECTL}_{\mathsf{nnf}}$, we also have* $\mathsf{nnf}(\neg\varphi) \in \mathscr{F}^{ECTL}_{\mathsf{nnf}}$*.*

Proof. By structural induction on the formulae, using the following well-known equivalences (e.g. [13]):

$$
\begin{array}{lll}
\neg\mathbf{T} \equiv \mathbf{F} & \neg\mathsf{A}\Box\varphi \equiv \mathsf{E}\Diamond\neg\varphi & \neg\mathsf{A}(\varphi\mathscr{R}\psi) \equiv \mathsf{E}(\neg\varphi\mathscr{U}\neg\psi) \\
\neg\mathbf{F} \equiv \mathbf{T} & \neg\mathsf{E}\Box\varphi \equiv \mathsf{A}\Diamond\neg\varphi & \neg\mathsf{E}(\varphi\mathscr{R}\psi) \equiv \mathsf{A}(\neg\varphi\mathscr{U}\neg\psi) \\
\neg\neg\varphi \equiv \varphi & \neg\mathsf{A}\Diamond\varphi \equiv \mathsf{E}\Box\neg\varphi & \neg\mathsf{A}\Box\Diamond\sigma \equiv \mathsf{E}\Diamond\Box\neg\sigma \\
\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi & \neg\mathsf{E}\Diamond\varphi \equiv \mathsf{A}\Box\neg\varphi & \neg\mathsf{E}\Box\Diamond\sigma \equiv \mathsf{A}\Diamond\Box\neg\sigma \\
\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi & \neg\mathsf{A}(\varphi\mathscr{U}\psi) \equiv \mathsf{E}(\neg\varphi\mathscr{R}\neg\psi) & \neg\mathsf{A}\Diamond\Box\sigma \equiv \mathsf{E}\Box\Diamond\neg\sigma \\
\neg\mathsf{A}\bigcirc\varphi \equiv \mathsf{E}\bigcirc\neg\varphi & \neg\mathsf{E}(\varphi\mathscr{U}\psi) \equiv \mathsf{A}(\neg\varphi\mathscr{R}\neg\psi) & \neg\mathsf{E}\Diamond\Box\sigma \equiv \mathsf{A}\Box\Diamond\neg\sigma \\
\neg\mathsf{E}\bigcirc\varphi \equiv \mathsf{A}\bigcirc\neg\varphi \quad \blacksquare &&
\end{array}
$$

The following result (e.g. [28]) is easily established using the above equivalences and the semantics of the constraints used in nnf.

Proposition 2.10 *(*nnf *preserves satisfiability and unsatisfiability). For any $\sigma \in \mathscr{F}^{CTL}_{\mathsf{nnf}}$ it holds that* $\mathsf{Mod}(\sigma) = \mathsf{Mod}(\mathsf{nnf}(\sigma))$*. Similarly, for any $\sigma \in \mathscr{F}^{ECTL}_{\mathsf{nnf}}$ it holds that* $\mathsf{Mod}(\sigma) = \mathsf{Mod}(\mathsf{nnf}(\sigma))$*.*

For simplicity, we will write $\sim\varphi$ instead of $\mathsf{nnf}(\neg\varphi)$. Also, for a finite set $\Phi = \{\varphi_1, \ldots, \varphi_n\}$, we let $\sim\Phi = \mathsf{nnf}(\neg\bigwedge_{i=1}^{n}\varphi_i)$.

For the formulation of our tableaux technique we will need a concept of a consistent (inconsistent) set of formulae, which is introduced in the following definition.

Definition 2.11 *(Syntactically consistent and inconsistent sets of formulae).* A set Σ of state formulae of CTL and ECTL in nnf is *(syntactically) inconsistent* (denoted $\mathsf{Incons}(\Sigma)$) if and only if $\mathbf{F} \in \Sigma$ or $\{\sigma, \sim\sigma\} \subseteq \Sigma$ for some σ. Otherwise, Σ is said to be *consistent*. $\blacksquare$

2.4. ECTL and CTL basic modalities

Next, we introduce a concept of a *basic modality* which reflects the restrictions on forming the basic admissible combinations of temporal operators in the scope of a path quantifier. We consider a basic modality of CTL or ECTL logic to be of the form QT, where T is a temporal operator. The structure QT is generated by the grammar rules for these logics in Definition 2.5. We can identify all basic modalities in a given formula φ by finding its most embedded modality(es), say M_1, then looking at the next basic modality in which M_1 is embedded, etc. For example, a basic modality for CTL is any admissible combination of Q and a temporal operator, i.e. $\mathsf{Q}\bigcirc$, $\mathsf{Q}\Diamond$, $\mathsf{Q}\mathscr{U}$, $\mathsf{Q}\Box$, and $\mathsf{Q}\mathscr{R}$, while ECTL basic modalities are those identified above for CTL and, additionally, the modalities that appear due to new admissible combinations $\Diamond\Box$ and $\Box\Diamond$ in the scope of a path quantifier –$\mathsf{Q}\Diamond\Box$ and $\mathsf{Q}\Box\Diamond$. If we analyse a CTL formula $\mathsf{E}\bigcirc\mathsf{A}\bigcirc p$ then the most embedded basic modality, M_1, would be $\mathsf{A}\bigcirc p$, which is embedded as $\mathsf{E}\bigcirc M_1$. These are generalised in Definition 2.12.

Definition 2.12 *(ECTL and CTL basic modalities).*

$$
\begin{array}{ll}
\mathsf{M}_{\mathsf{CTL}} & ::= c \mid \mathsf{Q}\bigcirc\mathsf{M} \mid \mathsf{Q}\Diamond\mathsf{M} \mid \mathsf{Q}(\mathsf{M}\mathscr{U}\mathsf{M}) \mid \mathsf{Q}\Box\mathsf{M} \mid \mathsf{Q}(\mathsf{M}\mathscr{R}\mathsf{M}). \\
\mathsf{M}_{\mathsf{ECTL}} & ::= \mathsf{M}_{\mathsf{CTL}} \mid \mathsf{Q}\Box\Diamond\mathsf{M} \mid \mathsf{Q}\Diamond\Box\mathsf{M}.
\end{array}
$$

where c stands for a purely classical formula (we can consider a purely classical formula as a zero-degree basic modality) and M stands for any basic modality of CTL in the definition of $\mathsf{M}_{\mathsf{CTL}}$ and of ECTL in the definition of $\mathsf{M}_{\mathsf{ECTL}}$. ■

In what follows, every CTL modality $\mathsf{Q}\mathscr{U}$ or $\mathsf{Q}\Diamond$ is called *eventuality* and $(\mathsf{Q}\bigcirc)^i$ stands for i consecutive occurrences of a basic modality $\mathsf{Q}\bigcirc$.

CTL tableau rules are based on fixpoint characterisation of its basic modalities: (in the equations below μ and ν stand for 'minimal fixpoint' and 'maximal fixpoint' operators, respectively)

$$\begin{array}{ll} \mathsf{E}\Box\varphi = \nu\rho(\varphi \wedge \mathsf{E}\bigcirc\rho) & \mathsf{E}(\varphi\mathscr{R}\psi) = \nu\rho(\psi \wedge (\varphi \vee \mathsf{E}\bigcirc\rho)) \\ \mathsf{A}\Box\varphi = \nu\rho(\varphi \wedge \mathsf{A}\bigcirc\rho) & \mathsf{A}(\varphi\mathscr{R}\psi) = \nu\rho(\psi \wedge (\varphi \vee \mathsf{A}\bigcirc\rho)) \\ \mathsf{E}\Diamond\varphi = \mu\rho(\varphi \vee \mathsf{E}\bigcirc\rho) & \mathsf{E}(\varphi\mathscr{U}\psi) = \mu\rho(\psi \vee (\varphi \wedge \mathsf{E}\bigcirc\rho)) \\ \mathsf{A}\Diamond\varphi = \mu\rho(\varphi \vee \mathsf{A}\bigcirc\rho) & \mathsf{A}(\varphi\mathscr{U}\psi) = \mu\rho(\psi \vee (\varphi \wedge \mathsf{A}\bigcirc\rho)) \end{array} \tag{5}$$

This fixpoint characterisation of basic CTL and ECTL modalities as maximal or minimal fixpoints give rise to their analytical classification as α- or β-formulae which are associated, in the tableau with α- and β-rules: $\mathsf{Q}\Box$, and $\mathsf{Q}\mathscr{R}$ as maximal fixpoints are classified as α-formulae while $\mathsf{Q}\Diamond$ and $\mathsf{Q}\mathscr{U}$ as minimal fixpoints are β-formulae. This is also reflected in the known equivalences:

$$\begin{array}{ll} \mathsf{E}\Box\varphi = \varphi \wedge \mathsf{E}\bigcirc\mathsf{E}\Box\varphi & \mathsf{E}(\varphi\mathscr{R}\psi) = \psi \wedge (\varphi \vee \mathsf{E}\bigcirc\mathsf{E}(\varphi\mathscr{R}\psi)) \\ \mathsf{A}\Box\varphi = \varphi \wedge \mathsf{A}\bigcirc A\Box\varphi & \mathsf{A}(\varphi\mathscr{R}\psi) = \psi \wedge (\varphi \vee \mathsf{A}\bigcirc\mathsf{A}(\varphi\mathscr{R}\psi)) \\ \mathsf{E}\Diamond\varphi = \varphi \vee \mathsf{E}\bigcirc\mathsf{E}\Diamond\varphi & \mathsf{E}(\varphi\mathscr{U}\psi) = \psi \vee (\varphi \wedge \mathsf{E}\bigcirc\mathsf{E}(\varphi\mathscr{U}\psi)) \\ \mathsf{A}\Diamond\varphi = \varphi \vee \mathsf{A}\bigcirc\mathsf{A}\Diamond\varphi & \mathsf{A}(\varphi\mathscr{U}\psi) = \psi \vee (\varphi \wedge \mathsf{A}\bigcirc\mathsf{A}(\varphi\mathscr{U}\psi)) \end{array} \tag{6}$$

3. Context-based tableau method for CTL

The tableau method determines whether a given set of CTL state formulae, Σ, is satisfiable or not. In addition, in the affirmative case a model (Kripke structure) of Σ can be generated from the tableau, whereas in the negative case we can generate a proof in the dual sequent calculus. In this section, we introduce the tableau method: we define the set of tableau rules and a general view and intuitions on how tableaux are constructed. In §4 we provide a precise algorithm for constructing tableaux in a systematic way. The generation of models and proofs will be explained in §7. Recall that we assume that every formula is in nnf.

We precede the formal introduction of the technique by its informal overview. The main concepts are informally introduced here and technically defined later in this section or in §4. A tableau for a set of CTL formulae is a graph, namely an AND-OR-Tree (for the account on AND-OR-Trees we refer an interested reader to [27]), where nodes are labelled by sets of CTL formulae. The initial node (or root) of the tableau is labelled by some given set of CTL formulae whose satisfiability we want to check. Non-terminal nodes are further expanded by applications of the tableau rules to their labelling sets. There are two kinds of *terminal nodes*. First, any node labelled by an inconsistent set of formulae (see Definition 2.11) is terminal. The second type are the so-called *loop-nodes*. Intuitively, we say that a branch (i.e. a path from the root to some node n) has a loop (or lasso) when the label of n (or some superset of it) has already appeared in this branch. In this case, n is called a loop-node. Loop-nodes are terminal whenever some precise eventuality fulfilment conditions hold in the branch. Intuitively, such conditions ensure that the (systematic) tableau for any satisfiable set of formulae represents a model of this set.

For the node expansion, we have the following types of tableau rules: α- and β-rules, the 'next-state' rule, which reflects a 'jump' from a 'state' to a 'pre-state', and, finally, characteristic to our approach, β^+-rules, where the use of the context (of an eventuality) is essential (recall that the context is a collection of state formulae accompanying the eventuality in the label of the node). The use of the context forces the soonest fulfillment of eventualities, preventing their useless delays. In our procedure, the application of β^+-rules to eventualities is essential to detect 'bad' loops. Only if β^+-rules have already been applied to every eventuality in the branch and there is no inconsistent node in this branch, then we can establish if all the eventualities have been fulfilled. When this check for fulfilment of eventualities is positive we have a 'good loop' and this branch would be part of a model of the input formula. Otherwise, when there is at least one unfulfilled eventuality, we choose one to which a corresponding β^+-rule has not been applied.

Next we proceed with formally defining the construction of the tableaux and introducing necessary concepts and tableau rules.

Definition 3.1 *(Tableau, consistent and incons. Node, closed and open branch).* A *tableau* for a set of CTL state formulae Σ is a labelled tree $\langle T, \tau, \Sigma\rangle$, where T is a tree, and τ is a mapping of the nodes of T to sets of state formulae, such that the following two conditions hold:

- The root is labelled by the set Σ.
- For any other node $m \in T$, its label $\tau(m)$ is a set of state formulae obtained as the result of the application of one of the rules in Figs. 1, 2 and 3 to its parent node n. When the applied rule is R, we term m an R-successor of n.

$$(\wedge)\ \frac{\Sigma, \sigma_1 \wedge \sigma_2}{\Sigma, \sigma_1, \sigma_2} \qquad (\mathsf{Q}\Box)\ \frac{\Sigma, \mathsf{Q}\Box\sigma}{\Sigma, \sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Box\sigma}$$

$$(\vee)\ \frac{\Sigma, \sigma_1 \vee \sigma_2}{\Sigma, \sigma_1 \mid \Sigma, \sigma_2} \qquad (\mathsf{Q}\mathscr{U})\ \frac{\Sigma, \mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)}{\Sigma, \sigma_2 \mid \Sigma, \sigma_1, \mathsf{Q}\bigcirc\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)}$$

$$(\mathsf{Q}\mathscr{R})\ \frac{\Sigma, \mathsf{Q}(\sigma_1 \mathscr{R} \sigma_2)}{\Sigma, \sigma_2, \sigma_1 \vee \mathsf{Q}\bigcirc\mathsf{Q}(\sigma_1 \mathscr{R} \sigma_2)} \qquad (\mathsf{Q}\Diamond)\ \frac{\Sigma, \mathsf{Q}\Diamond\sigma}{\Sigma, \sigma \mid \Sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Diamond\sigma}$$

Fig. 1. α- and β-rules.

Let $\Sigma, \mathcal{A}, \mathcal{E}$ be an elementary set of formulae where

- Σ is a set of literals,
- $\mathcal{A}$ is a possibly empty set of $\mathsf{A}\bigcirc$ formulae $\{\mathsf{A}\bigcirc\sigma_1, \ldots, \mathsf{A}\bigcirc\sigma_\ell\}$, and
- $\mathcal{E}$ is a non-empty set of $\mathsf{E}\bigcirc$ formulae $\{\mathsf{E}\bigcirc\sigma'_1, \ldots, \mathsf{E}\bigcirc\sigma'_k\}$.

$$(\bigcirc\mathsf{E})\ \frac{\Sigma, \mathcal{A}, \mathcal{E}}{\mathcal{A}^{\downarrow}, \sigma'_1\ \&\ \ldots\ \&\ \mathcal{A}^{\downarrow}, \sigma'_k} \qquad (\bigcirc\mathsf{A})\ \frac{\Sigma, \mathcal{A}}{\mathcal{A}^{\downarrow}}$$

where $\mathcal{A}^{\downarrow} = \{\sigma_1, \ldots, \sigma_\ell\}$. Hence, $\mathcal{A}^{\downarrow}$ is empty if and only if $\mathcal{A}$ is empty.

Fig. 2. Next-state rules ('&' joins AND-successors in the conclusion of ($\bigcirc$E)).

A node n of a tree T is *consistent* if its label, $\tau(n)$, is a (syntactically) consistent set of formulae (see Definition 2.11), else n is *inconsistent*. If a branch b of T, contains an inconsistent node, then b is *closed* else b is *open*. ■

Fig. 1 follows the standard for the tableaux classification of rules into α-rules and β-rules - this is based on the analytic classification of CTL modalities and reflects their interpretation as fixpoints (see equations (6)) of the formula (in the node label) that is 'designated' for the rule application. In the systematic tableau construction, at each node, the designated formula is chosen following some strategy that we specify later (in 4). An α- or β-rule is applied to a node labelled by a set of formulae Σ, φ, where φ is a designated formula that determines the rule (to be applied), and Σ is a possibly empty set of formulae that accompany φ in the label of the node. Then, if φ is an α-formula - $\wedge$, $\mathsf{Q}\Box$, or $\mathsf{Q}\mathscr{R}$ - then a corresponding α-rule applies, while if φ is a β-formula - $\vee$, $\mathsf{Q}\mathscr{U}$, or $\mathsf{Q}\Diamond$- then a corresponding β-rule applies. These applications of α- and β-rules generate the set of formulae in the conclusion of the rule as label(s) for the successor node(s): one successor in case of an α-rule, or two successors in case of a β-rule. In β-rules we use | to emphasise that successors are OR-siblings.

When a node n is labelled by an *elementary set of formulae* – i.e. a set which is exclusively formed by literals and formulae of the form $Q\bigcirc\sigma$ – then this structure is analogous to a 'state' in the terminology of [36]; it enables us to construct the successors of n corresponding to 'pre-states' [36].

According to the next proposition we are guaranteed to reach such a tree structure, where the last node of every branch, at this stage of the construction, is a state.

