A Really Temporal Logic*†

Rajeev Alur
Thomas A. Henzinger

TR 92-1310
November 1992

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This research was supported in part by an IBM graduate fellowship to the second author, by the National Science Foundation under grants CCR-8812595 and CCR-9200794, by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211 and by the United States Air Force Office of Scientific Research under contracts AFOSR-88-0281 and AFOSR-92-NM255.
†This paper will appear in *The Journal of the Association for Computing Machinery*. A preliminary version appeared in the proceedings of the 30th IEEE Symposium on *Foundations of Computer Science* (1989).

A Really Temporal Logic*†

Rajeev Alur
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Thomas A. Henzinger
Computer Science Department
Cornell University
Ithaca, NY 14853

Abstract. We introduce a temporal logic for the specification of real-time systems. Our logic, TPTL, employs a novel quantifier construct for referencing time: the *freeze quantifier* binds a variable to the time of the local temporal context.

TPTL is both a natural language for specification and a suitable formalism for verification. We present a tableau-based decision procedure and a model checking algorithm for TPTL. Several generalizations of TPTL are shown to be highly undecidable.

1 Introduction

Linear temporal logic is a widely accepted language for specifying properties of reactive systems and their behavior over time [Pnu77, OL82, MP92]. The tableau-based satisfiability algorithm for its propositional version, PTL, forms the basis for the automatic verification and synthesis of finite-state systems [LP84, MW84].

PTL is interpreted over models that abstract away from the actual times at which events occur, retaining only temporal ordering information about the states of a system. The analysis of systems with hard real-time requirements, such as bounded response time, calls, however, for the development of formalisms with explicit time. Several attempts have been made to introduce time explicitly in PTL, and to interpret it over models that associate a time with every system state [BH81, PH88, Koy90, Ost90]. While these logics allow the specification of typical real-time requirements, most of the important decidability and complexity questions have not been answered. In particular, it has not been understood which timing constraints may be permitted in PTL without sacrificing the decidability of the verification problem.

Our objective is the development of a real-time extension of PTL that admits a generalization of the PTL-based tools for algorithmic verification. To begin with, a notational extension of PTL must be capable of relating the times of different system states. One commonly proposed method employs *first-order* temporal logic, with one of the state variables representing time [PH88, Ost90,

*This research was supported in part by an IBM graduate fellowship to the second author, by the National Science Foundation under grants CCR-8812595 and CCR-9200794, by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, and by the United States Air Force Office of Scientific Research under contracts AFOSR-88-0281 and AFOSR-92-NM255.

†This paper will appear in *The Journal of the Association for Computing Machinery*. A preliminary version appeared in the proceedings of the 30th IEEE Symposium on *Foundations of Computer Science* (1989).

LA92]. We claim that the unconstrained quantification of time variables allowed by this approach does not restrict the user to reasonable and readable specifications.

Instead, we propose a novel, restricted, form of quantification — we call it *freeze quantification* — in which every variable is bound to the time of a particular state. Freeze quantification identifies, so we argue, precisely the subclass of "intended" specifications, and it leads to a concise and readable notation. For instance, the typical time-bounded response requirement that every request p is followed by a response q within 10 time units, can be asserted by the formula

$$\Box x.(p \rightarrow \Diamond y.(q \wedge y \leq x + 10))$$

(read "whenever there is a request p, and the variable x is frozen to the current time, the request is followed by a response q, at time y, such that y is at most $x + 10$").

Secondly, we need to identify how expressive a theory of time may be added, in this fashion, to PTL without sacrificing its elementary complexity. Our main results are twofold: we develop a near-optimal decision procedure and a model-checking algorithm for real-time PTL by restricting both the syntax of the timing constraints and the precision of the time model, and we show that these restrictions cannot be relaxed without losing decidability. In particular, adding to PTL the theory of the natural numbers with successor, ordering, and congruence operations, yields the EXPSPACE-complete real-time temporal logic TPTL. The tableau method for PTL can be generalized to TPTL. However, allowing either addition over time or a dense time domain results in highly undecidable (Π_1^1-complete) logics.

Thus we lay out a theoretical basis for the automatic verification of finite-state real-time systems and, simultaneously, identify a boundary between the decidability and undecidability of finite-state formalisms with explicit time.

Alternative approaches to the automatic verification of real-time systems using temporal logic include work on the branching-time logic RTCTL [EMSS89] and the explicit-clock logic XCTL [HLP90]. RTCTL makes the simplifying assumption of modeling synchronous real-time systems, all of whose events occur with the ticks of a global clock. In the case of XCTL, all formulas are quantifier-free and their variables are implicitly universally quantified, which makes it difficult to compose requirements, like asserting that an implementation implies a specification. We will find both restrictions unnecessary.

Since an earlier version of this paper was published [AH89], many new results concerning real-time temporal logics have been obtained [ACD90, AH90, AFH91, WME92]. We point to [Hen90] for a complete axiomatization of TPTL and to [AH92] for a survey of recent results.

2 Timed Temporal Logic

We define *Timed Propositional Temporal Logic*, TPTL, and demonstrate its adequacy as a real-time specification language.

2.1 Timed state sequences

The formulas of TPTL are interpreted over timed state sequences. Let P be a set of proposition symbols $(p, q, r, \ldots)$ and let N be the set of nonnegative integers. A state is an interpretation for the propositions in P. A timed state sequence is an infinite sequence of states, each of which is labeled with a time from the discrete time domain $\mathsf{T} = \mathsf{N}$.

Definition 1 (Timed state sequence) *A state sequence* $\sigma = \sigma_0\sigma_1\sigma_2\ldots$ *is an infinite sequence of states* $\sigma_i \subseteq P$, $i \geq 0$. *A time sequence* $\tau = \tau_0\tau_1\tau_2\ldots$ *is an infinite sequence of times* $\tau_i \in \mathsf{T}$, $i \geq 0$, *such that*

Monotonicity $\tau_i \leq \tau_{i+1}$ *for all* $i \geq 0$, *and*

Progress *for all* $t \in \mathsf{T}$ *there is some* $i \geq 0$ *such that* $\tau_i > t$.

A timed state sequence $\rho = (\sigma, \tau)$ *is a pair consisting of a state sequence* σ *and a time sequence* τ.

By σ^i $[\tau^i]$ we denote the state [time] sequence that results from the state sequence σ [time sequence τ] by deleting the first i elements. We let $\rho^i = (\sigma^i, \tau^i)$ and use the convention that $\tau_{-1} = 0$.

At this point, a few words about our model of time are in order. Time is discrete but not a state counter; rather, the time between successive states of a timed state sequence may remain the same, or it may increase by an arbitrary amount. While a state counter would suffice to model synchronous real-time systems, the events of asynchronous processes take place in a dense time domain. Reasoning about dense time, on the other hand, may be prohibitively difficult (see Section 4). As a compromise, the *fictitious-clock* (or *digital-clock*) *assumption* for real-time systems has enjoyed increasing popularity [AH92, HMP92]: the true, dense times of events are recorded with the finite precision of a discrete clock. Our definition of timed state sequences is chosen sufficiently general to accommodate the fictitious-clock assumption (states between successive clock ticks can be labeled with identical times).

2.2 Syntax and semantics of TPTL

We are given an infinite supply V of variables $(x, y, z, \ldots)$. The formulas of TPTL are built from proposition symbols and timing constraints by boolean connectives, temporal operators, and freeze quantifiers.

Definition 2 (Syntax of TPTL) *The terms* π *and formulas* ϕ *of* TPTL *are inductively defined as follows:*

$$\pi \ := \ x + c \mid c$$

$$\phi \ := \ p \mid \pi_1 \leq \pi_2 \mid \pi_1 \equiv_d \pi_2 \mid \mathit{false} \mid \phi_1 \rightarrow \phi_2 \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2 \mid x.\phi$$

for $x \in V$, $p \in P$, *and integer constants* $c, d \in \mathsf{N}$, $d \neq 0$.

The timing constraints. The timing constraints of TPTL are of the form $\pi_1 \leq \pi_2$ and $\pi_1 \equiv_d \pi_2$ ("time π_1 is congruent to time π_2 modulo the constant d"). The abbreviations x (for $x + 0$), $=$, $<$, $>$, $\geq$, *true*, $\neg$, $\wedge$, and $\vee$ are defined as usual. If each term of a formula ϕ contains a variable, we say that ϕ contains *no absolute time references*. On the other hand, each term of a formula may contain at most one variable. While from a logical point of view, this restriction confines TPTL to the *successor* operation on time, we do not define terms using a unary successor operator; rather, for determining the length of a formula, we assume that all constants are given in a reasonably succinct (e.g., binary) encoding. The size of a formula will be important for locating the computational complexity of problems whose input includes formulas of TPTL.

The temporal operators. TPTL is based on the two temporal operators of PTL [GPSS80]. The *next* formula $\bigcirc p$ asserts about a timed state sequence that the second state in the sequence satisfies the proposition p. The *until* formula $p \mathcal{U} q$ asserts about a timed state sequence that there is a state satisfying the proposition q, and all states before this q-state satisfy the proposition p.

Additional temporal operators are defined as usual. In particular, the *eventually* operator $\Diamond\phi$ stands for $\mathit{true}\,\mathcal{U}\,\phi$, and the *always* operator $\Box\phi$ stands for $\neg\Diamond\neg\phi$.

