Once and *For All*

Orna Kupferman
Department of Computer Science
The Technion
Haifa 32000, Israel
E-mail: ornab@cs.technion.ac.il

Amir Pnueli
Department of Computer Science
Weizmann Institute of Science
Rehovot 76100, Israel
E-mail: amir@wisdom.weizmann.ac.il

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 [Pnu77]. 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 [Lam80]. 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 modalities that refer to both past and future [Pri57, Kam68]. Striving for minimality, computer scientists usually use temporal logics that provide only future-time modalities. Since program computations have a definite starting time and since, in this case, past-time modalities add no expressive power to linear temporal logics [GPSS80], this seems practical. Nevertheless, enriching linear temporal logics with past-time modalities makes the formulation of specifications more intuitive and does not increase the complexity of the validity and the model-checking problems [LPZ85, Var88].

Is the same true of branching temporal logics? Examining this question, we found in the literature several logics that extend branching temporal logics with past-time modalities. Yet, as we soon specify, we did not find a logic that meets our understanding of past in a branching-time model: we distinguish 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.

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 [Pra76], 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 [Str81, Var85, VW86]: 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* [VW86], 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 modalities [PW84], and in the reverse operators in [Sti87].

A logic *PCTL**, which augments the branching temporal logic CTL* with past-time modalities, is introduced in [HT87]. Formulas of PCTL* are interpreted over *computation trees*. Thus, PCTL* corresponds to the linear-past interpretation. However, the semantics of PCTL* makes the usage of past-time modalities very limited. Actually, past cannot go beyond the present. For example, the PCTL* formula $EXEYtrue$ ("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 [LS94]). Another augmentation of CTL* with past-time modalities is the *ockhamist computational logic* (OCL), presented in [ZC93]. We found the semantics of OCL unsatisfactory, as it is interpreted over structures which are not fusion closed.

In this paper, we consider the logics CTL$^\star_{bp}$ and CTL$^\star_{lp}$, as well as their sub-languages CTL$_{bp}$ and CTL$_{lp}$. Syntactically, CTL$^\star_{bp}$ and CTL$^\star_{lp}$ are exactly the same: both extend the full branching temporal logic CTL* with past-time modalities. Semantically, we have two completely different interpretations. Formulas of 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. Accordingly, we regard CTL$^\star_{bp}$ formulas as describing a single computation which is a partially ordered set [PW84]. 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 [PW84], 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, the CTL$^\star_{bp}$ formula $AG(term \rightarrow ((EPterm_1) \wedge \ldots \wedge (EPterm_n))$ states that the entire system can terminate only after all its n processes have terminated. Formulas of 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. Accordingly, we regard $\mathrm{CTL}^\star_{lp}$ 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 non-determinism or interleaving due to concurrency. For example, the $\mathrm{CTL}^\star_{lp}$ 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 $\mathrm{CTL}^\star$, which is insensitive to unwinding (that is, unwinding of a Kripke structure preserves the set of $\mathrm{CTL}^\star$ formulas it satisfies), augmenting a branching temporal logic with past-time modalities 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 modalities. As was shown in [GPSS80], past-time modalities 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 $\mathrm{CTL}^\star$.

We consider the *model-checking* and the *satisfiability* problems for the logics CTL_{bp} and CTL_{lp}. While the model-checking problem for CTL_{bp} is in linear time, as the one for CTL [CES86], augmenting CTL with linear past makes its model-checking problem PSPACE-hard. Typically, the gap follows from the fact that CTL_{lp} actually subsumes the expressive power of linear temporal logic. Since the linear-past (rather than the branching-past) approach corresponds to the natural way branching temporal logics have been used to represent computations, these news are, all in all, sad. For consolation, we show that branching past does not increase the cost also in the case of $\mathrm{CTL}^\star$, thus the model-checking problem for $\mathrm{CTL}^\star_{bp}$ is PSPACE-complete, as is the one for $\mathrm{CTL}^\star$ [EL85]. Comfort can also be found in the fact that the satisfiability problem for both CTL_{bp} and CTL_{lp} is EXPTIME-complete, as is the one for CTL. 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 modalities preserves its exponential satisfiability. Nevertheless, our procedure for CTL_{bp} demonstrates how the fact that past is finite makes life much easier. The procedure for CTL_{lp} is also very simple, using a translation of CTL_{lp} formulas into formulas in CTL augmented with existential quantification over history variables.