Proposition 3.2. *Any set of CTL state formulae has a tableau T such that the last node of every branch is labelled by an elementary set of state formulae.*

Proof. Repeatedly apply to every expandable node any α- or β-rule until all expandable nodes are labelled by elementary sets of formulae. Then, the appropriate next-state rule must be applied to every expandable node depending on the number of formulae starting by $\mathsf{E}\bigcirc$ appearing at the node. ■

Proposition 3.2 enables the application of the so-called 'next-state rules' depicted in Fig. 2. Applying rule ($\bigcirc$E) we split the current branch at node n where the set

$$\Sigma, \mathsf{A}\bigcirc\sigma_1, \ldots, \mathsf{A}\bigcirc\sigma_\ell, \mathsf{E}\bigcirc\sigma'_1, \ldots, \mathsf{E}\bigcirc\sigma'_k \tag{7}$$

is satisfied, $0 \leq l$ and $k \geq 1$, into k branches: the number of branches is equal to the number of $\mathsf{E}\bigcirc$ constraints, where the successors of n along these branches, say $m_1, \ldots, m_k$ are AND-successors of n. We label each AND-successor m_j $(1 \leq j \leq k)$ in the following way. As any constraint $\mathsf{A}\bigcirc\sigma_i$ $(1 \leq i \leq l)$ in the premise of the rule would propagate σ_i to any successor node, then each of these successor nodes, m_j should have the set $\sigma_1, \ldots, \sigma_l$ as part of its label. The next part of the label of m_j is coming from the corresponding $\mathsf{E}\bigcirc\sigma'_s$ $(1 \leq s \leq k)$ as this existential constraint only determines the label for one of the AND-successors. Thus, each AND-successor m_j of n has its label of the form $\sigma_1, \ldots, \sigma_l, \sigma'_s$. The rule ($\bigcirc$E) splits branches

$$(\mathsf{Q}\mathscr{U})^+\ \frac{\Sigma, \mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)}{\Sigma, \sigma_2 \ \mid\ \Sigma, \sigma_1, \mathsf{Q}\bigcirc\mathsf{Q}((\sigma_1 \wedge {\sim}\Sigma')\mathscr{U}\sigma_2)} \qquad (\mathsf{Q}\Diamond)^+\ \frac{\Sigma, \mathsf{Q}\Diamond\sigma}{\Sigma, \sigma \ \mid\ \Sigma, \mathsf{Q}\bigcirc\mathsf{Q}((\sim\Sigma')\mathscr{U}\sigma)}$$

where $\Sigma' = \Sigma \setminus \{(\mathsf{A}\bigcirc)^i \mathsf{A}\Box\sigma \in \Sigma \mid i \geq 0\}$

Fig. 3. β^+-Rules.

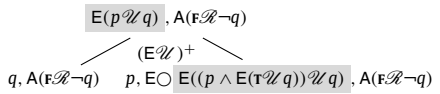


Fig. 4. Application of rule $(\mathsf{E}\mathscr{U})^+$.

in a 'conjunctive' way, and we use the symbol & to represent the generation of AND-successors of node n. Thus, the graphs generated with the application of the 'next-state' rule ($\bigcirc\mathsf{E}$) are indeed AND-OR trees. When $k = 0$ in the elementary set of formulae that labels node n (7), the rule ($\bigcirc\mathsf{A}$) is applied. The application of both rules ($\bigcirc\mathsf{E}$) and ($\bigcirc\mathsf{A}$) represents a 'jump' to the next state, hence the set of literals Σ in (7) disappears in every child produced by these rules.

Next we extend our set of tableau rules with the new two rules named as β^+-rules (Fig. 3). Note that the $(\mathsf{Q}\Diamond)^+$ rule can be derived from the application of the $(\mathsf{Q}\mathscr{U})^+$ to the CTL formula $\mathbf{T}\mathscr{U}\sigma$. These rules, similarly to β-rules, also split a branch into two branches. Moreover, whenever a β^+-rule -$(Q\,\mathscr{U})^+$ or $(Q\,\Diamond)^+$- is applicable, so is the respective β-rule -$(Q\,\mathscr{U})$ or $(Q\,\Diamond)$. Which rules, β-rules or β^+-rules are applied at each step of the tableau construction, does not affect the correctness. However, for completeness, some strategy is required. It is easy to see that the extension with β^+-rules preserves the property given in Proposition 3.2.

These β^+-rules are the only rules in our system that make use of the context -their application forces the eventualities to be satisfied as soon as possible (from the point of the tableau construction where an eventuality is selected to be expanded with a β^+-rule). The context is given by the possibly empty set Σ that accompanies the designated formula, which in this case is an eventuality. Inside one of formulae of the right-hand child of each β^+-rule we add a conjunct ${\sim}\Sigma'$ that is calculated by deleting some specific formulae from Σ (the context). In particular, if Σ is empty, then so is Σ' and the formula ${\sim}\Sigma'$ is the constant $\mathbf{F}$.

In what follows, any formula of the form $\mathsf{Q}((\sigma_1 \wedge {\sim}\Sigma')\mathscr{U}\sigma_2)$ (respectively, $\mathsf{Q}((\sim\Sigma')\mathscr{U}\sigma)$) is called *the contextualised variant* of $\mathsf{Q}(\sigma_1\mathscr{U}\sigma_2)$ (resp. $\mathsf{Q}\Diamond\sigma$) provided that $\mathsf{Q}\bigcirc\mathsf{Q}((\sigma_1 \wedge {\sim}\Sigma')\mathscr{U}\sigma_2)$ (respectively, $\mathsf{Q}\bigcirc\mathsf{Q}((\sim\Sigma')\mathscr{U}\sigma)$) has been obtained by the application of the corresponding β^+-rule to a formula $\mathsf{Q}(\sigma_1\mathscr{U}\sigma_2)$ (resp. $\mathsf{Q}\Diamond\sigma$) with a context Σ. Note that the contextualised variants will appear after one application of the rule ($\bigcirc\mathsf{E}$) or ($\bigcirc\mathsf{A}$), which removes all $\mathsf{Q}\bigcirc$ prefixes.

Recall that ${\sim}\Sigma'$ is the nnf of the negation of the conjunction of all formulae in Σ' that are left from Σ after performing the set-theoretical difference constraint indicated in the formulation of the rule. The idea now is that ${\sim}\Sigma'$ should also be satisfied until σ_2 becomes satisfied. This prevents the repetition of the context while σ_2 is 'delayed'. Note that Σ' does not include the $\mathsf{A}\Box$ constraint (prefixed by any sequence of the $\mathsf{A}\bigcirc$ constraints) because these formulae would be necessarily repeated along any branch - indeed, if we use ${\sim}\Sigma$ instead of ${\sim}\Sigma'$ we will generate a branch for each $\mathsf{A}\Box$ that will be immediately closed.

Example 3.3. Consider $\Sigma = \{q, \mathsf{E}\bigcirc\neg p, \mathsf{A}\Box a\}$ to be the context of the formula $\mathsf{A}\Diamond p$. Then, $\Sigma' = \{q, \mathsf{E}\bigcirc\neg p\}$ and ${\sim}\Sigma'$ is the formula $\neg q \vee \mathsf{A}\bigcirc p$. Consequently, the contextualised variant $\mathsf{A}((\neg q \vee \mathsf{A}\bigcirc p)\mathscr{U}p)$ of $\mathsf{A}\Diamond p$, preceded by $\mathsf{A}\bigcirc$, is introduced in the branch where $\mathsf{A}\Diamond p$ is delayed. This variant says that p should be fulfilled (in all branches) after the next state, but while p is not fulfilled some formula in the context $\{q, \mathsf{E}\bigcirc\neg p, \mathsf{A}\Box a\}$ should be 'violated'. Since $\mathsf{A}\Box a$ is a formula that cannot be violated in any branch (it is in the initial node), then either $\neg q$ or $\neg\mathsf{E}\bigcirc\neg p$ (in nnf, $\mathsf{A}\bigcirc p$) should be satisfied in all path until p is satisfied. If the context of eventuality $\mathsf{A}\Diamond p$ was empty or, for example, $\{\mathsf{A}\Box a\}$, then the contextualised variant of $\mathsf{A}\Diamond p$ would be $\mathsf{A}(\mathbf{F}\mathscr{U}p)$. ■

Next, we illustrate in Example 3.4 the application of the β^+-rule $(\mathsf{E}\mathscr{U})^+$ to a node labelled by $\{\mathsf{E}(p\mathscr{U}q), \mathsf{A}(\mathbf{F}\mathscr{R}\neg q)\}$. A tableau for $\{\mathsf{E}(p\mathscr{U}q), \mathsf{A}(\mathbf{F}\mathscr{R}\neg q)\}$ is also exhibited in [1,22]. In Section 4 we will use this tableau as a running example, complete its construction, and compare it with the tableau presented in [1,22]. From now on, we highlight with the gray colour the formula (or subformula) to which the β^+-rule is (or will be) applied.

Example 3.4. In Fig. 4, the context of the eventuality $\mathsf{E}(p\mathscr{U}q)$ in the root is the formula $\mathsf{A}(\mathbf{F}\mathscr{R}\neg q)$. The rule $(\mathsf{E}\mathscr{U})^+$ splits the tableau into two branches. In the right-hand successor the middle formula $\mathsf{E}\bigcirc\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathscr{U}q))\mathscr{U}q)$ contains the contextualised variant of the eventuality $\mathsf{E}(p\mathscr{U}q)$ that is constructed by the conjunction of p and $\mathsf{nnf}(\neg\mathsf{A}(F\mathscr{R}\neg q)) = \mathsf{E}(\mathbf{T}\mathscr{U}q)$ as new left-hand component of the until formula. The effect is that along the future states (from the next one), while q is not satisfied, not only p should be satisfied, but also the negation of the current context (i.e. $\mathsf{E}(\mathbf{T}\mathscr{U}q)$) should be satisfied. ■

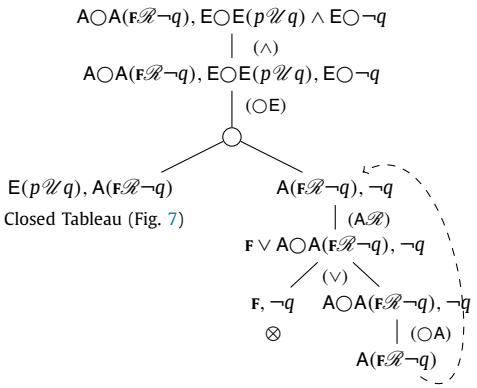


Fig. 5. A closed tableau for $\{\mathsf{A}\bigcirc\mathsf{A}(\mathbf{F}\mathscr{S}\neg q), \mathsf{E}\bigcirc\mathsf{E}(p\mathscr{U}q) \wedge \mathsf{E}\bigcirc\neg q\}$.

4. Systematic tableau construction

In this section we define a recursive algorithm, $\mathscr{A}^{sys}$, that constructs a *fully expanded systematic tableau* for a given set of formulae Σ. Intuitively, 'fully expanded' means that we 'complete' the formation of the tableau in the sense that every expandable node has been already expanded. In §7 we explain how this algorithm, with more parameters and results, returns a model when the resulting tableau is open or, otherwise, a proof in the dual sequent calculus that corresponds with the closed tableau. We first define all the main concepts involved in the algorithm.

A branch is a finite linear structure – formed by the successive nodes, from the root to a leaf– inside the tree-shaped structure of the tableau. When the leaf of a branch has occurred previously in the branch, it is called a loop-node, and the sequence of nodes finitely represents an infinite loop branch. We will load the current branch by 'stages' instead of 'node-by-node' mode.

Definition 4.1 *(Stage).* Given a branch b of a tableau T, a *stage* in b is every maximal subsequence of successive nodes $n_i, n_{i+1}, \ldots, n_j$ in b such that $\tau(n_k)$ is not a $(\bigcirc\mathsf{E})$-child or $(\bigcirc\mathsf{A})$-child of $\tau(n_{k-1})$, for all k such that $i < k \leq j$. We denote by $\mathsf{Stages}(b)$ the *sequence of all stages* of b. The successor relation on $\mathsf{Stages}(b)$ is induced by the successor relation on b. The labelling function τ is extended to stages as the union of the original τ applied to every node in a stage. ■

When the input is a satisfiable set of formulae, the systematic tableau aims to obtain one loop-node since it represents a cycle in a cyclic Kripke structure that could be part of a model of the input set of formulae.

Definition 4.2 *(Loop-node).* Let $b = n_0, n_1, \ldots, n_i$ (where $i > 0$) be a tableau branch and $\mathsf{Stages}(b) = s_0, s_1, \ldots, s_m$ (where $m > 0$ and $n_i \in s_m$). Then, n_i is a *loop-node* if there exists some $0 \leq j < m$ such that $\tau(n_i) \subseteq \tau(s_j)$. We say that s_j is the *companion stage* of the node n_i and $s_j, \ldots, s_m$ are the stages in the loop. ■

Example 4.3. To illustrate the notions of stage and loop-node, let us consider the tree (tableau) in Fig. 5, which contains our running example as a sub-tree. Note that the application of rule $(\bigcirc\mathsf{E})$, at step 2, generates two AND-successors (AND-edges are denoted with a big circle). The left successor is our running example and will be fully expanded later (Fig. 7). The right-most branch is formed by five nodes. This branch has three stages, the first one is formed by the first two nodes, hence its label is the union of their labels, i.e. the set $\{\mathsf{A}\bigcirc\mathsf{A}(\mathbf{F}\mathscr{S}\neg q), \mathsf{E}\bigcirc\mathsf{E}(p\mathscr{U}q) \wedge \mathsf{E}\bigcirc\neg q, \mathsf{E}\bigcirc\mathsf{E}(p\mathscr{U}q), \mathsf{E}\bigcirc\neg q\}$. The second stage in that branch is labelled by $\{\mathsf{A}(\mathbf{F}\mathscr{S}\neg q), \neg q, \mathsf{A}\bigcirc\mathsf{A}(\mathbf{F}\mathscr{S}\neg q)\}$. The last node $\mathsf{A}(\mathbf{F}\mathscr{S}\neg q)$ is a loop-node because it is included in the second stage, which is its companion stage. ■

When a loop-node is found, the fulfilment of eventualities along the branch must be ensured. When this fulfillment condition holds we say that the branch is eventuality-covered. It is obvious that universal eventualities ($\mathsf{A}\mathscr{U}$ or $\mathsf{A}\Diamond$) should be fulfilled in all branches that go across the node where these eventualities appear, as reflected by rules $(\bigcirc\mathsf{Q})$. However, as a consequence of the distribution of existential formulae in different branches performed by the rule $(\bigcirc\mathsf{E})$, an existential eventuality ($\mathsf{E}\mathscr{U}$ or $\mathsf{E}\Diamond$) should be fulfilled only along the branch it belongs to after the $(\bigcirc\mathsf{E})$ splitting, but not in the other branches. Indeed, more than one existential eventuality could appear in the stages of a tableau branch b, however not all of them should be fulfilled in b but, on the contrary, some of them could be 'delayed' and then, by an application of rule $(\bigcirc\mathsf{E})$, could be 'sent' to a different tableau branch.

Example 4.4. For example, in Fig. 5 after the splitting by $(\bigcirc\mathsf{E})$ in two AND-successors, the existential eventuality $\mathsf{E}(p\mathscr{U}q)$ goes to the left subtree, hence it could not be satisfied in the right subtree, where $\mathsf{E}\bigcirc\neg q$ forces to satisfy $\neg q$. ■

Hence, the notion of eventuality coverage (in branches) differentiates existential eventualities from universal ones es explained above.

Definition 4.5 *(Eventuality fulfillment and eventuality-covered branch).* Let b be a tableau branch such that $\mathsf{Stages}(b) = s_0, \ldots, s_n$. An eventuality $\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)$ (resp. $\mathsf{Q}(\Diamond\sigma)$) is *fulfilled* in the branch b if and only if $\sigma_2 \in \tau(s_k)$ (resp. $\sigma \in \tau(s_k)$) for some $0 \leq k \leq n$. The branch b is *eventuality-covered* if and only if the following two conditions hold:

- Every eventuality $\mathsf{A}\mathscr{U}$ or $\mathsf{A}\Diamond$ that occurs at some stage of b is fulfilled in b.
- For every eventuality $\varphi = \mathsf{E}(\sigma_1 \mathscr{U} \sigma_2)$ (resp. $\varphi = \mathsf{E}\Diamond\sigma$) that occurs in some stage of b either φ is fulfilled in b or $\mathsf{E}\bigcirc\varphi \notin s_k$ for some $0 \leq k \leq n$. ■

We only need to check the above two conditions for those eventualities that have not been selected in the branch, because loops cannot contain an unfilled eventuality that has been selected at some point.

In the next example we have a branch with two existential eventualities.

Example 4.6. Consider a branch of five nodes labelled by the following sets:

1. $\neg p, \neg q,\ \mathsf{E}\Diamond p\ ,\ \mathsf{E}\Diamond q$
2. $\neg p, \neg q, \mathsf{E}\bigcirc\ \mathsf{E}((p \vee q \vee \mathsf{A}\Box\neg q)\mathscr{U} p)\ ,\ \mathsf{E}\Diamond q$
3. $\neg p, \neg q, \mathsf{E}\bigcirc\ \mathsf{E}((p \vee q \vee \mathsf{A}\Box\neg q)\mathscr{U} p)\ ,\ \mathsf{E}\bigcirc\mathsf{E}\Diamond q$
4. $\mathsf{E}((p \vee q \vee \mathsf{A}\Box\neg q)\mathscr{U} p)$
5. p

This branch would be generated by successively applying the rules $(\mathsf{E}\Diamond)^+$ to the selected eventuality $\mathsf{E}\Diamond p$ in 1, $(\mathsf{E}\Diamond)$ to eventuality $\mathsf{E}\Diamond q$ in 2, $(\bigcirc\mathsf{E})$ in 3, and $(\mathsf{E}\Diamond)^+$ in 4, and taking only one of the children at each step. This branch is eventuality covered, though $\mathsf{E}\Diamond q$ is not fulfilled in the branch. Indeed the application of the rule $(\bigcirc\mathsf{E})$ at item 3 generates two AND-branches and the other branch starts with node $\mathsf{E}\Diamond q$. ■

Since $(\bigcirc\mathsf{E})$-children are AND-siblings, whenever one of them does not have a possible model, the parent node is unsatisfiable. On the contrary, to ensure the satisfiability of a node where $(\bigcirc\mathsf{E})$ is applied it should have a collection of satisfiable branches that includes all the $(\bigcirc\mathsf{E})$-successors of any node labelled by an elementary set of formulae. These collections of branches are called *bunches*. Next, we define the notion of a bunch and how bunches determine whether a tableau is open, closed and fully expanded.