The freeze quantifier. A variable x can be bound by a freeze quantifier "$x.$", which "freezes" x to the time of the local temporal context. Let $\phi(x)$ be a formula in which the variable x occurs freely. Then $x.\phi(x)$ asserts about the timed state sequence $\rho = (\sigma, \tau)$ that $\phi(\tau_0)$ is satisfied by ρ, where the formula $\phi(\tau_0)$ is obtained from $\phi(x)$ by replacing all free occurrences of the variable x with the constant τ_0. For example, in the formula

$$\Diamond x.(p \wedge x < 10),$$

the variable x is bound to the time of a state in which the proposition p is "eventually" satisfied; it asserts that p is satisfied in some state before time 10. Similarly, the formula

$$\Box x.(p \rightarrow x \leq 10)$$

asserts that whenever the proposition p is satisfied in a state, then the time is at most 10 (i.e., p is not satisfied after time 10).

This intuition is captured formally by the following definition.

Definition 3 (Semantics of TPTL**)** *Let $\rho = (\sigma, \tau)$ be a timed state sequence and let $\mathcal{E}: V \rightarrow \mathsf{T}$ be an interpretation (environment) for the variables. The pair $(\rho, \mathcal{E})$ satisfies the* TPTL*-formula ϕ iff $\rho \models_{\mathcal{E}} \phi$, where the satisfaction relation $\models$ is inductively defined as follows:*

$\rho \models_{\mathcal{E}} p$ *iff* $p \in \sigma_0$;

$\rho \models_{\mathcal{E}} \pi_1 \leq \pi_2$ *iff* $\mathcal{E}(\pi_1) \leq \mathcal{E}(\pi_2)$;

$\rho \models_{\mathcal{E}} \pi_1 \equiv_d \pi_2$ *iff* $\mathcal{E}(\pi_1) \equiv_d \mathcal{E}(\pi_2)$;

$\rho \not\models_{\mathcal{E}} \mathit{false}$;

$\rho \models_{\mathcal{E}} \phi_1 \rightarrow \phi_2$ *iff* $\rho \models_{\mathcal{E}} \phi_1$ *implies* $\rho \models_{\mathcal{E}} \phi_2$;

$\rho \models_{\mathcal{E}} \bigcirc\phi$ *iff* $\rho^1 \models_{\mathcal{E}} \phi$;

$\rho \models_{\mathcal{E}} \phi_1 \mathcal{U} \phi_2$ *iff* $\rho^i \models_{\mathcal{E}} \phi_2$ *for some* $i \geq 0$, *and* $\rho^j \models_{\mathcal{E}} \phi_1$ *for all* $0 \leq j < i$;

$\rho \models_{\mathcal{E}} x.\phi$ *iff* $\rho \models_{\mathcal{E}[x:=\tau_0]} \phi$.

Here $\mathcal{E}(x + c) = \mathcal{E}(x) + c$ and $\mathcal{E}(c) = c$. Moreover, $\mathcal{E}[x := t]$ denotes the environment that agrees with the environment $\mathcal{E}$ on all variables except x, and maps x to $t \in \mathsf{T}$.

A TPTL-formula is *closed* if every occurrence of a variable x is within the scope of a freeze quantifier "$x.$". We shall henceforth consider only closed formulas of TPTL.

The truth value of a closed formula is completely determined by a timed state sequence alone. The timed state sequence ρ is a *model* of the (closed) TPTL-formula ϕ, denoted by $\rho \models \phi$, if the pair $(\rho, \mathcal{E})$ satisfies ϕ for any environment $\mathcal{E}$. The formula ϕ is *satisfiable* [*valid*] if $\rho \models \phi$ for some [every] timed state sequence ρ. Two formulas are *equivalent* if they have the same models.

2.3 TPTL as a specification language

We compare TPTL to alternative extensions of PTL with explicit time references. In particular, we show that freeze quantification can be viewed as a constrained form of classical (i.e., universal and existential) quantification.

TPTL versus first-order temporal logic

The freeze quantifier allows us to relate the times of different states. A typical real-time requirement for a reactive system is that a multi-valued switch must be turned from position p to position q within 10 time units. In TPTL this condition can be expressed by the formula

$$\Box x.(p \rightarrow p\mathcal{U}\, y.(q \wedge y \leq x + 10)). \tag{1}$$

Using standard first-order temporal logic with a state variable *now* that assumes the value of the current time in every state, we may attempt to write the condition (1) as

$$\Box((p \wedge now = x) \rightarrow p\mathcal{U}(q \wedge now \leq x + 10))$$

or, in closer resemblance to the TPTL-formula (1),

$$\Box((p \wedge now = x) \rightarrow p\mathcal{U}(q \wedge now = y \wedge y \leq x + 10)). \tag{2}$$

The meaning of these formulas (i.e., their truth values over timed state sequences) depends on the interpretation of the variables x and y. The explicit quantification of variables, however, is typically omitted from first-order temporal specifications [PH88, Ost90]. Moreover, specification languages are often restricted to formulas with implicit or explicit quantifier *prefixes* [Har88, HLP90].

The condition (2) is an example that the choice of quantifiers that provides the intended meaning may not be obvious and, indeed, may not correspond to any prefix. In particular, the following quantification of (2) yields a formula that is equivalent to (1):

$$\Box\forall x.((p \wedge now = x) \rightarrow p\mathcal{U}\,\exists y.(q \wedge now = y \wedge y \leq x + 10)) \tag{3}$$

(to be precise, a timed state sequence ρ and an environment $\mathcal{E}$ satisfy the formula $\exists x.\phi$ iff $\rho \models_{\mathcal{E}[x:=t]} \phi$ for some $t \in \mathsf{T}$).

On the other hand, no quantifier prefix makes the formula (2) equivalent to (1). For example, (1) does not imply the stronger condition

$$\forall x.\exists y.\Box((p \wedge now = x) \rightarrow p\mathcal{U}(q \wedge now = y \wedge y \leq x + 10)). \tag{4}$$

The difference is subtle: while the formula (1) asserts that every p-state of time x is followed by p-states and, eventually, a q-state of time $y \leq x + 10$, the formula (4) demands more; that if there is a p-state of time x, then there is a time $y \leq x + 10$ such that every p-state of time x is followed by p-states and, eventually, a q-state of time y. For instance, the timed state sequence

$$(\{p\},0) \longrightarrow (\{q\},0) \longrightarrow (\{p\},0) \longrightarrow (\{q\},1) \longrightarrow (\{\},2) \longrightarrow (\{\},3) \longrightarrow \cdots$$

(presented as a sequence of state-time pairs) satisfies (1) but not (4).

In general, TPTL identifies a fragment of the first-order temporal logic with the state variable *now*. TPTL includes precisely those first-order temporal formulas in which each variable in V is, immediately upon introduction, frozen to the time in the local temporal context (i.e., the value of *now*): the TPTL-formula $x.\phi$ is equivalent to the first-order temporal formula

$$\forall x.(x = now \;\rightarrow\; \phi)$$

or, equivalently, to the formula

$$\exists x.(x = now \wedge \phi).$$

In other words, TPTL restricts all time references to times of states, rather than permitting quantification over the entire time domain. It is precisely this restriction that allows (and limits) us to express timing constraints between states by concise and readable specifications (compare (1) with (3)). It is also this restriction that leads to a generalization of the PTL-based tableau algorithms for verification. While the tableau method for TPTL will be developed in Section 3, the validity problem for TPTL with unconstrained classical quantification was recently shown to be nonelementary [AH90].

TPTL **versus bounded temporal operators**

Several researchers have proposed to add an infinite supply of real-time modalities such as $\Diamond_{\leq\delta}$ ("eventually within δ time units") to PTL [PH88, Koy90] or branching-time logics [EMSS89]. These bounded temporal operators are definable in TPTL. For instance, the bounded-eventuality operator $\Diamond_{\leq\delta}\phi$ can be expressed by the TPTL-formula

$$x.\Diamond y.(y \leq x + \delta \wedge \phi).$$

While bounded temporal operators always relate the times of adjacent temporal contexts, TPTL admits constraints between distant contexts. For example, the formula

$$\Box x.(p \;\rightarrow\; \Diamond(q \wedge \Diamond y.(r \wedge y \leq x + 10))$$

asserts that every p-state is followed by a q-state and, later, by an r-state, and the time difference between the p-state and the corresponding r-state is at most 10.

For reasoning about synchronous real-time systems, "next state" can be identified with "next time" and timing constraints may be expressed in PTL using the *next* operator. In this case, bounded temporal operators are abbreviations for nested *next* formulas [EMSS89]. We can restrict TPTL to the synchronous case by postulating

$$\Box x. \bigcirc y. y = x + 1.$$

3 Timed Tableaux

We present a tableau-based decision procedure for TPTL. We then justify the doubly-exponential-time cost of the decision procedure by showing that the validity problem for TPTL is EXPSPACE-complete. Finally, we demonstrate how the tableau techniques can be applied to verify TPTL-properties of real-time systems.

3.1 Decision procedure for TPTL

First we observe that to solve the validity problem for a formula, it suffices to check if its negation is satisfiable. Throughout this subsection, we are given a formula ϕ of TPTL and wish to determine if ϕ is satisfiable. The tableau method searches systematically for a model of ϕ. It originated with the propositional calculus [Smu68] and was first applied to obtain a decision procedure for a modal logic of computation in the case of dynamic logic [Pra80].

We follow the standard presentation of the tableau-based decision procedure for PTL [BMP81, Wol83] and begin by constructing the initial tableau for ϕ. Checking the satisfiability of ϕ can then be reduced to checking if the finite initial tableau for ϕ contains certain infinite paths. The tableau method for PTL is, in fact, subsumed by our procedure as the special case in which ϕ contains no timing constraints.

Preliminary assumptions

For the moment, we assume that

1. ϕ contains no absolute time references; that is, every term in ϕ contains a variable. Thus we may perform simple arithmetic manipulations so that all timing constraints in ϕ are of the form $x \leq y + c$ or $x + c \leq y$ or $x \equiv_d y + c$, for nonnegative integers $d > c \geq 0$.
2. ϕ contains only the temporal operators $\bigcirc$ and $\Box$; that is, the first argument of every occurrence of the *until* operator $\mathcal{U}$ in ϕ is *true*.