2 Branching logics with past operators

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

The logic $CTL^\star_{bp}$ extends $\mathrm{CTL}^\star$ by allowing past-time operators. As $\mathrm{CTL}^\star$, 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 $\mathrm{CTL}^\star_{bp}$: *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 $\mathrm{CTL}^\star_{bp}$ state formula is either:

- *True*, *False* (represented in the sequel as t and f, respectively), or p, for $p \in AP$.
- $\neg\varphi_1$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are $\mathrm{CTL}^\star_{bp}$ state formulas.
- $E\psi_1$, where ψ_1 is a $\mathrm{CTL}^\star_{bp}$ path formula.

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

- A $\mathrm{CTL}^\star_{bp}$ 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 $\mathrm{CTL}^\star_{bp}$ path formulas.

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

We use the following abbreviations in writing formulas:

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

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 negations must be immediately preceded by a path quantifier. CTL_{bp} allows also the temporal operators Y, S, and their negations. 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 negations, 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 $\langle w, w^0\rangle \in R$, and $L : W \rightarrow 2^{AP}$ 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 $i \geq 0$, 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 $w \in W$, $w \models t$ and $w \not\models f$.
- 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 $j \geq 0$ 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 $k \geq j$ such that $\pi, k \models \psi_2$ and $\pi, i \models \psi_1$ for all $j \leq i < k$.
- $\pi, j \models \psi_1 S \psi_2$ iff there exists $0 \leq k \leq j$ such that $\pi, k \models \psi_2$ and $\pi, i \models \psi_1$ for all $k < i \leq j$.

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. For a Kripke structure K, we say that $K \models \varphi$ iff $w^0 \models \varphi$.

We consider also the logic QCTL_{bp}, obtained by adding quantifiers to CTL_{bp}. 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. We use $\forall p \psi$ to abbreviate $\neg\exists p \neg\psi$.

2.2 Linear past: the logic $\mathrm{CTL}^\star_{lp}$

The logic $\mathrm{CTL}^\star_{lp}$ has the same syntax as $\mathrm{CTL}^\star_{bp}$. We define its semantic 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*. Below, we define formally computation trees and the semantics of $\mathrm{CTL}^\star_{lp}$.

A *tree* is a set $T \subseteq \mathbb{N}^*$ such that if $x \cdot c \in T$ where $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $x \in T$, and for all $0 \leq c' < c$, $x \cdot c' \in T$. 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 $c \in \mathbb{N}$ are the *successors* of x. We consider here trees in which each node has at least one successor. A *path* ρ of a tree T is a set $\rho \subseteq T$ such that $\epsilon \in \rho$ and for every $x \in \rho$ there exists a unique $c \in \mathbb{N}$ such that $x \cdot c \in \rho$. For a path ρ and $j \geq 0$, let ρ_j denote the node of length j in ρ. Given an alphabet Σ, a *Σ-labeled tree* is a pair $\langle T, V \rangle$ where T is a tree and $V : T \rightarrow \Sigma$ maps each node of T to a letter in Σ. A computation tree is a Σ-labeled 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 $x \models \varphi$ to indicate that a state formula φ holds at node $x \in T$. We use $\rho, j \models \psi$ to indicate that a path formula ψ holds in position j of the the path $\rho \subseteq T$. The relation $\models$ is defined similarly to the one of $\mathrm{CTL}^\star_{bp}$, taking a node x here instead a state w there, and a path ρ here, instead π there. In particular, we have:

- $x \models E\psi_1$ iff there exist a path ρ and $j \geq 0$ such that $\rho_j = x$ and $\rho, j \models \psi_1$.
- $\rho, j \models \varphi$ 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 be to 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 have in them a single, and outermost, future-time operator. We consider also the logic EQCTL_{lp}, obtained by adding existential quantifiers to CTL_{lp}. Precisely, if ψ is a CTL_{lp} formula and $p_1, \ldots, p_n$ are atomic propositions, then $\exists p_1 \ldots \exists p_n \psi$ is an EQCTL_{lp} formula. The semantics of $\exists p_1 \ldots \exists p_n \psi$ is given by $\langle T, V \rangle \models \exists p_1 \ldots \exists p_n \psi$ 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 from 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 exists a Kripke structure K and a formula φ such that $K \models \varphi$ and $\langle T_K, V_K \rangle \not\models \varphi$.

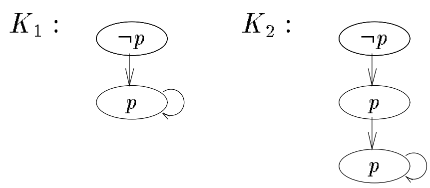

Figure 1: The Kripke structures K_1 and K_2.

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

Proof: Consider the Kripke structure K_1 appearing in Figure 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 $\langle T_{K_1}, V_{K_1} \rangle \models AF(p \wedge AYp)$. $\square$

Note that since $CTL^\star_{bp}$ 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 $w^0 \not\models EYt$ and $\epsilon \not\models EYt$. Clearly, for all $w \neq w^0$ and $x \neq \epsilon$, we have $w \models EYt$ and $x \models EYt$. Thus, both $CTL^\star_{bp}$ and $CTL^\star_{lp}$ can characterize the starting point of the past.

The logics QCTL and EQCTL are also sensitive to unwinding. Consider the formula $\varphi = \exists q AG(p \leftrightarrow AXq)$ and consider the Kripke structures K_1 and K_2 appearing in Figure 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

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 $K \models \varphi_2$. 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 $j \geq 0$, 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 ($L_1 \geq L_2$) 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 $L_1 \geq L_2$ and $L_2 \geq L_1$, and L_1 is *strictly more expressive than* L_2 ($L_1 > L_2$) if both $L_1 \geq L_2$ and $L_2 \not\geq L_1$.

In this section we consider the expressive power of branching temporal logics with past with respect to branching temporal logics without past. Our results are summarized in the hierarchy presented in Figure 2. In the Figure, we use $L_1 \leftarrow L_2$ to indicate that $L_1 > L_2$, we use $L_1 \leftrightarrow L_2$ to indicate that $L_1 = L_2$, and we use L_1 — L_2 to indicate that $L_1 \not\geq L_2$ and $L_2 \not\geq L_1$.

We first prove that $CTL^\star_{lp} = CTL^\star$. While $CTL^\star_{lp}$ is interpreted over computation trees, $CTL^\star$ is interpreted over Kripke structures. However, as $CTL^\star$ is insensitive to unwinding, this causes no difficulty. We use the Separation Theorem for LTL_p, quoted below.

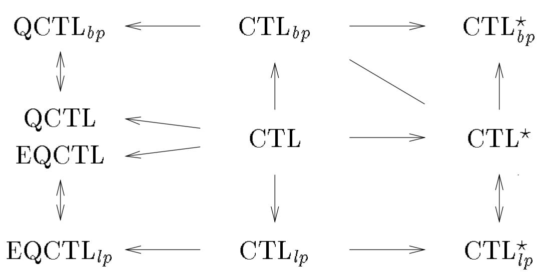

Figure 2: Hierarchy of expressive power.

Theorem 3.2 *[Gab87] Any LTL_p formula is congruent to a boolean combination of past and future LTL_p formulas.*

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

Proof: By the Separation Theorem, ψ is congruent to a boolean combination, ψ', of future and past LTL_p formulas. Without loss of generality, ψ' is of the form $\bigvee_{1 \leq i \leq n}(p_i \wedge q_i)$, where for all $1 \leq i \leq n$, p_i is a past LTL_p formula and q_i is a future LTL_p formula. As past is linear, path quantification over past LTL_p formulas can be eliminated using the congruences below.