Definition 4.7 *(Bunch, closed bunch and closed tableau).* A *bunch* H is a collection of branches which is maximal with respect to $(\bigcirc\mathsf{Q})$-successors, i.e. every $(\bigcirc\mathsf{A})$-successor and every $(\bigcirc\mathsf{E})$-successor of any node in H is also in H. A bunch H is *closed* if and only if at least one of its branches is closed, otherwise it is open. A tableau is closed if, and only if, all its bunches are closed. ■

Definition 4.8 *(Fully expanded bunch and tableau).* A branch b is *fully expanded* if and only if either b is closed (see Definition 3.1) or the last node in b is a loop-node and b is eventuality-covered. A bunch is fully expanded if all its branches are fully expanded. A tableau is fully expanded if all its bunches are fully expanded. ■

Example 4.9. The tableau in Fig. 5 is closed in spite of the open sub-tableau at the right $(\bigcirc\mathsf{E})$-successor of the second node. Any bunch in this tableau should include at least one branch across the left node $\{\mathsf{E}(p\mathscr{U}q), \mathsf{A}(\mathbf{F}\mathscr{R}\neg q)\}$ and at least one branch across the right node $\{\mathsf{A}(\mathbf{F}\mathscr{R}\neg q), \neg q\}$. Independently of the branch chosen in the right-hand tree (the closed or the open one), any branch in the left-hand subtree is closed (the details of the closed tableau for $\{\mathsf{E}(p\mathscr{U}q), \mathsf{A}(\mathbf{F}\mathscr{R}\neg q)\}$ are given later). Therefore, any bunch in the tableau of Fig. 5 is closed. ■

Any open tableau has at least one open bunch, formed by one or more open branches. Open branches are ended in a loop-node. Open bunches represent models, specifically *cyclic models* as defined in Definition 2.7.

Example 4.10. In Fig. 6 we depict an open tableau that is a slight modification of the closed tableau in Example 4.3 (Fig. 5). This tableau has three open branches. One open branch is crossing the left-hand $(\bigcirc\mathsf{E})$-child of the second node. Two open branches cross the right-hand child of that node. Hence, there are two possible open bunches, depending on which of the latter two branches we choose. ■

Next, we introduce the recursive Algorithm 1, called $\mathscr{A}^{sys}$, that constructs a *fully expanded systematic tableau* for any input Σ and returns a Boolean value saying if such tableau is closed. The current branch of the tableau construction is passed

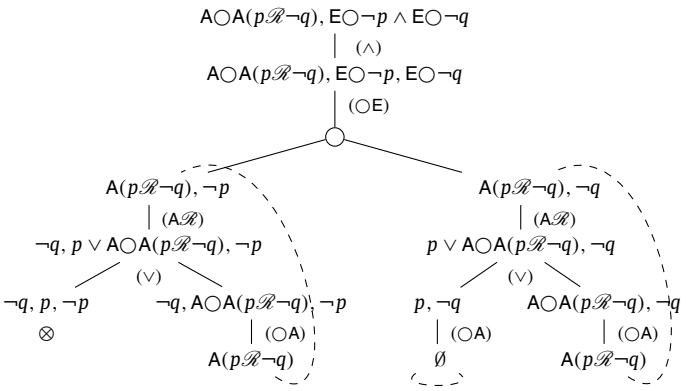


Fig. 6. An open tableau for $\{\mathsf{A}\bigcirc\mathsf{A}(p\mathscr{S}\neg q), \mathsf{E}\bigcirc\neg p \wedge \mathsf{E}\bigcirc\neg q\}$.

through the recursive calls. We can see the branches as lists of stages, which in turn are lists of formulae. Hence, in the first call the branch that receives $\mathscr{A}^{sys}$ as input is $[[\Sigma]]$. We illustrate the steps of the algorithm with details of the construction of the tableaux in Figs. 5, 6 and 7.

Algorithm 1: $\mathscr{A}^{sys}$.
Input = Σ: set of formulae, b: Branch.
Output = *is_closed*: boolean, b': Branch.

```
1  if Σ = ∅ then
2  |  is_closed, b' := FALSE, b;
3  else if Incons(Σ) then
4  |  is_closed, b' := TRUE, b;
5  else if Σ ⊆ τ(s) for some stage s in b & Ev_Covered(b) then
6  |  is_closed, b' := FALSE, b;
7  else if β+_is_applicable(Σ) then
8  |  select_eventuality(Σ) ;
9  |  is_closed, b' := apply_β+_rule(Σ, b)
10 else if α_β_is_applicable(Σ) then
11 |  is_closed, b' := apply_α_β_rule(Σ, b)
12 else // Σ is an elementary set
13 |  Let Σ = Φ, A○(Ψ), E○σ1, ..., E○σk, Φ is a set of literals and k ≥ 0.
14 |  if k ≥ 1 then
15 |  |  Let Σi = Ψ, σi for all 1 ≤ i ≤ k;
16 |  |  n := k;
17 |  else
18 |  |  Σ1, n := Ψ, 1
19 |  end
20 |  i, is_closed :=0, FALSE;
21 |  while is_closed = FALSE & i < n do
22 |  |  is_closed, b' := 𝒜^sys(Σi, b + [[Σi]]);
23 |  |  i := i + 1 ;
24 |  end
25 end
```

Lines 1-6 give the three non-recursive (or simple) cases of $\mathscr{A}^{sys}$. Lines 1-2 deal with the case of the empty input which is trivially satisfiable. Note that by application of the rule $(\bigcirc\mathsf{A})$ when $\mathscr{A}$ is empty we could get this case. Indeed, this is a special case of loop-node labelled by the empty set, whose companion is itself. In Fig. 6 there is a node $\{p, \neg q\}$ where $(\bigcirc\mathsf{A})$ is applied and the loop on the empty set is produced. Lines 3-4 deal with the terminal nodes by inconsistency. For example, the four terminal nodes in Fig. 7 correspond to this case. Three of their labels contain $q, \neg q$ or $\mathbf{F}$ or both. The fourth (right-most) terminal node contains the inconsistent set $\{\mathsf{E}(\mathbf{T}\mathscr{U}q), \mathsf{A}(\mathbf{F}\mathscr{R}\neg q)\}$.

In line 5, Algorithm 1 detects a 'good loop'. An example of this is the last node in the right-most branch in Fig. 5.

Otherwise, there is either a 'bad loop' (i.e. a loop in a non-eventuality-covered branch) or not a loop, and the algorithm tries to apply (if possible) a β^+ rule to some selected eventuality. This is the goal of lines 7-9.

Line 7 stands for the first checked case whenever some tableau rule must be applied to Σ. The rules α, β could really be applied in any order. However, next-state rules can only be applied to elementary sets of formulas. For generating simplest contextualised variants, we just prioritised the application of the β^+-rules. Note that at each stage at most one eventuality can be selected, and at most one β^+ rule is applied. More precisely, the call *select_eventuality*(Σ) selects an unfulfilled eventuality to force its fulfilment by application of the corresponding β^+-rule. Since β^+-rules keep the contextualised vari-

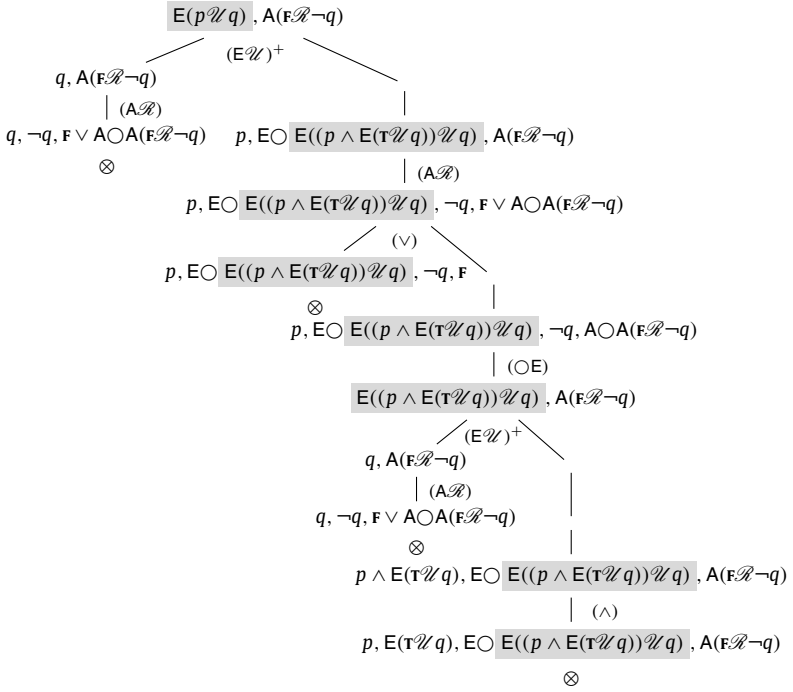


Fig. 7. A closed tableau for $\{\mathsf{E}(p\,\mathscr{U}\,q), \mathsf{A}(\mathbf{F}\mathscr{R}\neg q)\}$.

ant as selected, the predicate call $\beta^+\_is\_applicable(\Sigma)$ gives true if and only if either there is already a selected eventuality $\mathsf{Q}\mathscr{U} \in \Sigma$ or else, both of the following hold: there exists a not selected $\mathsf{Q}\bigcirc\mathsf{Q}\mathscr{U} \in \Sigma$ and there exists a (non-fulfilled) eventuality to be selected. In the former case, $select\_eventuality(\Sigma)$ keeps the selection. In the latter case, it does perform a selection.

Example 4.11. For example, in Fig. 7 the eventuality $\mathsf{E}(p\,\mathscr{U}\,q)$ is selected in the root, and its contextualised variant $\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathscr{U}\,q))\mathscr{U}\,q)$ is kept selected along the right-most branch. ■

The application of a specific β or a β^+ rule R, inside the calls of the procedures $apply\_\beta^+\_rule(\Sigma, b)$ or $apply\_\alpha\_\beta\_rule(\Sigma, b)$, produces two R-children, namely Σ_1 and Σ_2, that are OR-siblings. Hence, in these cases, lines 9 and 11, first do the recursive call $is\_closed, b' := \mathscr{A}^{sys}(\Sigma_1, b_1)$. Then, only if is_closed is true, do the call $is\_closed, b' := \mathscr{A}^{sys}(\Sigma_2, b_2)$. The above branches b_1 and b_2 stand for the adequate actualisation according to Σ_1 and Σ_2. As an example, in Algorithm 2, we provide the details for applying the rule $(\mathsf{E}\mathscr{U})^+$ to input $\Sigma, \mathsf{E}(\sigma_1\,\mathscr{U}\,\sigma_2)$. Note that branches are conveniently updated in recursive calls. In fact, function $update(b, \Sigma)$ adds to the last stage of b the formulae in Σ that previously did not occur in that stage.

Algorithm 2: Apply $(\mathsf{E}\mathscr{U})^+$ to $\Phi = \Sigma, \mathsf{E}(\sigma_1\,\mathscr{U}\,\sigma_2)$.

```
1  Σ := Φ \ {E(σ1 𝒰 σ2)};
2  is_closed, b' := 𝒜^sys(Σ1, b1) where Σ1 = Σ ∪ {σ2} and b1 = update(b, Σ1);
3  if is_closed = TRUE then
4  |  is_closed, b' := 𝒜^sys(Σ2, b2) where Σ2 = Σ ∪ {σ1, E((σ1 ∧ ∼Σ') 𝒰 σ2)}
5  |                                          and b2 = update(b, Σ2);
6  end
```

Example 4.12. In the tableau of Fig. 7, the first application of $(\mathsf{E}\mathscr{U})^+$ to the root node, calls $\mathscr{A}^{sys}(\Sigma_1, b)$ for $\Sigma_1 = \{q, \mathsf{A}(\mathbf{F}\mathscr{R}\neg q)\}$ which returns true in is_closed after the construction of the left-most sub-tree. Then, the call $\mathscr{A}^{sys}(\Sigma_2, b)$ where Σ_2 contains the contextualised variant, constructs the right sub-tree to return also true. Hence, the whole tableau is closed. Note that, in Fig. 7, there is a second call to apply $(\mathsf{E}\mathscr{U})^+$ that works in a similar way. At this second application the selected eventuality $\mathsf{E}((p \wedge (\mathsf{E}(\mathbf{T}\mathscr{U}\,q))\mathscr{U}\,q)$, which is kept selected from the previous nodes in the branch, has the same context that at the first application. Therefore, as we simplify $\varphi \wedge \varphi$ to φ, the contextualised variant for the next step becomes unchanged. ■

For each rule that splits branches (i.e. β or β^+) there is an application algorithm similar to Algorithm 2. For the α-rules, that enlarge the branch with one node, a recursive call to $\mathscr{A}^{sys}$ on the only one child suffices.

In line 10, the algorithm checks and applies applicable α- or β-rules. When no α- or β-rules are applicable, it means that Σ is an elementary set of formulae (line 12). Therefore, in lines 13-23, the rule $(\bigcirc\mathsf{E})$ or $(\bigcirc\mathsf{A})$ is applied, by iterating

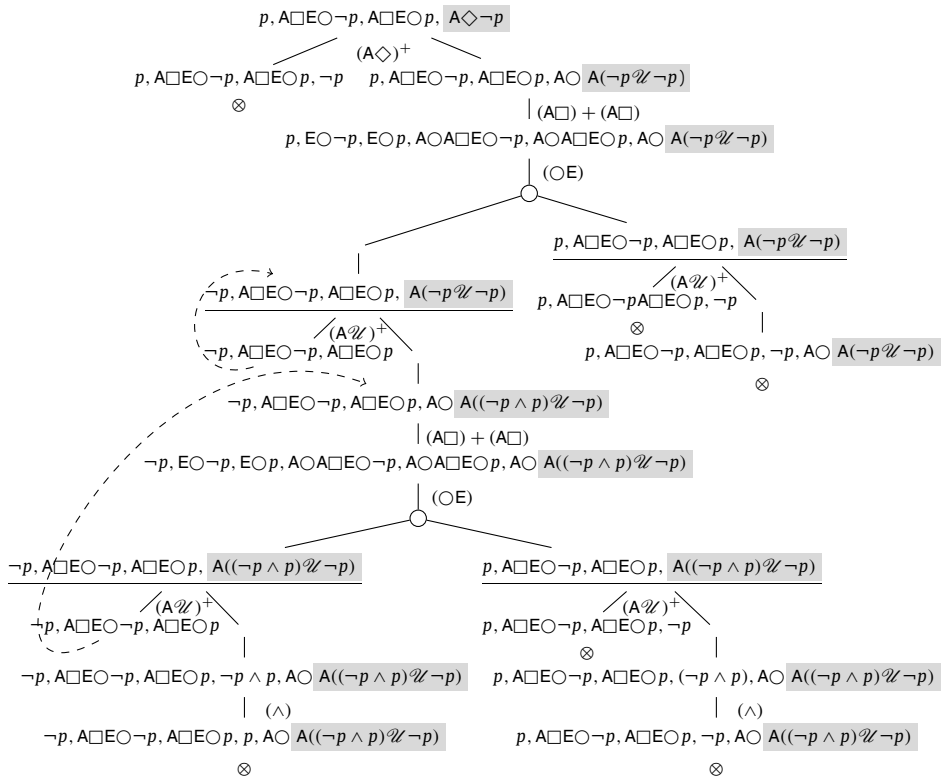


Fig. 8. A closed tableau for $\{p, \mathsf{A}\Box\mathsf{E}\bigcirc p, \mathsf{A}\Box\mathsf{E}\bigcirc\neg p, \mathsf{A}\Diamond\neg p\}$.

recursive calls to $\mathscr{A}^{sys}(\Sigma_i, b + [[\Sigma_i]])$ for each child labelled by Σ_i. Note that, each application of a tableau rule ($\bigcirc\mathsf{Q}$) produces a recursive call for a ($\bigcirc\mathsf{Q}$)-child Σ_i, where the current branch b is actualised to include a new stage containing just Σ_i (this is expressed in the algorithm by $b + [[\Sigma_i]]$). Let us also observe that ($\bigcirc\mathsf{E}$)-children are AND-siblings, hence the iteration on the ($\bigcirc\mathsf{E}$)-children terminates as soon as one of them is closed. On the contrary, if every ($\bigcirc\mathsf{E}$)-child is open the tableau should have a collection of open branches including all the ($\bigcirc\mathsf{E}$)-successors of any node labelled by an elementary set of formulae, i.e. and open bunch. Any open bunch of the systematic tableau, constructed by the algorithm $\mathscr{A}^{sys}$ introduced in this section, enables the construction of a model for the initial set of formulae.