While neither of the two restrictions is essential, they simplify the exposition of the decision procedure. Later we will accommodate both absolute time references and *until* operators. We also assume that ϕ is of the form $z.\phi'$; this can be easily achieved, if necessary, by prefixing ϕ with any variable z that does not occur freely in ϕ.

A timed state sequence $\rho = (\sigma, \tau)$ is Δ-*bounded*, for a constant $\Delta \in \mathbb{N}$, if $\tau_i \leq \tau_{i-1} + \Delta$ for all $i \geq 0$; that is, the time of the initial state of a Δ-bounded timed state sequence is at most Δ and the time increases from a state to its successor state by no more than Δ.

To begin with, we restrict ourselves to Δ-bounded models for checking satisfiability. This case has finite-state character: the times that are associated with states can be modeled by finitely many (new) time-difference propositions $Prev_\delta$, $0 \leq \delta \leq \Delta$, that represent in the initial state, the initial time δ, and in all other states, the time increase δ from the predecessor state. Formally, we can capture the (state and) time information in a timed state sequence $\rho = (\sigma, \tau)$ by a state sequence $\hat{\sigma}$ with

$$\hat{\sigma}_i = \sigma_i \cup \{Prev_{\tau_i - \tau_{i-1}}\}$$

for all $i \geq 0$. This reduction of timed state sequences to state sequences allows us to adopt the tableau techniques for PTL. At the conclusion of this section we will show how we can find an appropriate constant Δ for the given formula ϕ.

Updating timing constraints

The key observation underlying the tableau method for PTL is that any formula can be split into two conditions: a non-temporal ("present") requirement on the initial state and a temporal ("future") requirement on the rest of a model (i.e., the successor state). For example, the eventuality $\Diamond\psi$ can be satisfied by either ψ or $\bigcirc\Diamond\psi$ being true in the initial state of a state sequence. Since the number of conditions generated in this way is finite, checking for satisfiability is reducible to checking for satisfiability in a finite structure, the initial tableau.

The splitting of TPTL-formulas into a present and a future (next-state) condition demands more care; to obtain the requirement on the successor state, all timing constraints need to be updated appropriately to account for the time increase δ from the initial state to its successor. Consider, for example, the formula $x.\Diamond y.\psi(x, y)$, and recall that the free occurrences of x in ψ are references to the initial time. This eventuality can be satisfied either by having the initial state satisfy $y.\psi(y, y)$, with all free occurrences of x in ψ replaced by y, or by having the next state satisfy the updated eventuality "$x.\Diamond y.\psi(x - \delta, y)$." For $\delta > 0$, a naive replacement of x by $x - \delta$ would, however, successively generate infinitely many new conditions. Fortunately, the monotonicity of time can be exploited to keep the tableau finite; the observation that y is always instantiated, in the "future," to a value greater than or equal to the initial time x, allows us to simplify timing assertions of the form $x \leq y + c$ and $y + (c + 1) \leq x$ to *true* and *false*, respectively.

We define, therefore, the formula $x.\psi(x)^\delta$ that results from updating all references in ψ to the initial time x by the time difference δ. For instance, if $x.\psi$ is the formula

$$x.\Box y.(p \rightarrow y.y \leq x + 5),$$

then $x.\psi^1$, $x.\psi^5$, and $x.\psi^6$ are the following formulas:

$$x.\Box y.(p \rightarrow y.y \leq x + 4),$$

$$x.\Box y.(p \rightarrow y.y \leq x),$$

$$x.\Box y.(p \rightarrow \mathit{false}).$$

In general, given a TPTL-formula $x.\psi$ and $\delta \in \mathsf{N}$, the TPTL-formula $x.\psi^\delta$ is defined inductively as follows:

- $x.\psi^0$ equals $x.\psi$.
- $x.\psi^{\delta+1}$ results from $x.\psi^\delta$ by replacing every term of the form $x + (c + 1)$ with $x + c$, and every subformula of the form $x \leq y + c$, $y + c + 1 \leq x$, and $x \equiv_d y + c$ with *true*, *false*, and $x \equiv_d y + ((c + 1)\ \mathit{mod}\ d)$, respectively, provided that the occurrence of x in the specified terms and formulas is free in ψ^δ.

The following lemma confirms that this transformation has the intended effect and updates all time references correctly; that is, the formula $x.\psi(x)^\delta$ expresses the condition "$x.\psi(x - \delta)$."

Lemma 1 (Time step) *Let $\rho = (\sigma, \tau)$ be a timed state sequence, let $\mathcal{E}$ be an environment, and let $\delta \in \mathsf{N}$ such that $\delta \leq \tau_0$. Then $\rho \models_{\mathcal{E}} x.\psi^{\delta}$ iff $\rho \models_{\mathcal{E}[x:=\tau_0-\delta]} \psi$ for every* TPTL*-formula $x.\psi$.*

Proof of Lemma 1 The proof proceeds by a straightforward induction on the structure of ψ. ■

Closure of a TPTL-formula

We collect all conditions that may arise by recursively splitting the formula ϕ into its present and future parts in the closure of ϕ. It suffices to define the closure for formulas whose outermost symbol is a freeze quantifier. The *closure* set $\mathit{Closure}(z.\phi')$ of the TPTL-formula $z.\phi'$ is the smallest set of formulas containing $z.\phi'$ that is closed under the following operation Sub:

$$\begin{aligned}
\mathit{Sub}(z.(\psi_1 \rightarrow \psi_2)) &= \{z.\psi_1, z.\psi_2\},\\
\mathit{Sub}(z.\bigcirc\psi) &= \{z.\psi^{\delta} \mid \delta \geq 0\},\\
\mathit{Sub}(z.\Box\psi) &= \{z.\psi, z.\bigcirc\Box\psi\},\\
\mathit{Sub}(z.x.\psi) &= \{z.\psi[x := z]\},
\end{aligned}$$

where the formula $\psi[x := z]$ results from ψ by replacing all free occurrences of x with z. Note that all formulas in a closure set are of the form $z.\psi$.

A constant $c > 0$ *occurs* in the TPTL-formula ϕ if ϕ contains a subformula of the form $x \leq y + (c-1)$ or $x + (c-1) \leq y$, or ϕ contains the predicate symbol $\equiv_c$. Let C be the largest constant that occurs in the formula ϕ. The closure set of ϕ is finite, because for all $\delta \geq C$ and for all formulas $z.\psi$ in $\mathit{Closure}(\phi)$, $z.\psi^{\delta}$ is $z.\psi^{C}$.

The size of the closure set of ϕ depends on both the structure of ϕ and the constants that occur in ϕ. We define k_ϕ, the product of all constants that occur in the TPTL-formula ϕ, inductively as follows:

$$\begin{aligned}
&k_{\pi_1 \leq \pi_2} = k_{\pi_1} \cdot k_{\pi_2} \ \text{ and } \ k_{\pi_1 \equiv_c \pi_2} = c \cdot k_{\pi_1} \cdot k_{\pi_2};\\
&k_{\mathit{false}} = 1 \ \text{ and } \ k_{\phi_1 \rightarrow \phi_2} = k_{\phi_1} \cdot k_{\phi_2};\\
&k_{\bigcirc\phi} = k_\phi,\ k_{\phi_1 \mathcal{U} \phi_2} = k_{\phi_1} \cdot k_{\phi_2}, \ \text{ and } \ k_{x.\phi} = k_\phi,
\end{aligned}$$

where $k_{x+c} = c + 1$ and $k_c = c + 1$.

Lemma 2 (Size of closure) *Let $n-1$ be the number of boolean, temporal, and freeze operators in the* TPTL*-formula ϕ, and let k be the product of all constants that occur in ϕ. Then $|\mathit{Closure}(\phi)| \leq 2nk$.*

Proof of Lemma 2 Given a formula $z.\phi'$, we define, by induction on the structure of ϕ', the set $D_{\phi'}$ of formulas that contains ϕ' and is closed under updating of timing constraints; the set $C_{\phi'}$ is, in addition, closed under subformulas:

$$\begin{aligned}
&C_p = D_p = \{p\},\\
&C_{x \leq y+c} = D_{x \leq y+c} = \{x \leq y+c, x \leq y+(c-1), \ldots x \leq y\},\\
&C_{x \equiv_d y+c} = D_{x \equiv_d y+c} = \{x \equiv_d y+(d-1), x \equiv_d y+(d-2), \ldots x \equiv_d y\},\\
&C_{\psi_1 \rightarrow \psi_2} = C_{\psi_1} \cup C_{\psi_2} \cup D_{\psi_1 \rightarrow \psi_2},\ D_{\psi_1 \rightarrow \psi_2} = D_{\psi_1} \rightarrow D_{\psi_2},\\
&C_{\bigcirc\psi} = C_\psi \cup D_{\bigcirc\psi},\ D_{\bigcirc\psi} = \bigcirc D_\psi,\\
&C_{\Box\psi} = C_\psi \cup D_{\Box\psi} \cup D_{\bigcirc\Box\psi},\ D_{\Box\psi} = \Box D_\psi,\\
&C_{x.\psi} = C_{\psi[x:=z]} \cup D_{x.\psi},\ D_{x.\psi} = x.D_\psi.
\end{aligned}$$

Here $x.E = \{x.\psi \mid \psi \in E\}$ for any set E of formulas; the other operators are applied to sets in an analogous fashion. The case of formulas of the form $x + c \leq y$ is treated similarly to the timing constraints $x \leq y + c$. Furthermore, let $E^* = E \cup \{true, false\}$.