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

□

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

Proof: Given a $CTL^\star_{lp}$ formula φ, we translate φ into an equivalent $CTL^\star$ 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 1 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 CTL* 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 f, replace pSq with q, and we end up with a CTL* formula. $\square$

As our semantics allows past to go beyond the present, Theorem 3.4 is much stronger than the PCTL* = CTL* result in [HT87]. In all the $L_1 < L_2$ relations in Theorems 3.5, 3.6, and 3.7 below, the $L_1 \leq L_2$ part follows by syntactic containment and we prove only strictness.

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

Proof: In [EH86], Emerson and Halpern show that the CTL* formula $AF(p \wedge Xp)$ has no equivalent of CTL. As $AXAF(p \wedge Yp) \sim AF(p \wedge Xp)$, we have CTL < 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 [Wol83], φ has no equivalent of CTL*. Hence, as CTL$_{lp}$ $\leq$ CTL*, we have CTL$_{lp}$ < EQCTL$_{lp}$.

To prove EQCTL$_{lp}$ = EQCTL, we prove that EQCTL$_{lp}$ $\leq$ 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 $j \geq 0$ 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. It is easy to see that the translation is linear. For example, the formula $AXAF(p \wedge Yp)$ is translated to the formula $\exists q AXAF(p \wedge q) \wedge \neg q \wedge AG(p \rightarrow AXq) \wedge AG(\neg p \rightarrow AX\neg q)$. $\square$

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 Figure 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.

$$label(\varphi, p) = spread(\varphi, p) \wedge$$

$$\forall r(spread(\varphi, r) \rightarrow AG(p \rightarrow r)).$$

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. As there, the translation is linear. 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. $\square$

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 [PW84]. 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 [EH86], Emerson and Halpern show that the $\mathrm{CTL}^\star$ formula $EGFp$ has no equivalent of CTL. It is easy to extend their proof to consider also CTL_{lp} and CTL_{bp}. Hence, $\mathrm{CTL}^\star_{lp} > \mathrm{CTL}_{lp}$ and $\mathrm{CTL}^\star_{bp} > \mathrm{CTL}_{bp}$. In the proof of Theorem 3.6, we point on a CTL_{bp} formula φ which distinguishes between a Kripke structure and its unwinding. Since $\mathrm{CTL}^\star$ is not sensitive to unwinding, φ has no equivalence of $\mathrm{CTL}^\star$. Hence, $\mathrm{CTL}^\star_{bp} > \mathrm{CTL}^\star$. $\square$

4 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 $K = \langle W, R, w^0, L\rangle$, determine whether K satisfies φ. When some of the logics are sensitive to unwinding, there are two possible interpretation of this problem. The first interpretation, which is the one appropriate to branching past, asks whether $w^0 \models \varphi$. 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.

Theorem 4.1 *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 [CES86], 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 modalities is symmetric to the one suggested in [CES86] for future-time modalities, switching successors and predecessors. Careful attention, however, should be payed to the fact that past is finite. While finiteness of the past does not influence checking of $E\varphi_1 S\varphi_2$, it does influence checking of $E\varphi_1 \tilde{S}\varphi_2$. When past is finite, a state w for which there exists a path from w^0 to w such that all the states in this path satisfy φ_2, satisfies $E\varphi_1\tilde{S}\varphi_2$. Accordingly, labeling w^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 modality $E\tilde{S}$ is handled using the same procedure that handles the modality ES. □

Theorem 4.2 *The model-checking problem for $CTL^\star_{bp}$ 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$_p$ [Var88] and it is based on the method of reducing branching-time model checking to linear-time model checking [EL85]. 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 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. □

Theorem 4.3 *The model-checking problem for CTL_{lp} is PSPACE-hard.*

Proof: We prove hardness in PSPACE using the same reduction used in [SC85] 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. □

5 Satisfiability

As with model checking, there are two interpretations of the satisfiability problem for a branching temporal logic which is sensitive to unwinding. The first interpretation, which is the one appropriate to branching past, asks whether there exists a Kripke structure K and a state w^0 in it, such that w^0 has no predecessors and $w^0 \models \varphi$. In the second 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 $\langle T, V\rangle \models \varphi$. In this section we consider satisfiability complexity for the two interpretations.

