Journal of Computer and System Sciences 78 (2012) 981–996

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Once and *for all* ☆

Orna Kupferman [a,*], Amir Pnueli [b,1], Moshe Y. Vardi [c]

[a] *Hebrew University, Israel*
[b] *Weizmann Institute, Israel*
[c] *Rice University, United States*

ARTICLE INFO

Article history:
Received 3 June 2010
Received in revised form 29 May 2011
Accepted 5 August 2011
Available online 17 August 2011

Keywords:
Temporal logic
Past-time operators
Expressive power
Decision procedures
Alternating automata

ABSTRACT

It has long been known that past-time operators add no expressive power to linear temporal logics. In this paper, we consider the extension of *branching temporal logics* with *past-time operators*. Two possible views regarding the nature of past in a branching-time model induce two different such extensions. In the first view, past is branching and each moment in time may have several possible futures and several possible pasts. In the second view, past is linear and each moment in time may have several possible futures and a unique past. Both views assume that past is finite. We discuss the practice of these extensions as specification languages, characterize their expressive power, and examine the complexity of their model-checking and satisfiability problems.

1. Introduction

Temporal logics, which are modal logics that enable the description of occurrence of events in time, serve as a classical tool for specifying behaviors of concurrent programs [1]. The appropriateness of temporal logics for algorithmic verification follows from the fact that finite-state programs can be modeled by finite *propositional Kripke structures*, whose properties can be specified using propositional temporal logic. This yields fully-algorithmic methods for synthesis and for reasoning about the correctness of programs. These methods consider two types of temporal logics: linear and branching [2]. In *linear temporal logics*, each moment in time has a unique possible future, while in *branching temporal logics*, each moment in time may split into several possible futures. Thus, if we model a program by a Kripke structure, then linear temporal logic formulas are interpreted over *paths* in the Kripke structure (and thus refer to a single computation of the program), while branching temporal logic formulas are interpreted over *states* in the Kripke structure (and thus refer to all the computations of the program).

Striving for maximality and correspondence to natural languages, philosophers developed temporal logics that provide temporal connectives that refer to both past and future [3,4]. Striving for minimality, computer scientists usually use temporal logics that provide only future-time connectives. Since program computations have a definite starting time and since, in this case, past-time connectives add no expressive power to linear temporal logics [5], this seems practical. Nevertheless, enriching linear temporal logics with past-time connectives makes the formulation of specifications more intuitive [6]. Furthermore, it is known that it also makes the specifications shorter: the logic LTL_p, which augments LTL with past-time

☆ The paper is based on Kupferman and Pnueli (1995) [10] and includes the solution to problems that were left open there; namely, model checking of CTL^*_{lp}, satisfiability of CTL^*_{lp}, and satisfiability of CTL^*_{bp}.

* Corresponding author.
E-mail address: ornak@cs.huji.ac.il (O. Kupferman).

[1] Amir Pnueli passed away on November 2, 2009.

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2011.08.006

connectives is exponentially more succinct than LTL [7]. At the same time, adding past connectives does not increase the complexity of the validity and the model-checking problems [6,8].

Adding past time constructs is also natural in branching temporal logics. The question we raise here is what the effect is of adding such connectives to branching temporal logics on the expressiveness and computational complexity of the logics. Examining this question, we found in the literature several logics that extend branching temporal logics with past-time connectives; see review in [9]. Yet, until the publication of [10] there was no systematic study of the past in branching temporal logics.[2]

We distinguished in [10] between two possible views regarding the nature of past. In the first view, *past is branching* and each moment in time may have several possible futures and several possible pasts. In the second view, *past is linear* and each moment in time may have several possible futures and a unique past. Both views assume that *past is finite*. Namely, they consider only paths that start at a definite starting time, since a computation always has a starting point. (For a different view of this point, see [9].)

Before going on with our two views, let us consider the branching temporal logics with past that we found in the literature. The original version of *propositional dynamic logic* (PDL), as presented by Pratt in [11], includes a *converse construction*. The converse construction reverses the program, thus enabling the specifications to refer to the past. As each state in a program may have several predecessors, *converse-PDL* corresponds to the branching-past interpretation. Beyond our aspiration to replace the PDL system with branching temporal logics used nowadays, our main complaint about the converse construction is that it allows infinite paths in the reversed programs. Thus, it does not reflect the (helpful, as we shall show) fact that programs have a definite starting time. As a result, combining the converse construction with other constructions, e.g. the *loop* construction and the *repeat* construction, results in quite complicated logics [12–14]: they do not satisfy the *finite model property*, their decidability becomes more expensive, and no model-checking procedures are presented for them. In addition, while *converse*-DPDL satisfies the *tree model property* [14], the logics we introduce for the branching-past interpretation do not satisfy it. So, intuitively, our branching past is "more branching". In spite of this, our logics satisfy the finite model property. The same tolerance towards paths that backtrack the transition relation without ever reaching an initial state is found in *POTL*, which augments the branching temporal logic $B(X, F, G)$ with past-time connectives [15], and in the reverse operators in [16].

A logic *PCTL**, which augments the branching temporal logic CTL* with past-time connectives, is introduced in [17]. Formulas of PCTL* are interpreted over *computation trees*. Thus, PCTL* corresponds to the linear-past interpretation. Nevertheless, quantification over the past in the semantics of PCTL* is very weak, making the past in PCTL* very weak. For example, the PCTL* formula $EXEY\mathit{true}$ ("exists a successor state for which there exists a predecessor state that satisfies *true*") is not satisfiable. It is not surprising then, that PCTL* is not more expressive than CTL* (a similar limited past is discussed in [18]). Another augmentation of CTL* with past-time connectives is the *Ockhamist computational logic* (OCL), presented in [19]. We found the semantics of OCL unsatisfactory, as it is interpreted over structures that are not fusion closed. (Fusion closure is a basic property of state-transition structures; see [20].)

In this paper, we consider the logics $\mathrm{CTL}^\star_{bp}$ and $\mathrm{CTL}^\star_{lp}$, as well as their sub-languages CTL_{bp} and CTL_{lp}. Syntactically, $\mathrm{CTL}^\star_{bp}$ and $\mathrm{CTL}^\star_{lp}$ are exactly the same: both extend the full branching temporal logic CTL* with past-time connectives. Semantically, we have two completely different interpretations. Formulas of $\mathrm{CTL}^\star_{bp}$ are interpreted over states of a Kripke structure with a definite initial state. Since each state in a Kripke structure may have several successors and predecessors, this interpretation induces a "branching reference" to both future and past. There are two ways to think of such branching structures. In [11], the branching reflects nondeterminism, and the structures are simply state-transition graphs, with $\mathrm{CTL}^\star_{bp}$ formulas being able to quantify both about forward paths and backward paths. For example, in this view, the CTL_{bp} formula $AG(\mathit{grant} \rightarrow EP(\mathit{req}))$ specifies that whenever the system is in a configuration in which a grant is given, there is a possibility to reach this configuration along a computation in which a request was issued (the temporal operator P stands for "past" – the dual of future-time operator "eventually"). Note that a configuration in which a grant is given may be reachable by several computation. The specification does not require a request to be issued along all the computations that lead to the configuration, and only states that at least one such computation exists. In [15], a different view for branching past is suggested. There, the structure describes one computation, with processes that can fork and join. The initial state corresponds to a single initial process, a state with several successors corresponds to creating new processes, and a state with several predecessors corresponds to merging processes. Unlike [15], which allows infinite paths going back in time, we consider only paths that start in the initial state and thus ignore computations which can be traversed backwards an infinite number of steps. For example, in this view, the CTL_{bp} formula $AG(\mathit{term} \rightarrow ((EP\mathit{term}_1) \wedge \cdots \wedge (EP\mathit{term}_n)))$ states that the entire system can terminate only after processes $1, \ldots, n$ have terminated. Indeed, the formula states that if the system has a configuration in which it terminates, then all processes have terminated within the sequence of forks and joins that has led to this configuration.

Formulas of $\mathrm{CTL}^\star_{lp}$, on the other hand, are interpreted over nodes of a computation tree obtained by, say, unwinding a Kripke structure. Since each node in a computation tree may have several successors but only one predecessor (except the root that has no predecessor), this interpretation induces a linear reference to past and a branching reference to future.

[2] Thus, for readers that still ponder about the title of this paper, [10] provided, once and for all, a systematic study of temporal logics that combine past-time operators (e.g., "once") with branching path quantification (e.g., "for all").

Accordingly, we regard $CTL^\star_{lp}$ formulas as describing a set of computations of a nondeterministic program, where each computation is a totally ordered set. The branching in the tree represents nondeterminism or interleaving due to concurrency. For example, the $CTL^\star_{lp}$ formula $AG(grant \rightarrow P(req))$ states that grant is given only upon request. Note that there is no path quantification over $P(req)$. It is clear from the semantics that a request should be found along the path that led from the root to the state in which the grant is received.

We investigate the *expressive power* of the two extensions. We first compare the two approaches to past. We show that, unlike the case of CTL*, which is insensitive to unwinding (that is, unwinding of a Kripke structure preserves the set of CTL* formulas it satisfies), augmenting a branching temporal logic with past-time connectives makes it sensitive to unwinding. We then consider the increase in the expressive power of branching temporal logics due to the addition of past-time connectives. As was shown in [5], past-time connectives do not increase the expressive power of linear temporal logic. We show here that when branching temporal logics are considered, the situation is diversified. In addition, we compare the power of past with the power of *history variables*, and check whether past can help CTL to compete successfully with CTL*.

We consider the *model-checking* and the *satisfiability* problems for the logics CTL_{bp} and CTL_{lp}. We show that, on the one hand, branching past does not make the problems harder. That is, the model-checking problem for CTL_{bp} is in linear time, and its satisfiability is in EXPTIME, coinciding with the known complexities for CTL [21,22]. Likewise, the model-checking problem for $CTL^\star_{bp}$ is in PSPACE, and its satisfiability is in 2EXPTIME, coinciding with the known complexities for CTL* [23, 24]. Note that since Pinter and Wolper have already established an exponential upper bound for satisfiability of POTL, it is not surprising that augmenting CTL with branching-past connectives preserves its exponential satisfiability. Nevertheless, our procedure for CTL_{bp} demonstrates how the fact that past is finite makes life much easier. On the other hand, linear past makes the model-checking problem more difficult. Indeed, the finite linear-past approach has been adopted by the verification community, used also in related logics (e.g., in NCTL, which augments CTL with a "from now on" modality [9], and in memoryful branching temporal logics, which enable past quantification to refer to computations that start in the initial state [25]), and its complexity has been studied [26]. In order to solve this problem we use an extension of the automata-theoretic approach to branching-time temporal logics [27] to automata with *satellites*. A similar extension was used in [25] in order to handle memoryful branching temporal logics. Essentially, a satellite is a deterministic automaton that runs in parallel to the alternating automaton for the formula and whose state space keeps track of the behavior of the computation since its beginning. Using alternating automata with satellites we are able to show a PSPACE and an EXPSPACE upper bounds for the model-checking problems of CTL_{lp} and $CTL^\star_{lp}$, respectively. We also prove matching lower bounds. We use the automata-theoretic approach also for showing a 2EXPTIME upper bound for $CTL^\star_{lp}$ satisfiability. As for CTL_{lp}, an EXPTIME satisfiability procedure is very simple, using a translation of CTL_{lp} formulas into formulas in CTL augmented with existential quantification over history variables.

Our complexity results are summarized in the table below. All results are tight.

	Model checking	Satisfiability
CTL_{bp}	linear time	EXPTIME
CTL_{lp}	PSPACE	EXPTIME
$CTL^\star_{bp}$	PSPACE	2EXPTIME
$CTL^\star_{lp}$	EXPSPACE	2EXPTIME

2. Branching logics with past operators

2.1. Branching past: the logic $CTL^\star_{bp}$

The logic $CTL^\star_{bp}$ extends CTL* by allowing past-time operators, with branching-time semantics. An argument in favor of this approach is that past and future are handled uniformly. On the other hand, it is argued in [9] that "branching past is awkward for behavior specifications". Indeed, it is more appropriate to think of this logic as a structural logic, appropriate for making assertions about the structure of a system, rather than its behavior.

As CTL*, it combines both branching-time and linear-time operators. A path quantifier E ("for some path") can prefix a formula composed of an arbitrary combination of the linear-time operators X ("next time"), U ("until"), Y ("yesterday"), and S ("since"). There are two types of formulas in $CTL^\star_{bp}$: *state formulas*, whose satisfaction is related to a specific state, and *path formulas*, whose satisfaction is related to a specific path. Formally, let AP be a set of atomic proposition names. A $CTL^\star_{bp}$ state formula is either:

- *true*, *false*, or p, for $p \in AP$.
- $\neg\varphi_1$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are $CTL^\star_{bp}$ state formulas.
- $E\psi_1$, where ψ_1 is a $CTL^\star_{bp}$ path formula.

