A Really Temporal Logic

RAJEEV ALUR AND THOMAS A. HENZINGER

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.

Categories and Subject Descriptors: C.3 [Special-Purpose and Application-Based Systems]—real-time systems; D.2.1 [Software Engineering]: Requirements Specifications—languages; D.2.4 [Software Engineering]: Program Verification—correctness proofs; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—complexity of proof procedures; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs—logics of programs; mechanical verification; specification techniques; F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic—model theory; F.4.2 [Mathematical Logic and Formal Languages]: Formal Languages—decision problems; I.2.2 [Artificial Intelligence]: Automatic Programming—program verification; I.7 [Computers in Other Systems]—real time

General Terms: Theory, Verification

Additional Key Words and Phrases: Dense time, discrete time, EXPSpace-completeness, linear-time temporal logic, model checking, real-time requirements, \Pi^p_1-completeness

1. Introduction

Linear temporal logic is a widely accepted language for specifying properties of reactive systems and their behavior over time [Manna and Pnueli, 1992; Owicki and Lamport, 1982; Pnueli, 1977]. The tableau-based satisfiability algorithm for its propositional version, PTL, forms the basis for the automatic verification and synthesis of finite-state systems [Lichtenstein and Pnueli, 1985; Manna and Wolper, 1984].

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 [Bernstein and Harter, 1981; Koymans, 1990; Ostroff, 1990; Pnueli and Harel, 1988]. Although 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 [Abadi and Lamport, 1992; Ostroff, 1990; Pnueli and Harel, 1988]. 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

\[
\square x.(p \rightarrow \Diamond y.(q \land y \leq x + 10))
\]

(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 (\( \Pi_1 \)-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 [Emerson et al., 1990] and the explicit-clock logic XCTL [Harel et al., 1990]. 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 [Alur and Henzinger, 1989], many new results concerning real-time temporal logics have been obtained [Alur et al., 1990, 1991; Alur and Henzinger, 1990; Wang et al., 1992]. We refer to Henzinger [1990] for a complete axiomatization of TPTL and to Alur and Henzinger [1992] 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 \( \mathbb{N} \) be the set of nonnegative integers. A state is an interpretation for the propositions in \( P \) (i.e., a subset of \( P \)). A timed state sequence is an infinite sequence of states, each of which is labeled with a time from the discrete time domain \( T = \mathbb{N} \).

**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 \( \tau_i \in T, i \geq 0 \), that satisfies the following two conditions:

1. **Monotonicity.** \( \tau_i \leq \tau_{i+1} \) for all \( i \geq 0 \), and
2. **Progress.** For all \( t \in T \), 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 \( \sigma \) and a time sequence \( \tau \).

By \( \sigma^i(\tau^i) \) we denote the state (time) sequence that results from the state sequence \( \sigma \) (time sequence \( \tau \)) 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. Although 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 [Alur and Henzinger, 1992; Henzinger et al., 1992]: 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 $\pi$ and formulas $\phi$ of TPTL are inductively defined as follows:

$$
\begin{align*}
\pi & := x + c \mid c \\
\phi & := p \mid \pi_1 \leq \pi_2 \mid \pi_1 =_d \pi_2 \mid \text{false} \mid \phi_1 \rightarrow \phi_2 \mid \bigcirc \phi \mid \phi_1 \bigotimes \phi_2 \mid x. \phi
\end{align*}
$$

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

2.2.1. The Timing Constraints. The timing constraints of TPTL are of the form $\pi_1 \leq \pi_2$ and $\pi_1 =_d \pi_2$ ("time $\pi_1$ is congruent to time $\pi_2$ modulo the constant $d$"). The abbreviations $x$ (for $x + 0$), $=$, $\leq$, $\geq$, $\text{true}$, $\neg$, $\land$, and $\lor$ are defined as usual. If each term of a formula $\phi$ contains a variable, we say that $\phi$ contains no absolute time references. On the other hand, each term of a formula may contain at most one variable. Although 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.

2.2.2. The Temporal Operators. TPTL is based on the two temporal operators of PTL [Gabbay et al., 1980]. 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 $p \bigcirc q$ 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 $\bigcirc \phi$ stands for $\text{true} \bigotimes \phi$, and the always operator $\Box \phi$ stands for $\neg \bigcirc \neg \phi$.