Theorem 5.1 *The satisfiability problem for CTL_{bp} is EXPTIME-complete.*

Proof: Hardness in EXPTIME follows from hardness of the satisfiability problem for CTL. To prove membership in EXPTIME we extend the tableau method for CTL used in [EH85]. The puzzle in this method lays in the fact that the quotient construction does not preserve modelhood. Let K' denote the quotient structure of a Kripke structure K. A formula of the form $A\varphi_1 U\varphi_2$ which is satisfied in K might not be satisfied in K' due to cycles introduced into it. Emerson and Halpern solve this puzzle by showing that if K is a model for a CTL formula φ, then K' is "almost" model for φ (a pseudo-Hintikka model), and that a model of size exponential in the size of φ can be constructed from it. This establishes a small model property for CTL and yields a tableau-based satisfiability procedure.

The fact that CTL$_{bp}$ assumes a finite past makes the extension of the above described method easy. The crucial point is that when past is finite, the quotient construction preserves modelhood with respect to all past-time modalities. Indeed, paths that get stuck in a cycle introduced into K' do not reach the initial state and can thus be ignored. Hence, if K is a model for a CTL$_{bp}$ formula, then K' is a pseudo-Hintikka model for it, in which the only eventualities that may not be fulfilled are of the form AU. We now sketch how, given K', we construct an exponential size model for φ from it. As a first step we construct a Kripke structure, K'', in which every state satisfies all its subformulas that have an outermost future modality (modulo the assumption that every state satisfies its subformulas that have an outermost past modality). This is done using the "matrix construction" for CTL [EH85]. Since each transition in K'' exists in K', then each state in K'' satisfies also all its subformulas that have an outermost and universal past modality. It is thus left to worry only about subformulas that have an outermost existential past modality. Some of these subformulas may be satisfied in K'' but for some it may be required to add transitions to K'. Still, these transitions are contained in the transitions of K' and can be added to K'' preserving the fulfillment of AU formulas. $\square$

Theorem 5.2 *The satisfiability problem for CTL$_{lp}$ is EXPTIME-complete.*

Proof: Hardness in EXPTIME follows from hardness of the satisfiability problem for CTL. To prove membership in EXPTIME we use the linear translation of CTL$_{lp}$ formulas into EQCTL formulas and reduce satisfiability of CTL$_{lp}$ to satisfiability of CTL. Let ψ be a CTL$_{lp}$ formula and let $\exists p_1 \ldots \exists p_n \varphi$ be its equivalent EQCTL formula. We show that ψ is satisfiable iff φ is satisfiable. Recall that if ψ is defined over the set AP of atomic propositions, then φ is defined over the set $AP \cup \{p_1, \ldots, p_n\}$. Also, while satisfaction of φ is checked with respect to Kripke structures, satisfaction of ψ is checked with respect to computation trees. Assume first that φ is satisfiable. Then, there exists a Kripke structure K such that $K \models \varphi$. Since CTL is not sensitive to unwinding, then $\langle T_K, V_K\rangle \models \varphi$. Clearly, $\langle T_K, V_K\rangle$ satisfies ψ too. Assume now that ψ is satisfiable. Then, there exists a computation tree $\langle T, V\rangle$ such that $\langle T, V\rangle \models \exists p_1 \ldots \exists p_n \varphi$. Hence, there exists a computation tree $\langle T, V'\rangle$, such that V' differs from V in at most the labeling of the p_i's and $\langle T, V'\rangle \models \varphi$. By the finite model property of CTL, this implies that there exists a Kripke structure in which φ is satisfied. $\square$

6 Discussion

Our complexity results are summarized in the table below:

	Model checking	Satisfiability
CTL$_{bp}$	linear time	EXPTIME-complete
CTL$_{lp}$	PSPACE-hard	EXPTIME-complete
CTL$^{\star}_{bp}$	PSPACE-complete	?
CTL$^{\star}_{lp}$	?	?