A $CTL^\star_{bp}$ path formula is either:

- A $CTL^\star_{bp}$ state formula.
- $\neg\psi_1$, $\psi_1 \vee \psi_2$, $X\psi_1$, $\psi_1 U \psi_2$, $Y\psi_1$, or $\psi_1 S \psi_2$, where ψ_1 and ψ_2 are $CTL^\star_{bp}$ path formulas.

We use the following abbreviations of dual connectives in writing formulas:

- $\wedge, \rightarrow$, and $\leftrightarrow$, interpreted in the usual way.
- $A\psi = \neg E\neg\psi$ ("in all paths").
- $F\psi = trueU\psi$ ("eventually in the future").
- $P\psi = trueS\psi$ ("sometime in the past").
- $G\psi = \neg F\neg\psi$ ("always in the future").
- $H\psi = \neg P\neg\psi$ ("always in the past").
- $\tilde{Y}\psi = \neg Y\neg\psi$ ("weak yesterday").
- $\psi_1\tilde{S}\psi_2 = \neg((\neg\psi_1)S(\neg\psi_2))$ ("dual since").

$\mathrm{CTL}^{\star}_{bp}$ is the set of state formulas generated by the above rules.

A *past formula* is a formula in which no future-time operators occur. Similarly, a *future formula* is a formula in which no past-time operators occur.

The logic CTL_{bp} is an extension of the branching temporal logic CTL. In CTL, the temporal operators X, U, and their duals must be immediately preceded by a path quantifier. CTL_{bp} allows also the temporal operators Y, S, and their duals. As in CTL, they must be immediately preceded by a path quantifier. Formally, it is the subset of $\mathrm{CTL}^{\star}_{bp}$ obtained by restricting the path formulas to be $X\varphi_1$, $\varphi_1 U\varphi_2$, $Y\varphi_1$, $\varphi_1 S\varphi_2$, or their duals, where φ_1 and φ_2 are CTL_{bp} state formulas.

We define the semantics of $\mathrm{CTL}^{\star}_{bp}$ with respect to a *Kripke structure* $K = \langle W, R, w^0, L\rangle$, where W is a set of states, $R \subseteq W \times W$ is a transition relation that must be total in its first element, w^0 is an initial state for which there exists no state w such that $\langle w, s^0\rangle \in R$, and $L : W \to 2^{AP}$ maps each state to a set of atomic propositions true in this state. For $\langle w_1, w_2\rangle \in R$, we say that w_2 is a *successor* of w_1, and w_1 is a *predecessor* of w_2. A *path* in K is an infinite sequence of states $\pi = w_0, w_1, \ldots$, such that $w_0 = w^0$ and for all $i \geqslant 0$, we have $\langle w_i, w_{i+1}\rangle \in R$.

We use $w \models \varphi$ to indicate that a state formula φ holds at state w. We use $\pi, j \models \psi$ to indicate that a path formula ψ holds at position j of the path π. The relation $\models$ is inductively defined as follows (assuming an agreed K).

- For all $w \in W$, $w \models true$ and $w \not\models false$.
- For an atomic proposition $p \in AP$, $w \models p$ iff $p \in L(w)$.
- $w \models \neg\varphi_1$ iff $w \not\models \varphi_1$.
- $w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$.
- $w \models E\psi_1$ iff there exist a path $\pi = w_0, w_1, \ldots$ and $j \geqslant 0$ such that $w_j = w$ and $\pi, j \models \psi_1$.
- $\pi, j \models \varphi$ for a state formula φ, iff $w_j \models \varphi$.
- $\pi, j \models \neg\psi_1$ iff $\pi, j \not\models \psi_1$.
- $\pi, j \models \psi_1 \vee \psi_2$ iff $\pi, j \models \psi_1$ or $\pi, j \models \psi_2$.
- $\pi, j \models X\psi_1$ iff $\pi, j+1 \models \psi_1$.
- $\pi, j \models Y\psi_1$ iff $j > 0$ and $\pi, j-1 \models \psi_1$.
- $\pi, j \models \psi_1 U\psi_2$ iff there exists $k \geqslant j$ such that $\pi, k \models \psi_2$ and $\pi, i \models \psi_1$ for all $j \leqslant i < k$.
- $\pi, j \models \psi_1 S\psi_2$ iff there exists $0 \leqslant k \leqslant j$ such that $\pi, k \models \psi_2$ and $\pi, i \models \psi_1$ for all $k < i \leqslant j$.

For a Kripke structure K, we say that $K \models \varphi$ iff $w^0 \models \varphi$. Note that the past-time operator Y is interpreted in the strong sense. That is, in order to satisfy a Y requirement, a state must have some predecessor. In contrast, the weak-yesterday operator does not include such a requirement, and thus it enables us to identify the initial state of a computation. Indeed, $\tilde{Y}\psi$ states that if we are not in the beginning of the computation, then ψ should hold. In particular, we are in the beginning of the computation iff the current position satisfies the formula $\tilde{Y}false$. Finally, the dual-since operator is the past-time counterpart of the future-time "release" operator.[3] Thus, a computation satisfies $\psi_1\tilde{S}\psi_2$ at a certain position iff, going backwards from this position, ψ_2 may not hold only after ψ_1 holds. Formally, $\pi, j \models \psi_1\tilde{S}\psi_2$ iff for all $0 \leqslant k \leqslant j$, if $\pi, k \models \psi_2$, then there is $k < i \leqslant j$ such that $\pi, i \models \psi_1$.

We consider also the logic QCTL_{bp}, obtained by adding explicit second-order quantifiers to CTL_{bp}, in which we can quantify over sets of states. Every CTL_{bp} formula is a QCTL_{bp} formula and, in addition, if ψ is a QCTL_{bp} formula and p is an atomic proposition occurring free in ψ, then $\exists p\psi$ is also a QCTL_{bp} formula. The semantics of $\exists p\psi$ is given by $K \models \exists p\psi$ iff there exists a Kripke structure K' such that $K' \models \psi$ and K' differs from K in at most the labels of p. Thus, the effect of $\exists p$ is to choose an arbitrary set of states and label them with p. We use $\forall p\psi$ to abbreviate $\neg\exists p\neg\psi$.

2.2. Linear past: the logic $CTL^{\star}_{lp}$

The logic $\mathrm{CTL}^{\star}_{lp}$ has the same syntax as $\mathrm{CTL}^{\star}_{bp}$. We define its semantics with respect to Kripke structures in which each state has a unique path leading from the initial state to it. We call such Kripke structures *computation trees*. (We can also

[3] We do not introduce "release" here and do introduce $\tilde{S}$ as we are going to use $\tilde{S}$ in a normal form for some variant of $\mathrm{CTL}^{\star}_{bp}$.

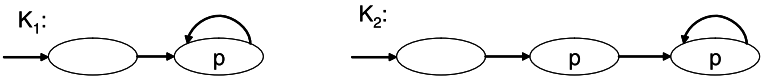

Fig. 1. The Kripke structures K_1 and K_2.

interpret formulas of $\mathrm{CTL}^\star_{lp}$ over Kripke structures, simply by referring to the unwindings of such structures.) Below, we define formally computation trees and the semantics of $\mathrm{CTL}^\star_{lp}$. Given a set D of *directions*, a *D-tree* is a set $T \subseteq D^*$ such that if $x \cdot c \in T$ where $x \in D^*$ and $c \in D$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot c$ where $c \in D$ are the *successors* of x. We consider here trees in which each node has at least one successor. (We assume, conventionally, that termination is modeled explicitly, say by a special proposition *prop*, so deadlocked nodes are not allowed.)

A *path* ρ of a tree T is a set $\pi \subseteq T$ such that $\epsilon \in \rho$ and for every $w \in \pi$ there exists a unique $c \in D$ such that $w \cdot c \in \pi$. For a path π and $j \geqslant 0$, let π_j denote the node of length j in π. Given a set D of directions and an alphabet Σ, a *Σ-labeled D-tree* is a pair $\langle T, V\rangle$ where T is a D-tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ. A computation tree is a Σ-labeled D-tree with $\Sigma = 2^{AP}$ for some set AP of atomic propositions.

We define the semantics of $\mathrm{CTL}^\star_{lp}$ with respect to a computation tree $\langle T, V\rangle$. We use $w \models \varphi$ to indicate that a state formula φ holds at node $w \in T$. We use $\pi, j \models \psi$ to indicate that a path formula ψ holds in position j of the path $\pi \subseteq T$. The relation $\models$ is defined similarly to the one of $\mathrm{CTL}^\star_{bp}$, with the following changes:

- $w \models E\psi_1$ iff there exist a path π and $j \geqslant 0$ such that $\rho_j = w$ and $\pi, j \models \psi_1$.
- $\pi, j \models \varphi$ for a state formula φ, iff $\rho_j \models \varphi$.

For a computation tree $\langle T, V\rangle$, we say that $\langle T, V\rangle \models \varphi$ iff $\epsilon \models \varphi$.

The logic LTL_p is an extension of the linear temporal logic LTL. It extends LTL by allowing also the past-time operators Y and S. The logic CTL_{lp} is the linear-past extension of CTL. As past is linear, path quantification of past-time operators is redundant. Thus, we require the temporal operators X and U to be preceded by a path quantifier, yet we impose no equivalent restriction on the operators Y and S. Note that this implies that in CTL_{lp}, path quantifiers are followed by LTL_p formulas that do no have nested future-time operators. We consider also the logic EQCTL_{lp}, obtained by adding existential second-order quantifiers to CTL_{lp}. Formally, if ψ is a CTL_{lp} formula and $p_1, \ldots, p_n$ are atomic propositions, then $\exists p_1 \cdots \exists p_n \psi$ is an EQCTL_{lp} formula. The semantics of $\exists p_1 \cdots \exists p_n \psi$ is given by $\langle T, V\rangle \models \exists p_1 \cdots \exists p_n \psi$ iff there exists a computation tree $\langle T, V'\rangle$, such that $\langle T, V'\rangle \models \psi$ and V' differs from V in at most the labels of the p_i's.

3. Expressiveness

3.1. Branching past versus linear past

A Kripke structure K can be unwound into an infinite computation tree. We denote by $\langle T_K, V_K\rangle$ the computation tree obtained by unwinding K. Each state in K may correspond to several nodes in $\langle T_K, V_K\rangle$, all having the same future (i.e., they root identical subtrees) yet differ in their past (i.e., they have different paths leading to them). Intuitively, unwinding of the Kripke structure has two implications: it makes past linear and it makes past finite. In order to show that the two approaches to past induce different logics, we show that satisfaction of $\mathrm{CTL}^\star_{bp}$ is *sensitive to unwinding*. Namely, we show that there exist a Kripke structure K and a formula φ such that $K \models \varphi$ and $\langle T_K, V_K\rangle \not\models \varphi$.

Theorem 3.1. *Satisfaction of $\mathrm{CTL}^\star_{bp}$ formulas is sensitive to unwinding.*

Proof. Consider the Kripke structure K_1 appearing in Fig. 1. The computation tree induced by K_1 is $\langle T_{K_1}, V_{K_1}\rangle$ where $T_{K_1} = 0^*$ and V_{K_1} is defined by $V_{K_1}(\epsilon) = \emptyset$ and $V_{K_1}(x) = \{p\}$ for all $x \in 0^+$. It is easy to see that while $K_1 \not\models AF(p \wedge AYp)$, we have $\langle T_{K_1}, V_{K_1}\rangle \models AF(p \wedge AYp)$. □

Note that since $\mathrm{CTL}^\star_{bp}$ assumes a finite past, both K_1 and $\langle T_{K_1}, V_{K_1}\rangle$ satisfy $AGAP\neg p$. Also, as both w^0 and ϵ have no predecessors, then $w^0 \not\models EYtrue$ and $\epsilon \not\models EYtrue$. Clearly, for all $w \neq w^0$ and $x \neq \epsilon$, we have $w \models EYtrue$ and $x \models EYtrue$. Thus, both $\mathrm{CTL}^\star_{bp}$ and $\mathrm{CTL}^\star_{lp}$ can characterize the starting point of the past.