Example 4.13. For example, in Fig. 5, if the left-most sub-tableau (i.e. the closed tableau in the figure) is firstly constructed, then its right-hand child would not be constructed, since any bunch would be necessarily closed (see Example 4.3). In Example 4.10 we show an open tableau that contains two open bunches. ■

Example 4.14. The call $\mathscr{A}^{sys}$ with input $\Sigma = \{\mathsf{E}(p\,\mathscr{U}\,q), \mathsf{A}(\mathsf{F}\mathscr{R}\neg q)\}$ constructs the closed tableau in Fig. 7 as explained along this section. A tableau for the same input Σ is also exhibited in [1,22]. Note the direct correspondence between our context-based tableau (Fig. 7) and the one in [1,22] – they have exactly the same nodes. The right-most branch, in our case, closes by (syntactical) inconsistency, likewise all the other branches. The difference is that, in this branch, the inconsistency comes from the use of the context in the selected eventuality. The corresponding branch in the tableau in [1,22] is closed by the detection of a "bad loop" using information loaded during the construction of the previously constructed branches. ■

Let us recall that in [7] some (subsumption-like) simplification rules were introduced to ensure the termination of the tableau method for the logic ECTL$^{\#}$. The method for CTL (and also for ECTL) requires the following simplification rule:

$$(\sqsubset\mathsf{Q}\mathscr{U})\ \{\mathsf{Q}((\sigma_1 \wedge \chi)\mathscr{U}\sigma_2), \mathsf{Q}(\sigma_1\mathscr{U}\sigma_2)\} \longrightarrow \{\mathsf{Q}((\sigma_1 \wedge \chi)\mathscr{U}\sigma_2)\} \tag{8}$$

By means of this rule, any contextualised variant of an eventuality φ subsumes the original eventuality φ that could repeatedly appear otherwise. Our algorithm systematically performs these simplifications in nodes.

To complete this section, we provide an example of systematic tableau construction involving a universal eventuality $\mathsf{A}\Diamond$ where bunches should cross along the two applications of ($\bigcirc\mathsf{E}$).

Example 4.15. Fig. 8 presents a closed fully-expanded tableau for the unsatisfiable set of formulae $\{p, \mathsf{A}\Box\mathsf{E}\bigcirc p, \mathsf{A}\Box\mathsf{E}\bigcirc\neg p,$

$\mathsf{A}\Diamond\neg p\}$. The systematic construction starts by selecting the unique eventuality and applying the rule $(\mathsf{A}\Diamond)^+$. The successive contextualised variants of this eventuality are kept selected and the rule $(\mathsf{A}\mathscr{U})^+$ is applied to them after each application of the next-state rule $(\bigcirc\mathsf{E})$. The rules $(\mathsf{A}\Box)$ and $(\wedge)$ are applied to reach the elementary sets where $(\bigcirc\mathsf{E})$ is applied. For saving space, we sometimes represent two applications of the rule $(\mathsf{A}\Box)$ in the same step.

It is worth noting that, in spite of the two open branches (see the two loops in Fig. 8), every bunch in the figure is closed. Indeed, any bunch should contain branches for crossing the four $(\bigcirc\mathsf{E})$-nodes (underlined in Fig. 8). It is easy to see that any bunch contains at least one closed branch. The "bad loop detection approach" ([1,22]) would create a "bad loop" branch similar to the largest branch of our tableau in which $\bigcirc p$ is satisfied. For that, the information of the other branches (where p is satisfied) should be used. ■

5. Soundness and completeness

The logics CTL and ECTL are sublogics of $\mathsf{ECTL}^{\#}$ and the tableau method presented here is the adaptation of the method in [7]. In this section we essentially adapt to CTL the soundness and completeness proofs developed in [7]. In §8 we extend both results to ECTL. To prove the soundness of the tableau method for CTL (Theorem 5.2), we show that every tableau rule in Figs. 1, 2 and 3 preserves satisfiability in the sense of the next Lemma 5.1.

Lemma 5.1 *(Soundness of the tableau rules for CTL). Consider all the rules in Figs. 1, and 2 and 3.*

1. *For any α-rule of the form $\dfrac{\Sigma}{\Sigma_1}$, we have* $\mathsf{Sat}(\Sigma)$ *if and only if* $\mathsf{Sat}(\Sigma_1)$.
2. *For any β-rule and any β^+-rule of the form $\dfrac{\Sigma}{\Sigma_1 \mid \Sigma_2}$, we have* $\mathsf{Sat}(\Sigma)$ *if and only if* $\mathsf{Sat}(\Sigma_1)$ *or* $\mathsf{Sat}(\Sigma_2)$.
3. *If Σ is a consistent set of literals, then*
   (a) $\mathsf{Sat}(\Sigma, \cup\{\mathsf{A}\bigcirc\sigma_1, \ldots, \mathsf{A}\bigcirc\sigma_\ell, \mathsf{E}\bigcirc\sigma'_1, \ldots, \mathsf{E}\bigcirc\sigma'_k\})$ *if and only if* $\mathsf{Sat}(\{\sigma_1, \ldots, \sigma_\ell, \sigma'_i\})$ *for all* $1 \leq i \leq k$.
   (b) $\mathsf{Sat}(\Sigma \cup \{\mathsf{A}\bigcirc\sigma_1, \ldots, \mathsf{A}\bigcirc\sigma_\ell\})$ *if and only if* $\mathsf{Sat}(\{\sigma_1, \ldots, \sigma_\ell\})$.

Proof. All these statements follow very easily from the 'systematic' application of the semantic definitions of the temporal modalities, except the 'if' direction' for the β^+-rules. Next we prove the 'if' direction of the rules $(\mathsf{Q}\mathscr{U})^+$ for $\mathsf{Q}=\mathsf{E}$ and $\mathsf{Q}=\mathsf{A}$, because this proof entails the proof for the rules $(\mathsf{Q}\Diamond)^+$ as particular cases (using the abbreviation $\Diamond\sigma = \mathsf{T}\mathscr{U}\sigma$).

For the 'if' direction of the rule $(\mathsf{E}\mathscr{U})^+$, let $\mathscr{K} \models \Sigma, \mathsf{E}(\sigma_1\mathscr{U}\sigma_2)$ and let x be the path in $\mathscr{K}$ such that $\mathscr{K}, x \models \Sigma, \sigma_1\mathscr{U}\sigma_2$. Then, let j be the least $i \geq 0$ such that $\mathscr{K}, x, i \models \sigma_2$. If $j = 0$, then $\mathscr{K}, x, 0 \models \Sigma, \sigma_2$. Otherwise, if $j > 0$ then $\mathscr{K}, x, m \models \sigma_1$, for all $0 \leq m < j$. Consider k to be the greatest of those m such that $\mathscr{K}, x, m \models \Sigma$. Hence, $\mathscr{K}, x, h \models \sim\Sigma$, for all h such that $k+1 \leq h < j$. In particular, by definition of Σ' (obtained from Σ) it is easy to see that $\mathscr{K}, x, h \models \sigma$ for every $\sigma \in (\Sigma \setminus \Sigma')$. Therefore, $\mathscr{K}, x, h \models \sim\Sigma'$, for all h such that $k+1 \leq h < j$. Consequently,

$$\mathscr{K}, x, k \models \Sigma, \sigma_1, \mathsf{E}\bigcirc\mathsf{E}((\sigma_1 \wedge \sim\Sigma')\mathscr{U}\sigma_2).$$

For the 'if' direction of rule $(\mathsf{A}\mathscr{U})^+$, let us suppose that

$$\mathsf{UnSat}(\Sigma \cup \{\sigma_2\}) \text{ and } \mathsf{UnSat}(\Sigma \cup \{\sigma_1, \mathsf{A}\bigcirc\mathsf{A}((\sigma_1 \wedge \sim\Sigma')\mathscr{U}\sigma_2)\}).$$

We will show that $\mathsf{UnSat}(\Sigma \cup \{\mathsf{A}(\sigma_1\mathscr{U}\sigma_2)\})$. For that, let us consider any arbitrary $\mathscr{K}$ such that $\mathscr{K} \models \Sigma$ to show that $\mathscr{K} \not\models \mathsf{A}(\sigma_1\mathscr{U}\sigma_2)$. By the above unsatisfiability hypothesis, if $\mathscr{K} \models \Sigma$, then both $\mathscr{K} \not\models \sigma_2$ and $\mathscr{K} \not\models \sigma_1 \wedge \mathsf{A}\bigcirc\mathsf{A}((\sigma_1 \wedge \sim\Sigma')\mathscr{U}\sigma_2)$. Then, there are two possible cases. First, if $\mathscr{K} \models \neg\sigma_1 \wedge \neg\sigma_2$, then it is obvious that $\mathscr{K} \not\models \mathsf{A}(\sigma_1\mathscr{U}\sigma_2)$. Second, if $\mathscr{K} \models \neg\sigma_2 \wedge \neg\mathsf{A}\bigcirc\mathsf{A}((\sigma_1 \wedge \sim\Sigma')\mathscr{U}\sigma_2)$, then there exists $x_1 \in \mathsf{fullpaths}(\mathscr{K})$ and $i_1 > 0$ that satisfy both $\mathscr{K}, x_1, j \models \neg\sigma_2$ for all j such that $0 \leq j \leq i_1$, and $\mathscr{K}, x_1, i_1 \models \neg\sigma_1 \vee \Sigma'$. Since all the formulae in $\Sigma \setminus \Sigma'$ are satisfied in all states along all paths, indeed $\mathscr{K}, x_1, i_1 \models \neg\sigma_1 \vee \Sigma$. Therefore, if $\mathscr{K}, x_1, i_1 \models \neg\sigma_1$, then obviously $\mathscr{K} \not\models \mathsf{A}(\sigma_1\mathscr{U}\sigma_2)$. Otherwise, if $\mathscr{K}, x_1, i_1 \models \Sigma$, applying the same reasoning for $\mathscr{K} \restriction x_1(i_1)$ as we did above for $\mathscr{K}$, we can conclude that there should be a path $x_2 \in \mathsf{fullpaths}(\mathscr{K} \restriction x_1(i_1))$ and some $i_2 > 0$ such that either $\mathscr{K} \restriction x_1(i_1), x_2, j \models \neg\sigma_2$ for all j such that $i_1 \leq j \leq i_2$ and $\mathscr{K} \restriction x_1(i_1), x_2, i_2 \models \neg\sigma_1 \vee \Sigma$. Hence, if $\mathscr{K} \restriction x_1(i_1), x_2, i_1 \models \neg\sigma_1$, then trivially $\mathscr{K} \not\models \mathsf{A}(\sigma_1\mathscr{U}\sigma_2)$. Otherwise, $\mathscr{K} \restriction x_1(i_1), x_2, i_1 \models \Sigma$. Hence, there are two possible scenarios: 1.) After a finite number of iterations we get a path $y = x_1^{<i_1}, x_2^{<i_2}, \cdots, x_k^{<i_k} \cdots$ such that $\mathscr{K}, y, j \models \neg\sigma_2$ for all j such that $0 \leq j \leq i_k$ and $\mathscr{K}, y, i_k \models \neg\sigma_1$. 2.) The infinite iteration of the second case yields a path $y = x_1^{<i_1}, x_2^{<i_2}, \cdots, x_k^{<i_k}, \cdots$ (that exists by the limit closure property) such that $\mathscr{K}, y, i \models \neg\sigma_2$ for all $i \geq 0$. In both scenarios we have $\mathscr{K} \not\models \mathsf{A}(\sigma_1\mathscr{U}\sigma_2)$ holds for any arbitrary $\mathscr{K}$ that satisfies Σ. Thus, $\mathsf{UnSat}(\Sigma \cup \{\mathsf{A}(\sigma_1\mathscr{U}\sigma_2)\})$. ■

According to Lemma 5.1 and the definition of the closed tableau, we prove the following result.

Theorem 5.2 *(Soundness of the tableau method for CTL). Given any set of state formulae Σ, if there exists a closed tableau for Σ then* $\mathsf{UnSat}(\Sigma)$.

Proof. In a closed tableau for Σ, at least one leaf in each bunch must have an inconsistent set of formulae that labels it. Therefore, this set is unsatisfiable. Then, by (the converse of) Lemma 5.1, the label of the root node, Σ, is unsatisfiable. ■

Next, we prove the refutational completeness of the tableau method for CTL (Theorem 5.7). For that, we firstly define the notion of a saturated stage and prove some auxiliary properties of the stages and bunches of the systematic tableau. These properties are necessary to prove that every open bunch in the systematic tableau represents a model of the initial set of formulae Σ (Lemma 5.6).

Definition 5.3 *($\alpha\beta^+$-saturated stage).* We say that a stage $s = n_i, \ldots, n_j$ in the tableau $\mathscr{A}^{sys}$ for Σ is $\alpha\beta^+$-saturated if and only if it satisfies the following conditions:

1. For all $\sigma_1 \wedge \sigma_2 \in \tau(s)$: $\{\sigma_1, \sigma_2\} \subseteq \tau(s)$.
2. For all $\mathsf{Q}\Box\sigma \in \tau(s)$: $\{\sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Box\sigma\} \subseteq \tau(s)$.
3. For all $\sigma_1 \vee \sigma_2 \in \tau(s)$: $\sigma_1 \in \tau(s)$ or $\sigma_2 \in \tau(s)$.
4. For all $\mathsf{Q}(\sigma_1 \mathscr{R} \sigma_2) \in \tau(s)$: $\{\sigma_2, \sigma_1 \vee \mathsf{Q}\bigcirc\mathsf{Q}(\sigma_1 \mathscr{R} \sigma_2)\} \subseteq \tau(s)$.
5. For all $\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2) \in \tau(s)$: $\sigma_2 \in \tau(s)$ or $\{\sigma_1, \mathsf{Q}\bigcirc\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)\} \subseteq \tau(s)$ or $\{\sigma_1, \mathsf{Q}\bigcirc\mathsf{Q}((\sigma_1 \wedge \sim\Sigma')\mathscr{U}\sigma_2)\} \subseteq \tau(s)$, where $\Sigma' = (\tau(n_i) \setminus \{\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)\}) \setminus \{(\mathsf{A}\bigcirc)^i \mathsf{A}\Box\varphi \mid i \geq 0 \text{ and } (\mathsf{A}\bigcirc)^i\mathsf{A}\Box\varphi \in \tau(n_i)\}$.
6. For all $\mathsf{Q}(\Diamond\sigma) \in \tau(s)$: $\sigma \in \tau(s)$ or $\{\mathsf{Q}\bigcirc\mathsf{Q}(\Diamond\sigma)\} \subseteq \tau(s)$ or $\{\mathsf{Q}\bigcirc\mathsf{Q}((\sim\Sigma')\mathscr{U}\sigma)\} \subseteq \tau(s)$, where $\Sigma' = (\tau(n_i) \setminus \{\mathsf{Q}\Diamond\sigma\}) \setminus \{(\mathsf{A}\bigcirc)^i\mathsf{A}\Box\varphi \mid i \geq 0 \text{ and } (\mathsf{A}\bigcirc)^i\mathsf{A}\Box\varphi \in \tau(n_i)\}$. ■

The construction of the systematic tableau applies exactly one β^+-rule to exactly one selected eventuality (if any) at the first node of the stage, and then applies exhaustively all the applicable α- and β-rules to the formulae in the stage, until the branch closes, or its leaf is labelled by an elementary set, or it contains a loop-node. Consequently, the following result can be trivially proved by construction.

Proposition 5.4. *Given any set of state formulae Σ, the systematic tableau $\mathscr{A}^{sys}$ for Σ is fully expanded.*

Proof. It is trivial, by construction, that every stage in $\mathscr{A}^{sys}$ is $\alpha\beta^+$-saturated. ■

Next we prove a crucial property of the systematic tableau management of eventualities by means of the selection policy.

Proposition 5.5. *Let b be an open branch of the tableau $\mathscr{A}^{sys}$ for Σ.*

1. *If a formula $\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)$ is selected at some stage $s_i \in \mathsf{Stages}(b)$, then there exists some stage $s_k \in \mathsf{Stages}(b)$ (for some $k \geq i$) such that $\sigma_2 \in \tau(s_k)$ and $\sigma_1 \in \tau(s_j)$ for all $j \in \{i, \ldots, k-1\}$.*
2. *If a formula $\mathsf{Q}(\Diamond\sigma)$ is selected at some stage $s_i \in \mathsf{Stages}(b)$, then there exists some stage $s_k \in \mathsf{Stages}(b)$ (for some $k \geq i$) such that $\sigma \in \tau(s_k)$.*

Proof. We will prove item 1. Item 2 is the particular case where $\sigma_1 = \mathbf{T}$ and $\sigma_2 = \sigma$. If $\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)$ is the selected formula at stage $s_i \in \mathsf{Stages}(b)$, by Algorithm 1, the set labelling the first node at each stage s_j $(j \geq i)$ of b has the form

$$\Sigma_{s_j}, \mathsf{Q}((\sigma_1 \wedge (\sim\Sigma_{s_i} \wedge \cdots \wedge \sim\Sigma_{s_{j-1}}))\mathscr{U}\sigma_2)$$

where each Σ_{s_j} is the context of the selected formula containing the contextualised variant of $\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)$ at the first node of each stage s_j. Since no other β^+-rule is applied each Σ_{s_j} is a subset of the finite set formed by all state formulae that are subformulae of some formula in Σ_{s_i} and their negations. Hence, there are a finite number of different Σ_{s_j}. Therefore, after finitely many applications of the β^+-rule, $\Sigma_{s_h} = \Sigma_{s_j}$, for some $h >= i$, for some $j \in \{i, \ldots, h-1\}$, and $\sigma_1 \wedge (\sim\Sigma_{s_i} \wedge \cdots \wedge \sim\Sigma_{s_{h-1}}) \in \tau(s_h)$. In particular, $\sim\Sigma_{s_h} \in \tau(s_h)$, hence, Σ_{s_h} must be inconsistent. Since b is open, this is a contradiction. This means that, for some $k \geq i$ the application of the corresponding β^+-rule should force that $\sigma_2 \in \tau(s_k)$. In addition, by Proposition 5.4 and Definition 5.3(5), $\sigma_1 \in \tau(s_j)$ for all $j \in \{i, \ldots, k-1\}$. ■

Lemma 5.6 *(Model existence). Let Σ be any set of formulae. For any fully expanded bunch H of the tableau $\mathscr{A}^{sys}$ for Σ, there exists a Kripke structure $\mathscr{K}_H$ such that $\mathscr{K}_H \models \Sigma$.*

Proof. Let H be any fully expanded bunch of $\mathscr{A}^{sys}$. We define $\mathscr{K}_H = (S, R, L)$ such that $S = \bigcup_{b \in H} \mathsf{Stages}(b)$ and for any $s \in S$: $L(s) = \{p \mid p \in \tau(n) \cap \mathsf{Prop} \text{ for some node } n \in s\}$. R is the relation induced in $\mathsf{Stages}(b)$ for each $b \in H$. Any branch in $b \in H$ is open, hence b ends in a loop-node. Moreover, every eventuality has been selected in some stage of b. Therefore,

there exists a (possibly empty) set Σ_ℓ such that for some $i \geq 0$: $b = s_0, s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_j, n_\ell$, where each s_h stands for a stage and n_ℓ is a non-expandable loop-node labelled by Σ_ℓ whose companion node is the first node at stage s_i. We are going to prove the following fact:

$$\mathscr{K}_H, s_a, 0 \models \sigma \text{ for any } a \in \{0, \ldots, j\} \text{ and any formula } \sigma \text{ in } L(s_a)$$

by structural induction on the formula σ.