2.2.3. 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 $\rho$, where the formula $\phi(\tau_0)$ is obtained from $\phi(x)$ by replacing all free occurrences of the variable $x$ with the constant $\tau_0$. For example, in the formula

$$\bigcirc x. (p \land x < 10),$$

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 $\mathcal{E}: V \rightarrow T$ be an interpretation (environment) for the variables. The
A Really Temporal Logic

A pair \((p, \mathcal{E})\) satisfies the TPTL-formula \(\phi\) iff \(\rho \models \mathcal{E} \phi\), where the satisfaction relation \(\models\) is inductively defined as follows:

\[
\begin{align*}
\rho \models \mathcal{E} p & \quad \text{iff} \quad p \in \sigma_0; \\
\rho \models \mathcal{E} \pi_1 \leq \pi_2 & \quad \text{iff} \quad \mathcal{E}(\pi_1) \leq \mathcal{E}(\pi_2); \\
\rho \models \mathcal{E} \pi_1 =_d \pi_2 & \quad \text{iff} \quad \mathcal{E}(\pi_1) =_d \mathcal{E}(\pi_2); \\
\rho \not\models \mathcal{E} \text{false}; \\
\rho \models \mathcal{E} \phi_1 \rightarrow \phi_2 & \quad \text{iff} \quad \rho \models \mathcal{E} \phi_1 \implies \rho \models \mathcal{E} \phi_2; \\
\rho \models \mathcal{E} (\phi \bigcirc \phi') & \quad \text{iff} \quad \rho \models \mathcal{E} \phi; \\
\rho \models \mathcal{E} \phi_1 \land \phi_2 & \quad \text{iff} \quad \rho \models \mathcal{E} \phi_2 \text{ for some } i \geq 0, \text{ and } \rho \models \mathcal{E} \phi_1 \text{ for all } 0 \leq j < i; \\
\rho \models \mathcal{E} \neg x.\phi & \quad \text{iff} \quad \rho \models \mathcal{E}[x := t_1 \ldots t_n] \phi.
\end{align*}
\]

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 \(t \in T\).

A TPTL-formula is closed if every occurrence of a variable \(x\) is within the scope of a freeze quantifier “\(x\)”.

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.

2.3.1. 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

\[
\Box x.(p \rightarrow p \neg y.(q \land y \leq x + 10)).
\]

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

\[
\Box((p \land \text{now} = x) \rightarrow p \neg (q \land \text{now} \leq x + 10))
\]

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

\[
\Box((p \land \text{now} = x) \rightarrow p \neg (q \land \text{now} = y \land y \leq x + 10)).
\]

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 [Ostroff, 1990; Pnueli and Harel, 1988]. Moreover, specification languages are often restricted to formulas with implicit or explicit quantifier prefixes [Harel, 1988; Harel et al., 1990].
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):

$$\Box \forall x.((p \land \text{now} = x) \rightarrow p \exists y. (q \land \text{now} = y \land y \leq x + 10))$$

(3)

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

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

$$\forall x. \exists y. \Box((p \land \text{now} = x) \rightarrow p \exists y. (q \land \text{now} = y \land y \leq x + 10)).$$

(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

$$((p), 0) \rightarrow ((q), 0) \rightarrow ((p), 0) \rightarrow ((q), 1) \rightarrow (\{\}, 2) \rightarrow (\{\}, 3) \rightarrow \ldots$$

(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 $\text{now}$. TPTL includes precisely those first-order temporal formulas in which each variable in $V$ is, immediately upon introduction, frozen to the time of the local temporal context (i.e., the value of $\text{now}$): the TPTL-formula $x.\phi$ is equivalent to the first-order temporal formula

$$\forall x.(x = \text{now} \rightarrow \phi)$$

or, equivalently, to the formula

$$\exists x.(x = \text{now} \land \phi).$$

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: 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 [Alur and Henzinger, 1990].