Observe that $D_\psi \subseteq C_\psi$. It is straightforward to show by induction on the structure of ϕ' that

(1) $\phi' \in D_{\phi'}$ and, hence, $\phi' \in C_{\phi'}$.
(2) For all $\delta \geq 0$, $z.\phi'^{\delta} \in z.D^*_{\phi'}$ and, therefore, $z.\phi'^{\delta} \in z.C^*_{\phi'}$.
(3) $z.C^*_{\phi'}$ is closed under *Sub*.

From (1) and (3) it follows that $Closure(z.\phi') \subseteq z.C^*_{\phi'}$. Thus, it suffices to show that $|D_{\phi'}| \leq k$ and $|C_{\phi'}| \leq 2nk$, which may again be done by induction on the structure of ϕ'. ■

Initial tableau of a TPTL-formula

Tableaux for TPTL are finite, directed state graphs (Kripke structures). Unlike the states of a timed state sequence, which determine the truth values of all propositions, the vertices of a tableau are labeled with arbitrary formulas of TPTL. The formulas that label a vertex of a tableau express conditions on the annotated state and its successor states. In addition, every vertex is labeled with a time-difference proposition $Prev_\delta$, $0 \leq \delta \leq \Delta$, that denotes the time increase from the predecessor states.

Formally, the vertices of a tableau for ϕ are the maximally consistent subsets of the finite universe

$$Closure^*(\phi) = Closure(\phi) \cup \{Prev_\delta \mid 0 \leq \delta \leq \Delta\}$$

of TPTL-formulas. A subset Φ of $Closure^*(\phi)$ is (maximally) *consistent* if it satisfies the following conditions, where all formulas range only over the finite set $Closure^*(\phi)$:

- $Prev_\delta$ is in Φ for precisely one $0 \leq \delta \leq \Delta$; this $\delta \in \mathbb{N}$ is referred to as δ_Φ.
- $z.(z \sim z + c)$ is in Φ iff $0 \sim c$ holds in $\mathbb{N}$ (for $\sim$ being one of $\leq$, $\geq$, and $\equiv_d$).
- $z.false$ is not in Φ.
- $z.(\psi_1 \rightarrow \psi_2)$ is in Φ iff either $z.\psi_1$ is not in Φ or $z.\psi_2$ is in Φ.
- $z.\Box\psi$ is in Φ iff both $z.\psi$ and $z.\bigcirc\Box\psi$ are in Φ.
- $z.x.\psi$ is in Φ iff $z.\psi[x := z]$ is in Φ.

The *initial tableau* $T(\phi)$ for the TPTL-formula ϕ is a directed graph whose vertices are the consistent subsets of $Closure^*(\phi)$, and which contains an edge from Φ to Ψ iff, for all formulas $z.\bigcirc\psi$ in $Closure^*(\phi)$,

$$z.\bigcirc\psi \in \Phi \quad \text{iff} \quad z.\psi^{\delta_\Psi} \in \Psi.$$

This definition ensures the global consistency of all temporal and real-time constraints in the initial tableau. The significance of the (finite) initial tableau $T(\phi)$ for the formula ϕ is that every model of ϕ corresponds to an infinite path through $T(\phi)$ along which all eventualities are satisfied in time, and vice versa. This implies a finite-model property for TPTL, in the sense that every satisfiable TPTL-formula ϕ is satisfied by a model whose state part, extended by the time-difference propositions $Prev_\delta$, is eventually periodic.

To be precise, an infinite path

$$\Phi : \quad \Phi_0 \rightarrow \Phi_1 \rightarrow \Phi_2 \rightarrow \cdots$$

through a tableau is a *ϕ-path* if it satisfies the following three conditions:

Initiality $\phi \in \Phi_0$.

Fairness All eventualities are satisfied along Φ in time or, equivalently, all missing invariances are violated along Φ in time; that is, for all $z.\Box\psi \in Closure^*(\phi)$ and $i \geq 0$,

$$z.\Box\psi \notin \Phi_i \quad \text{implies} \quad z.\psi^\delta \notin \Phi_j \text{ for some } j \geq i \text{ with } \delta = \sum_{i<k\leq j} \delta_{\Phi_k}.$$

Progress $\delta_{\Phi_i} > 0$ for infinitely many $i \geq 0$.

Every ϕ-path in the initial tableau for ϕ can be reduced to a ϕ-path that is eventually periodic. Moreover, the length of the period is bounded by the following lemma. This will prove to be important for obtaining an upper bound on the complexity of TPTL.

Lemma 3 (Length of ϕ-paths) *Suppose that the initial tableau $T(\phi)$ for the* TPTL*-formula ϕ consists of m vertices. If $T(\phi)$ contains a ϕ-path, then it contains a ϕ-path of the form*

$$\Phi_0 \rightarrow \Phi_1 \rightarrow \cdots \rightarrow \Phi_i \rightarrow (\Phi_{i+1} \rightarrow \cdots \rightarrow \Phi_l)^\omega$$

for $l \leq (2nk+1)m$, where n is the number of temporal operators in ϕ and k is the product of all constants that occur in ϕ.

Proof of Lemma 3 Consider the infinite ϕ-path $\Phi = \Phi_0\Phi_1\ldots$, and choose i to be the smallest j such that Φ_j occurs infinitely often in Φ. There are no more than nk invariances in $Closure^*(\phi)$; hence Φ_i lacks at most nk invariances $z.\Box\psi_l$, each one of which is violated by some vertex Ψ_l of the infinite suffix $\Phi_i\Phi_{i+1}\ldots$ of Φ. Let $\Phi^0 = \Phi_0\ldots\Phi_i$, $\Phi^{2l-1} = \Phi_i\ldots\Psi_l$, and $\Phi^{2l} = \Psi_l\ldots\Phi_i$, for all $1 \leq l \leq nk$, be finite segments of Φ that contain no other (i.e., inner) occurrences of Φ_i. Delete all loops in every segment Φ^j, thus obtaining the finite sequences $\hat{\Phi}^j$, $0 \leq j \leq 2nk$, each of length at most $m+1$. It is not hard to see that the result of deleting duplicated vertices (i.e., Φ_i) from

$$\hat{\Phi}^0(\hat{\Phi}^1\hat{\Phi}^2\ldots\hat{\Phi}^{2nk})^\omega$$

is a ϕ-path of the desired form. ■

Tableau decision procedure

The following main lemma suggests a decision procedure for TPTL: to determine if the TPTL-formula ϕ is satisfiable, construct the initial tableau $T(\phi)$ and check if it contains any ϕ-paths.

Lemma 4 (Initial tableau for TPTL**)**

> *(1) [Correctness] If the initial tableau $T(\phi)$ for the* TPTL*-formula ϕ contains a ϕ-path, then ϕ is satisfiable.*
>
> *(2) [Completeness] If ϕ has a Δ-bounded model, then $T(\phi)$ contains a ϕ-path.*

Proof of Lemma 4 The proof makes essential use of both directions of the *time-step* lemma, Lemma 1. Let $\mathcal{E}$ be any environment.

(1) Given a ϕ-path $\Phi = \Phi_0\Phi_1\ldots$ through the initial tableau $T(\phi)$, define the timed state sequence $\rho = (\sigma, \tau)$ such that, for all $i \geq 0$, $p \in \sigma_i$ iff $z.p \in \Phi_i$, and $\tau_i = \tau_{i-1} + \delta_{\Phi_i}$. Note that the time sequence τ satisfies the *progress* condition because Φ does. We show, by induction on the structure of ψ, that $\psi \in \Phi_i$ iff $\rho^i \models \psi$ for all $i \geq 0$ and $\psi \in \mathit{Closure}^*(\phi)$. Since $\phi \in \Phi_0$, it follows that ρ is a model of ϕ.

For a proposition $z.p \in \mathit{Closure}^*(\phi)$, we have $z.p \in \Phi_i$ iff $p \in \sigma_i$ iff $\rho^i \models z.p$. Let $\sim$ be one of $\leq$, $\geq$, $\equiv_d$, or its negation. By the consistency of Φ_i, $z.(z \sim z + c) \in \Phi_i$ iff $0 \sim c$ iff $\rho^i \models z.(z \sim z + c)$. This completes the base cases.

By the consistency of Φ_i, $z.(\psi_1 \rightarrow \psi_2) \in \Phi_i$ iff either $z.\psi_1 \notin \Phi_i$ or $z.\psi_2 \in \Phi_i$. By the induction hypothesis, this is the case iff either $\rho^i \not\models z.\psi_1$ or $\rho^i \models z.\psi_2$; that is, iff $\rho^i \models z.(\psi_1 \rightarrow \psi_2)$.

Now assume that $z.\bigcirc\psi \in \mathit{Closure}^*(\phi)$ and let $\delta = \delta_{\Phi_{i+1}}$; that is, $\tau_{i+1} = \tau_i + \delta$. Then $z.\bigcirc\psi \in \Phi_i$ iff $z.\psi^\delta \in \Phi_{i+1}$. By the induction hypothesis, this is the case iff $\rho^{i+1} \models z.\psi^\delta$. By Lemma 1, this is the case iff $\rho^{i+1} \models_{\mathcal{E}[z:=\tau_i]} \psi$; that is, iff $\rho^i \models z.\bigcirc\psi$.

For the case that $z.\Box\psi \in \mathit{Closure}^*(\phi)$, we first prove that $z.\Box\psi \in \Phi_i$ iff $z.\psi^{\tau_j - \tau_i} \in \Phi_j$ for all $j \geq i$. Let $\delta_j = \tau_j - \tau_i$ and note that $\delta_j = \sum_{i<k\leq j} \delta_{\Phi_k}$ by our choice of τ.