The base of the induction, for $\sigma = p \in \mathsf{Prop}$, follows by definition of $\mathscr{K}_H$. The cases where σ has one of the forms $\sigma_1 \wedge \sigma_2$, $\mathsf{Q}\Box\sigma$, $\sigma_1 \vee \sigma_2$ and $\mathsf{Q}(\sigma_1 \mathscr{R} \sigma_2)$ are trivial by Definition 5.3 and the induction hypothesis. Hence, to complete the inductive proof we will show that $\mathscr{K}_H, s_a, 0 \models \mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)$ for any $\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2) \in L(s_a)$. The case for all $\mathsf{Q}\Diamond\sigma \in L(s_a)$ follows as a particular case by $\Diamond\sigma \equiv \mathbf{T}\mathscr{U}\sigma$. Consider any $\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2) \in L(s_a)$. Since b is eventuality-covered and n_ℓ is a loop-node, $\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)$ must be the selected eventuality at some node between the states s_a and s_j. Hence, by Proposition 5.5 and the definition of $\mathscr{K}_H$, there should be a state $s_k \in S$ (for some $a \leq k \leq j$) such that $\sigma_2 \in L(s_k)$ and $\sigma_1 \in L(s_z)$ for all $z \in \{a, \ldots, k-1\}$. Then, by induction hypothesis, $\mathscr{K}_H, s_k, 0 \models \sigma_2$ and $\mathscr{K}_H, s_z, 0 \models \sigma_1$ for all $z \in \{a, \ldots, k-1\}$. Therefore, $\mathscr{K}_H, s_a, 0 \models \mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2)$.

To complete the proof, we show that the successor relation between states in $\mathscr{K}_H$ is well-defined. For that, consider any tableau node in any stage s_a that is labelled by an elementary set

$$\{\Sigma, \mathsf{A}\bigcirc\sigma_1, \ldots, \mathsf{A}\bigcirc\sigma_n, \mathsf{E}\bigcirc\sigma'_1, \ldots, \mathsf{E}\bigcirc\sigma'_k\}$$

where Σ is a consistent set of literals, by rule ($\bigcirc\mathsf{E}$), s_a has (in $\mathscr{K}_H$) a successor state s^i_{a+1}, for each $i \in \{1, \ldots, k\}$, such that $L(s^i_{a+1}) = \{\sigma_1, \ldots, \sigma_n, \sigma'_i\}$. We can assume (by the above proved fact) that $\mathscr{K}_H, s^i_{a+1}, 0 \models \{\sigma_1, \ldots, \sigma_n, \sigma'_i\}$ for all $i \in \{1, \ldots, k\}$. Therefore, we can infer that $\mathscr{K}_H, s_a, 0 \models \{\Sigma, \mathsf{A}\bigcirc\sigma_1, \ldots, \mathsf{A}\bigcirc\sigma_n, \mathsf{E}\bigcirc\sigma'_1, \ldots, \mathsf{E}\bigcirc\sigma'_k\}$. ■

Next, we prove the refutational completeness of the tableau method.

Theorem 5.7 *(Refutational completeness for CTL). For any set of state formulae Σ, if $\mathsf{UnSat}(\Sigma)$ then there exists a closed tableau for Σ.*

Proof. Suppose the contrary, i.e. there exists no closed tableau for Σ. Then the systematic tableau $\mathscr{A}^{sys}$ for Σ is open. Hence, there is at least one fully expanded bunch H in $\mathscr{A}^{sys}$. By Lemma 5.6, there exists a Kripke structure $\mathscr{K}_H$ such that $\mathscr{K}_H \models \Sigma$. Consequently, $\mathsf{Sat}(\Sigma)$. ■

Finally, we prove the completeness of our tableau method for CTL, and the first step here is to show that the method terminates.

Theorem 5.8 *(Termination of the tableau method for CTL). For any set of state formulae Σ, the construction of the fully expanded tableau $\mathscr{A}^{sys}$ for Σ terminates.*

Proof. Tableau rules produce a finite branching, hence König's Lemma, applies. Therefore, it suffices to prove that every branch is finite. By Proposition 5.5, the application of a β^+-rule to a selected formula stops after a finite number of steps. Since the number of selectable eventualities in any open branch is finite, any open branch is eventuality-covered after a finite number of eventuality selections. Recall that we assume the eventuality selection strategy to be fair. ■

Theorem 5.9 *(Completeness of the tableau method for CTL). For any set of state formulae Σ, if Σ is satisfiable then there exists a (finite) open fully expanded tableau for Σ.*

Proof. By Theorems 5.2 and 5.8, and the fact that the fully expanded systematic tableau $\mathscr{A}^{sys}$ for Σ is finite and cannot be closed. ■

6. The dual sequent calculus for CTL

In this section we introduce a sequent calculus for CTL, called $\mathscr{C}_\vdash$, that is dual to the tableau method presented in previous sections. Indeed, the sequent calculus, given in Fig. 9, is simply a reformulation of the tableau rules as a one-sided sequent calculus, the right-hand side of every sequent is the constant $\mathbf{F}$.

The calculus $\mathscr{C}_\vdash$ contains classical rules, such as ($\wedge$) and ($\vee$) for Booleans, and (*Ctd*) and ($\mathbf{F}$) for contradictions. The last two rules correspond to the branch closing conditions of the tableau method and they are premise-free, while all the other rules have at least one *premise*.

In the rules ($\mathsf{Q}\mathscr{R}$), ($\mathsf{Q}\Box$), ($\mathsf{Q}\mathscr{U}$), ($\mathsf{Q}\Diamond$), ($\mathsf{Q}\mathscr{U}$)$^+$, and ($\mathsf{Q}\Diamond$)$^+$, the symbol Q again denotes either of the path quantifiers, E or A. Therefore, each of these rules stands for a pair of analogous rules for each path quantifier, which correspond directly to the tableau rules of the same name.

$$(\wedge)\ \frac{\Sigma, \sigma_1, \sigma_2 \vdash \mathbf{F}}{\Sigma, \sigma_1 \wedge \sigma_2 \vdash \mathbf{F}} \qquad (\vee)\ \frac{\Sigma, \sigma_1 \vdash \mathbf{F} \mid \Sigma, \sigma_2 \vdash \mathbf{F}}{\Sigma, \sigma_1 \vee \sigma_2 \vdash \mathbf{F}} \qquad (Ctd)\ \frac{}{\Sigma, \sigma, \sim\sigma \vdash \mathbf{F}} \qquad (\mathbf{F})\ \frac{}{\Sigma, \mathbf{F} \vdash \mathbf{F}}$$

$$(\mathsf{Q}\mathscr{R})\ \frac{\Sigma, \sigma_2, \sigma_1 \vee \mathsf{Q}\bigcirc\mathsf{Q}(\sigma_1 \mathscr{R} \sigma_2) \vdash \mathbf{F}}{\Sigma, \mathsf{Q}(\sigma_1 \mathscr{R} \sigma_2) \vdash \mathbf{F}} \qquad (\mathsf{Q}\Box)\ \frac{\Sigma, \sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Box\sigma \vdash \mathbf{F}}{\Sigma, \mathsf{Q}\Box\sigma \vdash \mathbf{F}}$$

$$(\mathsf{Q}\mathscr{U})\ \frac{\Sigma, \sigma_2 \vdash \mathbf{F} \quad \Sigma, \sigma_1, \mathsf{Q}\bigcirc\mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2) \vdash \mathbf{F}}{\Sigma, \sigma_1 \mathscr{U} \sigma_2 \vdash \mathbf{F}} \qquad (\mathsf{Q}\Diamond)\ \frac{\Sigma, \sigma \vdash \mathbf{F} \quad \Sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Diamond\sigma \vdash \mathbf{F}}{\Sigma, \mathsf{Q}\Diamond\sigma \vdash \mathbf{F}}$$

$$(\mathsf{Q}\mathscr{U})^+\ \frac{\Sigma, \sigma_2 \vdash \mathbf{F} \quad \Sigma, \sigma_1, \mathsf{Q}\bigcirc\mathsf{Q}((\sigma_1 \wedge \sim\Sigma')\mathscr{U}\sigma_2) \vdash \mathbf{F}}{\Sigma, \mathsf{Q}(\sigma_1 \mathscr{U} \sigma_2) \vdash \mathbf{F}}$$

$$(\mathsf{Q}\Diamond)^+\ \frac{\Sigma, \sigma \vdash \mathbf{F} \quad \Sigma, \mathsf{Q}\bigcirc\mathsf{Q}((\sim\Sigma')\mathscr{U}\sigma) \vdash \mathbf{F}}{\Sigma, \mathsf{Q}\Diamond\sigma \vdash \mathbf{F}}$$

$$(\bigcirc\mathsf{E})\ \frac{\Sigma, \sigma \vdash \mathbf{F}}{\Sigma_0, \mathsf{A}\bigcirc\Sigma, \mathsf{E}\bigcirc\sigma \vdash \mathbf{F}} \qquad (\bigcirc\mathsf{A})\ \frac{\Sigma \vdash \mathbf{F}}{\Sigma_0, \mathsf{A}\bigcirc\Sigma \vdash \mathbf{F}}$$

Fig. 9. The Sequent Calculus for $\mathscr{C}_{\vdash}$. In $(\mathsf{Q}\mathscr{U})^+$ and $(\mathsf{Q}\Diamond)^+$, $\Sigma' = \Sigma \setminus \{(\mathsf{A}\bigcirc)^i \mathsf{A}\Box\sigma \in \Sigma \mid i \geq 0\}$. In $(\bigcirc\mathsf{E})$ and $(\bigcirc\mathsf{A})$, Σ_0 is a set of literals.

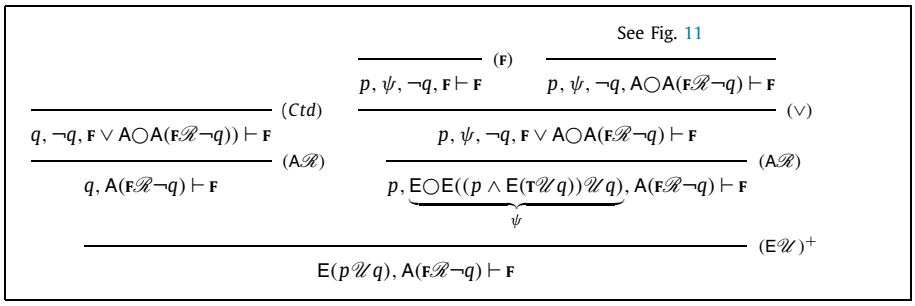


Fig. 10. Sequent proof for $\{\mathsf{E}(p\mathscr{U}q), \mathsf{A}(\mathbf{F}\mathscr{R}\neg q)\}$.

Note that the children of an elementary set with more than one formula starting by $\mathsf{E}\bigcirc$ are AND-siblings, so that in the calculus it is enough to refute one of this AND-children. This corresponds to the tableau notion of a closed bunch (Definition 4.7).

The soundness and completeness of $\mathscr{C}_{\vdash}$ easily follows from its duality with the tableau method for CTL presented in the previous sections.

Theorem 6.1 *(Soundness). For any set of CTL formulae Σ, if there exists a sequent proof of $\Sigma \vdash \mathbf{F}$, then* $\mathsf{UnSat}(\Sigma)$.

Proof. By induction on the length of the sequent calculus proof, it suffices to prove that every sequent rule is correct in the sense that if the set of formulae in the left-hand-side of each premise is unsatisfiable then the set of formulae of the conclusion is unsatisfiable. For the rules (Ctd) and $(\mathbf{F})$ is trivial. For the rest of the sequent rules the proof is similar to the proofs of the analogous tableaux rules in Lemma 5.1. ■

Theorem 6.2 *(Completeness). For any set of CTL formulae Σ, if* $\mathsf{UnSat}(\Sigma)$, *then there exists a sequent proof of $\Sigma \vdash \mathbf{F}$.*

Proof. If $\mathsf{UnSat}(\Sigma)$, then the tableau $\mathscr{A}^{sys}$ for Σ is fully expanded and closed. Each leaf of $\mathscr{A}^{sys}$ corresponds to an axiom obtained by application of the sequent rule (Ctd) or $(\mathbf{F})$. A sequent proof of $\Sigma \vdash \mathbf{F}$ is obtained from axioms by an application of the sequent rule corresponding to a tableau rule in the construction of $\mathscr{A}^{sys}$. ■

Example 6.3. In Fig. 10, we show the sequent proof that corresponds to the closed tableau in Fig. 7. ■

7. *MomoCTL*: implementation and experimentation

In this section we describe our implementation of the context-based tableau for CTL, called *MomoCTL*, and also compare its performance results with Gore's CTL-prover [1,22], called *TreeTab*. The benchmarks used were presented in [23] and they are available in http://users.cecs.anu.edu.au/~rpg/CTLComparisonBenchmarks/.

$$\dfrac{\dfrac{}{q, \neg q, \mathbf{F} \vee \mathsf{A}\bigcirc\mathsf{A}(\mathbf{F}\mathscr{R}\neg q) \vdash \mathbf{F}}\,(Ctd)}{\dfrac{}{q, \mathsf{A}(\mathbf{F}\mathscr{R}\neg q) \vdash \mathbf{F}}\,(\mathsf{A}\mathscr{R})} \qquad \dfrac{\dfrac{}{p, \mathsf{E}(\mathbf{T}\mathscr{U} q), \psi, \mathsf{A}(\mathbf{F}\mathscr{R}\neg q) \vdash \mathbf{F}}\,(Ctd)}{p \wedge \mathsf{E}(\mathbf{T}\mathscr{U} q), \psi, \mathsf{A}(\mathbf{F}\mathscr{R}\neg q) \vdash \mathbf{F}}\,(\wedge)$$

$$\dfrac{q, \mathsf{A}(\mathbf{F}\mathscr{R}\neg q) \vdash \mathbf{F} \qquad p \wedge \mathsf{E}(\mathbf{T}\mathscr{U} q), \psi, \mathsf{A}(\mathbf{F}\mathscr{R}\neg q) \vdash \mathbf{F}}{\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathscr{U} q))\mathscr{U} q), \mathsf{A}(\mathbf{F}\mathscr{R}\neg q) \vdash \mathbf{F}}\,(\mathsf{E}\mathscr{U}^{+})$$

$$\dfrac{\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathscr{U} q))\mathscr{U} q), \mathsf{A}(\mathbf{F}\mathscr{R}\neg q) \vdash \mathbf{F}}{p, \psi, \neg q, \mathsf{A}\bigcirc\mathsf{A}(\mathbf{F}\mathscr{R}\neg q) \vdash \mathbf{F}}\,(\bigcirc\mathsf{E})$$

Fig. 11. Sequent proof for $\{p, \psi, \neg q, \mathsf{A}\bigcirc\mathsf{A}(\mathbf{F}\mathscr{R}\neg q)\}$ where $\psi = \mathsf{E}\bigcirc\mathsf{E}((p \wedge \mathsf{E}(\mathbf{T}\mathscr{U} q))\mathscr{U} q)$.