2.3.2. 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 $\delta$ time units") to PTL [Koymans, 1990; Pnueli and Harel, 1988], or branching-time logics [Emerson et al., 1990]. 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

$$x.\Diamond_{\leq \delta} y.(y \leq x + \delta \land \phi).$$

Although bounded temporal operators always relate the times of adjacent temporal contexts, TPTL admits constraints between distant contexts. For
example, the formula
\[ \square x. (p \rightarrow \lozenge (q \land \lozenge y. (r \land y \leq x + 10))) \]
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 [Emerson et al., 1990]. We can restrict TPTL to the synchronous case by postulating
\[ \square x. \lozenge y. y = x + 1. \]

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 \( \phi \) of TPTL and wish to determine if \( \phi \) is satisfiable. The tableau method searches systematically for a model of \( \phi \). It originated with the propositional calculus [Smullyan, 1968], and was first applied to obtain a decision procedure for a modal logic of computation in the case of dynamic logic [Pratt, 1980].

We follow the standard presentation of the tableau-based decision procedure for PTL [Ben-Ari et al., 1981, Manna and Wolper, 1984] and begin by constructing the initial tableau for \( \phi \). Checking the satisfiability of \( \phi \) can then be reduced to checking if the finite initial tableau for \( \phi \) contains certain infinite paths. The tableau method for PTL is, in fact, subsumed by our procedure as the special case in which \( \phi \) contains no timing constraints.

3.1.1. Preliminary Assumptions. For the moment, we assume that

1. \( \phi \) contains no absolute time references; that is, every time in \( \phi \) contains a variable. Thus, we may perform simple arithmetic manipulations so that all timing constraints in \( \phi \) 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. \( \phi \) contains only the temporal operators \( \bigcirc \) and \( \square \); that is, the first argument of every occurrence of the until operator \( \bigotimes \) in \( \phi \) is true.

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

A timed state sequence \( \rho = (s, \tau) \) is \( \Delta \)-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 \( \Delta \)-bounded
timed state sequence is at most $\Delta$ and the time increases from a state to its successor state by no more than $\Delta$.

To begin with, we restrict ourselves to $\Delta$-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 $\text{Prev}_i$, $0 \leq \delta \leq \Delta$, that represent in the initial state, the initial time $\delta$, and in all other states, the time increase $\delta$ from the predecessor state. Formally, we can capture the (state and) time information in a timed state sequence $p = (\sigma, \tau)$ by a state sequence $\hat{\sigma}$ with

$$\hat{\sigma}_i = \sigma_i \cup \{\text{Prev}_{i-\delta_i}\}$$

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 show how we can find an appropriate constant $\Delta$ for the given formula $\phi$.

3.1.2. Updating Timing Constraints. The key observation underlying the tableau method for PTL is that any formula can be split into two conditions: a nontemporal (“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 $\psi$ or $\Box \Diamond \psi$ 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 $\delta$ 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 $\psi$ 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 $\psi$ 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 $\psi$ on the initial time $x$ by the time difference $\delta$. 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:

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

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

$$x.\Box y. (p \rightarrow \text{false}).$$
In general, given a TPTL-formula \( x.\psi \) and \( \delta \in \mathbb{N} \), 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 \( x \leq y + (c + 1) \leq x \) with \( x \equiv y + (c + 1) \) and \( x \equiv (c + 1) \mod d \), respectively, provided that the occurrence of \( x \) in the specified terms and formulas is free in \( \psi^\delta \).

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 \( \mathscr{E} \) be an environment, and let \( \delta \in \mathbb{N} \) such that \( \delta \leq \tau_0 \). Then \( \rho \models \mathscr{E} x.\psi^\delta \) iff \( \rho \models \mathscr{E}|x - \tau_0 - \delta| \psi \) for every TPTL-formula \( x.\psi \).

**Proof of Lemma 1.** The proof proceeds by a straightforward induction on the structure of \( \psi \). □

### 3.1.3. Closure of a TPTL-Formula

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

- \( \text{Sub}(z.(\psi_1 \rightarrow \psi_2)) = \{z.\psi_1, z.\psi_2\} \)
- \( \text{Sub}(z.\bigcirc \psi) = \{z.\psi^\delta \mid \delta \geq 0\} \)
- \( \text{Sub}(z.\square \psi) = \{z.\psi, z.\bigcirc \square \psi\} \)
- \( \text{Sub}(z.x.\psi) = \{z.\psi[x := z]\} \)

where the formula \( \psi[x := z] \) results from \( \psi \) 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 \( \phi \) if \( \phi \) contains a subformula of the form \( x \leq y + (c - 1) \) or \( x + (c - 1) \leq y \), or \( \phi \) contains the predicate symbol \( =_c \). Let \( C \) be the largest constant that occurs in the formula \( \phi \). The closure set of \( \phi \) is finite, because for all \( \delta \geq C \) and for all formulas \( z.\psi \) in \( \text{Closure}(\phi) \), \( z.\psi^\delta \) is \( z.\psi^C \).