The logics QCTL and EQCTL are also sensitive to unwinding. Consider the formula $\varphi = \exists qAG(p \leftrightarrow AXq)$ and consider the Kripke structures K_1 and K_2 appearing in Fig. 1. Though K_2 can be obtained by unwinding K_1, we have $K_1 \not\models \varphi$ and $K_2 \models \varphi$. In the sequel, we compare QCTL with CTL_{bp}, and thus interpret it over Kripke structures, and compare EQCTL with CTL_{lp}, and thus interpret it over computation trees.

3.2. Expressive power

In this section we consider the expressive power of branching temporal logics with past with respect to branching temporal logics without past.

We say that two formulas φ_1 and φ_2 are *equivalent* ($\varphi_1 \sim \varphi_2$) if for every Kripke structure K, we have $K \models \varphi_1$ iff $K \models \varphi_2$. For path formulas we need to refine this notion since they are satisfied with respect to paths. We say that two path formulas ψ_1 and ψ_2 are *congruent* ($\psi_1 \approx \psi_2$) if for every Kripke structure K, path π in it, and position $j \geqslant 0$, we have $\pi, j \models \psi_1$ iff $\pi, j \models \psi_2$. The notions of equivalence and congruence are defined similarly for logics with linear past, only that we define them with respect to computation trees. When comparing expressive power of two logics L_1 and L_2, we say that L_1 is *more expressive than* L_2 ($L_1 \geqslant L_2$) provided that for every formula φ_2 of L_2, there exists an equivalent formula φ_1 of L_1. Also, L_1 is *as expressive as* L_2 ($L_1 = L_2$) if both $L_1 \geqslant L_2$ and $L_2 \geqslant L_1$, and L_1 is *strictly more expressive than* L_2 ($L_1 > L_2$) if both $L_1 \geqslant L_2$ and $L_2 \not\geqslant L_1$.

We first prove that $\mathrm{CTL}^\star_{lp} = \mathrm{CTL}^\star$. While $\mathrm{CTL}^\star_{lp}$ is interpreted over computation trees, $\mathrm{CTL}^\star$ is interpreted over Kripke structures. However, since $\mathrm{CTL}^\star$ is insensitive to unwinding, this causes no difficulty, as the structures can be unwound to the limit, yielding trees, without affecting the semantics of $\mathrm{CTL}^\star$ formulas. We use the *Separation Theorem* for LTL_p:

Theorem 3.2 *(Separation Theorem). (See [28].) Every LTL_p formula is congruent to a boolean combination of past and future LTL_p formulas.*

Lemma 3.3. *Let $E\psi$ be a $\mathrm{CTL}^\star_{lp}$ formula all of whose strict state subformulas are in $\mathrm{CTL}^\star$. Then, $E\psi$ is congruent to a disjunction of formulas of the form $\theta \wedge E\varphi$, where θ is a past LTL_p formula and $E\varphi$ is a $\mathrm{CTL}^\star$ formula.*

Proof. By the Separation Theorem, ψ is congruent to a boolean combination, ψ', of future and past LTL_p formulas. (Strictly speaking, the Separation Theorem applies only to $\mathrm{CTL}^\star_{lp}$ formulas, and ψ is not a $\mathrm{CTL}^\star_{lp}$ formula. Nevertheless, the theorem does apply as we can treat state subformulas as atomic propositions.) Without loss of generality, ψ' is of the form $\bigvee_{1 \leqslant i \leqslant n}(\theta_i \wedge \varphi_i)$, where, for all $1 \leqslant i \leqslant n$, we have that θ_i is a past LTL_p formula and φ_i is a future LTL_p formula. As past is linear, path quantification over past LTL_p formulas can be eliminated using the congruence below.

$$E \bigvee_{1 \leqslant i \leqslant n} (p_i \wedge q_i) \approx \bigvee_{1 \leqslant i \leqslant n} E(p_i \wedge q_i) \approx \bigvee_{1 \leqslant i \leqslant n} (p_i \wedge Eq_i). \qquad \square$$

Theorem 3.4. *$CTL^\star_{lp} = CTL^\star$.*

Proof. Given a $\mathrm{CTL}^\star_{lp}$ formula φ, we translate φ into an equivalent $\mathrm{CTL}^\star$ formula. The translation proceeds from the innermost state subformulas of φ, using Lemma 3.3 to propagate past outward. Formally, we define the depth of a state subformula ξ in φ as the number of nested E's in ξ, and proceed by induction over this depth. Subformulas of depth one have atomic propositions as their subformulas and therefore they satisfy the lemma's condition. Also, at the end of step i of the induction, all subformulas of depth i are written as disjunctions of formulas of the form $p \wedge Eq$ where p is a past LTL_p formula and Eq is a $\mathrm{CTL}^\star$ formula. Thus, propagation can continue. In particular, at the end of the inductive propagation, φ is written as such a disjunction. Then, as the past formulas refer to the initial state, we replace Yq with *false*, replace pSq with q, and we end up with a $\mathrm{CTL}^\star$ formula. $\square$

As our past connectives are more powerful than the ones used in [17], Theorem 3.4 is much stronger than the $\mathrm{PCTL}^\star = \mathrm{CTL}^\star$ result there.

In all the $L_1 < L_2$ relations in Theorems 3.5, 3.6, and 3.7 below, the $L_1 \leqslant L_2$ part follows by syntactic containment and we prove only strictness.

Theorem 3.5. *$CTL < CTL_{lp} < EQCTL_{lp} = EQCTL$.*

Proof. In [29], Emerson and Halpern show that the $\mathrm{CTL}^\star$ formula $AF(p \wedge Xp)$ has no equivalent in CTL. As $AXAF(p \wedge Yp) \sim AF(p \wedge Xp)$, we have $\mathrm{CTL} < \mathrm{CTL}_{lp}$. The specification "$q$ holds at all even places" is expressible in EQCTL_{lp} using the formula $\varphi = \exists p(p \wedge AG(p \rightarrow AXAXp) \wedge AG(p \rightarrow q))$. Extending Wolper's result from [30], φ has no equivalent of $\mathrm{CTL}^\star$. Hence, as $\mathrm{CTL}_{lp} \leqslant \mathrm{CTL}^\star$, we have $\mathrm{CTL}_{lp} < \mathrm{EQCTL}_{lp}$.

To prove $\mathrm{EQCTL}_{lp} = \mathrm{EQCTL}$, we prove that $\mathrm{EQCTL}_{lp} \leqslant \mathrm{EQCTL}$. Equivalence then follows by syntactic containment. Given an EQCTL_{lp} formula ψ, we translate ψ into an equivalent EQCTL formula. Let φ be a formula of the form $Y\varphi_1$ or $\varphi_1 S \varphi_2$, and let p be a fresh atomic proposition. We define the formula $label(\varphi, p)$ as follows:

- $label(Y\varphi_1, p) = \neg p \wedge AG(\varphi_1 \rightarrow AXp) \wedge AG(\neg\varphi_1 \rightarrow AX\neg p)$.
- $label(\varphi_1 S \varphi_2, p) = (p \leftrightarrow \varphi_2) \wedge AG(p \rightarrow AX(p \leftrightarrow (\varphi_1 \vee \varphi_2))) \wedge AG(\neg p \rightarrow AX(p \leftrightarrow \varphi_2))$.

The definition of $label(\varphi, p)$ guarantees that if a computation tree $\langle T, V\rangle$ satisfies $label(\varphi, p)$, then for every node $x \in T$, we have $x \models p$ iff for every path $\rho \subseteq T$ and $j \geqslant 0$ with $\rho_j = x$, we have $\rho, j \models \varphi$. The above observation is the key to our translation. Given ψ, we translate it into an equivalent EQCTL formula by replacing its path subformulas φ of the form $Y\varphi_1$ or $\varphi_1 S\varphi_2$, with a fresh atomic proposition p_φ, conjuncting the resulted formula with $label(\varphi, p_\varphi)$, and prefixing it with $\exists p_\varphi$. Replacement continues for the past formulas in $label(\varphi, p_\varphi)$, if exist. For example, the formula $AXAF(p \wedge Yp)$ is translated to the formula $\exists q AXAF(p \wedge q) \wedge \neg q \wedge AG(p \rightarrow AXq) \wedge AG(\neg p \rightarrow AX\neg q)$. □

Theorem 3.6. *$CTL < CTL_{bp} < QCTL_{bp} = QCTL$.*

Proof. Consider the CTL_{bp} formula $\varphi = EF((EYp) \wedge EY\neg p)$ and consider the Kripke structures K_1 and K_2 presented in Fig. 1. It is easy to see that $K_1 \models \varphi$ and $K_2 \not\models \varphi$. As K_2 can be obtained by unwinding K_1 and as CTL is not sensitive to unwinding, no CTL formula can distinguish between K_1 and K_2. Hence $CTL < CTL_{bp}$. As with linear past, $CTL_{bp} < QCTL_{bp}$ follows from the inexpressibility of "q holds at all even places" in CTL_{bp}.

To prove $QCTL_{bp} = QCTL$, we suggest a translation of $QCTL_{bp}$ formulas into QCTL formulas. We assume a normal form for $QCTL_{bp}$ in which the allowed past operators are EY, ES, and $E\tilde{S}$. Intuitively, we would have liked to do something similar to the translation of CTL_{lp} formulas into EQCTL. However, since a state in a Kripke structure may have several predecessors, which do not necessarily agree on the formulas true in them, we cannot do it. Instead, we label K in two steps: for every past formula φ, we first label with p_φ all the states that satisfy φ. Then we require p_φ to be the least such labeling, guaranteeing that *only* states that satisfy φ are labeled. Let φ be a formula of the form $EY\varphi_1$, $E\varphi_1 S\varphi_2$, or $E\varphi_1\tilde{S}\varphi_2$, and let p be a fresh atomic proposition. We define the formula $spread(\varphi, p)$ as follows:

- $spread(EY\varphi_1, p) = AG(\varphi_1 \rightarrow AXp)$.
- $spread(E\varphi_1 S\varphi_2, p) = AG(\varphi_2 \rightarrow p) \wedge AG(p \rightarrow AX((\varphi_1 \vee \varphi_2) \rightarrow p))$.
- $spread(E\varphi_1\tilde{S}\varphi_2, p) = (p \leftrightarrow \varphi_2) \wedge AG(p \rightarrow AX(\varphi_2 \rightarrow p)) \wedge AG((\varphi_2 \wedge \varphi_1) \rightarrow p)$.

The definition of $spread(\varphi, p)$ guarantees that if a Kripke structure K satisfies $spread(\varphi, p)$, then for every state w for which $w \models \varphi$, we have $w \models p$. We now define the formula $label(\varphi, p)$ which guarantees that the labeling of p is tight.

$$label(\varphi, p) = spread(\varphi, p) \wedge \forall r\big(spread(\varphi, r) \rightarrow AG(p \rightarrow r)\big).$$

If a Kripke structure K satisfies $label(\varphi, p)$, then for every state w, we have $w \models p$ iff $w \models \varphi$. Once $label(\varphi, p)$ is defined, we proceed as in the linear-past case. Note that the fact that past is finite plays a crucial role in our translation. Only thanks to it we are able to determine labeling of $E\varphi_1\tilde{S}\varphi_2$ in w^0 and then to spread labeling forward. □

As QCTL satisfies the finite model property, Theorem 3.6 implies that CTL_{bp} satisfies the finite model property as well. The logic POTL, which essentially differs from CTL_{bp} in allowing infinite past, does not satisfy this property [15]. Indeed, the fact that CTL_{bp} assumes a finite past, makes it an "easy" language. On the other hand, CTL_{bp} does not satisfy the tree model property. To see this, consider the formula $EF((EYp) \wedge (EY\neg p))$, which is satisfied in K_1, yet no computation tree can satisfy it. As EQCTL does satisfy the tree model property, we could not do without universal quantifiers in the translation.

Theorem 3.7. *$CTL^\star_{lp} > CTL_{lp}$, $CTL^\star_{bp} > CTL_{bp}$, and $CTL^\star_{bp} > CTL^\star$.*

Proof. In [29], Emerson and Halpern show that the CTL* formula $EGFp$ has no equivalent formula in CTL. It is easy to extend their proof to consider also CTL_{lp} and CTL_{bp}. Hence, $CTL^\star_{lp} > CTL_{lp}$ and $CTL^\star_{bp} > CTL_{bp}$. In the proof of Theorem 3.6, we point to a CTL_{bp} formula φ that distinguishes between a Kripke structure and its unwinding. Since CTL* is not sensitive to unwinding, φ has no equivalence of CTL*. Hence, $CTL^\star_{bp} > CTL^\star$. □

In conclusion, while linear past does not add to the expressive power of CTL*, it adds to CTL the power of linear logic, and strictly increases its expressive power. As for branching past, it adds power to both CTL and CTL*. In both the linear and branching views, however, the contribution of past does not go beyond the contribution of quantified atomic propositions. Finally, the expressiveness superiority of CTL* with respect to CTL is maintained in the presence of both linear and branching past.