Algorithm 3: $\mathscr{E}$.
Input = Σ: set of formulae, b: Branch, P: Proof, M: Model.
Output = *is_closed*: boolean, b': Branch, P': Proof, M': Model.

if $\Sigma = \emptyset$ **then**
  $is\_closed, b' :=$ FALSE, b;
  $M' := EmptyLeaf\_Model$;
else if (F) applies to Σ **then**
  $is\_closed, b' :=$ TRUE, b;
  $P' := ProofLeaf((\mathbf{F}), \Sigma)$;
else if (Ctd) applies to Σ **then**
  $is\_closed, b' :=$ TRUE, b;
  $P' := ProofLeaf((\text{Ctd}), \Sigma)$;
else if $\Sigma \subseteq \tau(s)$ for some stage s in b & Ev_Covered(b) **then**
  $is\_closed, b' :=$ FALSE, b;
  $M' := ModelLeaf(s, b)$;
else if $\beta^{+}\_is\_applicable(\Sigma)$ **then**
  $select\_eventuality(\Sigma)$;
  $is\_closed, b', P', M' := apply\_\beta^{+}\_rule(\Sigma, b, P, M)$
else if $\alpha\_\beta\_is\_applicable(\Sigma)$ **then**
  $is\_closed, b', P', M' := apply\_\alpha\_\beta\_rule(\Sigma, b, P, M)$
else // Σ is an elementary set
  Let $\Sigma = \Phi, \mathsf{A}\bigcirc(\Psi), \mathsf{E}\bigcirc\sigma_1, \ldots, \mathsf{E}\bigcirc\sigma_k$ where $k \geq 0$;
  if $k \geq 1$ **then**
    Let $\Sigma_i = \Psi, \sigma_i$ for all $1 \leq i \leq k$;
    $n := k$;
  else
    $\Sigma_1, n := \Psi, 1$
  end
  $i, is_closed :=$0, FALSE;
  $ListModels := EmptyList$;
  while $is\_closed =$ FALSE & $i < n$ **do**
    $i := i + 1$;
    $is\_closed, b', P_i, M_i := \mathscr{E}(\Sigma_i, b + [[\Sigma_i]], P, M)$;
    if $is\_closed =$ FALSE **then** $add(M_i, ListModels)$;
  end
  if $is\_closed =$ TRUE **then**
    if $k = 0$ **then** $P' := ProofNode((\bigcirc\mathsf{A}), \Sigma, [P_i])$;
    else $P' := ProofNode((\bigcirc\mathsf{E}), \Sigma, [P_i])$;
  else
    $M' := ModelNode(Atoms(\Sigma), ListModels)$
  end
end

7.1. Implementation

Recall that our aim was to develop deductive techniques for branching-time logics with a reasoning mechanism capable of providing both certificates and counterexamples. Thus, we have implemented a prototype that, given a set of formulae Σ as input, decides whether Σ is satisfiable or unsatisfiable and returns either a Kripke structure that certifies the satisfiability of Σ or a sequent calculus proof that refutes Σ.

First we extend the algorithm $\mathscr{A}^{sys}$ (see Algorithm 1) to $\mathscr{E}$ (see Algorithm 3), so that depending on the value of *is_closed*, $\mathscr{E}$ is able to return either a proof or a model, but not both.

We can easily construct a model from an open tableau. Models are represented as trees. Each branch of the tree corresponds to a branch of the open tableau, which ends with a leaf pointing to some ancestor in the branch. Each non-leaf

node in a branch of the tree is the set of atoms appearing in one of the stages of the corresponding branch in the tableau. Example 7.1 displays the model returned by $\mathscr{E}$ when executed over a set of formulae.

The representation of a proof is also a tree whose nodes are steps of the proof. Thus, each leaf contains the inconsistent sequent and the name of the premise-free rule (Ctd) or (F) that applies to it. Non-leaf nodes contain the sequent Σ and the name of the rule (R) that applies to prove that $\Sigma \vdash$ F. Its children contain the proofs of the subgoals that result from applying (R) to $\Sigma \vdash$ F. In the initial call to algorithm $\mathscr{E}$, the proof and the model are both empty, therefore to decide the satisfiability of a set Σ and generate a proof or a model, we call $\mathscr{E}(\Sigma, [[\Sigma]], EmptyProof, EmptyModel)$. Note also that the branch contains just one stage that consists of Σ.

Lines 1-12 are the non-recursive cases. In line 3, the algorithm returns a model of the empty set of formulae, which is a linear model that consists of an empty state (every atom is false in it) that is infinitely repeated. In lines 6 and 9, it returns just one sequent which is a proof by one of the premise-free rules (F) and (Ctd), respectively. In line 10, the algorithm detects that Σ gets an eventuality-covered loop at the stage s in the current branch b. Therefore, the algorithm creates a branch of the potential model with the atoms that are loaded (as formulae) at each stage of b from s to the current node. This branch terminates by a leaf which is just a pointer to stage s.

Recursive constructions of proofs and models, are performed in lines 15, 17 and 26-36 of Algorithm 3.

Algorithm 4: Apply $(\mathsf{E}\mathscr{U})^+$ to $\Phi = \Sigma, \mathsf{E}(\sigma_1 \mathscr{U} \sigma_2)$ that returns a proof, when it exists.

```
1  Σ := Φ \ {E(σ1 U σ2)};
2  is_closed, b', P1, M' := E(Σ1, b1, P, M) where
3                                   Σ1 = Σ ∪ {σ2} and b1 = update(b, Σ1);
4  if is_closed = TRUE then
5  |  is_closed, b', P2, M' = E(Σ2, b2, P, M) where
6  |                                 Σ2 = Σ ∪ {σ1, E((σ1 ∧ ∼Σ') U σ2)} and
7  |                                 b2 = update(b, Σ2);
8  |  if is_closed = TRUE then
9  |  |  P' := ProofNode((E U)+, Σ, [P1, P2]);
10 |  end
11 end
```

The calls to $apply\_\beta^+\_rule$ and $apply\_\alpha\_\beta\_rule$ apply the corresponding rule to a designated formula in the node label, and recursively call $\mathscr{E}$ with the children produced by the applied rule. For example, see Algorithm 4, to apply the β^+-rule $(\mathsf{E}\mathscr{U})^+$ to $\Phi = \Sigma, \mathsf{E}(\sigma_1 \mathscr{U} \sigma_2)$. The procedure $\mathscr{E}$ is called on parameters $\Sigma_1 = \Sigma \cup \{\sigma_2\}$, the extended branch b_1, with the proof P, and the model M. Both P and M where the parameters of the call $apply\_\beta^+\_rule(\Sigma, b, P, M)$ (in line 15). If this recursive call returns in is_closed the value true, $\mathscr{E}$ is called with $\Sigma_2 = \Sigma \cup \{\sigma_1, \mathsf{E}((\sigma_1 \wedge \sim\Sigma')\mathscr{U}\sigma_2)\}$ and the value of this call is assigned to is_closed.

The recursive calls for Σ_1 and Σ_2 also construct either a proof or a model depending on the value of is_closed. When both calls report that is_closed is true, then P_1 contains a proof for the sequent Σ_1 and P_2 contains a proof for the sequent Σ_2, then Algorithm 4 (line 9) returns in P' a proof for Σ whose last step is the application of the rule $(\mathsf{E}\mathscr{U})^+$ to the sequents proved by P_1 and P_2. Consequently, when at least one of the two calls (for Σ_1 and Σ_2) reports that the tableau is not closed, the tableau for Σ is open and Algorithm 4 returns the model M'.

The application of α, β and β^+ rules does not construct models, they only load atoms in stages that will be collected and structured according to the application of the next-state rules when a cycle is detected. In lines 26-36 of Algorithm 3, one next-state rule is applied to an elementary set Σ with a number $k \geq 0$ of formulae $\mathsf{E}\bigcirc\sigma_i$. If $k = 0$ then $n := 1$ and, depending on the value of is_closed for the only recursive call, the algorithm constructs a proof for Σ from the returned proof P_1 of Σ_1 whose last step is the application of the rule ($\bigcirc$A). Otherwise, if $k > 0$, then $n := k$ and there is a recursive call for each AND-child of Σ, namely $\Sigma_1, \ldots, \Sigma_n$. If some of these calls for Σ_i returns in is_closed the value true, then it also returns in P_i a proof for Σ_i. Therefore it constructs in P' a proof for Σ that applies the rule ($\bigcirc$A). Otherwise, it collects a list of models $M_1, \ldots, M_n$, one for each of the $n \geq 1$ calls with respective parameters $\Sigma_1, \ldots, \Sigma_n$. Note that each call returns that is_closed is false. Then, this list of $n \geq 1$ models is used to compose a model of Σ whose root contains the atoms of Σ and this root has n children $M_1, \ldots, M_n$. Recall that the leaves of our models are either empty leaves (line 3) or pointers to some previous stage in the branch (line 12).

Example 7.1. Let Σ be the following set of formulae.

$$\{\, \mathsf{A}\Box(a \rightarrow \neg(b \vee c)),\ \mathsf{A}\Box(b \rightarrow \neg(a \vee c)),\ \mathsf{A}\Box(c \rightarrow \neg(b \vee a)),$$
$$\mathsf{E}\Diamond\mathsf{A}\Box a,\ \mathsf{E}\Diamond\mathsf{A}\Box b,\ \mathsf{E}\Diamond\mathsf{A}\Box c \,\}$$

Our prototype returns the following Kripke structure as a model of Σ that certifies its satisfiability.

```
State 0: {}
Subtree 1 of State 0
---State 1: {c} --> cycle to State 1
```

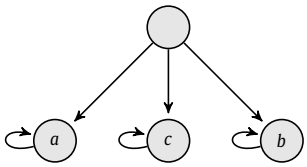


Fig. 12. Graphic representation of the Kripke structure returned by the prototype.

```
Subtree 2 of State 0
---State 1: {b} --> cycle to State 1
Subtree 3 of State 0
---State 1: {a} --> cycle to State 1
```

Fig. 12 shows a graphical representation of this model. ■

Algorithm 3 is the basis of our implementation, however, we have implemented some improvements mainly aimed at the efficiency and the usability in CMC. The formulae of the form $\mathsf{A}\Box\varphi$, that we call *global invariants*, are the most common in a model checking problem. In model checking, the formula that characterises a computation tree of a system is a (usually large) conjunction (collection) of global invariants. To provide shorter, clearer and readable proofs in CMC applications, we take advantage of the fact that, for each invariant $\mathsf{A}\Box\sigma$, either this formula itself or $\mathsf{A}\bigcirc\mathsf{A}\Box\sigma$ are in the node of the tableau, and both formulae are at every stage. Hence, proofs are easier to follow if we discharge all global invariants (and all $(\mathsf{A}\bigcirc\varphi)$ where φ is a global invariant). Indeed, the user could consult the system specification to follow the reasoning, though we specify in the proof when a rule is applied to an invariant formula (see sequent 2 in Example 7.2). Moreover, we do not load global invariants along the stages of the current branch.

Example 7.2. Consider the set of formulae $\Sigma = \{p, \mathsf{A}\Box(p \rightarrow \mathsf{A}\bigcirc p), \mathsf{E}\Diamond\neg p\}$. In the model checking framework the first formula could be seen as the initial state condition, the second formula as the model specification, and the third as the negation of the property $\mathsf{A}\Box p$, which the user is asking to check. Our prototype on this input Σ returns the expected result – that Σ is unsatisfiable (therefore, the property $\mathsf{A}\Box p$ holds in any model of $\{p, \mathsf{A}\Box(p \rightarrow \mathsf{A}\bigcirc p)\}$) and also gives the following 'detailed' proof as certificate:

```
0.E◇¬p, p. apply (E◇+)
-1.¬p, p. by (Ctd)
-2.E○(E(¬p𝒰¬p)), p. apply (A□) to Inv: A□(¬p ∨ A○p)
--3.¬p ∨ A○p, E○(E(¬p𝒰¬p)), p. apply (∨)
---4.E○(E(¬p𝒰¬p)), ¬p, p. by (Ctd)
---5.E○(E(¬p𝒰¬p)), A○p, p. apply (○E)
----6.E(¬p𝒰¬p), p. apply (E𝒰+)
-----7.¬p, p. by (Ctd)
-----8.E○(E(¬p𝒰¬p)), ¬p, p. by (Ctd)
```

It is worth to note that sequent 2 is derived by the application of $(\mathsf{E}\Diamond^{+})$ to sequent 0. In 0, the context of $\mathsf{E}\Diamond\neg p$ is just $\{p\}$. This context cannot be repeated from the next-state onwards until the eventuality would be fulfilled. Therefore, the contextualised variant $\mathsf{E}(\neg p\,\mathscr{U}\neg p)$ (of $\mathsf{E}\Diamond\neg p$) should be fulfilled from the next-state on. ■

Our prototype provides proofs with two grades of granularity: small-step proofs and big-step proofs. A small-step proof includes all rule applications (as the proof in Example 7.2). A big-step proof includes only the sequents before the application of a next-step rule, i.e. the elementary sets that have been refuted. In particular, the big-step version for Example 7.2 consists of just one elementary sequent:

```
---5. E○(E(¬p𝒰¬p)), A○p, p.
```

Big-step proofs are useful when proofs are very long and especially in the CMC, when the user wants just to see the evolution of the system that leads to contradiction. In the previous example, the unique sequent informs that the system evolves to satisfy $p, \mathsf{A}\bigcirc p$ while it should satisfy $\mathsf{E}\bigcirc\mathsf{E}(\neg p\,\mathscr{U}\neg p)$ (to satisfy the initial eventuality $\mathsf{E}\Diamond\neg p$), and that this leads to contradiction. Next, we show the big-step version of a larger proof.

Example 7.3. The following set of formulae is obtained by substituting in Example 7.1 the formula $\mathsf{E}\Diamond\mathsf{A}\Box a$ by $\mathsf{A}\Box\mathsf{E}\Diamond\mathsf{A}\Box a$, which makes it unsatisfiable

$$\{\, \mathsf{A}\Box(a \rightarrow \neg(b \vee c)),\ \mathsf{A}\Box(b \rightarrow \neg(a \vee c)),\ \mathsf{A}\Box(c \rightarrow \neg(b \vee a)),\ \mathsf{A}\Box\mathsf{E}\Diamond\mathsf{A}\Box a,\ \mathsf{E}\Diamond\mathsf{A}\Box b,\ \mathsf{E}\Diamond\mathsf{A}\Box c \,\}$$

The small-step proof provided by our prototype has about 350 sequents, however the big-step proof contains the following five sequents:

```
--76. E○E(F𝒰A□b), E○(E◇A□a), A○A□c, ¬b, ¬a, c
---145. E○E((A□E◇¬b)𝒰A□c), E○E◇A□a, A○A□b, ¬c, ¬a, b
--240. E○E((A□E◇¬b)𝒰A□c)), E○E◇A□b, E○E◇A□a, ¬c, ¬b, ¬a
----278. E○E((a ∨ b)𝒰A□a), A○A□c, ¬b, ¬a, c
-----345. E○E(F𝒰A□c), E○E◇A□a, A○A□E◇¬b, ¬c, ¬b, ¬a
```

Since all sequents of a big-step proof are elementary sets of formulae, to which the appropriated next-state rule is applied, we do not report that implicit information. In the proof above, (○E) is applied to the indicated five sequents because all these sequents have one or more E○ formulae. The two first lines (steps 76 and 145) show that constructing a path to fulfill A□b, while A□c is satisfied, is not possible. The last three lines (steps 240, 278 and 345) show that constructing a path to fulfill A□c is also impossible, because once A□c is fulfilled, A□a cannot be fulfilled. ■

We have implemented the sets of formulae by ordered sequences of formulae. For that we have defined an ad-hoc ordering on formulae (in nnf) to quickly detect a selected eventuality and to check if a sequence is elementary by just looking at the first element. In addition, we exploit the ordering to detect more quickly that a sequence is not a subsequence of another sequence, hence branches also keep ordered sequences of stages, in particular to detect cycles. We have also implemented some rules of subsumption (such as φ subsumes $\varphi \vee \psi$) in sequences and in the construction of the contextualised variants of eventualities –the latter is very important for the feasibility of our method. Of course, we have implemented the subsumption-like simplification rule (8) (See §4) by which any contextualised variant subsumes the original eventuality.

Our systematic tableau construction produces many repetitions of subtrees. We prevent to repeat the refutation (closed tableau) of any sequent that is a child of an elementary sequent. When one of them is refuted, it is loaded in a set called 'As Proved Above (APA)'. In the output proofs, we use the word APA to indicate that a sequent has been previously refuted. Moreover, we consider as refuted any sequent that is a superset of another sequent, already refuted. In other words, if a sequent $\Sigma \in$ APA and $\Sigma \subseteq \Sigma'$, then the sequent Σ' is immediately classified as refuted. Since loading all refuted sequents is really costly, we choose some sequents as candidates to be loaded in APA. This choice clearly determines the efficiency of the prototype. In the current prototype, we only add to APA the sequents produced after the application of the next-state rule. Further experimentation is needed to evaluate and compare the performance of the implementation with different selection criteria of the sequents-candidates for the APA.

Most of the code (written in Java) is automatically generated from the language Dafny [32]. Though Dafny is a program verifier of functional correctness, we note that our implementation is only partially verified, i.e. only some crucial properties have been verified by the time of writing this paper. For example, the ordering in formulae has been proved to be total. The methods that exploit this ordering to perform more efficient operations in sequences (e.g. insertion, deletion) have been proved to preserve the order. Other methods that decide properties of sequences more efficiently thanks to the order (e.g. elementarity check) have also been proved correct. The verification of the functional correctness of our prototype remains as an encouraging and challenging future work.

7.2. Experimental results

In order to assess the feasibility of our context-based tableau, we have tested the prototype *MomoCTL* on the collection of benchmarks, namely GBext, borrowed from http://users.cecs.anu.edu.au/~rpg/CTLComparisonBenchmarks/ that was created for the comparison of various CTL-provers made in [23]. In this section we report on our performance results and compare them with the other one-pass tableau for CTL ([1]) that is called *TreeTab* in [23]. It is worth noting that our current Java code has been automatically generated from Dafny, therefore it is not an optimised Java code. GBext is very well elaborated and gives a comprehensive, fair, and objective collection of CTL theorem-proving problems. We expect that an in-depth analysis of our results will allow us to identify what type of heuristics, strategies, etc. are convenient to improve *MomoCTL*.