> We use induction on j to show that $z.\Box\psi \in \Phi_i$ implies $z.\Box\psi^{\delta_j} \in \Phi_j$ for all $j \geq i$. Suppose that $z.\Box\psi^{\delta_j} \in \Phi_j$ for an arbitrary $j \geq i$. By the consistency of Φ_j, also $z.\bigcirc\Box\psi^{\delta_j} \in \Phi_j$ and therefore $z.\Box\psi^{\delta_{j+1}} \in \Phi_{j+1}$. Invoking again the consistency of Φ_j, we conclude that $z.\psi^{\delta_j} \in \Phi_j$ for all $j \geq i$.
>
> On the other hand, suppose that $z.\Box\psi \notin \Phi_i$. Since Φ is a ϕ-path, there is some $j \geq i$ such that $z.\psi^{\delta_j} \notin \Phi_j$.

By the induction hypothesis, it follows that $z.\Box\psi \in \Phi_i$ iff $\rho^j \models z.\psi^{\delta_j}$ for all $j \geq i$. By Lemma 1, this is the case iff $\rho^j \models_{\mathcal{E}[z:=\tau_i]} \psi$ for all $j \geq i$; that is, iff $\rho^i \models z.\Box\psi$.

Finally, consider the case that $z.x.\psi \in \mathit{Closure}^*(\phi)$. In this case, $z.x.\psi \in \Phi_i$ iff $z.\psi[x := z] \in \Phi_i$. By the induction hypothesis, this is the case iff $\rho^i \models z.\psi[x := z]$; that is, iff $\rho^i \models z.x.\psi$.

(2) Let $\rho = (\sigma, \tau)$ be a Δ-bounded model of ϕ. The subsets Φ_i, for $i \geq 0$, of $\mathit{Closure}^*(\phi)$ are defined as follows: $\mathit{Prev}_{\tau_i - \tau_{i-1}} \in \Phi_i$, and $\psi \in \Phi_i$ iff $\rho^i \models \psi$ for all $\psi \in \mathit{Closure}^*(\phi)$. We show that $\Phi = \Phi_0\Phi_1\ldots$ is a ϕ-path through the initial tableau $T(\phi)$.

By inspecting the consistency rules, it is evident that every Φ_i is (maximally) consistent. To prove that Φ is an infinite path through $T(\phi)$, we also have to show that there is an edge from Φ_i to Φ_{i+1} for all $i \geq 0$. Suppose that $z.\bigcirc\psi \in \mathit{Closure}^*(\phi)$ and let $\delta = \tau_{i+1} - \tau_i$. Then $z.\bigcirc\psi \in \Phi_i$ iff $\rho^i \models z.\bigcirc\psi$ iff $\rho^{i+1} \models_{\mathcal{E}[z:=\tau_i]} \psi$. By Lemma 1, this is the case iff $\rho^{i+1} \models z.\psi^\delta$; that is, iff $z.\psi^\delta \in \Phi_{i+1}$. Since also $\mathit{Prev}_\delta \in \Phi_{i+1}$, the initial tableau for ϕ contains an edge from Φ_i to Φ_{i+1}.

We now show that the infinite path Φ is indeed a ϕ-path. It satisfies the *progress* condition because the time sequence τ does. To see that $\phi \in \Phi_0$, observe that ρ is a model of ϕ. It remains to be established that all eventualities in Φ are satisfied in time. Suppose that $z.\Box\psi \in Closure^*(\phi)$ and $z.\Box\psi \notin \Phi_i$; that is, $\rho^i \models z.\Diamond\neg\psi$ and therefore $\rho^j \models_{\mathcal{E}[z:=\tau_i]} \neg\psi$ for some $j \geq i$. Let $\delta = \tau_j - \tau_i$; thus $\delta = \sum_{i<k\leq j} \delta_{\Phi_k}$. Then $\rho^j \not\models z.\psi^\delta$ by Lemma 1, which implies that $z.\psi^\delta \notin \Phi_j$. ∎

After constructing the initial tableau $T(\phi)$ for the formula ϕ, we delete all vertices that are not on a ϕ-path. This can be achieved by a straightforward modification of the standard techniques for marking all vertices of a graph that lie on an infinite path along which all eventualities are satisfied [Wol83]. The remaining state graph is called the *final tableau* for ϕ. It follows that a TPTL-formula ϕ has a Δ-bounded model iff its final tableau is not empty.

The procedure for finding the final tableau is polynomial in the size of the initial tableau, which contains $O(\Delta \cdot 2^{nk})$ vertices, each of size $O(nk)$, where $n-1$ is the number of operators in ϕ and k is the product of all constants that occur in ϕ. Thus, provided that Δ is dominated by 2^{nk}, the initial $T(\phi)$ can be constructed and checked for ϕ-paths in deterministic time exponential in nk. We show next that Δ can indeed be bounded by k.

Bounding the time step-width

Given a formula ϕ, we finally determine the bound Δ on the time increase between two successive states such that the satisfiability of ϕ is not affected; that is, we choose the constant $\Delta \in \mathbb{N}$ such that ϕ is satisfiable iff it has a Δ-bounded model.

Let c be the largest constant in ϕ that occurs in a subformula of the form $x \leq y + (c-1)$ or $x + (c-1) \leq y$, and let $\equiv_{c_1}, \ldots \equiv_{c_m}$ be all the congruence predicates that occur in ϕ. If the time increase δ between two states is greater than or equal to c, it obviously suffices to know the residues of δ modulo $c_1, \ldots c_m$ in order to update, in a tableau, all timing constraints correctly. Indeed, for checking the satisfiability of ϕ, the arbitrary step-width δ can be bounded by taking the smallest representative for each of the finitely many congruence classes.

Lemma 5 (Bounded time increase) *If the* TPTL*-formula ϕ is satisfiable and k is the product of all constants that occur in ϕ, then ϕ has a k-bounded model.*

Proof of Lemma 5 We can, in fact, derive the tighter bound $c + k' \leq k$, for the least common multiple k' of all c_i, $1 \leq i \leq m$. Given a model $\rho = (\sigma, \tau)$ of ϕ, let the time sequence τ' be such that, for all $i \geq 0$, $\tau_i = \tau'_{i-1} + (\tau_i - \tau_{i-1})$ if $\tau_{i+1} - \tau_i < c$; otherwise choose τ'_i to be the smallest $\delta \geq \tau'_i + c$ with $\delta \equiv_{k'} \tau_i$. It is easy to see that $\rho' = (\sigma, \tau')$ is also a model of ϕ. ∎

Combining this result with the tableau method developed above, we arrive at the conclusion that the satisfiability of the TPTL-formula ϕ is decidable in deterministic time exponential in nk. Moreover, Lemma 3 implies that every satisfiable formula ϕ is satisfiable in a model whose size is, in the sense mentioned above, exponential in nk.

Remember that we have restricted ϕ to contain no *until* operators and no absolute time references. We now show that both assumptions can be relaxed.

Until operators

First, let us address formulas that include *until* operators. We take the *closure* of a formula ϕ with *until* operators to be closed under the operation

$$Sub(z.(\psi_1 \mathcal{U} \psi_2)) = \{z.\psi_1, z.\psi_2, z.\bigcirc(\psi_1 \mathcal{U} \psi_2)\}$$

and add the following condition on the *consistency* of a set $\Phi \subseteq \mathit{Closure}^*(\phi)$ of formulas:

$$z.(\psi_1 \mathcal{U} \psi_2) \text{ is in } \Phi \text{ iff either } z.\psi_2 \text{ is in } \Phi, \text{ or both } z.\psi_1 \text{ and } z.\bigcirc(\psi_1 \mathcal{U} \psi_2) \text{ are in } \Phi.$$

Finally, the *fairness* requirement on a ϕ-path $\Phi_0\Phi_1\Phi_2\ldots$ is generalized to

$$z.(\psi_1 \mathcal{U} \psi_2) \in \Phi_i \text{ implies } z.\psi_2^\delta \in \Phi_j \text{ for some } j \geq i \text{ with } \delta = \textstyle\sum_{i<k\leq j} \delta_{\Phi_k}$$

for all $i \geq 0$. It is not hard to check that the Lemmas 2, 3, 4, and 5 allow the addition of *until* operators in this way.

Absolute time references

Secondly, let us accommodate absolute time references. Instead of generalizing the tableau method to constant terms, which contain no variable, we can use a simple observation. Suppose that we test the formula ϕ for satisfiability. Let x be a variable that does not occur in ϕ and replace every variable-free term c in ϕ with the term $x+c$, thus obtaining the new formula ϕ^R (which may contain free occurrences of x). The following lemma allows us to reduce the satisfiability problem for ϕ to the satisfiability problem for the formula $x.\bigcirc\phi^R$, which contains no absolute time references.

Lemma 6 (Absolute time references) *A* TPTL*-formula ϕ is satisfiable iff the formula $x.\bigcirc\phi^R$ is satisfiable, where x does not occur in ϕ.*

Proof of Lemma 6 (1) Let $\rho = (\sigma, \tau)$ be a timed state sequence. We define the timed state sequence $\rho' = (\sigma', \tau')$ such that $\tau'_0 = 0$, and $\sigma'_{i+1} = \sigma_i$ and $\tau'_{i+1} = \tau_i$ for all $i \geq 0$. Clearly, if $\rho \models \phi$, then $\rho' \models x.\bigcirc\phi$.

(2) Let $\rho = (\sigma, \tau)$ be a timed state sequence. We define the timed state sequence $\rho' = (\sigma', \tau')$ such that $\sigma'_i = \sigma_{i+1}$ and $\tau'_i = \tau_{i+1} - \tau_0$ for all $i \geq 0$. Clearly, if $\rho \models x.\bigcirc\phi$, then $\rho' \models \phi$. ■

Note that the transformation from ϕ to ϕ^R does not increase the number of operators in ϕ nor the product of all constants that occur in ϕ. The following theorem summarizes our results about the tableau method.