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

\[
k_{\pi_1 \leq \pi_2} = k_{\pi_1} \cdot k_{\pi_2} \quad \text{and} \quad k_{x = y} = c \cdot k_{x_1} \cdot k_{x_2};
k_{\text{false}} = 1 \quad \text{and} \quad k_{\phi_1 \rightarrow \phi_2} = k_{@\phi_1} \cdot k_{\phi_2};
k_{\bigcirc \phi} = k_{\phi}, \quad k_{\phi_1 \bigcirc = \phi_2} = k_{@\phi_1} \cdot k_{\phi_2}; \quad \text{and} \quad k_{x.\phi} = k_{\phi},
\]

where \( k_{x.\phi} = c + 1 \) 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 \( \phi \), and let \( k \) be the product of all constants that occur in \( \phi \). Then \( |\text{Closure}(\phi)| \leq 2nk \).
Proof of Lemma 2. Given a formula \( z \cdot \phi' \), we define, by induction on the structure of \( \phi' \), the set \( D_{\phi'} \) of formulas that contains \( \phi' \) and is closed under updating of timing constraints: the set \( C_{\phi'} \) is, in addition, closed under subformulas:

\[
C_p = D_p = \{p\},
\]

\[
C_{x \leq y + c} = D_{x \leq y + c} = \{x \leq y + c, x \leq y + (c - 1), \ldots, x \leq y\},
\]

\[
C_{x = y + c} = D_{x = y + c} = \{x = y + (d - 1), x = y + (d - 2), \ldots, x = y\},
\]

\[
C_{x \neq y + c} = C_{x \neq y} \cup D_{x \neq y},
\]

\[
C_{x < y + c} = C_{x < y} \cup D_{x < y + c},
\]

\[
C_{x > y + c} = C_{x > y} \cup D_{x > y + c},
\]

\[
C_{x = y} = C_{x = y} \cup D_{x = y},
\]

\[
C_{x > y} = C_{x > y} \cup D_{x > y},
\]

\[
C_{x < y} = C_{x < y} \cup D_{x < y},
\]

\[
C_{x} = C_{x} \cup D_{x},
\]

\[
C_{\square \psi} = C_{\square \psi} \cup D_{\square \psi},
\]

\[
C_{\dashv \phi} = C_{\dashv \phi} \cup D_{\dashv \phi},
\]

\[
C_{\dashv \phi} = C_{\dashv \phi} \cup D_{\dashv \phi},
\]

Here \( x \cdot E = \{x, \phi | \psi \in E\} \) 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 \( D_{\phi} \subseteq C_{\phi} \). It is straightforward to show by induction on the structure of \( \phi' \) that

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

From (1) and (3) it follows that \( Closure(z \cdot \phi') \subseteq z \cdot C_{\phi} \). 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 \( \phi' \). □

3.1.4. 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 \( \text{Prev}_\delta \), \( 0 \leq \delta \leq \Delta \), that denotes the time increase from the predecessor states.

Formally, the vertices of a tableau for \( \phi \) are the maximally consistent subsets of the finite universe

\[
Closure^*(\phi) = Closure(\phi) \cup \{\text{Prev}_\delta | 0 \leq \delta \leq \Delta\}
\]

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

1. \( \text{Prev}_\delta \) is in \( \Phi \) for precisely one \( 0 \leq \delta \leq \Delta \); this \( \delta \in \mathbb{N} \) is referred to as \( \delta_\phi \).
2. \( z.(z \sim z + c) \) is in \( \Phi \) iff \( 0 \sim c \) holds in \( \mathbb{N} \) (for \( \sim \) being one of \( \leq \), \( \geq \), and \( = \)).
3. \( z.\text{false} \) is not in \( \Phi \).
4. \( z.(\psi_1 \rightarrow \psi_2) \) is in \( \Phi \) iff either \( z.\psi_1 \) is not in \( \Phi \) or \( z.\psi_2 \) is in \( \Phi \).
5. \( z.\square \psi \) is in \( \Phi \) iff both \( z.\psi \) and \( z.\boxdot \square \psi \) are in \( \Phi \).
6. \( z.x.\psi \) is in \( \Phi \) iff \( z.\psi[x := z] \) is in \( \Phi \).
The initial tableau $T(\phi)$ for the TPTL-formula $\phi$ is a directed graph whose vertices are the consistent subsets of $\text{Closure}^*(\phi)$, and which contains an edge from $\Phi$ to $\Psi$ iff, for all formulas $z \circ \psi$ in $\text{Closure}^*(\phi)$,