In GBext there are three logically equivalent versions of each formula, which indicates that in [23], the comparison performed takes into account the syntactic form of the input. As expected, when translated to our input format –a set of formulae in nnf– the three versions give similar performance result. Therefore, we really use one version of each benchmark in GBext and this reduced set is called GB. We have automatically translated each unique formula in each file in GB to a file with the logically equivalent set of formulae in nnf. The obtained collection is denoted as MB. Both GB and MB collections are divided into different classes, some of which are further divided into subclasses for satisfiable and unsatisfiable instances or for different types of formula patterns. MB and *MomoCTL* are available at https://github.com/alexlesaka/MomoCTL. The executable (.JAR) runs full *MomoCTL* when it is called with a single file .ctl (i.e. it returns certificates). However, when that .JAR runs over a folder of benchmarks files, it runs the version of *MomoCTL* that returns just SAT/UNSAT values.

Both *TreeTab* and *MomoCTL* have been applied, respectively, to GB and MB. To make a fair comparison, we have run the version of *MomoCTL* that does not return certificates. We know that the delay between this simplified version and full *MomoCTL* is about 2%. These experiments were performed with an Intel Xeon Dual core 2.60 GHz CPU with 64 GB RAM computer. As the authors of [23] did, we imposed a stack limit of 512 MB and a 1000 seconds time limitation for each problem instance.

In the rest of this section, we compare *MomoCTL* and *TreeTab* performance results on the most significant classes and subclasses of MB and GB, which are: *Alternating Bit Protocol (abp), Business Processes (busproc), Exponential Formulae (exp_sat and exp_unsat), Montali's Formulae (montali_sat and montali_usat), Pattern_AE, and Reskill.*

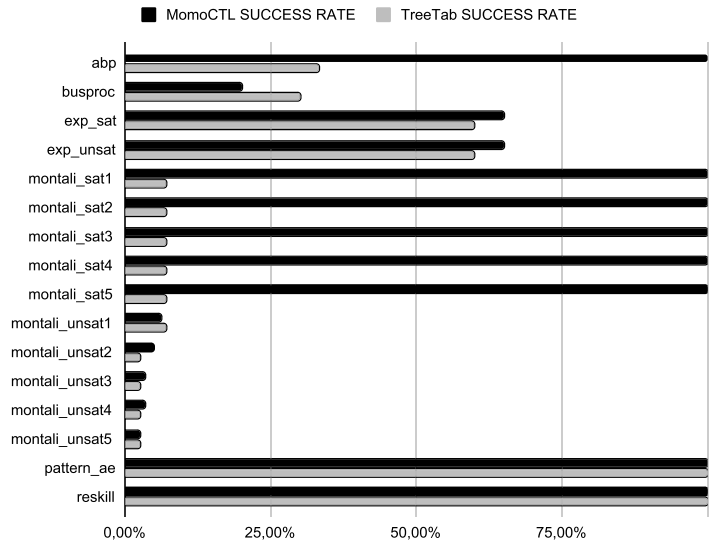


Fig. 13. Percentage of solved instances (by class) within time limit.

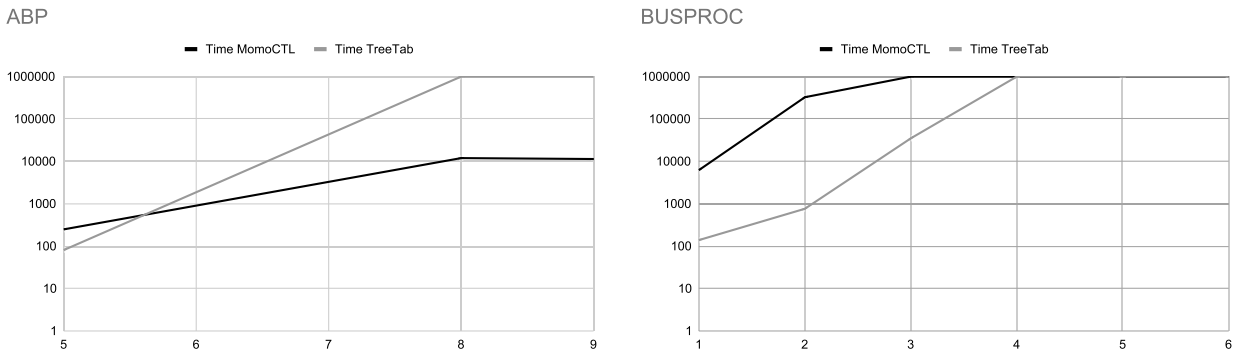


Fig. 14. abp and busproc formulae.

In Fig. 13, we show the percentage of files in each class that are solved within the time limit, i.e. those that return a value –satisfiable or unsatisfiable– and do not produce timeout or some error –e.g. lack of memory. We use gray colour for *TreeTab* and black colour for *MomoCTL*. Instances are ordered by the increasing complexity. As we will detail below, for *MomoCTL* it takes longer than for *TreeTab* to solve smaller instances, but the former solves (in 1000 s.) greater instances –in *classes abp, exp_sat, exp_unsat, montali_sat*– that *TreeTab* cannot solve (in 1000 s.). *TreeTab* solves more instances than *MomoCTL* in *busproc and montali_unsat1*. Next, for each class, we compare the running times of both CTL-provers and analyse possible causes of their differences. In the plots, the abscissa is the number of different variable symbols in the input and the ordinate is the running time in milliseconds. The plots are given using a logarithmic scale (the same scale used in [23]).

1. Alternating Bit Protocol (abp). This class has three instances that encode whether three different properties are valid for a protocol specification (see [23]). All instances are unsatisfiable and the problem is quite difficult for CTL-provers based on tableaux. Tableaux methods necessarily should produce a huge number of closed branches. In the simplest problem (*abp5*), each branch contains four eventualities and a releases-formula that cannot be fulfilled in any order, therefore 120 combinations should be tested. The other two instances (*abp8* and *abp9*) add to *abp5* one extra eventuality with the releases-formula included in it, which increases the combinations up to 720. As shown in Fig. 14, only the simplest instance (*abp5*) is solved by *TreeTab* in 0.081 s. while *MomoCTL* is able to solve the three instances in 0.249 s., 11.952 s., and 13.370 s. respectively.

2. Business Processes (busproc). This class arises from a synthesis problem ([4]). All the instances have a huge amount of different models that exponentially increases with the input size. For tableau methods, to find just one model it suffices to decide the satisfiability. Consequently, (as showed in [23]) tableau methods perform better than other approaches for satisfiable inputs. However, CTL models of *busproc* problems are non-linear Kripke-structures. Fig. 14 compares *TreeTab* and *MomoCTL*. *TreeTab* solves the three first instances –*busproc1* in 0.139 s., *busproc2* in 0.762 ms. and *busproc3* in 34.991 s. *MomoCTL* only solves *busproc1* and *busproc2* in 6,226 s. and 326,316 s. respectively. In the case of *busproc1*, (the full

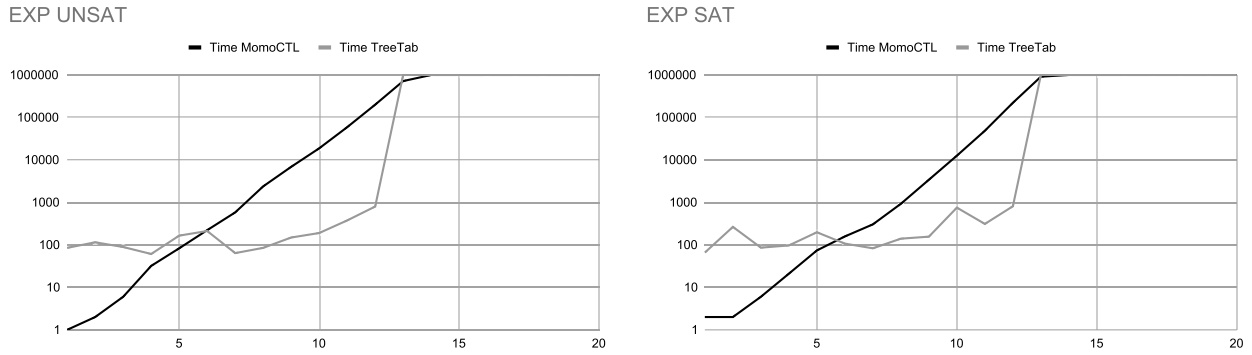


Fig. 15. Exponential satisfiable and unsatisfiable formulae.

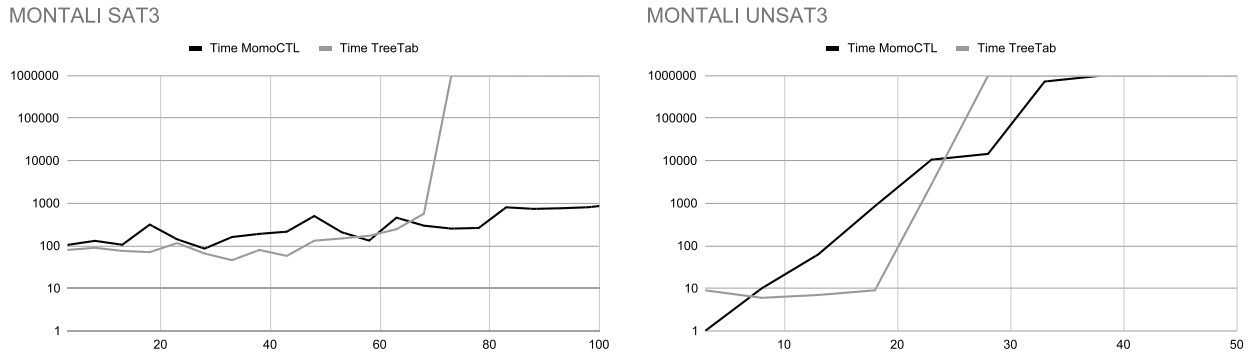


Fig. 16. Montali's satisfiable and unsatisfiable formulae with depth 3.

version) returns a model of depth five with four branches. For this construction, *MomoCTL* collected about 400 refuted sequents, hence in the tableau there are many closed branches that are constructed before the first model is found.

Analysing these refuted sequents (for *busproc1* and other benchmarks with many models), we saw that most of them are attempts to fulfill an eventuality at some state where it can not be fulfilled, although there are many possibilities to explore to get to decide it. We conclude that *MomoCTL* strategy of forcing eventualities to be satisfied "as soon as possible" does not work very well with satisfiable problems with a big amount of models. Indeed, contexts could produce a big explosion of tableau OR-branches when the selected eventuality is delayed to the next step. We think that *MomoCTL* should be equipped with some heuristics for eventuality selection. The class *busproc* is a useful collection of problems for our further work.

3. Exponential Formulae. Both satisfiable and unsatisfiable exponential formulae have a similar difficulty for *TreeTab* and *MomoCTL*. The subclass of satisfiable instances has models with exponentially larger paths. The tableau for unsatisfiable instances builds an exponentially increasing number of non-repeated tableau branches that are closed.

In Fig. 15 we can see that *MomoCTL* has an exponential growth of the running time from small instances in both subclasses, while *TreeTab* does not. Nevertheless *TreeTab* runs out of memory from size 12, whereas *MomoCTL* is able to solve the instance of size 13 inside the time limit. Allowing more time, *MomoCTL* solves the unsatisfiable instance of size 14 in 1,551 s. This is thanks to the strategy of not repeating refutations that have been made previously. To solve the satisfiable instance of size 14, *MomoCTL* needs 6,000 s. because the model of this formula is extraordinarily large. Therefore, *MomoCTL* solves more difficult instances as the time increases. This is one of the main features of our prototype which makes it different from the *TreeTab*.

4. Montali's Formulae. These formulae are CTL-reformulations of LTL-specifications of business processes ([31]) that has been used for comparing LTL-provers ([20]). Montali's formulae are parameterised by n and m.

$$\varphi_1^i = \mathsf{A}\Diamond p_i \qquad \varphi_m^i = \mathsf{A}\Diamond(p_i \wedge \mathsf{A}\bigcirc\varphi_{m-1}^i) \qquad \varphi_n = \wedge_{i=0}^{n-1}\mathsf{A}\Box(\neg p_i \vee \mathsf{A}\bigcirc\mathsf{A}\Diamond p_{i+1})$$

Satisfiable instances are of the form $\varphi_m^0 \wedge \varphi_n$, whereas unsatisfiable instances have the form $\varphi_m^0 \wedge \varphi_n \wedge \neg\varphi_m^n$.

Fig. 16 shows the results for satisfiable and unsatisfiable subclasses where $m = 3$ and n is in the abscissa. For $m = 1, 2, 4, 5$ the plots are very similar. In the case of satisfiable formulae, *MomoCTL* finds a model in the first (leftmost) branch when doing the depth-first search.

Regarding unsatisfiable formulae, *MomoCTL* is able to respond more (until size 33) but it requires more time than *TreeTab*, which runs out of memory at size 23. Note the similarity of the plots for *montali_unsat* and exponential formulae. As we mentioned above, *MomoCTL* can solve larger and larger instances as time increases. The main reason is that the

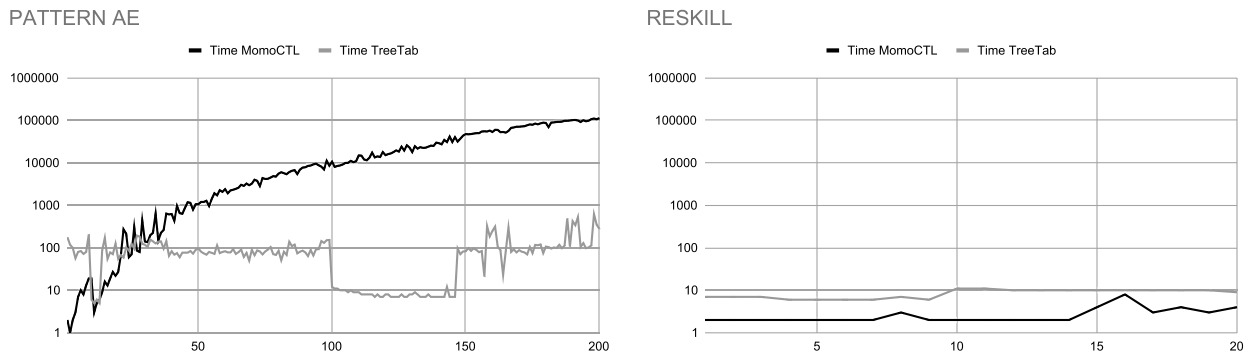


Fig. 17. Pattern_AE and Reskill formulae.

systematic forcing of eventualities takes more advantage as the number of eventualities grows – they produce always-formulae that reduce the search space.

5. Pattern_AE and Reskill. In [23], the authors introduce these two classes and show that methods based on BDDs and resolution perform badly on inputs containing many conflicting $\mathsf{E}\Box$ temporal formulae (in the case of *pattern_ae*) and when there are many potential resolution-steps in a satisfiable formula (in the case of *reskill*). The class *pattern_ae* contains formulae of the form

$$(\bigwedge_{i=0}^{n} \mathsf{A}\Box\mathsf{E}\Box p_i) \vee (\bigwedge_{i=0}^{n} \mathsf{A}\Box\mathsf{E}\Box\neg p_i).$$

while the class *reskill* contains these kinds of formulae

$$\neg p \wedge (\neg p \vee (p \wedge \mathsf{A}\Box(\bigwedge_{i=0}^{n-1} \mathsf{A}\Diamond q_i) \wedge \bigwedge_{i=0}^{n-1} \mathsf{A}\Box(\neg q_i \vee \bigvee_{j=0, j\neq i}^{n-1} \mathsf{E}\bigcirc q_j) \wedge \bigwedge_{i=0}^{n-1} \mathsf{A}\Box(\neg q_i \vee \bigvee_{j=0, j\neq i}^{n-1} \neg q_j)))$$

The instances of both classes are satisfiable. Fig. 17 shows that both –*TreeTab* and *MomoCTL*– run very fast on the two classes of formulae. In *pattern_ae*, *MomoCTL* is faster than *TreeTab* in small instances until $n = 25$. *MomoCTL* performs with a moderate increase of time, while *TreeTab* is more constant with some insignificant increase or decrease of the number of steps. The model returned by *MomoCTL* to pattern_ae of size n is:

```
State 0: {p_0,...,p_{n-1}} --> cycle to State 0
```

Both *MomoCTL* and *TreeTab* perform extremely fast on the class *reskill* because they are able to find the simplest model, which is returned by *MomoCTL* as follows:

```
State 0: {}
---State 1:{} --> cycle to State 1
```

Finally, we again remark that *MomoCTL* does not run out of memory on any benchmark. Indeed, with unlimited execution time, *MomoCTL* solves some benchmarks, on which *TreeTab* runs out of memory. We believe that these errors of *TreeTab* are due to the fact that the branches it produces are too large. We know that the way how we use contexts to handle eventualities prevents the generation of such extra large branches. However, the price we pay is the increased number of the shorter branches to analyse. This makes *MomoCTL* run slower than *TreeTab* for some not very complex problems. We are convinced that a more intelligent eventuality selection procedure, in addition to different heuristic strategies could solve this problem and improve notably MomoCTL's performance.