As noted earlier, we can interpret the logics with linear past over Kripke structures by referring to their unwindings. This makes it possible to compare linear and branching past. The results in this section imply that $CTL^\star_{bp} > CTL^\star_{lp}$. Our conjecture is that CTL_{bp} and CTL_{lp} are incomparable.

4. From formulas to automata

In [27], the authors describe an automata-theoretic framework for reasoning about branching temporal logics. By translating formulas of a branching temporal logic $\mathcal{L}$ to alternating tree automata, it is possible to reduce the satisfiability and

model-checking problems for $\mathcal{L}$ to the nonemptiness and 1-letter nonemptiness problems, respectively, for the corresponding automata. This leads to tight complexity bounds for many branching temporal logics. In particular, an analysis of the structure of the alternating automata obtained for the logic CTL* (the hesitation condition, as we elaborate below) leads to tight space complexities. In [25], the authors extend the framework to handle memoryful branching temporal logics. Memoryful branching temporal logic enables a reference to the behavior of computations since their beginning, which requires the automaton to remember the past. Simple solutions for the need to refer to the past, like two-way automata or a naive maintenance of information about the past in the state space of the alternating automaton, result in automata that are too big [31,26]. Instead, it is suggested in [25] to augment alternating automata by a deterministic *satellite* that maintains information about the past. In this section we show how a similar idea can work also for branching temporal logics with linear past, and translate CTL$^\star_{lp}$ and CTL$_{lp}$ formulas to hesitant alternating automata with satellites. Further, by considering *two-way* trees, which model both forward and backward transitions, we can translate CTL$^\star_{bp}$ and CTL$_{bp}$ formulas to alternating automata with satellites that can be used in order to solve the satisfiability problem for branching temporal logic with branching past.

4.1. Alternating automata

Automata over infinite trees (tree automata) run over Σ-labeled D-trees that have no leaves [32]. *Alternating tree automata* generalize nondeterministic tree automata and were first introduced in [33]. Their application to temporal reasoning was proposed in [34] and explored further in [27]. Here we define *symmetric* alternating automata, which cannot distinguish between the different successors of a node, and send copies to the successors in either a universal or an existential manner [35].

Let $\Omega = \{\Box, \Diamond\}$, and let $\mathcal{B}^+(\Omega \times Q)$ be the set of positive Boolean formulas over $\Omega \times Q$; i.e., Boolean formulas built from elements in $\Omega \times Q$ using $\wedge$ and $\vee$, where we also allow the formulas *true* and *false* and, as usual, $\wedge$ has precedence over $\vee$. For a set $S \subseteq \Omega \times Q$ and a formula $\theta \in \mathcal{B}^+(\Omega \times Q)$, we say that S *satisfies* θ iff assigning *true* to elements in S and assigning *false* to elements in $(\Omega \times Q) \setminus S$ makes θ true.

Consider a set D of directions. In a symmetric alternating automaton $\mathcal{A}$ over Σ-labeled D-trees, with a set Q of states, the transition function δ maps an automaton state $q \in Q$ and an input letter $\sigma \in \Sigma$ to a formula in $\mathcal{B}^+(\Omega \times Q)$. Intuitively, an atom $\langle \Box, q' \rangle$ corresponds to copies of the automaton in state q', sent to all the successors of the current node. An atom $\langle \Diamond, q' \rangle$ corresponds to a copy of the automaton in state q', sent to some successor of the current node. When, for instance, the automaton is in state q, reads a node x with successors $x \cdot d_1, \ldots, x \cdot d_n$, and $\delta(q, V(x)) = (\Box, q_1) \wedge (\Diamond, q_2) \vee (\Diamond, q_2) \wedge (\Diamond, q_3)$, the automaton can either send n copies in state q_1 to all the successors of x and a copy in state q_2 to one of the successors, or send a copy in state q_2 to one of the successors and a copy in state q_3 to one (possibly the same) successor. So, while nondeterministic tree automata send exactly one copy to each successor, symmetric alternating automata can send several copies to the same successor. On the other hand, symmetric alternating automata cannot distinguish between left and right and can send copies to successor nodes only in either a universal or an existential manner.

Formally, a symmetric alternating automaton is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_{in}, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(\Omega \times Q)$ is a transition function, $q_{in} \in Q$ is an initial state, and α specifies the acceptance condition (a condition that defines a subset of Q^ω). Let $T = D^*$. If $q \in Q$, we use the notation $\mathcal{A}^q$ to denote the automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q, \alpha \rangle$, where q is the initial state. A *run* of a symmetric alternating automaton $\mathcal{A}$ on an input Σ-labeled D-tree $\langle T, V \rangle$ is a tree $\langle T_r, r \rangle$ (to be formally defined shortly) in which each node is labeled by an element of $D^* \times Q$. Unlike T, in which each node has exactly $|D|$ children, the tree T_r may have nodes with many children and may also have leaves. Thus, $T_r \subset \mathcal{N}^*$ and a path in T_r may be either finite, in which case it ends in a leaf, or infinite. Each node of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T; in contrast, in a run of a nondeterministic automaton on $\langle T, V \rangle$ there is a one-to-one correspondence between the nodes of the run and the nodes of the tree. The labels of a node and its children have to satisfy the transition function. Formally, the run is a $(D^* \times Q)$-labeled $\mathcal{N}$-tree $\langle T_r, r \rangle$ that satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_{in})$.
2. Let $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$. Then there is a (possibly empty) set $S \subseteq \Omega \times Q$, such that S satisfies θ, and for all $(c, s) \in S$, the following hold:
   - If $c = \Box$, then for each $d \in D$, there is $j \in \mathcal{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot d, s)$.
   - If $c = \Diamond$, then for some $d \in D$, there is $j \in \mathcal{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot d, s)$.

For example, if $\langle T, V \rangle$ is a $\{1, 2\}$-tree with $V(\varepsilon) = a$ and $\delta(q_{in}, a) = \Diamond q_1 \wedge \Box q_2$, then the nodes of $\langle T_r, r \rangle$ at level 1 include one of the labels $(1, q_1)$ or $(2, q_1)$, and include both labels $(1, q_2)$ and $(2, q_2)$. Note that if $\theta = \mathit{true}$, then y need not have children. This is the reason why T_r may have leaves. Also, since there exists no set S as required for $\theta = \mathit{false}$, we cannot have a run that takes a transition with $\theta = \mathit{false}$.

Each infinite path ρ in $\langle T_r, r \rangle$ is labeled by a word in Q^ω. Let $\mathit{inf}(\rho)$ denote the set of states in Q that appear in $r(\rho)$ infinitely often. A run $\langle T_r, r \rangle$ is accepting iff all its infinite paths satisfy the acceptance condition. In *Büchi* automata, $\alpha \subseteq Q$, and an infinite path ρ satisfies α iff $\mathit{inf}(\rho) \cap \alpha \neq \emptyset$. In *co-Büchi* automata, $\alpha \subseteq Q$, and an infinite path ρ satisfies

α iff $inf(\rho) \cap \alpha = \emptyset$. We also refer to *generalized Büchi* automata, where $\alpha \subseteq 2^Q$ and an infinite path ρ satisfies α iff $inf(\rho) \cap F \neq \emptyset$ for all sets $F \in \alpha$.

An automaton accepts a tree iff there exists an accepting run on it. The language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of all labeled trees that $\mathcal{A}$ accepts. The *nonemptiness problem* is to decide, given $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A}) \neq \emptyset$. When $|\Sigma| = 1$, we refer to the problem as 1-*letter nonemptiness*.

In [27], the authors introduce *hesitant alternating automata* (HAAs, for short) and show that CTL* formulas can be translated to such automata. An HAA is an alternating automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, where $\alpha = \langle G, B \rangle$ with $G \subseteq Q$ and $B \subseteq Q$. That is, the acceptance condition of HAAs consists of a pair of sets of states. As in weak alternating automata [34], there exist a partition of Q into disjoint sets $Q_1, \ldots, Q_m$ and a partial order $\leqslant$ on these sets such that transitions lead to sets that are lower in the partial order. Formally, for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, \sigma)$, for some $\sigma \in \Sigma$, we have $Q_j \leqslant Q_i$. In addition, each set Q_i is classified as either *transient*, *existential*, or *universal*, such that for each set Q_i and for all $q \in Q_i$ and $\sigma \in \Sigma$, the following hold:

1. If Q_i is a transient set, then $\delta(q, \sigma)$ contains no states of Q_i.
2. If Q_i is an existential set, then $\delta(q, \sigma)$ only contains disjunctively related states of Q_i. Thus, no state of Q_i is in a scope of a $\Box$, and if $\delta(q, \sigma)$ is rewritten in DNF, then there are no two states of Q_i in the same disjunct.
3. If Q_i is a universal set, then $\delta(q, \sigma)$ only contains conjunctively related elements of Q_i. Thus, no state of Q_i is in a scope of a $\Diamond$, and if $\delta(q, \sigma)$ is rewritten in CNF, then there are no two states of Q_i in the same conjunct.

It follows that every infinite path π of a run $\langle T_r, r \rangle$ gets trapped within some existential or universal set Q_i. The path then satisfies an acceptance condition $\langle G, B \rangle$ if and only if either Q_i is an existential set and $inf(\pi) \cap G \neq \emptyset$, or Q_i is a universal set and $inf(\pi) \cap B = \emptyset$. Note that the acceptance condition of HAAs combines the Büchi and the co-Büchi condition: existential sets refer to the Büchi condition G and universal sets refer to the co-Büchi condition B. The number m of sets in the partition of Q is referred to as the *depth* of the HAA.

Given a transition function δ, let $\tilde{\delta}$ denote the dual function of δ. That is, for every q and σ with $\delta(q, \sigma) = \theta$, let $\tilde{\delta}(q, \sigma) = \tilde{\theta}$, where $\tilde{\theta}$ is obtained from θ by switching $\Box$ and $\Diamond$, switching $\vee$ and $\wedge$, and switching *true* and *false*. If, for example, $\theta = \Box p \vee (true \wedge \Diamond q)$ then $\tilde{\theta} = \Diamond p \wedge (false \vee \Box q)$,

Lemma 4.1. *(See [27].) Given an HAA $\mathcal{A} = \langle \Sigma, Q, \delta, q_{in}, \langle G, B \rangle \rangle$, the alternating automaton $\tilde{\mathcal{A}} = \langle \Sigma, Q, \tilde{\delta}, q_{in}, \langle B, G \rangle \rangle$ is an HAA that complements $\mathcal{A}$. Thus, $\tilde{\mathcal{A}}$ accepts exactly all trees that $\mathcal{A}$ rejects.*

Symmetric alternating automata are powerful enough for deciding satisfiability and model-checking of CTL* formulas [27]. In order to handle linear past, we need an extension of symmetric alternating automata to *two-way* automata [31]. This approach, however, would lead to nondeterministic automata of a doubly-exponential size. While this is good enough for the satisfiability problem, it is not satisfactory for CTL$^\star_{lp}$ model checking. Instead, we use *hesitant alternating automata with satellites*, which were introduced in [25] with a similar motivation — model checking of memoryful branching temporal logics. We show that this framework can establish upper bounds for both satisfiability and model checking.