$$z \circ \psi \in \Phi \iff z \psi^\delta \in \Psi.$$ 

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 $\phi$ is that every model of $\phi$ 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 $\phi$ is satisfied by a model whose state part, extended by the time-difference propositions $\text{Prev}_t$, is eventually periodic.

To be precise, an infinite path

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

through a tableau is a $\phi$-path if it satisfies the following three conditions:

1. **Initiality.** $\phi \in \Phi_0$.
2. **Fairness.** All eventualities are satisfied along $\Phi$ in time or, equivalently, all missing invariances are violated along $\Phi$ in time; that is, for all $z \circ \psi \in \text{Closure}^*(\phi)$ and $i \geq 0$, $z \circ \psi \notin \Phi_i$ implies $z \psi^\delta \in \Phi_j$ for some $j \geq i$ with $\delta = \sum_{i < \delta} \delta_{\Phi_i}$.
3. **Progress.** $\delta_{\Phi_i} > 0$ for infinitely many $i \geq 0$.

Every $\phi$-path in the initial tableau for $\phi$ can be reduced to a $\phi$-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 $\phi$-Paths).** Suppose that the initial tableau $T(\phi)$ for the TPTL-formula $\phi$ consists of $m$ vertices. If $T(\phi)$ contains a $\phi$-path, then it contains a $\phi$-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 $\phi$ and $k$ is the product of all constants that occur in $\phi$.

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

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

is a $\phi$-path of the desired form. □
3.1.5. Tableau Decision Procedure. The following main lemma suggests a decision procedure for TPTL: to determine if the TPTL-formula $\phi$ is satisfiable, construct the initial tableau $T(\phi)$ and check if it contains any $\phi$-paths.

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

(1) [Correctness]. If the initial tableau $T(\phi)$ for the TPTL-formula $\phi$ contains a $\phi$-path, then $\phi$ is satisfiable.

(2) [Completeness]. If $\phi$ has a $\Delta$-bounded model, then $T(\phi)$ contains a $\phi$-path.

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

(1) Given a $\phi$-path $\Phi = \Phi_0\Phi_1 \cdots$ 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 $\tau_i = \tau_{i-1} + \delta_p$. Note that the time sequence $\tau$ satisfies the progress condition because $\Phi$ does. We show, by induction on the structure of $\psi$, that for all $i \geq 0$ and $\psi \in \text{Closure}^*(\phi)\psi \in \Phi_i$, iff $\rho_i \models \psi$. Since $\phi \in \Phi_0$, it follows that $\rho$ is a model of $\phi$.

For a proposition $z.p \in \text{Closure}^*(\phi)$, we have $z.p \in \Phi_i$ iff $p \in \sigma_i$, iff $\rho_i \models z.p$. Let $\sim \rho \models z.p \in \Phi_i$, iff $p \models z.p$. Let $\gamma = \rho_i \models z.p \in \Phi_i$, iff $p \models z.p$.

Now assume that $z.\psi \in \text{Closure}^*(\phi)$ and let $\delta = \delta_{\Phi_i+1}$; that is, $\tau_{i+1} = \tau_i + \delta$. Then $z.\psi \in \Phi_i$, iff $z.\psi \in \Phi_{i+1}$. By the induction hypothesis, this is the case iff $\rho_{i+1} \models z.\psi$. By Lemma 1, this is the case iff $\rho_{i+1} \models z.\psi$.

For the case that $z.\overline{x} \psi \in \text{Closure}^*(\phi)$, we first prove that $z.\overline{x} \psi \in \Phi_i$ iff $z.\psi_{i+1} \in \Phi_i$ for all $i \geq 0$. Let $\delta = \delta_{\Phi_i+1}$ and note that $\delta_i = \sum_{k \leq i} \delta_{\phi_i}$ by our choice of $\tau$.