We would like to comment here on the problems which are still open (and which we hope to close in the full version). Probably the most interesting one is finding a tight bound for CTL$_{lp}$ model checking. While an EXPTIME upper bound is straightforward (e.g., by the linear translation to EQCTL), we did not find an EXPTIME lower bound. A good excuse for this is our conjecture that the problem is PSPACE-complete. In more detail, the automata-theoretic framework from [BVW94] can be used to reduce the model-checking problem for CTL$_{lp}$ to the 1-letter nonemptiness problem of weak alternating automata. Unlike CTL, where the automata are of linear size, the automata for CTL$_{lp}$ are of exponential size. We believe that, as with CTL, the structure of these automata enables a space-efficient 1-letter nonemptiness procedure. Automata-theoretic techniques can also be used, we conjecture, to achieve 2EXPTIME algorithms for the satisfiability problem of both CTL$^{\star}_{bp}$ and CTL$^{\star}_{lp}$, as well as for the model-checking problem for CTL$^{\star}_{lp}$. The idea, as suggested to us by

Moshe Vardi, is that each state of the automaton is associated not only with formulas that should hold on the future (as is usually the case with automata-theoretic techniques), but also with formulas whose satisfaction in the past is guaranteed.

References

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *CAV, Proc. 5th Int. Workshop*, 1994.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[EH85] E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30:1–24, 1985.

[EH86] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

[EL85] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *POPL, Proc. of the Twelfth ACM Symposium*, pages 84–96, 1985.

[Gab87] D. Gabbay. The declarative past and imperative future. In *Temporal Logic in Specification*, LNCS 398, 1987.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *POPL, Proc. of the 7th ACM Symposium*, pages 163–173, 1980.

[HT87] Th. Hafer and W. Thomas. Computation tree logic CTL*and path quantifiers in the monadic theory of the binary tree. In *Proc. 14th ICALP, volume 267 of LNCS*, pages 269–279, 1987.

[Kam68] J.A.W. Kamp. *Tense Logic and the Theory of Order*. PhD thesis, UCLA, 1968.

[Lam80] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *POPL, Proc. of the 7th ACM Symposium*, pages 174–185, 1980.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, LNCS 193, pages 196–218, 1985

[LS94] F. Laroussinie and Ph. Schnoebelen. A hierarchy of temporal logics with past. In *Proceedings of STACS*, 1994.

[Pnu77] A. Pnueli. The temporal logic of programs. In *FOCS, Proc. 18th IEEE Symposium*, pages 46–57, 1977.

[Pra76] V.R. Pratt. Semantical considerations on floyd-hoare logic. In *FOCS, Proc. of the 17th IEEE Symposium*, pages 109–121, 1976.

[Pri57] A. Prior. *Time and Modality*. Oxford University Press, 1957.

[PW84] S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations. In *PODC, Proc. of the 3rd ACM Symposium*, pages 28–37, 1984.

[SC85] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *J. ACM*, 32:733–749, 1985.

[Sti87] C. Stirling. Comparing linear and branching time temporal logics. In *Temporal Logic in Specification*, LNCS 398, pages 1-20, 1987.

[Str81] R.S. Street. Propositional dynamic logic of looping and converse. In *Proc. of the 13th ACM STOC*, pages 375–383, 1981.

[Var85] M.Y. Vardi. The timing of converse: Reasoning about two-way computations. In *Logics of Programs*, LNCS 193, pages 413–424, 1985

[Var88] M.Y. Vardi. A temporal fixpoint calculus. In *POPL Proc. 15th ACM Symposium*, pages 250–259, 1988.

[VW86] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–21, April 1986.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.

[ZC93] A. Zanardo and J. Carno. An ockhamist computational logic: past-sensitive necessitation in CTL*. *Journal of Logic and Computation*, 3(3):294–268, June 1993.