A *satellite* for an HAA $\mathcal{A} = \langle \Sigma, Q, \delta, q_{in}, \alpha \rangle$ is a deterministic word automaton $\mathcal{U} = \langle \Sigma, Q', \delta', q'_{in} \rangle$ with no acceptance condition. Thus, the transition function $\delta' : Q' \times \Sigma \to Q'$ maps a state and a letter to a single successor state. For a node $x = d_1 \cdots d_k$ of a Σ-labeled D-tree $\langle T, V \rangle$, let $word\_to(x) = V(\varepsilon) \cdot V(d_1) \cdot V(d_1 \cdot d_2) \cdots V(d_1 \cdot d_2 \cdots d_k)$ be the word that labels the path from the root to x. We extend the domain of δ' to $Q' \times \Sigma^*$ in the standard way. When the HAA reads a node x of the input tree, its transitions may depend on $\delta'(q'_{in}, word\_to(x))$, namely on the state in which $\mathcal{U}$ would have been if we had run it along the paths of the tree. Formally, the transition function of $\mathcal{A}$ is $\delta : Q \times \Sigma \times Q' \to \mathcal{B}^+(\Omega \times Q)$, and is such that when $\mathcal{A}$ is in state q as it reads the node x, it proceeds according to $\delta(q, V(x), \delta'(q'_{in}, word\_to(x)))$. Note that the transition is performed in two phases. In the first phase, $\mathcal{U}$ reads the letter $V(x)$ — the last letter in $word\_to(x)$, and updates its state. Then, $\mathcal{A}$ reads $V(x)$ and the updated state of $\mathcal{U}$, and updates its state. Technically, an HAA with a satellite is equivalent to the HAA obtained by taking the product of $\mathcal{A}$ and $\mathcal{U}$: the new state space is $Q \times Q'$, the initial state is $\langle q_{in}, q'_{in} \rangle$, and whenever the product is in state $\langle q, q' \rangle$ and reads a node x, the transition from $\langle q, q' \rangle$ is obtained from $\delta(q, V(x), \delta'(q', V(x)))$ by replacing each atom $\Box s$ or $\Diamond s$ by $\Box(s, \delta'(q', V(x)))$ or $\Diamond(s, \delta'(q', V(x)))$, respectively. The acceptance condition of the product HAA is induced by α. The partition of the state space to sets, the partial order on the sets, and their classification into transient, universal, and existential are induced by these in $\mathcal{A}$.

Note that satellites are only a convenient way to describe HAA in which the state space can be partitioned to two components, one of which is deterministic, independent of the other, has no influence on the acceptance, and runs on all the branches of the tree. In particular, Lemma 4.1 holds also for HAA with satellites. It is sometimes convenient to describe the HAA and its satellite separately. In addition to clarity, the separation to $\mathcal{A}$ and $\mathcal{U}$ enables a tighter analysis of the complexity of the nonemptiness problem. Recall that the solution of the emptiness problem for alternating automata involves alternation removal, which results in a nondeterministic automaton with exponentially many states. While the product of an HAA with n states and a satellite with n' states has nn' states, there is a need to pay the exponential price of alternation removal in the process of the nonemptiness check only for $\mathcal{A}$. Formally, we have the following.

Theorem 4.2. *(See [25].)*

- *The nonemptiness problem for an HAA with n states and a satellite with n' states can be solved in time $2^{O(n \log n' + n^2 \log n)}$.*
- *The 1-letter nonemptiness problem for an HAA with n states, depth m, and a satellite with n' states can be solved in space $O(m \log^2(nn'))$.*

(The definition of satellite here differs in a minor way from the definition in [25]. The difference has essentially no impact in the algorithms and complexity results described in Theorem 4.2.)

4.2. From $CTL^\star_{lp}$ and CTL_{lp} to HAAs with satellites

For an LTL_p formula ψ, the *closure* of ψ, denoted by $cl(\psi)$, is the set of ψ's subformulas and their negations ($\neg\neg\psi$ is identified with ψ). Formally, $cl(\psi)$ is the smallest set of formulas that satisfy the following.

- $\psi \in cl(\psi)$.
- If $\psi_1 \in cl(\psi)$ then $\neg\psi_1 \in cl(\psi)$.
- If $\neg\psi_1 \in cl(\psi)$ then $\psi_1 \in cl(\psi)$.
- If $\psi_1 \wedge \psi_2 \in cl(\psi)$ then $\psi_1 \in cl(\psi)$ and $\psi_2 \in cl(\psi)$.
- If $X\psi_1 \in cl(\psi)$ or $Y\psi_1 \in cl(\psi)$ then $\psi_1 \in cl(\psi)$.
- If $\psi_1 U \psi_2 \in cl(\psi)$ or $\psi_1 S \psi_2 \in cl(\psi)$ then $\psi_1 \in cl(\psi)$ and $\psi_2 \in cl(\psi)$.

For example, $cl(p \wedge ((Xp)Sq))$ is

$$\big\{p \wedge \big((Xp)Sq\big), \neg\big(p \wedge \big((Xp)Sq\big)\big), p, \neg p, (Xp)Sq, \neg\big((Xp)Sq\big), Xp, \neg Xp, q, \neg q\big\}.$$

Lemma 4.3. *Consider an LTL_p formula ψ. There is a nondeterministic generalized Büchi automaton $\mathcal{U}_{cl(\psi)} = \langle 2^{AP}, \{q_0\} \cup Q, \delta, q_0, \alpha\rangle$ such that $Q \subseteq 2^{cl(\psi)}$, and for every word $w \in (2^{AP})^\omega$, if $q_0, C_0, C_1, \ldots$ is an accepting run of $\mathcal{U}_{cl(\psi)}$ on w, then for each $\theta \in cl(\psi)$ and each $i \geqslant 0$, we have that $\theta \in C_i$ iff $w, i \models \theta$.*

Proof. The construction is a variation on the standard translation of LTL formulas to generalized Büchi automata [36,37]. In the standard translation we are trying to check whether ψ holds at time 0. Here, on the other hand, we have to keep track of all the formulas in $cl(\psi)$. Note that the automaton is denoted by $\mathcal{U}_{cl(\psi)}$ and not by $\mathcal{U}_\psi$, since the automaton checks all formulas in $cl(\psi)$.

Given an LTL formula ψ over AP, we define $\mathcal{U}_{cl(\psi)} = \langle 2^{AP}, \{q_0\} \cup Q, \delta, q_0, \alpha\rangle$, where

- We say that a set $C \subseteq cl(\psi)$ is *good in* $cl(\psi)$ if C is a maximal set of formulas in $cl(\psi)$ that does not have propositional inconsistency. Thus, C satisfies the following conditions.
  1. For all $\psi_1 \in cl(\psi)$, we have $\psi_1 \in C$ iff $\neg\psi_1 \notin C$, and
  2. for all $\psi_1 \wedge \psi_2 \in cl(\psi)$, we have $\psi_1 \wedge \psi_2 \in C$ iff $\psi_1 \in C$ and $\psi_2 \in C$.

  The set $Q \subseteq 2^{cl(\psi)}$ is the set of all the good sets in $cl(\psi)$.
- Let C' be a good set in $cl(\psi)$, and let $\sigma \in 2^{AP}$ be a letter. Then, $C' \in \delta(q_0, \sigma)$ if the following hold:
  1. $\sigma = C' \cap AP$,
  2. for all $Y\psi_1 \in cl(\psi)$, we have $\neg Y\psi_1 \in C'$, and
  3. for all $\psi_1 S \psi_2 \in cl(\psi)$, we have $\psi_1 S \psi_2 \in C'$ iff $\psi_2 \in C'$.
- Let C and C' be two good sets in $cl(\psi)$, and let $\sigma \in 2^{AP}$ be a letter. Then, $C' \in \delta(C, \sigma)$ if the following hold:
  1. $\sigma = C' \cap AP$,
  2. for all $X\psi_1 \in cl(\psi)$, we have $X\psi_1 \in C$ iff $\psi_1 \in C'$,
  3. for all $Y\psi_1 \in cl(\psi)$, we have $Y\psi_1 \in C'$ iff $\psi_1 \in C$,
  4. for all $\psi_1 U \psi_2 \in cl(\psi)$, we have $\psi_1 U \psi_2 \in C$ iff either $\psi_2 \in C$ or both $\psi_1 \in C$ and $\psi_1 U \psi_2 \in C'$, and
  5. for all $\psi_1 S \psi_2 \in cl(\psi)$, we have $\psi_1 S \psi_2 \in C'$ iff either $\psi_2 \in C'$ or both $\psi_1 \in C'$ and $\psi_1 S \psi_2 \in C$.
- The acceptance condition guarantees that fulfillment of eventualities is not delayed forever. Accordingly, as in [36,37], every formula of the form $\psi_1 U \psi_2 \in cl(\psi)$ contributes to α the set

$$F_{\psi_1 U \psi_2} = \big\{C \in Q\colon\ \psi_2 \in C \text{ or } \neg(\psi_1 U \psi_2) \in C\big\}.$$

Note that in the construction of [36], the initial state is a good set that contains the formula ψ. Here, on the other hand, we use a generic initial state q_0. In a sense, the run of the automaton here is "shifted to the right by one" with respect to the construction in [36]. In particular, note that the letter that labels the transition from state C to state C' agrees with the atomic proposition in C' (rather than with those in C, as is the case in [36]). The reason for this shift is explained later.

Let $w \in (2^{AP})^*$ and $q_0, C_0, C1, \ldots$ be an accepting run of $\mathcal{U}_{cl(\psi)}$ on w. It follows from the results in [36,37] that for each $\theta \in cl(\psi)$ and each $i \geqslant 0$, we have that $\theta \in C_i$ iff $w, i \models \theta$. $\square$

As in [36], the construction is exponential in the length of ψ. Note that since the automaton $\mathcal{U}_{cl(\psi)}$ checks the truth of all subformulas in $cl(\psi)$, we cannot use here the size-minimizing heuristics of [37].

Thus, by Lemma 4.3, if we want to check whether a node x of a computation tree $\langle T, V\rangle$ satisfies a $\mathrm{CTL}^\star_{lp}$ formula of the form $E\xi$, then we can consult $\mathcal{U}_{cl(\xi)}$, see the set of states it reaches after reading the path from the root to x, and guess a direction to proceed to with one of the states that contain ξ. Consulting $\mathcal{U}_{cl(\xi)}$ is done by following its subset construction in a satellite that runs in parallel with the HAA.

Another technical obstacle we have to address is the fact that the path formulas of $\mathrm{CTL}^\star_{lp}$ may have state formulas (rather than atomic propositions) as their subformulas. Accordingly, when we construct an automaton $\mathcal{U}_\xi$ for a formula of the form $E\xi$, we assume that its alphabet is subsets of ξ's subformulas, rather than subsets of AP (we later take care of this assumption). To formalize this, we first need some definitions. For a $\mathrm{CTL}^\star_{lp}$ formula ψ, let $sf(\psi)$ be the set of state subformulas of ψ. For two state formulas θ and φ of ψ, we say that θ is *maximal* in φ if θ is a strict state subformula of φ and there exists no state formula "between them", namely, there exists no strict subformula ξ of φ such that θ is a strict subformula of ξ. We denote by $max(\varphi)$ the set of all formulas maximal in φ. For example, $max(A((X\neg p)U(EYq))) = \{\neg p, EYq\}$. Consider a $\mathrm{CTL}^\star_{lp}$ formula ψ and a computation tree $\langle T, V\rangle$. We say that a $2^{sf(\psi)}$-labeled tree $\langle T, g\rangle$ is *sound* for ψ if for all $x \in T$ and $\theta \in sf(\psi)$, we have that $x \models \theta$ iff $\theta \in g(x)$. Thus, a node x is labeled by exactly all the formulas in $sf(\psi)$ that are satisfied in x.

Theorem 4.4. *Given a $\mathrm{CTL}^\star_{lp}$ formula ψ, we can construct an HAA $\mathcal{A}_\psi$ with $2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states, such that $\mathcal{A}_\psi$ runs on $2^{sf(\psi)}$-labeled trees and accepts exactly all trees that are sound for ψ and satisfy ψ.*

Proof. We first define the satellite $\mathcal{U}$ for $\mathcal{A}_\psi$. Consider a subformula of ψ of the form $E\xi$. Let $\mathcal{U}_{cl(\xi)} = \langle 2^{sf(\xi)}, Q_\xi, M_\xi, q^{in}_\xi, \alpha_\xi\rangle$ be the nondeterministic generalized Büchi automaton that corresponds to $cl(\xi)$, as defined in Lemma 4.3. Note that $\mathcal{U}_{cl(\xi)}$ regards the state formulas maximal in ξ as atomic propositions ($\mathcal{U}_{cl(\xi)}$ ignores the other formulas in $sf(\psi)$). Let $\mathcal{U}^d_{cl(\xi)} = \langle 2^{sf(\psi)}, 2^{Q_\xi}, M^d_\xi, \{q^{in}_\xi\}\rangle$ be the deterministic automaton with no acceptance condition obtained by applying the subset construction to $\mathcal{U}_{cl(\xi)}$. Thus, for all $S \in 2^{Q_\xi}$ and $\sigma \in 2^{sf(\psi)}$, we have that $M^d_\xi(S, \sigma) = \bigcup_{s\in S} M_\xi(s, \sigma)$. Now, the satellite $\mathcal{U} = \langle 2^{sf(\psi)}, Q', \delta', q'_{in}\rangle$ is the crossproduct of all the automata $\mathcal{U}^d_{cl(\xi)}$ above (for all the subformulas $E\xi$ of ψ). Intuitively, $\mathcal{U}$ supplies to $\mathcal{A}_\psi$ the information required in order to evaluate path formulas on paths that start in the root of the tree and visit the current node. When there is a need to check that $E\xi$ holds in some node x, the automaton $\mathcal{A}_\psi$ guesses a state $C \subseteq cl(\xi)$ such that $\xi \in C$ and C is a member of the current state $s \in 2^{Q_\xi}$ of $\mathcal{U}^d_\xi$ (recall that 2^{Q_ξ} is a component in the state space of the satellite) and it executes $\mathcal{U}^C_{cl(\xi)}$ along some path that starts in x. By Lemma 4.3, a correct guess of C and the path, as well as a successful run of $\mathcal{U}^C_{cl(\xi)}$ along that path, are possible iff the position $|x|$ in the path satisfies ξ, which corresponds to the semantics of $E\xi$.