We use induction on $j$ to show that $z.\overline{x} \psi \in \Phi_i$ implies $z.\overline{x} \psi_{i+1} \in \Phi_i$ for all $j \geq i$. Suppose that $z.\overline{x} \psi_{i+1} \in \Phi_i$ for an arbitrary $j \geq i$. By the consistency of $\Phi_i$, also $z.\overline{x} \psi_{i+1} \in \Phi_i$, and therefore $z.\overline{x} \psi_{i+1} \in \Phi_{i+1}$. Invoking again the consistency of $\Phi_i$, we conclude that $z.\psi_{i+1} \in \Phi_i$ for all $j \geq i$.

On the other hand, suppose that $z.\overline{x} \psi \notin \Phi_i$. Since $\Phi$ is a $\phi$-path, there is some $j \geq i$ such that $z.\psi_{i+1} \in \Phi_j$.

By the induction hypothesis, it follows that $z.\overline{x} \psi \in \Phi_i$ iff $\rho_i \models z.\psi_{i+1}$ for all $j \geq i$. By Lemma 1, this is the case iff $\rho_i \models z.\psi_{i+1}$ for all $j \geq i$; that is, iff $\rho_i \models z.\overline{x} \psi$.

Finally, consider the case that $z.x.\psi \in \text{Closure}^*(\phi)$. 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 $\Delta$-bounded model of $\phi$. The subsets $\Phi_i$, for $i \geq 0$, of $\text{Closure}^*(\phi)$ are defined as follows: $\text{Prev}_{i+1} \in \Phi_i$, and $\psi \in \Phi_i$, for all $\psi \in \text{Closure}^*(\phi)$, iff $\rho_i \models \psi$. We show that $\Phi = \Phi_0\Phi_1 \cdots$ is a $\phi$-path through the initial tableau $T(\phi)$.

By inspecting the consistency rules, it is evident that every $\Phi_i$ is consistent. To prove that $\Phi$ is an infinite path through $T(\phi)$, we also have to show that, for all $i \geq 0$, there is an edge from $\Phi_i$ to $\Phi_{i+1}$. Suppose that $z.\varphi \psi \in \text{Closure}^*(\phi)$ and let $\delta = \tau_{i+1} - \tau_i$. Then $z.\varphi \psi \in \Phi_i$ iff $\rho_i \models z.\varphi \psi$ iff $\rho_i \models z.\varphi \psi$. \hfill \Box
A Really Temporal Logic

\[ \Rightarrow \bar{\varepsilon}_{[\tau_j - \tau_i]} \psi. \] By Lemma 1, this is the case iff \( \rho^{i+1} \models z.\psi^5 \); that is, iff \( z.\psi^5 \in \Phi_{i+1} \). Since also \( P_{\phi} \in \Phi_{i+1} \), the initial tableau for \( \phi \) contains an edge from \( \Phi_i \) to \( \Phi_{i+1} \).

We now show that the infinite path \( \Phi \) is indeed a \( \phi \)-path. It satisfies the progress condition because the time sequence \( \tau \) does. To see that \( \phi \in \Phi_0 \), observe that \( \rho \) is a model of \( \phi \). It remains to be established that all eventualities in \( \Phi \) are satisfied in time. Suppose that \( z.\square \psi \in \text{Closure}^k(\phi) \) and \( z.\square \psi \notin \Phi_j \); that is, \( \rho^i \models z.\diamond \neg \psi \) and therefore \( \rho^j \models z.\psi^5 \) for some \( j \geq i \). Let \( \delta = \tau_j - \tau_i \); thus \( \delta = \sum_{i < k \leq j} \delta_{k-1} \). Then \( \rho^j \models \neg z.\psi^5 \) by Lemma 1, which implies that \( z.\psi^5 \notin \Phi_j \).

After constructing the initial tableau \( T(\phi) \) for the formula \( \phi \), we delete all vertices that are not on a \( \phi \)-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 [Manna and Wolper, 1984]. The remaining state graph is called the final tableau for \( \phi \). It follows that a TPTL-formula \( \phi \) has a \( \Delta \)-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 \( \phi \) and \( k \) is the product of all constants that occur in \( \phi \). Thus, provided that \( \Delta \) is dominated by \( 2^{nk} \), the initial tableau \( T(\phi) \) can be constructed and checked for \( \phi \)-paths in deterministic time exponential in \( nk \). We show next that \( \Delta \) can indeed be bounded by \( k \).