Theorem 1 (Deciding TPTL**)** *The validity problem for a (closed)* TPTL*-formula ϕ can be decided in deterministic time exponential in nk, where $n-1$ is the number of boolean, temporal, and freeze operators in ϕ, and k is the product of all constants that occur in ϕ.*

Note that the length ℓ of any formula ϕ, whose constants are presented in a logarithmic (e.g., binary) encoding, is within constant factors of $n + \log k$. Thus we have a decision procedure for TPTL that is doubly exponential in ℓ (although only singly exponential in n, the "untimed" part and, therefore, singly exponential for PTL). The algorithm we have outlined can, of course, be improved in many ways. In particular, we may avoid the construction of the entire initial tableau by starting with the initial state, which contains ϕ, and successively adding new states only when needed [Wol83]. This stepwise procedure, however, does not lower the doubly exponential deterministic-time bound; as we will show in the following subsection, the decision problem for TPTL is EXPSPACE-hard.

We also point out that while the *monotonicity* condition on timed state sequences is essential for the tableau method to work, the *progress* condition on timed state sequences (and ϕ-paths) can be omitted.

3.2 Complexity of TPTL

The following theorem establishes TPTL as being exponentially harder to decide than its untimed base PTL, which has a PSPACE-complete decision problem [SC85]. The extra exponential is caused by the succinct representation of time constants in TPTL and is typical for many real-time specification languages [AH90].

Theorem 2 (Complexity of TPTL**)** *The validity problem for* TPTL *is EXPSPACE-complete (with respect to polynomial-time reduction).*

Proof of Theorem 2 The proof proceeds in two parts; we first show that TPTL is in EXPSPACE, and then that it is EXPSPACE-hard. The first part follows the argument that PTL is in PSPACE, which builds on a nondeterministic version of the tableau-based decision procedure [Wol83]; the hardness part is patterned after a proof that the universality problem of regular expressions with exponentiation is EXPSPACE-hard [HU79].

[EXPSPACE] It suffices to show that the complementary problem of checking the satisfiability of a TPTL-formula is in nondeterministic EXPSPACE and, hence, by Savitch's theorem, in (deterministic) EXPSPACE.

In particular, it can be checked in nondeterministic singly exponential space if the initial tableau $T(\phi)$ contains a ϕ-path of the form stated in Lemma 3. In trying to construct such a ϕ-path nondeterministically, at each stage only the current vertex, the "loop-back" vertex, and a vertex counter have to be retained in order to construct a successor vertex, loop back, or, if the vertex counter exceeds the maximal length of the loop, fail. Since both the size of each vertex and the length of the loop have, by Lemma 2 and Lemma 3, respectively, (singly) exponential representations in the length of ϕ, it follows that this nondeterministic procedure requires only exponential space.

[EXPSPACE-hardness] Consider a deterministic 2^n-space-bounded Turing machine M. For each input X of length n, we construct a TPTL-formula ϕ_X of length $O(n \cdot log\ n)$ that is valid iff M accepts X. By a standard complexity-theoretic argument, using the hierarchy theorem for space, it follows that there is a constant $c > 0$ such that every Turing machine that solves the validity problem for formulas ϕ of length ℓ takes space $S(\ell) \geq 2^{c\ell/log\ \ell}$ infinitely often. Thus it suffices to construct, given the initial tape contents X,

1. a sufficiently succinct formula ϕ_X that describes the (unique) computation of M on X as an infinite sequence of propositions, and
2. a sufficiently succinct formula ϕ_{ACCEPT} that characterizes the computation of M on X as accepting.

Then the implication

$$\phi_X \rightarrow \phi_{ACCEPT}$$

is valid iff the machine M accepts the input X.

We use a proposition p_i and a proposition q_j for every tape symbol i and machine state j of M, respectively. In particular, p_0 and q_0 correspond to the special tape symbol "blank" and the initial state of M. We use the following abbreviations for formulas:

$$\begin{array}{ll} \hat{p}_i: & p_i \wedge \bigwedge_{i' \neq i} \neg p_{i'} \wedge \bigwedge \neg q_j, \\ r_{i,j}: & p_i \wedge q_j \wedge \bigwedge_{i' \neq i} \neg p_{i'} \wedge \bigwedge_{j' \neq j} \neg q_{j'}, \\ s: & \bigwedge \neg p_i \wedge \bigwedge \neg q_j. \end{array}$$

We represent configurations of M by $\hat{p}$-state sequences of length 2^n that are separated by s-states; the position of the read-write head is marked by an r-state. The computation of M on X is completely determined by the following two conditions:

(1) it starts with the initial configuration, and
(2) every configuration follows from the previous one by a move of M.

Both conditions can be expressed in TPTL. Take ϕ_X to consist of $\Box x. \bigcirc y. y = x + 1$, forcing time to resemble a state counter, and the following two conjuncts, which correspond to the requirements (1) and (2), respectively:

$$\begin{array}{ll} \phi_{INITIAL}: & s \wedge \bigcirc r_{X_1,0} \wedge x. \bigwedge_{2 \leq i \leq n} \Box y. (y = x + i \rightarrow \hat{p}_{X_i}) \wedge \\ & x. \Box y. (x + n < y \leq x + 2^n \rightarrow \hat{p}_0), \\ \phi_{MOVE}: & \Box x. (s \rightarrow \Diamond y. (y = x + 2^n + 1 \wedge s)) \wedge \\ & \bigwedge_{P,Q,R} \Box x. (P \wedge \bigcirc Q \wedge \bigcirc \bigcirc R \rightarrow \Diamond y. (y = x + 2^n + 2 \wedge f_M(P, Q, R))). \end{array}$$

Here P, Q, and R each range over the propositions $\hat{p}_i$, $r_{i,j}$, and s, and $f_M(P, Q, R)$ refers to the transition function of M. For instance, if M writes, in state j on input i', the symbol k onto the tape, moves to the right, and enters state j', then $f_M(\hat{p}_i, r_{i',j}, \hat{p}_{i''}) = \hat{p}_k$ and $f_M(r_{i',j}, \hat{p}_{i''}, \hat{p}_{i'''}) = r_{i'',j'}$.

The computation of M on X is accepting iff it contains the accepting state F, which is expressible in TPTL by the formula

$$\phi_{ACCEPT}: \quad \Diamond \bigvee_i r_{i,F}.$$

The lengths of $\phi_{INITIAL}$, ϕ_{MOVE}, and ϕ_{ACCEPT} are $O(n \cdot log\ n)$, $O(n)$, and $O(1)$, respectively (recall that constants are represented in binary), thus implying the desired $O(n \cdot log\ n)$-bound for ϕ_X. ∎

3.3 Real-time verification

Researchers have proposed a variety of different languages for defining real-time systems [AH92]. Instead of siding with a particular syntax, we represent finite-state real-time systems by abstract state graphs with timing information. Typically, it is not difficult to compile a given concrete syntax into timed state graphs (consult, for example, [Hen91] for the translation of timed transition systems into timed state graphs).

Model checking is an algorithmic verification technique that compares the temporal-logic specification of a system against a state-graph description of the system. Suppose that a system S is represented as a finite state graph T_S; that is, all possible runs of S correspond to infinite paths through T_S that meet certain fairness conditions. Furthermore, suppose that the specification of S is given as a formula ϕ of a linear temporal logic. The verification problem asks if all possible runs of the system S satisfy the specification ϕ.

In the case of PTL, the tableau construction can be used to solve the verification problem [LP84]. The initial tableau $T(\neg\phi)$ for the negated specification $\neg\phi$ captures precisely the models of the formula $\neg\phi$. Hence the system S meets the specification ϕ iff there is no infinite path that is common to both finite state graphs, T_S and $T(\neg\phi)$, and that corresponds both to a possible run of S and a model of $\neg\phi$. This question is answered by constructing the product of the two state graphs T_S and $T(\neg\phi)$ and checking if it contains an infinite path that meets certain fairness conditions.

We generalize the PTL-algorithm to check if a timed state graph meets a TPTL-specification. We then show that the problem of checking if a TPTL-formula ϕ is satisfied by all paths in a given structure is EXPSPACE-complete, and thus, in general, equally hard as deciding if ϕ is satisfied by all timed state sequences. The practitioner will note that complexity of the model checking problem is doubly exponential only in the size of the formula, which is typically much smaller than the size of the structure.

Timed state graphs

We represent finite-state real-time systems by finite, directed state graphs (Kripke structures) whose vertices (locations) are labeled with sets of propositions. Each location is labeled with a state and a time-difference proposition $Prev_\delta$ or $Prev_{\geq 0}$, which indicates that the time difference from the predecessor location is either exactly δ time units or unspecified, respectively.

Definition 4 (Timed state graph) *A timed state graph $T = \langle L, \mu, \nu, L_0, \mathcal{T}\rangle$ consists of*

- *a finite set L of locations;*
- *a state labeling function $\mu: L \rightarrow 2^P$ that labels every location $\ell \in L$ with a state $\mu_\ell \subseteq P$;*
- *a time labeling function ν that labels every location $\ell \in L$ with a time-difference proposition ν_ℓ, which is either $Prev_\delta$ for some $\delta \in \mathbb{N}$, or $Prev_{\geq 0}$;*
- *a set $L_0 \subseteq L$ of initial locations;*
- *a set $\mathcal{T} \subseteq L^2$ of transitions.*

A timed state sequence $\rho = (\sigma, \tau)$ is a computation of the timed state graph T if there is an infinite path $\ell_0\ell_1\ell_2\ldots$ through T such that for all $i \geq 0$, $\sigma_i = \mu_{\ell_i}$ and if $\nu_{\ell_i} = Prev_\delta$, then $\tau_i = \tau_{i-1} + \delta$.