As in the case of the HAA for CTL* [27], we construct $\mathcal{A}_\psi$ by induction on the structure of ψ. With each state subformula φ of ψ, we associate an HAA $\mathcal{A}'_\varphi$ composed from HAAs associated with state formulas maximal in φ. We assume that the state sets of composed HAAs are disjoint (otherwise, we rename states). The HAA $\mathcal{A}'_\varphi$ assumes that the tree is sound for the formulas in $max(\varphi)$ and only checks the satisfaction of φ under this assumption. We then define $\mathcal{A}_\psi$ as the intersection of $\mathcal{A}'_\psi$ with an automaton that checks, by sending copies to the different $\mathcal{A}'_\varphi$ automata, that the input tree is indeed sound with respect to ψ.

- If $\varphi = p$ for $p \in AP$, then $\mathcal{A}'_\varphi$ is the one-state HAA that goes to *true* when it reads σ with $p \in \sigma$ and goes to *false* otherwise.
- If $\varphi = \neg\varphi_1$, then $\mathcal{A}'_\varphi$ is $\tilde{\mathcal{A}}_{\varphi_1}$ — the HAA obtained by dualizing the HAA $\mathcal{A}_{\varphi_1}$ for φ_1. If $\varphi = \varphi_1 \vee \varphi_2$, then $\mathcal{A}'_\varphi$ has an initial state that sends all the copies sent by $\mathcal{A}'_{\varphi_1}$ or all the copies sent by $\mathcal{A}'_{\varphi_2}$. Formally, $\mathcal{A}'_\varphi = \langle 2^{sf(\psi)}, Q^1 \cup Q^2 \cup \{q_{in}\}, \delta, q_{in}, \langle G^1 \cup G^2, B^1 \cup B^2\rangle\rangle$, where $\mathcal{A}'_{\varphi_i} = \langle 2^{sf(\psi)}, Q^i, \delta^i, q^i_{in}, \langle G^i, B^i\rangle\rangle$, q_{in} is a new state and δ is defined as follows. For states in Q^1 and Q^2, the transition function δ agrees with δ^1 and δ^2, respectively. For the state q_{in} and for all $\sigma \in 2^{sf(\psi)}$ and $q' \in Q'$, we have $\delta(q_{in}, \sigma, q') = \delta(q^1_{in}, \sigma, q') \vee \delta(q^2_{in}, \sigma, q')$. Thus, in the state q_{in}, the HAA $\mathcal{A}'_\varphi$ sends all the copies sent by $\mathcal{A}'_{\varphi_1}$ or all the copies sent by $\mathcal{A}'_{\varphi_2}$. The singleton $\{q_{in}\}$ constitutes a transient set, with the ordering $\{q_{in}\} > Q_i$ for all the sets Q_i in Q^1 and Q^2.
- If $\varphi = E\xi$, where ξ is a $\mathrm{CTL}^\star_{lp}$ path formula, we proceed as follows. Let $\mathcal{U}_{cl(\xi)} = \langle 2^{sf(\psi)}, Q_\xi, M_\xi, q^{in}_\xi, \alpha_\xi\rangle$ be the nondeterministic generalized Büchi automaton that corresponds to ξ, as defined in Lemma 4.3. We now translate $\mathcal{U}_{cl(\xi)}$ to a Büchi (rather than generalized Büchi) automaton. Technically, this means that we have to take k copies of the state space, for the number k of sets in α_ξ, thus the states in Q_ξ are no longer in $2^{cl(\xi)}$ and are rather in $2^{cl(\xi)} \times \{1, \ldots, k\}$ [38]. For simplicity, we still use the notation $\varphi \in s$, for a subformula φ and a state $s \in Q_\xi$, to indicate that $\varphi \in s'$, for the state s' for which $s \in \{s'\} \times \{1, \ldots, k\}$. Recall that the state space Q' of the satellite $\mathcal{U}$ is the product of the state spaces of

the deterministic automata for the path formulas. Thus, each state $q' \in Q'$ has a component for each of these automata, and in particular for $\mathcal{U}^d_{cl(\xi)}$. Consider a state $q' \in Q'$. Let $q'_{|\xi}$ be the state of $\mathcal{U}^d_{cl(\xi)}$ in q'. Note that $q'_{|\xi} \in 2^{Q_\xi}$. Then, $\mathcal{A}'_\varphi = \langle 2^{sf(\psi)}, Q_\xi, \delta', q_{in}, \langle \alpha_\xi, \emptyset \rangle \rangle$ is defined so that from its initial state q_{in}, it consults the satellite's state q' and executes $\mathcal{U}_{cl(\xi)}$ along a single path, starting from some state in $q'_{|\xi}$ that contains ξ. Formally, for all $\sigma \in 2^{sf(\psi)}$ and $q' \in Q'$, we have

$$\delta'(q_{in}, \sigma, q') = \bigvee_{s \in q'_{|\xi} : \xi \in s} \bigvee_{s' \in M_\xi(s,\sigma)} \Diamond s'.$$

Also, for all $q \in Q_\xi$, we have $\delta'(q, \sigma, q') = \bigvee_{q_i \in M_\xi(q,\sigma)} \Diamond q_i$. If $M_\xi(q, \sigma) = \emptyset$, then $\delta(q, \sigma, q') = \mathit{false}$. Note that the only transition in which the input from the satellite is taken into an account is the transition from q_{in}, where $\mathcal{A}'_\varphi$ chooses a state from $q_{|\xi}$ to proceed with. Note also that Q_ξ constitutes a single existential set. The Büchi acceptance condition of $\mathcal{U}^d_{cl(\xi)}$ requires visiting accepting states of $\mathcal{U}^d_{cl(\xi)}$ infinitely often. Thus, the accepting states of $\mathcal{U}^d_{cl(\xi)}$ go into the set G of $\mathcal{A}'_\varphi$. The HAA $\mathcal{A}'_\varphi$ accepts a $2^{sf(\psi)}$-labeled tree from node x iff x satisfies φ, assuming the input tree is sound for the formulas in $\mathit{max}(\varphi)$.

We now add to $\mathcal{A}'_\psi$ transitions that check that the input tree is indeed sound for ψ, thus the letter read at node x describes the set of formulas satisfied in x. For this purpose, we add a new state q_{check}. Whenever $\mathcal{A}_\psi$ is in state q_{check} and reads a letter σ, it sends copies sent from the initial states of the HAA $\mathcal{A}'_{\varphi_i} = \langle 2^{sf(\psi)}, Q^i, \delta^i, q^i_{in}, \langle G^i, B^i \rangle \rangle$, for all $\varphi_i \in \sigma$, sends copies sent from the initial states of the HAA $\tilde{\mathcal{A}}'_{\varphi_i} = \langle 2^{sf(\psi)}, \tilde{Q}^i, \tilde{\delta}^i, q^i_{in}, \langle B^i, G^i \rangle \rangle$, for all $\varphi_i \notin \sigma$ (and also sends a copy that stays in q_{check} to all the successors). Formally, for all $\sigma \in 2^{sf(\psi)}$ and $q' \in Q'$, we have

$$\delta(q_{check}, \sigma, q') = \Box q_{check} \wedge \bigwedge_{\varphi_i \in \sigma} \delta^i(q^i_{in}, \sigma, q') \wedge \bigwedge_{\varphi_i \notin \sigma} \tilde{\delta}^i(q^i_{in}, \sigma, q').$$

The HAA $\mathcal{A}_\psi = \langle 2^{sf(\psi)}, Q, \delta, q_{in}, \langle G, B \rangle \rangle$ is such that Q is the union of $\{q_{in}, q_{check}\}$ with the union of the state spaces of $\mathcal{A}'_\theta$, for $\theta \in sf(\psi)$. The initial state q_{in} checks for the satisfaction of ψ and for the soundness with respect to ψ, thus $\delta(q_{in}, \sigma, q') = \delta^\psi(q^\psi_{in}, \sigma, q') \wedge \delta(q_{check}, \sigma, q')$, where δ^ψ and q^ψ_{in} are the transition function and initial state of $\mathcal{A}'_\psi$. Finally, $G = \bigcup_{\varphi_i \in sf(\psi)} G^i$ and $B = \bigcup_{\varphi_i \in sf(\psi)} B^i$. The state q_{in} is transient and the state q_{check} constitute a singleton universal set of the HAA.

The arguments about the correctness of the construction, its size, and its depth, are similar to these in [27,25]. □

For satisfiability, we can simply check the automaton $\mathcal{A}_\psi$ for nonemptiness; see Section 6. Note that $\mathcal{A}_\psi$ runs on $2^{sf(\psi)}$-labeled trees. For model checking, however, we need automata that run on 2^{AP}-labeled trees.

Theorem 4.5. *Given a CTL$^\star_{lp}$ formula ψ, we can construct an HAA $\mathcal{A}^{AP}_\psi$ with $2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states, such that $\mathcal{A}^{AP}_\psi$ runs on 2^{AP}-labeled trees and accepts exactly all trees that satisfy ψ.*

Proof. The construction is similar to the one described in Theorem 4.4, only that we have to adjust the HAA and its satellite to the alphabet 2^{AP}. In the proof of Theorem 4.4 we took advantage of the fact that the nodes of the input tree are labeled by subsets of $sf(\psi)$. In many automata-theoretic constructions, cf. [14], the automata itself guess the labeling. The difficult here is that we cannot expect an HAA to guess a labeling, as different branches of the run tree may take different guesses. Instead, we let the satellite guess the richer labels, and then let the HAA check the guess. Thus, the satellite is nondeterministic — on top of its deterministic transition we add a guess of the subset of $sf(\psi)$ to be read in the successor node. Yet, running the HAA on a 2^{AP}-labeled tree, and letting it check the guesses, guarantees that the word w_x leading to node x can be viewed as a word in $(2^{sf(\psi)})^*$ rather than a word in $(2^{AP})^*$. Accordingly, $\delta'(q_{in}, w_x)$ is a singleton, as in the case of a deterministic satellite.

Formally, if in the construction in Theorem 4.4 we ended up with a satellite $\mathcal{U} = \langle 2^{sf(\psi)}, Q', \delta', q'_{in} \rangle$ and HAA $\mathcal{A}_\psi = \langle 2^{sf(\psi)}, Q, \delta, q_{in}, \langle G, B \rangle \rangle$, now we have a satellite $\mathcal{U}_{AP} = \langle 2^{AP}, Q' \times 2^{sf(\psi)}, \delta'_{AP}, q'_{in} \rangle$ where for all $\langle q', \sigma \rangle \in Q' \times 2^{sf(\psi)}$ and $\sigma' \in 2^{AP}$, we have $\delta'_{AP}(\langle q', \sigma \rangle, \sigma') = \{\delta'(q', \sigma)\} \times 2^{sf(\psi)}$, and $\mathcal{A}^{AP}_\psi = \langle 2^{AP}, Q, \delta^{AP}, q_{in}, \langle G, B \rangle \rangle$, where for all $q \in Q$, $\sigma' \in 2^{AP}$, and $\langle q', \sigma \rangle \in Q' \times 2^{sf(\psi)}$, we have $\delta^{AP}(q, \sigma', \langle q', \sigma \rangle) = \delta(q, \sigma, q') \wedge \bigwedge_{\varphi_i \in \sigma} \delta^i(q^i_{in}, \sigma, q') \wedge \bigwedge_{\varphi_i \notin \sigma} \tilde{\delta}^i(q^i_{in}, \sigma, q')$. □

Recall that in CTL$_{lp}$ formulas, every temporal operator is preceded by a path quantifier. When, however, the temporal operators are Y or S (that is, refer to the past), the path quantifiers are redundant and nesting of states formulas whose temporal operators are past operators amounts to a state formula whose path formula is an LTL$_p$ formula that has only past-time operators. As we shall see now, for such formulas we can construct deterministic satellites of only an exponential size. The proof is identical to the one of Lemma 4.3, and we only have to argue that the constructed automaton is deterministic. This easily follows from the fact that the only non-propositional formulas in $cl(\psi)$ are of the form $Y\psi_1$ or $\psi_1 S \psi_2$. Indeed, in

the definition of δ in the proof, membership of formulas of the form $Y\psi_1$ in C' is determinized uniquely, and membership of formulas of the form $\psi_1 S \psi_2$ in C' is determined uniquely once the membership of ψ_1 and ψ_2 is determined. (Here we use the fact that the run of the automaton is "shifted to the right by one" after starting from the initial state q_0. Had we not required this shift, we'd had to guess C_0, losing the determinism of the automaton.)

