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 $\mathrm{LTL}_p$, 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 $\mathrm{CTL}^*_{lp}$, satisfiability of $\mathrm{CTL}^*_{lp}$, and satisfiability of $\mathrm{CTL}^*_{bp}$.

* 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*$^\star$, which augments the branching temporal logic CTL$^\star$ with past-time connectives, is introduced in [17]. Formulas of PCTL$^\star$ are interpreted over *computation trees*. Thus, PCTL$^\star$ corresponds to the linear-past interpretation. Nevertheless, quantification over the past in the semantics of PCTL$^\star$ is very weak, making the past in PCTL$^\star$ very weak. For example, the PCTL$^\star$ 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$^\star$ is not more expressive than CTL$^\star$ (a similar limited past is discussed in [18]). Another augmentation of CTL$^\star$ 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 $\mathrm{CTL}_{bp}$ and $\mathrm{CTL}_{lp}$. Syntactically, $\mathrm{CTL}^\star_{bp}$ and $\mathrm{CTL}^\star_{lp}$ are exactly the same: both extend the full branching temporal logic CTL$^\star$ 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 $\mathrm{CTL}_{bp}$ 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 $\mathrm{CTL}_{bp}$ 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").