A TPTL-formula ϕ is *satisfied* [*valid*] in the timed state graph T if some [all] computations of T are models of ϕ. The problem of *model checking* is to determine if a formula is valid in a timed state graph.

Model checking

We are given a timed state graph T_S and a TPTL-formula ϕ. Let k be the product of all constants in ϕ, and recall that k is the largest constant δ for which the initial tableau $T(\phi)$ for ϕ contains a time-difference proposition $Prev_\delta$.

We define the *product* $T = T_S \times T(\phi)$ of the two structures T_S and $T(\phi)$ to be a finite, directed graph whose vertices are pairs of T_S-vertices and $T(\phi)$-vertices. Each vertex (ℓ, Φ) of T consists of a location ℓ of T_S and a vertex Φ of $T(\phi)$ such that

- The state information in ℓ and Φ is compatible; that is, $p \in \sigma_\ell$ iff $z.p \in \Phi$ for all propositions $p \in P$.
- The time information in ℓ and Φ is compatible; that is, either $\nu_\ell = Prev_{\geq 0}$, or $\nu_\ell = \delta_\Phi$, or $\delta_\Phi = Prev_k$ and $\nu_\ell = Prev_{k'}$ for some $k' \geq k$.

The product T contains an edge from the vertex (ℓ_1, Φ_1) to the vertex (ℓ_2, Φ_2) iff T_S contains an edge from ℓ_1 to ℓ_2 and $T(\phi)$ contains an edge from Φ_1 to Φ_2.

The size of the product $T_S \times T(\phi)$ is clearly linear in the product of the sizes of T_S and $T(\phi)$.

An infinite path through the product $T_S \times T(\phi)$ is a *ϕ-path* if its second projection is a ϕ-path through $T(\phi)$; it is an *initialized ϕ-path* if, in addition, it starts at a vertex whose first projection is an initial vertex of T_S. The following lemma, which follows immediately from Lemma 4 and the proof of Theorem 1, confirms that our product construction has the intended effect.

Lemma 7 (Tableau product) *The timed state graph T_S satisfies the* TPTL*-formula ϕ iff the product $T_S \times T(\phi)$ contains an initialized ϕ-path.*

This lemma suggests a model checking algorithm. To see if all runs of a finite-state real-time system S satisfy a TPTL-formula ϕ:

1. Construct the timed state graph T_S for S.
2. Construct the initial tableau $T(\neg\phi)$ for the negated formula $\neg\phi$.
3. Construct the tableau product $T = T_S \times T(\neg\phi)$.
4. Check if T contains an initialized $\neg\phi$-path. The system S meets the specification ϕ iff this is not the case.

According to different notions of system fairness, various variants of computations through timed state graphs can be defined, and checked for, as in the untimed case [LP84].

Since a structure can be checked for ϕ-paths in polynomial time, the running time of the model checking algorithm is determined by the size of the tableau product T, which contains $O(|T_S| \cdot |T(\phi)|)$ vertices. The size of T_S typically is exponentially larger than the description of S itself [Hen91]; we have seen that the size of $T(\phi)$ can be two exponentials larger than ϕ.

Theorem 3 (Model checking) *The problem if a* TPTL*-formula ϕ is valid in a timed state graph T can be decided in deterministic time linear in the size of the T and doubly exponential in the length of ϕ.*

Complexity of model checking

Theorem 4 (Complexity of model checking) *The problem of deciding if a* TPTL*-formula is valid in a timed state graph is EXPSPACE-complete.*

Proof of Theorem 4 [EXPSPACE] The given timed state graph T satisfies the TPTL-formula ϕ iff the product $T \times T(\neg\phi)$ contains an initialized $\neg\phi$-path. It is easy to see that nondeterministic singly exponential space suffices to check for the desired $\neg\phi$-path. The EXPSPACE bound follows.

[EXPSPACE-hardness] To reduce the validity problems for TPTL to model checking, it suffices to give a timed state graph T of constant size such that formula ϕ is valid iff ϕ is valid in T. Simply choose $T = \langle 2^P, \mu, \nu, 2^P, 2^P \times 2^P \rangle$ to be the complete graph over all subsets of P, and label all locations with the time-difference proposition $Prev_{\geq 0}$. ■

4 Undecidable Real-time Properties

At last, let us justify our decisions to restrict the semantics of TPTL to the discrete time domain $\mathbb{N}$ and to restrict the syntax of the timing constraints to successor, ordering, and congruence operations. Indeed, both decisions seem overly limiting for real-time specification languages. For example, without addition of time values the property that "the time difference between subsequent p-states increases forever" cannot be defined. We show, however, that both restrictions are necessary to obtain verification algorithms.

We consider two natural extensions of TPTL, a syntactic one (allowing addition over time) and a semantic one (interpreting TPTL-formulas over a dense time domain). Both extensions are shown to be Π_1^1-complete, by reducing a Σ_1^1-hard problem of 2-counter machines to the respective satisfiability problems. It follows that they cannot even be (recursively) axiomatized (for an exposition of the analytical hierarchy consult [Rog67]).

4.1 A Σ_1^1-complete problem

A *nondeterministic 2-counter machine* M consists of two counters C and D, and a sequence of n instructions, each of which may increment or decrement one of the counters, or jump, conditionally upon one of the counters being zero. After the execution of a non-jump instruction, M proceeds nondeterministically to one of two specified instructions. We represent the configurations of M by triples $\langle i, c, d\rangle$, where $0 \leq i < n$, $c \geq 0$, and $d \geq 0$ are the current values of the location counter and the two counters C and D, respectively. The consecution relation on configurations is defined in the obvious way. A *computation* of M is an infinite sequence of related configurations, starting with the initial configuration $\langle 0, 0, 0\rangle$. The computation is *recurring* if it contains infinitely many configurations with the value of the location counter being 0.

The problem of deciding if a nondeterministic Turing machine has, over the empty tape, a computation in which the starting state is visited infinitely often, has been shown Σ_1^1-complete [HPS83]. Along the same lines we obtain the following result.

Lemma 8 (Complexity of 2-counter machines) *The problem of deciding if a given nondeterministic 2-counter machine has a recurring computation, is Σ_1^1-hard.*

Proof of Lemma 8 Every Σ_1^1-formula is equivalent to a Σ_1^1-formula χ of the form

$$\exists f. (f(0) = 1 \wedge \forall x. g(f(x), f(x+1))),$$

for a recursive predicate g [HPS83]. For any such χ we can construct a nondeterministic 2-counter machine M that has a recurring computation iff χ is true.

Let M start by computing $f(0) = 1$, and proceed, indefinitely, by nondeterministically guessing the next value of f. At each stage, M checks whether $f(x)$ and $f(x+1)$ satisfy g, and if (and only if) so, it jumps to instruction 0. Such an M exists, because 2-counter machines can, being universal, compute the recursive predicate g. It executes the instruction 0 infinitely often iff a function f with the desired properties exists. ■

4.2 Encoding computations of 2-counter machines

We show that the satisfiability problem for several extensions of TPTL is Σ_1^1-complete. First, we observe that the satisfiability of a formula ϕ can, in all cases, be phrased as a Σ_1^1-sentence, asserting the existence of a model for ϕ. Any timed state sequence ρ for ϕ can be encoded, in first-order arithmetic, by finitely many infinite sets of natural numbers; say, one for each proposition p in ϕ, characterizing the states in which p holds, and one to encode pairs of state numbers and associated times. It is a routine matter to express, as a first-order predicate, that ϕ holds in ρ. We conclude that satisfiability is in Σ_1^1.

To show that the satisfiability problem of a logic is Σ_1^1-hard, it suffices, given a nondeterministic 2-counter machine M, to construct a formula ϕ_M such that ϕ_M is satisfiable iff M has a recurring computation. We demonstrate the technique of encoding recurring computations of M by showing that the *monotonicity* constraint on time is necessary for the decidability of TPTL.

Theorem 5 (Nonmonotonic time) *Relaxing the monotonicity condition for time sequences renders the satisfiability problem for* TPTL Σ_1^1*-complete.*

Proof of Theorem 5 We encode a computation Γ of M by the "time" sequence τ such that, for all $k \geq 0$, $\tau_{4k} = i$, $\tau_{4k+1} = n + c$, $\tau_{4k+2} = n + d$, and $\tau_{4k+3} = n + k$ for the k-th configuration $\langle i, c, d \rangle$ of Γ. The sequence τ satisfies the *progress* condition, but not the *monotonicity* condition.

It is not difficult to express, by a TPTL-formula ϕ_M, that τ encodes a recurring computation of M. First specify the initial configuration, by

$$\phi_{INIT}: \quad x.\, x = 0 \ \wedge\ \bigcirc x.\, x = n \ \wedge\ \bigcirc^2 x.\, x = n \ \wedge\ \bigcirc^3 x.\, x = n$$

(we abbreviate a sequence of m *next* operators by $\bigcirc^m$). Then ensure proper consecution by adding a $\Box$-conjunct ϕ_i for every instruction i of M. For instance, the instruction 1 that increments the counter C and proceeds, nondeterministically, to either instruction 2 or 3, contributes the conjunct

$$\phi_1: \quad \Box x. \left(x = 1 \ \rightarrow \ \left(\begin{array}{l} \bigcirc^4 y.\, (y = 2 \ \vee\ y = 3) \ \wedge \\ \bigcirc y.\ \bigcirc^4 z.\, z = y + 1 \ \wedge \\ \bigcirc^2 y.\ \bigcirc^4 z.\, z = y \ \wedge \\ \bigcirc^3 y.\ \bigcirc^4 z.\, z = y + 1 \end{array} \right) \right).$$

The recurrence condition can be expressed by a $\Box\Diamond$-formula:

$$\phi_{RECUR}: \quad \Box\Diamond x.\, x = 0.$$

Clearly, the conjunction ϕ_M of these $n+2$ formulas is satisfiable iff M has a recurring computation. ■

Note that this proof does not require any propositions. It follows that first-order temporal logic with a single state variable ranging over the natural numbers is Π_1^1-complete, provided the underlying assertion language has at least successor (in addition to equality) as a primitive.