Lemma 4.6. *Consider an LTL$_p$ formula ψ that has only past-time operators. There is a deterministic Büchi automaton $\mathcal{U}_{cl(\psi)} = \langle 2^{AP}, \{q_0\} \cup Q, \delta, q_0, \alpha\rangle$ such that $Q \subseteq 2^{cl(\psi)}$, and if $w \in (2^{AP})^\omega$ and $q_0, C_0, C1, \ldots$ is an accepting run of $\mathcal{U}_{cl(\psi)}$ on w, then for each $\theta \in cl(\psi)$ and each $i \geqslant 0$, we have that $\theta \in C_i$ iff $w, i \models \theta$.*

By [27], it is possible to translate a CTL formula ψ to an HAA with linearly many states. Each state of the HAA is associated with a subformula in $cl(\psi)$ and the transitions follow the semantics of CTL. Having defined satellites for pure-past LTL$_p$ formulas, the translation for CTL$_{lp}$ is then similar, only that whenever the HAA is in a state associated with a past formula it consults the satellite in order to know whether the formula is satisfied.

Theorem 4.7. *Given a CTL$_{lp}$ formula ψ, we can construct an HAA $\mathcal{A}^{AP}_{\psi}$ with $O(|\psi|)$ states, depth $O(|\psi|)$, and a satellite with $2^{O(|\psi|)}$ states, such that $\mathcal{A}^{AP}_{\psi}$ runs on 2^{AP}-labeled trees and accepts exactly all trees that satisfy ψ.*

4.3. From CTL$^\star_{bp}$ and CTL$_{bp}$ to HAAs with satellites

Since tree automata run on trees (rather than on structures with a branching past), an automata-theoretic framework for temporal logic with branching past requires some form of a tree-model property for CTL$^\star_{bp}$. In order to establish a tree-model property for CTL, one simply unwinds Kripke structures, as is done, for example, in [14]. Such unwinding is sufficient to capture linear past, as any node in the tree is reachable by a unique path from the root. In order to handle branching past, we need an augmented unwinding technique used in [14,31]. The idea is that each node x in the tree, which corresponds to a state s in a Kripke structure K, has as children in the tree both its successors and predecessors in K. That is, if the set of successors and predecessors of w in K is $s_1, s_2, \ldots$, then x has children $x_1, x_2, \ldots$, which correspond to $s_1, s_2, \ldots$. In order to distinguish between successors and predecessors, we add a special proposition *succ*. If a tree node x correspond to a state s, then a child y of x corresponds to a state that is a successor of s if *succ* is true at y. Otherwise, the state that corresponds to y is a predecessor of s. We call the resulting tree a *two-way* tree. So a two-way tree is simply a labeled tree with the special proposition *succ*. We denote by $\langle T'_K, V'_K\rangle$ the two-way computation tree obtained by unwinding K both forward and backward.

Because we unwind both forwards and backwards, a state s in K may correspond to several nodes in $\langle T'_K, V'_K\rangle$. Of special interest to us are nodes in $\langle T'_K, V'_K\rangle$ that correspond to the initial state s^0 of K. We mark all these nodes by a special proposition *init*. Recall that the initial state s^0 does not have predecessors. In our unwound trees, nodes marked with *init* may have predecessor nodes (that is, successors not labeled by *succ*), but such nodes will be ignored by the automaton and play no role. (Allowing nodes marked with *init* to have predecessors make the construction simpler.) Note that since all nodes marked with *init* correspond to the initial state s^0, they all need to satisfy the same state subformulas of the input formula ψ. That is, if $\langle T'_K, V'_K\rangle$ is labeled by subformulas of ψ rather than only atomic propositions, then the label of all these nodes have to agree with $V'_K(\varepsilon)$.

We can now construct alternating automata on two-way trees that use satellites in order to keep track of satisfaction of subformulas along the computation from the root and take care of the other computations in the past by traversing paths along backward nodes. Since our motivation is a solution to the satisfiability problem, the hesitation structure of the automaton is not crucial (in the constructions for temporal logics with linear past, the motivation is model checking, and the hesitation condition is crucial for obtaining tight space complexity bounds). Still, in order to keep the translation similar to the one for linear past, we construct HAA. Also, since we care for satisfiability, we do not have to go all the way to automata with alphabet 2^{AP} and we construct automata whose alphabet refer to satisfaction of subformulas, along with *init* and *succ*.

Theorem 4.8. *Given a CTL$^\star_{bp}$ formula ψ, we can construct an HAA $\mathcal{A}_\psi$ with $2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states, such that $\mathcal{A}_\psi$ runs on $2^{sf(\psi)\cup\{succ,init\}}$-labeled two-way trees and accepts exactly all two-way trees that are sound for ψ and are two-way unwindings of Kripke structures that satisfy ψ.*

Proof. Let $\mathcal{U}_\psi$ be a satellite for the path subformulas of ψ. In addition to the standard behavior of a satellite, as described in the proof of Theorem 4.4, the satellite here also remembers the label $V(\varepsilon)$. Whenever $\mathcal{U}_\psi$ reads a node labeled with *init*, the HAA $\mathcal{A}_\psi$ checks that its label agrees with $V(\varepsilon)$. If there is disagreement, the satellite goes into a special rejecting state. This ensures that all nodes labeled with *init* satisfy precisely the same state subformulas.

As in the proof of Theorem 4.4, the definition of $\mathcal{A}_\psi$ is by induction on the structure of ψ. The only difference is in the handling of formulas of the form $E\xi$. Consider a subformula $\varphi = E\xi$. In contrast to CTL$^\star_{lp}$, where $\mathcal{A}_\psi$ examined the single path that started at the root of the tree, here $\mathcal{A}_\psi$ has to examine all paths that start at an initial node. The automaton $\mathcal{A}_\psi$

disjunctively checks if one of the following two possibilities holds. First, ξ may hold on the path that started at the root. This can be checked by consulting the satellite, as in Theorem 4.4. Second, ξ may hold on another path that starts at an initial node. This can be checked as follows. The HAA $\mathcal{A}_\psi$ guesses a state C of the satellite that contains ξ and it launches two copies conjunctively. One copy simulates $\mathcal{U}_\xi$ starting from the state C and moving "forward" (that is, this copy proceeds only along nodes that are labeled by *succ*), and it accepts if the acceptance conditions α_ξ is satisfied. The other copy also starts at the state C, but it goes "backwards" (that is, this copy proceeds only along nodes that are not labeled by *succ*), and it simulates U_ξ backwards, by reversing the direction of its transitions. This copy accepts if it reaches a node labeled *init* in an initial state of $\mathcal{U}_\xi$. Note that both copies accept iff there is a path that starts in an initial state for which ξ is satisfied in the position that corresponds to the current node.

The analysis of the number of states of $\mathcal{A}_\psi$ is similar to the one in Theorem 4.4, except that here the satellite checks the consistency among all nodes labeled by *init*, and thus has to remember the label $V(\varepsilon)$. Since there are exponentially many such labels, it does not blow up the state space of the satellite. □

Note that checking $E\xi$, the path leading to the current node may be checked in two different ways: both by the satellite and by following it when the HAA follows $\mathcal{U}_\xi$ in a backward manner. One may be tempted to think that the second way is sufficient and we do not need a satellite. To see why the satellite is essential, one has to remember that when we dualize the automaton in order to get an HAA for $A\neg\xi$, we have to make sure that the universal quantifier applies to the path from the root. Consider for example the CTL_{bp} formula $\psi = EX(p \wedge EXAY\neg p)$. Clearly, ψ is not satisfiable. By checking only the second way of satisfaction, the universal quantifier in $AY\neg p$ is not forced to include the path from the root, and may proceed only along backward path in which $\neg p$ holds.

A similar translation holds for CTL_{bp}. In fact, here, the satellite for each subformula is of size at most 2. Hence we have the following.

Theorem 4.9. *Given a CTL_{bp} formula ψ, we can construct an HAA $\mathcal{A}_\psi$ with $O(|\psi|)$ states, depth $O(|\psi|)$, and a satellite with $2^{O(|\psi|)}$ states, such that $\mathcal{A}_\psi$ runs on $2^{sf(\psi)\cup\{succ,init\}}$-labeled two-way trees and accepts exactly all trees that are sound for ψ and are two-way unwinding of Kripke structures that satisfy ψ.*

5. Model checking

The model-checking problem for a variety of branching temporal logics can be stated as follows: given a branching temporal logic formula φ and a finite Kripke structure $K = \langle W, R, s^0, L\rangle$, determine whether K satisfies φ. When some of the logics are sensitive to unwinding, there are two possible interpretations of this problem. The first interpretation, which is the one appropriate to branching past, asks whether $s^0 \models \varphi$. In the second interpretation, which is the one appropriate to linear past, we are given φ and K and are asked to determine whether $\langle T_K, V_K\rangle \models \varphi$. In this section we consider model-checking complexity for the two interpretations. We start with branching past, which does not make the model-checking problem more difficult:

Theorem 5.1. *The model-checking problem for CTL_{bp} is in linear time.*

Proof. We present a model-checking procedure for CTL_{bp}. Our procedure is a simple extension of the efficient model-checking procedure for CTL in [21], and is of complexity linear in both the length of the formula and the size of the Kripke structure being checked. As there, the algorithm labels with a formula φ exactly all the states that satisfy φ. This is done by recursively labeling the Kripke structure with the subformulas of φ. Once the Kripke structure is labeled with the subformulas of φ, it is possible to label it also with φ. Handling of past-time connectives is symmetric to the one suggested in [21] for future-time connectives, switching successors and predecessors. Careful attention, however, should be paid to the fact that past is finite and initialized. For example: when past is finite, a state s for which there exists a path from s^0 to *is* such that all the states in this path satisfy φ_2, satisfies $E\varphi_1\tilde{S}\varphi_2$. Accordingly, labeling s^0 with a fresh atomic proposition *init*, we have $E\varphi_1\tilde{S}\varphi_2 \sim E\varphi_2 S(\varphi_2 \wedge (init \vee \varphi_1))$. Thus, the connectives $E\tilde{S}$ is handled using the same procedure that handles the connectives ES. □

Theorem 5.2. *The model-checking problem for $\mathrm{CTL}^\star_{bp}$ is PSPACE-complete.*

Proof. Hardness in PSPACE follows from hardness of the model-checking problem for CTL*. To prove membership in PSPACE, we present a PSPACE model-checking algorithm. Our algorithm uses the PSPACE model-checking algorithm for LTL_p [8] and it is based on the method of reducing branching-time model checking to linear-time model checking [23]. According to this method, nested formulas of the form $E\xi$ are evaluated by recursive descent. For example, in order to model check $EXEXGp$, we first model check $EXGp$ using the model checker for LTL and label every state that satisfies it with a fresh atomic proposition q. Then, we model check EXq. In order to adopt this method for $\mathrm{CTL}^\star_{bp}$, we should guarantee that the model checker for LTL_p considers only paths that start in the initial state. This can be easily done by labeling the initial state with a fresh atomic proposition *init* and conjuncting each linear-time formula checked with $Pinit$. For example, in order to

check $EXEXPp$, we first model check, in PSPACE, the formula $E((XPp) \wedge (Pinit))$ and label every state that satisfies it with a fresh atomic proposition q. Then we model check, again in PSPACE, the formula $E((Xq) \wedge (Pinit))$. It is easy to see that the overall complexity is PSPACE. As in [27], the space complexity in the structure is polylogarithmic. □

We can now move to linear past, where we apply the translation of CTL_{lp} and $\mathrm{CTL}^{\star}_{lp}$ formulas to HAA.