8. Extending the tableau method and the sequent calculus to ECTL

In this section we explain a (relatively easy) way to extend the CTL tableau method and its dual sequent calculus to the more expressive logic ECTL. This is achieved by adding the new tableau rules given in Fig. 20 and their dual sequents rules in Fig. 21. These rules correspond to the following logical equivalences for the basic modalities that extend CTL to ECTL:

$$\begin{array}{ll} \mathsf{E}\Box\Diamond\sigma \equiv \mathsf{E}\Diamond\sigma \wedge \mathsf{E}\bigcirc\mathsf{E}\Box\Diamond\sigma & \mathsf{E}\Diamond\Box\sigma \equiv \mathsf{E}\Box\sigma \vee (\mathsf{E}\Diamond\sigma \wedge \mathsf{E}\bigcirc\mathsf{E}\Diamond\Box\sigma) \\ \mathsf{A}\Box\Diamond\sigma \equiv \mathsf{A}\Diamond\sigma \wedge \mathsf{A}\bigcirc\mathsf{A}\Box\Diamond\sigma & \mathsf{A}\Diamond\Box\sigma \equiv \mathsf{A}\Box\sigma \vee (\mathsf{A}\Diamond\sigma \wedge \mathsf{A}\bigcirc\mathsf{A}\Diamond\Box\sigma) \end{array} \quad (9)$$

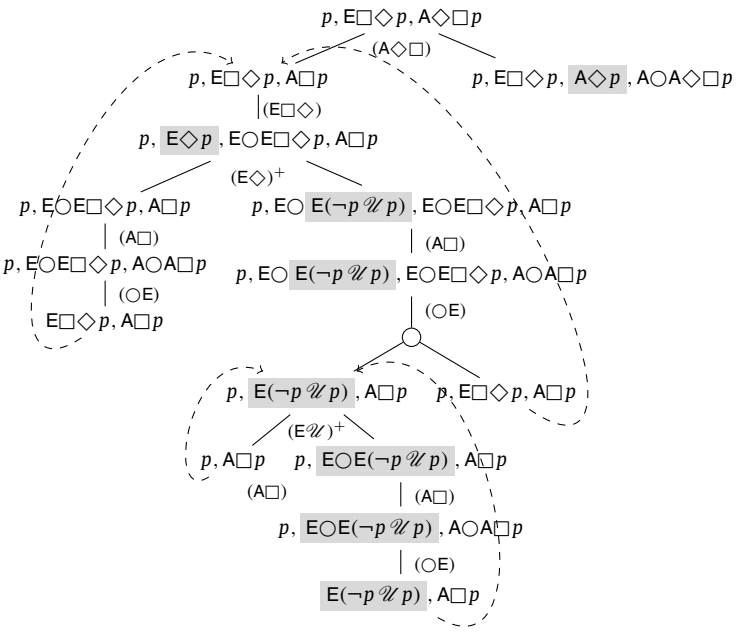


Fig. 18. ECTL systematic tableau for $\{p, \mathsf{E}\Box\Diamond p, \mathsf{A}\Diamond\Box p\}$.

Fig. 19. Graphic representation of the model for $\{p, \mathsf{E}\Box\Diamond p, \mathsf{A}\Diamond\Box p\}$.

$$(\mathsf{Q}\Box\Diamond)\ \frac{\Sigma, \mathsf{Q}\Box\Diamond\sigma}{\Sigma, \mathsf{Q}\Diamond\sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Box\Diamond\sigma} \qquad (\mathsf{Q}\Diamond\Box)\ \frac{\Sigma, \mathsf{Q}\Diamond\Box\sigma}{\Sigma, \mathsf{Q}\Box\sigma \mid \Sigma, \mathsf{Q}\Diamond\sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Diamond\Box\sigma}$$

Fig. 20. Additional tableau rules for ECTL.

The rule $(\mathsf{Q}\Box\Diamond)$ is added to the set of α-rules and the rule $(\mathsf{Q}\Diamond\Box)$ is added to the set of β-rules. There are no additional β^+-rules. Indeed the eventualities $\mathsf{Q}\Diamond\sigma$ introduced by the rules in Fig. 20 are CTL-modalities that are handled by the β^+-rules of the method for CTL.

Next, we illustrate that issue with an example of systematic tableau for an ECTL input where additional rules for ECTL are applied, along with CTL rules.

Example 8.1. Fig. 18 partially shows an open tableau with the application of two of the rules added to extend CTL to ECTL. We just outline some branches to illustrate how the ECTL-rules $(\mathsf{A}\Diamond\Box)$, and $(\mathsf{E}\Box\Diamond)$ (in Fig. 20) are applied. The tableau for the right-most node, which is also open, is not depicted for space reasons. Note that our algorithm constructs just the left-most branch to decide that the label of the root is satisfiable and returns a very simple model depicted in Fig. 19, which is a cyclic Kripke structure with the single fullpath $\{p\}^w$. ■

We extend soundness and completeness results (for CTL) to ECTL. Firstly, we extend Lemma 5.1 to the rules in Fig. 20.

Lemma 8.2. *For any ECTL set of state formulae Σ and any state formula σ:*

1. $\mathsf{Sat}(\Sigma \cup \{\mathsf{Q}\Box\Diamond\sigma\})$ *if and only if* $\mathsf{Sat}(\Sigma \cup \{\mathsf{Q}\Diamond\sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Box\Diamond\sigma\})$.
2. $\mathsf{Sat}(\Sigma \cup \{\mathsf{Q}\Diamond\Box\sigma\})$ *if and only if* $\mathsf{Sat}(\Sigma \cup \{\mathsf{Q}\Box\sigma\})$ *or* $\mathsf{Sat}(\Sigma \cup \{\mathsf{Q}\Diamond\sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Diamond\Box\sigma\})$.

Proof. It follows by 'systematic' application of the semantic definitions of the modalities $\mathsf{Q}\Box\Diamond$ and $\mathsf{Q}\Diamond\Box$ given by the equivalences (9) in §8. ■

$$(\mathsf{Q}\Box\Diamond)\ \frac{\Sigma, \mathsf{Q}\Diamond\sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Box\Diamond\sigma \vdash \mathbf{F}}{\Sigma, \mathsf{Q}\Box\Diamond\sigma \vdash \mathbf{F}}$$

$$(\mathsf{Q}\Diamond\Box)\ \frac{\Sigma, \mathsf{Q}\Box\sigma \vdash \mathbf{F} \quad \Sigma, \mathsf{Q}\Diamond\sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Diamond\Box\sigma \vdash \mathbf{F}}{\Sigma, \mathsf{Q}\Diamond\Box\sigma \vdash \mathbf{F}}$$

Fig. 21. Additional sequent rules for ECTL.

As a consequence of Lemma 8.2, the soundness Theorem 5.2 easily extends to ECTL. For refutational completeness of ECTL, we firstly extend Definition 5.3 with the following additional conditions for a stage to be $\alpha\beta^{+}$-saturated:

Definition 8.3. We say that a stage $s = n_i, \ldots, n_j$ in the ECTL systematic tableau for Σ is $\alpha\beta^{+}$-saturated if and only if it satisfies the conditions in Definition 5.3 and the following two additional conditions:

1. For all $\mathsf{Q}\Box\Diamond\sigma \in \tau(s)$: $\{\mathsf{Q}\Diamond\sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Box\Diamond\sigma\} \subseteq \tau(s)$.
2. For all $\mathsf{Q}\Diamond\Box\sigma \in \tau(s)$: $\{\mathsf{Q}\Box\sigma\} \subseteq \tau(s)$ or $\{\mathsf{Q}\Diamond\sigma, \mathsf{Q}\bigcirc\mathsf{Q}\Diamond\Box\sigma\} \subseteq \tau(s)$. ■

It is obvious that the conditions 7. and 8. are satisfied in any stage of the systematic tableau by construction. Using these conditions, it is routine to prove that $\mathscr{K}_H$, as defined in Lemma 5.6, satisfies the fact that: $\mathscr{K}_H, s_a, 0 \models \sigma$ for any $a \in \{0, \ldots, j\}$ and any formula of the form $\mathsf{Q}\Box\Diamond\sigma, \mathsf{Q}\Diamond\Box\sigma$ that belongs to $L(s_a)$. Therefore, refutational completeness (i.e. Theorem 5.7) extends to ECTL. Finally, for the termination result (see the proof in Theorem 5.8), it suffices to ensure that the rules in Fig. 20 do not affect the behaviour of the β^{+}-rules on the selected eventualities in the sense that Proposition 5.5 is preserved. Furthermore, although each application of a rule in Fig. 20 introduces a new $\mathsf{Q}\Diamond\sigma$, the simplification rule $(\sqsubset\mathsf{Q}\mathscr{W})$ (see (8) in Section 8) ensures that any occurrence of an eventuality is subsumed by its contextualised variant. Therefore, Proposition 5.5 holds and hence Theorem 5.8 extends to ECTL.

The calculus $\mathscr{G}_{\vdash}$ is also extended by adding the sequent rules that are dual to the tableau rules in Fig. 20. These additional sequent rules are given in Fig. 21 and the duality of the extended tableau and sequent calculus for ECTL is trivial.

9. Conclusion

We introduced sound and complete systems of tableaux and sequent calculus for temporal logics CTL and ECTL, illustrating the methods on a number of examples, and reporting on their implementation *MomoCTL* and experimental results. *MomoCTL* is available at https://github.com/alexlesaka/MomoCTL.

The distinctive feature of the tableau methods presented in the paper, is that the core tableau construction is based on the concept of a context of an eventuality. The method developed in the paper is much simpler than the analogous technique obtained earlier (see [7]) for a richer logic - ECTL$^{\#}$ where two types of contexts (both outer and inner contexts) are used. Indeed, the construction presented in this paper only uses the "outer" context, however, similar to ECTL$^{\#}$, generates tableaux as AND-OR trees. Our context-based approach to deduction systems for branching-time temporal logics preserves the classical duality of tableaux and sequent calculus, enabling a framework where both certifying proofs and models can be constructed as byproducts of the satisfiability test. This makes our approach specifically important in the area of CMC.

The methods presented in this paper have double-exponential time worst case complexity. Indeed, a trivial adaptation of [19] allows us to say that the so-called *closure* –the set of all formulae that could appear in a tableau– has in the worst case size $O(2^{O(2^n)})$, where n is the size of the input (this complexity characterisation matches the one of [1,22]). However, in practice the worst case is very unusual. For example, when the context of an eventuality mostly contains modalities A$\Box$ –which is typical in reactive systems specifications and model checking practical problems– the number of possible contexts is much smaller and consequently the overall performance of the technique developed is much better.

Our experimentation with *MomoCTL* on a large and interesting suite of benchmarks (http://users.cecs.anu.edu.au/~rpg/CTLComparisonBenchmarks/) has given encouraging results and many ideas for future improvements. We plan to perform more experimentation and efficiency improvements, before extending the prototype to cover the methods for logics ECTL, ECTL^{+}, and ECTL$^{\#}$ [7].

Future work also includes the mechanical check of proof certificates extending the ideas in [2], the graphical representation of models, and the correspondence of the one-side sequent calculus with a two-side sequent calculus (in line with what is done in [19] for the linear temporal logic). The formal verification of the complete functional correctness of the presented algorithms in Dafny is also in our plans.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have included a link to share the application developed in this work: https://github.com/alexlesaka/MomoCTL

References

[1] P. Abate, R. Goré, F. Widmann, One-pass tableaux for computation tree logic, in: N. Dershowitz, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, Springer, Berlin, Heidelberg, 2007, pp. 32–46.
[2] A. Abuin, A. Bolotov, U. Díaz-de-Cerio, M. Hermo, P. Lucio, Towards certified model checking for PLTL using one-pass tableaux, in: Johann Gamper, Sophie Pinchinat, Guido Sciavicco (Eds.), 26th International Symposium on Temporal Representation and Reasoning, TIME 2019, October 16-19, 2019, Málaga, Spain, in: LIPIcs, vol. 147, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 12:1–12:18.
[3] A. Abuin, A. Bolotov, M. Hermo, P. Lucio, One-pass context-based tableaux systems for CTL and ECTL, in: E. Muñoz-Velasco, A. Ozaki, M. Theobald (Eds.), 27th International Symposium on Temporal Representation and Reasoning, TIME 2020, September 23-25, 2020, Bozen-Bolzano, Italy, in: LIPIcs, vol. 178, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 14:1–14:20.
[4] A. Awad, R. Goré, J. Thomson, M. Weidlich, An iterative approach for business process template synthesis from compliance rules, in: Proceedings of the 23rd International Conference on Advanced Information Systems Engineering, CAiSE'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 406–421.
[5] C. Baier, J.-P. Katoen, Principles of Model Checking, The MIT Press, 2008.
[6] M. Ben-Ari, A. Pnueli, Z. Manna, The temporal logic of branching time, Acta Inform. 20 (3) (September 1983) 207–226.
[7] A. Bolotov, M. Hermo, P. Lucio, Branching-time logic ECTL# and its tree-style one-pass tableau: extending fairness expressibility of ECTL+, Theor. Comput. Sci. 813 (2020) 428–451.
[8] M.C. Browne, E. Clarke, O. Grümberg, Characterizing finite Kripke structures in propositional temporal logic, Theor. Comput. Sci. 59 (1–2) (jul 1988) 115–131.
[9] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, Symbolic model checking: 1020 states and beyond, Inf. Comput. 98 (2) (1992) 142–170.
[10] E.M. Clarke, E.A. Emerson, Using branching time temporal logic to synthesise synchronisation skeletons, Sci. Comput. Program. 2 (1982) 241–266.
[11] E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Trans. Program. Lang. Syst. 8 (2) (1986) 244–263.
[12] E.M. Clarke, O. Grumberg, D.A. Peled, Model Checking, MIT Press, London, Cambridge, 1999.
[13] E.A. Emerson, Temporal and modal logic, in: Jan van Leeuwen (Ed.), Handbook of Theoretical Computer Science (vol. B), MIT Press, Cambridge, USA, 1990, pp. 995–1072.
[14] E.A. Emerson, J.Y. Halpern, Decision procedures and expressiveness in the temporal logic of branching time, J. Comput. Syst. Sci. 30 (1) (1985) 1–24.
[15] E.A. Emerson, J.Y. Halpern, Sometimes and not never revisited: on branching versus linear time temporal logic, J. ACM 33 (1) (1986) 151–178.
[16] M. Fitting, Tableau methods of proof for modal logics, Notre Dame J. Form. Log. 13 (2) (1972) 237–247.
[17] D.M. Gabbay, Expressive functional completeness in tense logic (preliminary report), in: Aspects of Philosophical Logic, Dordrecht, 1981, pp. 91–117, Reidel.
[18] D.M. Gabbay, The declarative past and imperative future: executable temporal logic for interactive systems, in: Behnam Banieqbal, Howard Barringer, Amir Pnueli (Eds.), Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings, in: Lecture Notes in Computer Science, vol. 398, Springer, 1989, pp. 409–448.
[19] J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, F. Orejas, Dual systems of tableaux and sequents for PLTL, J. Log. Algebraic Program. 78 (8) (2009) 701–722.
[20] V. Goranko, A. Kyrilov, D. Shkatov, Tableau tool for testing satisfiability in ltl: implementation and experimental analysis, in: Proceedings of the 6th Workshop on Methods for Modalities (M4M-6 2009), in: Electronic Notes in Theoretical Computer Science, vol. 262, 2010, pp. 113–125.
[21] V. Goranko, A. Zanardo, From linear to branching-time temporal logics: transfer of semantics and definability, Log. J. IGPL 15 (1) (2007) 53–76.
[22] R. Goré, And-or tableaux for fixpoint logics with converse: LTL, CTL, PDL and CPDL, in: Stéphane Demri, Deepak Kapur, Christoph Weidenbach (Eds.), Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014, Proceedings, in: Lecture Notes in Computer Science, vol. 8562, Springer, 2014, pp. 26–45.
[23] R. Goré, J. Thomson, F. Widmann, An experimental comparison of theorem provers for CTL, in: C. Combi, M. Leucker, F. Wolter (Eds.), Eighteenth International Symposium on Temporal Representation and Reasoning, TIME 2011, Lübeck, Germany, September 12-14, 2011, IEEE, 2011, pp. 49–56.
[24] R. Goré, Tableau methods for modal and temporal logics, in: Marcello D'Agostino, Dov M. Gabbay, Reiner Hähnle, Joachim Posegga (Eds.), Handbook of Tableau Methods, Springer, Netherlands, Dordrecht, 1999, pp. 297–396.
[25] A. Griggio, M. Roveri, S. Tonetta, Certifying proofs for ltl model checking, in: 2018 Formal Methods in Computer Aided Design (FMCAD), 2018, pp. 1–9.
[26] R. Kashima, An axiomatization of ECTL, J. Log. Comput. 24 (1) (2014) 117–133.
[27] R. Kowalski, AND/OR graphs, theorem-proving graphs and bi-directional search, in: Machine Intelligence 7, 1972, pp. 167–194.
[28] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems - Specification, Springer, 1992.
[29] N. Markey, Temporal logics, in: Course Notes, Master Parisien de Recherche en Informatique, Paris, France, 2013.
[30] A. Mebsout, C. Tinelli, Proof certificates for SMT-based model checkers for infinite-state systems, in: Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design, FMCAD '16, 2016, pp. 117–124.
[31] M. Montali, P. Torroni, M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, Verification from Declarative Specifications Using Logic Programming, ICLP, vol. '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 440–454.
[32] K. Rustan, M. Leino. Dafny, An automatic program verifier for functional correctness, in: Edmund M. Clarke, Andrei Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, in: Lecture Notes in Computer Science, vol. 6355, Springer, April 2010, pp. 348–370.
[33] S. Schwendimann, A new one-pass tableau calculus for PLTL, in: Harrie C.M. de Swart (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX '98, Oisterwijk, the Netherlands, May 5-8, 1998, Proceedings, in: Lecture Notes in Computer Science, vol. 1397, Springer, 1998, pp. 277–292.
[34] A.P. Sistla, E.M. Clarke, The complexity of propositional linear temporal logics, J. ACM 32 (3) (1985) 733–749.
[35] R.M. Smullyan, First-Order Logic, Springer-Verlag, Berlin, Germany, New York, 1968.
[36] P. Wolper, The tableau method for temporal logic: an overview, Log. Anal. 28 (110–111) (1985) 119–136.