4.3 Presburger TPTL

We show that a certain extremely modest relaxation of the syntax of timing constraints leads to a highly undecidable logic. Consequently, TPTL with addition over time is undecidable.

Theorem 6 (Presburger TPTL) *If the syntax of* TPTL *is extended to allow multiplication by* 2, *the satisfiability problem becomes* Σ_1^1*-complete.*

Proof of Theorem 6 To encode computations of M, we use the propositions $p_1, \ldots p_n$, r_1, and r_2, precisely one of which is true in any state; hence we may identify states with propositions. The configuration $\langle i, c, d\rangle$ of M is represented by the finite sequence $p_i\, r_1^c\, r_2^d$ of states.

The initial configuration (p_0) as well as the recurrence condition ($\Box\Diamond p_0$) can be easily expressed in PTL. The crucial property that allows a temporal logic to specify the consecution relation of configurations, and thus the set of computations of M, is the ability to copy an arbitrary number of r-states. In real-time temporal logics, the times that are associated with a state sequence can be used for copying. With the availability of multiplication by 2, we are able to have the k-th configuration of a computation correspond, for all $k \geq 0$, to the finite sequence of states that is mapped to the time interval $[2^k, 2^{k+1})$. First, we force the time to increase by a strictly positive amount between successive states ($\Box x. \bigcirc y. y > x$), to ensure that every state is uniquely identifiable by its time. Then we can copy groups of r-states by establishing a one-to-one correspondence of r_j-states ($j = 1, 2$) at time t and time $2t$; clearly there are enough time gaps to accommodate an additional r_j-state when required by an increment instruction.

For instance, the instruction 1 that increments the counter C and proceeds, nondeterministically, to either instruction 2 or 3, can be expressed as follows:

$$\Box x. (p_1 \rightarrow (\psi_1 \wedge \psi_2 \wedge \psi_3(r_1) \wedge \psi_3(r_2) \wedge \psi_4)),$$

where

$$\begin{array}{ll}
\psi_1: & \Diamond z. (z = 2x \wedge (p_2 \vee p_3)), \\
\psi_2: & \Box y_1. \bigcirc y_2. (y_2 < 2x \rightarrow \Diamond z_1. (z_1 = 2y_1 \wedge \bigcirc z_2. z_2 = 2y_2)), \\
\psi_3(r): & \Box y. (y < 2x \wedge r \rightarrow \Diamond z. (z = 2y \wedge r)), \\
\psi_4: & \Box y_1. \bigcirc y_2. (y_2 = 2x \rightarrow \Diamond z_1. (z_1 = 2y_1 \wedge \bigcirc r_1 \wedge \bigcirc \bigcirc z_2. z_2 = 2y_2)).
\end{array}$$

The first conjunct ψ_1 ensures the proper progression to one of the two specified instructions, 2 or 3; the second one ψ_2 establishes a one-to-one correspondence between states in successive intervals representing configurations, while the formula $\psi_3(r)$ copies r-states. The last conjunct ψ_4 adds, finally, an r_1-state at the end of the successor configuration, as required by the increment operation. ■

We can modify this proof by reducing time to a state counter ($\Box x. \bigcirc y. y = x + 1$), and letting all propositions be false in the resulting additional (padding) states. Thus, the satisfiability problem for TPTL with multiplication by 2 is Σ_1^1-hard even if time is replaced by a state counter. As a corollary we infer that the first-order theory of the natural numbers with $\leq$, multiplication by 2, and monadic predicates is Π_1^1-complete. A similar result has been obtained independently in [Hal91], where it is shown that Presburger arithmetic becomes Π_1^1-complete with the addition of a single unary predicate.

The proof technique we used to show Presburger TPTL undecidable can be applied to many real-time specification languages, including the logics RTL [JM86], GCTL [Har88], RTTL [Ost90], and MTL [Koy90]. All of these formalisms admit addition over time as a primitive, which renders them undecidable (see [AH90]).

4.4 Dense TPTL

An alternative way of extending the expressive power of TPTL is to relax the semantics by adopting a dense time domain; that is, between any two given points in time there is another time point. We show that the resulting logic is, again, highly undecidable.

Theorem 7 (Dense TPTL) *If* TPTL *is interpreted over the rational numbers (i.e.,* $\mathsf{T} = \mathsf{Q}$*), the satisfiability problem becomes* Σ_1^1*-complete.*

Proof of Theorem 7 The proof depends, once more, on the ability to copy groups of r-states. This time, we are able to have the k-th configuration of a computation of M correspond, for all $k \geq 0$, to the finite sequence of states that is mapped to the time interval $[k, k+1)$, because dense time allows us to squeeze arbitrarily many states into every interval of length 1. Again, we identify every state with a unique time, and can then establish a one-to-one correspondence of r_j-states $(j = 1, 2)$ at time t and time $t + 1$. In fact, we may simply replace all occurrences of multiplication by 2 in the Presburger-TPTL formula encoding the recurring computations of M, by a successor operation, in order to obtain the desired dense-TPTL formula ϕ_M. ■

This proof goes through for any time domain $(\mathsf{T}, \prec, S)$ such that $(\mathsf{T}, \prec)$ is a dense linear order, and S is a unary function over T that satisfies the following two first-order axioms:

$$\forall x. x \prec S(x),$$

$$\forall x, y. (x \prec y \rightarrow S(x) \prec S(y)).$$

To show that, for arbitrary dense time domains, the satisfiability problem is in Σ_1^1, a standard Löwenheim-Skolem argument is necessary to infer the existence of countable models.

Acknowledgements. We thank Zohar Manna, Amir Pnueli, and David Dill for their guidance. Moshe Vardi and Joe Halpern gave us very helpful advice for refining our undecidability results; in particular, they pointed out to us the Σ_1^1-completeness of a problem on Turing machines.

References

[ACD90] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking for real-time systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.

[AFH91] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Tenth Annual Symposium on Principles of Distributed Computing*, pages 139–152. ACM Press, 1991.

[AH89] R. Alur and T.A. Henzinger. A really temporal logic. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 164–169. IEEE Computer Society Press, 1989.

[AH90] R. Alur and T.A. Henzinger. Real-time logics: complexity and expressiveness. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 390–401. IEEE Computer Society Press, 1990.

[AH92] R. Alur and T.A. Henzinger. Logics and models of real time: a survey. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 74–106. Springer-Verlag, 1992.

[BH81] A. Bernstein and P.K. Harter, Jr. Proving real-time properties of programs with temporal logic. In *Proceedings of the Eighth Annual Symposium on Operating System Principles*, pages 1–11. ACM Press, 1981.

[BMP81] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. In *Proceedings of the Eighth Annual Symposium on Principles of Programming Languages*, pages 164–176. ACM Press, 1981.

[EMSS89] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. Presented at the First Annual Workshop on Computer-aided Verification, Grenoble, France, 1989.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the Seventh Annual Symposium on Principles of Programming Languages*, pages 163–173. ACM Press, 1980.

[Hal91] J.Y. Halpern. Presburger arithmetic with unary predicates is Π^1_1-complete. *The Journal of Symbolic Logic*, 56(2):637–642, 1991.

[Har88] E. Harel. Temporal analysis of real-time systems. Master's thesis, The Weizmann Institute of Science, Rehovot, Israel, 1988.

[Hen90] T.A. Henzinger. Half-order modal logic: how to prove real-time properties. In *Proceedings of the Ninth Annual Symposium on Principles of Distributed Computing*, pages 281–296. ACM Press, 1990.

[Hen91] T.A. Henzinger. *The Temporal Specification and Verification of Real-time Systems.* PhD thesis, Stanford University, 1991.

[HLP90] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit-clock temporal logic. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 402–413. IEEE Computer Society Press, 1990.

[HMP92] T.A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In W. Kuich, editor, *ICALP 92: Automata, Languages, and Programming*, Lecture Notes in Computer Science 623, pages 545–558. Springer-Verlag, 1992.

[HPS83] D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[JM86] F. Jahanian and A.K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, SE-12(9):890–904, 1986.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time Systems*, 2(4):255–299, 1990.

[LA92] L. Lamport and M. Abadi. An old-fashioned recipe for real time. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 1–27. Springer-Verlag, 1992.

[LP84] O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. In *Proceedings of the 11th Annual Symposium on Principles of Programming Languages*, pages 97–107. ACM Press, 1984.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.

[MW84] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, 1984.

[OL82] S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, 1982.

[Ost90] J.S. Ostroff. *Temporal Logic of Real-time Systems.* Research Studies Press, 1990.

[PH88] A. Pnueli and E. Harel. Applications of temporal logic to the specification of real-time systems. In M. Joseph, editor, *Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 331, pages 84–98. Springer-Verlag, 1988.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[Pra80] V.R. Pratt. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20(2):231–254, 1980.

[Rog67] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill Book Company, 1967.

[SC85] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Smu68] R.M. Smullyan. *First-order Logic.* Springer-Verlag, 1968.

[WME92] F. Wang, A.K. Mok, and E.A. Emerson. Asynchronous propositional temporal logic. In *Proceedings of the 12th ICSE*, 1992.

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