Theorem 5.3. *The model-checking problem for CTL_{lp} is PSPACE-complete.*

Proof. We start with the upper bound. Consider a CTL_{lp} formula ψ. By Theorem 4.7, there is an HAA $\mathcal{A}_\psi$ with $O(|\psi|)$ states, depth $O(|\psi|)$, and a satellite with $2^{O(|\psi|)}$ states, such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of computation trees satisfying ψ. Consider a Kripke structure K. By [27], K satisfies ψ iff the 1-letter HAA obtained by taking the product of K with $\mathcal{A}_\psi$ is not empty. The product has $O(|K| \cdot |\psi|)$ states, depth $O(|\psi|)$, and a satellite with $2^{O(|\psi|)}$ states. Thus, by Theorem 4.2, its 1-letter nonemptiness problem can be solved in space $|\psi| \cdot \log^2(|K| \cdot |\psi| \cdot 2^{O(|\psi|)})$, hence the PSPACE upper bound. Note that the space complexity is only polylogarithmic in the structure, and the polynomial dependency is in the length of the formula, which is usually much smaller.

We prove hardness in PSPACE using the same reduction used in [39] for proving that model checking for LTL is PSPACE-hard. There, Sistla and Clarke associate with a polynomial space Turing machine M and an input word w, a Kripke structure K and an LTL formula ψ, such that $K \models \psi$ iff M accepts w. The formula ψ uses the X operator to describe the possible successors of a configuration of M and uses the F operator to ensure that an accepting configuration is eventually reached. A similar formula, that uses the operators F and Y can be written in CTL_{bp}. The formula is of the form $EF\xi$, where ξ is a past LTL_p formula, asserting that the current configuration is accepting, and that it has been reached by a valid run of M on w. As ψ, the length of ξ is polynomial in M and w. □

Theorem 5.4. *The model-checking problem for $\mathrm{CTL}^{\star}_{lp}$ is EXPSPACE-complete.*

Proof. The lower bound follows from the EXPSPACE-hardness results for model checking mCTL*, a memoryful branching-time logic studied in [25], as it is shown there that mCTL* can be viewed as a fragment of $\mathrm{CTL}^{\star}_{lp}$. See also [26]. For the upper bound, consider a $\mathrm{CTL}^{\star}_{lp}$ formula ψ. By Theorem 4.5, there is an HAA $\mathcal{A}_\psi$ with $2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states, such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of computation trees satisfying ψ. Consider a Kripke structure K. By [27], K satisfies ψ iff the 1-letter HAA obtained by taking the product of K with $\mathcal{A}_\psi$ is not empty. The product has $|K| \cdot 2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states. Thus, by Theorem 4.2, its 1-letter nonemptiness problem can be solved in space $|\psi| \cdot \log^2(|K| \cdot |\psi| \cdot 2^{2^{O(|\psi|)}})$, and hence the EXPSPACE complexity. Again, note that the exponential complexity is only in the formula, and that the dependency in the size of the structure is only polylogarithmic. □

6. Satisfiability

As with model checking, there are two interpretations of the satisfiability problem for a branching temporal logic that is sensitive to unwinding. In the first interpretation, which is the one appropriate to linear past, we are given φ and are asked to determine whether there exists a computation tree $\langle T, V\rangle$ such that $\langle T, V\rangle \models \varphi$. The second interpretation, which is the one appropriate to branching past, asks whether there exist a Kripke structure K and a state s^0 in it, such that s^0 has no predecessors and $s^0 \models \varphi$. In this section we consider satisfiability complexity for the two interpretations. We show that adding past to CTL and CTL*, regardless of the interpretation, does not change the complexity of the satisfiability problem.

Theorem 6.1. *The satisfiability problems for CTL_{lp} and CTL_{bp} are EXPTIME-complete.*

Proof. For both logics, hardness in EXPTIME follows from hardness of the satisfiability problem for CTL [40].

To prove membership in EXPTIME, consider a CTL_{lp} (or a CTL_{bp}) formula ψ. By Theorem 4.7 (respectively, Theorem 4.9), there is an HAA $\mathcal{A}_\psi$ with $O(|\psi|)$ states and a satellite with $2^{O(|\psi|)}$ states such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of $2^{sf(\psi)}$-labeled trees that are sound for ψ and satisfy ψ. The HAA $\mathcal{A}_\psi$ is nonempty iff ψ is satisfiable. By Theorem 4.2, the nonemptiness of $\mathcal{A}_\psi$ can be decided in time $2^{O(|\psi|)}$, so we are done. □

Theorem 6.2. *The satisfiability problems for $\mathrm{CTL}^{\star}_{lp}$ and $\mathrm{CTL}^{\star}_{bp}$ are 2EXPTIME-complete.*

Proof. Hardness in EXPTIME follows from hardness of the satisfiability problem for CTL* [41]. To prove membership in 2EXPTIME, consider a $\mathrm{CTL}^{\star}_{lp}$ (or a $\mathrm{CTL}^{\star}_{bp}$) formula ψ. By Theorem 4.4 (respectively, Theorem 4.8), there is an HAA $\mathcal{A}_\psi$ with $2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of $2^{sf(\psi)}$-labeled trees that

are sound for ψ and satisfy ψ. The HAA $\mathcal{A}_\psi$ is nonempty iff ψ is satisfiable. By Theorem 4.2, the nonemptiness of $\mathcal{A}_\psi$ can be decided in time $2^{2^{O(|\psi|)}}$, so we are done. □

We note that for $\mathrm{CTL}^\star_{lp}$, the satisfiability problem was solved also in [26]; we include here a proof based on our framework for the sake of completeness.

References

[1] A. Pnueli, The temporal logic of programs, in: Proc. 18th IEEE Symp. on Foundations of Computer Science, 1977, pp. 46–57.
[2] L. Lamport, "Sometimes" is sometimes "not never" — On the temporal logic of programs, in: Proc. 7th ACM Symp. on Principles of Programming Languages, 1980, pp. 174–185.
[3] A. Prior, Time and Modality, Oxford University Press, 1957.
[4] J. Kamp, Tense logic and the theory of order, PhD thesis, UCLA, 1968.
[5] D. Gabbay, A. Pnueli, S. Shelah, J. Stavi, On the temporal analysis of fairness, in: Proc. 7th ACM Symp. on Principles of Programming Languages, 1980, pp. 163–173.
[6] O. Lichtenstein, A. Pnueli, L. Zuck, The glory of the past, in: Logics of Programs, in: Lecture Notes in Comput. Sci., vol. 193, Springer, 1985, pp. 196–218.
[7] N.M.F. Laroussinie, P. Schnoebelen, Temporal logic with forgettable past, in: Proc. 17th IEEE Symp. on Logic in Computer Science, 2002, pp. 383–392.
[8] M. Vardi, A temporal fixpoint calculus, in: Proc. 15th ACM Symp. on Principles of Programming Languages, 1988, pp. 250–259.
[9] F. Laroussinie, P. Schnoebelen, Specification in CTL + past for verification in CTL, Inform. and Comput. 156 (1–2) (2000) 236–263.
[10] O. Kupferman, A. Pnueli, Once and for all, in: Proc. 10th IEEE Symp. on Logic in Computer Science, 1995, pp. 25–35.
[11] V. Pratt, Semantical considerations on Floyd–Hoare logic, in: Proc. 17th IEEE Symp. on Foundations of Computer Science, 1976, pp. 109–121.
[12] R. Street, Propositional dynamic logic of looping and converse, in: Proc. 13th ACM Symp. on Theory of Computing, 1981, pp. 375–383.
[13] M. Vardi, The taming of converse: Reasoning about two-way computations, in: Logic of Programs Workshop, vol. 193, Springer, 1985, pp. 413–424.
[14] M. Vardi, P. Wolper, Automata-theoretic techniques for modal logics of programs, J. Comput. System Sci. 32 (2) (1986) 182–221.
[15] S. Pinter, P. Wolper, A temporal logic for reasoning about partially ordered computations, in: Proc. 3rd ACM Symp. on Principles of Distributed Computing, 1984, pp. 28–37.
[16] C. Stirling, Comparing linear and branching time temporal logics, in: B. Banieqbal, H. Barringer, A. Pnueli (Eds.), Temporal Logic in Specification, vol. 398, Springer, 1987, pp. 1–20.
[17] T. Hafer, W. Thomas, Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree, in: Proc. 14th Int. Colloq. on Automata, Languages, and Programming, in: Lecture Notes in Comput. Sci., vol. 267, Springer, 1987, pp. 269–279.
[18] F. Laroussinie, P. Schnoebelen, A hierarchy of temporal logics with past, Theoret. Comput. Sci. 148 (2) (1995) 303–324.
[19] A. Zanardo, J. Carno, An ockhamist computational logic: past-sensitive necessitation in CTL*, J. Logic Comput. 3 (3) (1993) 249–268.
[20] E. Emerson, E. Clarke, Characterizing correctness properties of parallel programs using fixpoints, in: Proc. 7th Int. Colloq. on Automata, Languages, and Programming, 1980, pp. 169–181.
[21] E. Clarke, E. Emerson, A. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Trans. Program. Lang. Syst. 8 (2) (1986) 244–263.
[22] E. Emerson, J. Halpern, Decision procedures and expressiveness in the temporal logic of branching time, J. Comput. System Sci. 30 (1985) 1–24.
[23] E. Emerson, C.-L. Lei, Modalities for model checking: Branching time logic strikes back, in: Proc. 12th ACM Symp. on Principles of Programming Languages, 1985, pp. 84–96.
[24] E. Emerson, A.P. Sistla, Deciding branching time logic, in: Proc. 16th ACM Symp. on Theory of Computing, 1984, pp. 14–24.
[25] O. Kupferman, M. Vardi, Memoryful branching-time logics, in: Proc. 21st IEEE Symp. on Logic in Computer Science, 2006, pp. 265–274.
[26] L. Bozzelli, The complexity of CTL* + linear past, in: Proc. 11th Int. Conf. on Foundations of Software Science and Computation Structures, in: Lecture Notes in Comput. Sci., vol. 4962, Springer, 2008, pp. 186–200.
[27] O. Kupferman, M. Vardi, P. Wolper, An automata-theoretic approach to branching-time model checking, J. ACM 47 (2) (2000) 312–360.
[28] D. Gabbay, The declarative past and imperative future, in: B. Banieqbal, H. Barringer, A. Pnueli (Eds.), Temporal Logic in Specification, in: Lecture Notes in Comput. Sci., vol. 398, Springer, 1987, pp. 407–448.
[29] E. Emerson, J. Halpern, Sometimes and not never revisited: On branching versus linear time, J. ACM 33 (1) (1986) 151–178.
[30] P. Wolper, Temporal logic can be more expressive, Information and Control 56 (1–2) (1983) 72–99.
[31] M. Vardi, Reasoning about the past with two-way automata, in: Proc. 25th Int. Colloq. on Automata, Languages, and Programming, in: Lecture Notes in Comput. Sci., vol. 1443, Springer, Berlin, 1998, pp. 628–641.
[32] W. Thomas, Automata on infinite objects, in: Handbook of Theoretical Computer Science, Elsevier Science Publishers, 1990, pp. 133–191.
[33] D. Muller, P. Schupp, Alternating automata on infinite trees, Theoret. Comput. Sci. 54 (1987) 267–276.
[34] D. Muller, A. Saoudi, P.E. Schupp, Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time, in: Proc. 3rd IEEE Symp. on Logic in Computer Science, 1988, pp. 422–427.
[35] T. Wilke, CTL^+ is exponentially more succinct than CTL, in: Proc. 19th Conf. on Foundations of Software Technology and Theoretical Computer Science, in: Lecture Notes in Comput. Sci., vol. 1738, Springer, 1999, pp. 110–121.
[36] M. Vardi, P. Wolper, Reasoning about infinite computations, Inform. and Comput. 115 (1) (1994) 1–37.
[37] R. Gerth, D. Peled, M. Vardi, P. Wolper, Simple on-the-fly automatic verification of linear temporal logic, in: P. Dembiski, M. Sredniawa (Eds.), Protocol Specification, Testing, and Verification, Chapman & Hall, 1995, pp. 3–18.
[38] Y. Choueka, Theories of automata on ω-tapes: A simplified approach, J. Comput. System Sci. 8 (1974) 117–141.
[39] A. Sistla, E. Clarke, The complexity of propositional linear temporal logic, J. ACM 32 (1985) 733–749.
[40] M. Fischer, R. Ladner, Propositional dynamic logic of regular programs, J. Comput. System Sci. 18 (1979) 194–211.
[41] M. Vardi, L. Stockmeyer, Improved upper and lower bounds for modal logics of programs, in: Proc. 17th ACM Symp. on Theory of Computing, 1985, pp. 240–251.