3.1.6. Bounding the Size of Time Steps. Given a formula \( \phi \), we finally determine the bound \( \Delta \) on the time increase between two successive states such that the satisfiability of \( \phi \) is not affected; that is, we choose the constant \( \Delta \in \mathbb{N} \) such that \( \phi \) is satisfiable iff it has a \( \Delta \)-bounded model.

Let \( c \) be the largest constant in \( \phi \) 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 \( \phi \). If the time increase \( \delta \) between two states is greater than or equal to \( c \), it obviously suffices to know the residues of \( \delta \) modulo \( c_1, \ldots, c_m \) in order to update, in a tableau, all timing constraints correctly. Indeed, for checking the satisfiability of \( \phi \), the arbitrary step-width \( \delta \) 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 \( \phi \) is satisfiable and \( k \) is the product of all constants that occur in \( \phi \), then \( \phi \) 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 \( \phi \), let the time sequence \( \tau' \) be such that, for all \( i \geq 0 \), \( \tau_i = \tau_{i-1} + (\tau_i - \tau_{i-1}) \) if \( \tau_i - \tau_{i-1} < c \); otherwise, choose \( \tau'_i \) 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 \( \phi \).

Combining this result with the tableau method developed above, we arrive at the conclusion that the satisfiability of the TPTL-formula \( \phi \) is decidable in deterministic time exponential in \( nk \). Moreover, Lemma 3 implies that every satisfiable formula \( \phi \) is satisfiable in a model whose size is, in the sense mentioned above, exponential in \( nk \).
Remember that we have restricted \( \phi \) to contain no until operators and no absolute time references. We now show that both assumptions can be relaxed.

### 3.1.7. Until Operators

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

\[
\text{Sub}(z.(\psi_1 \& \psi_2)) = \{ z.\psi_1, z.\psi_2, z.\bigcirc(\psi_1 \& \psi_2) \}
\]

and add the following condition on the consistency of a set \( \Phi \subseteq \text{Closure}^*(\phi) \) of formulas: \( z.(\psi_1 \& \psi_2) \) is in \( \Phi \) iff either \( z.\psi_2 \) is in \( \Phi \), or both \( z.\psi_1 \) and \( z.\bigcirc(\psi_1 \& \psi_2) \) are in \( \Phi \).

Finally, the fairness requirement on a \( \phi \)-path \( \Phi_0 \Phi_1 \Phi_2 \cdots \) is generalized to the condition that \( z.(\psi_1 \& \psi_2) \in \Phi_i \) implies that, for all \( i \geq 0 \), \( z.\psi_2^i \in \Phi_i \) for some \( j \leq i \) with \( \delta = \sum_{i, z, k < j} \delta_{\phi_i} \). It is not hard to check that the Lemmas 2, 3, 4, and 5 allow the addition of until operators in this way.

### 3.1.8. 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 \( \phi \) for satisfiability. Let \( x \) be a variable that does not occur in \( \phi \) and replace every variable-free term \( c \) in \( \phi \) with the term \( x + c \), thus obtaining the new formula \( \phi^R \) (which may contain free occurrences of \( x \)). The following lemma allows us to reduce the satisfiability problem for \( \phi \) 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 \( \phi \) is satisfiable iff the formula \( x.\bigcirc \phi^R \) is satisfiable, where \( x \) does not occur in \( \phi \).

**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 \( \sigma_i' = \sigma_i \) and \( \tau_i' = \tau_i \) 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 \( \rho' \models \phi \). \( \square \)

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

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

Note that the length \( l \) of any formula \( \phi \) whose constants are presented in a logarithmic (e.g., binary) encoding, is within a constant factor 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 \( \phi \), and successively adding new states only when needed [Manna and Wolper, 1984]. This stepwise procedure, however, does not lower the doubly exponential deterministic-time bound; as we 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 \( \phi \)-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 [Sistla and Clarke, 1985]. The extra exponential is caused by the succinct representation of time constants in TPTL and is typical for many real-time specification languages [Alur and Henzinger, 1990].

**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 [Manna and Wolper, 1984]; the hardness part is patterned after a proof that the universality problem for regular expressions with exponentiation is EXPSPACE-hard [Hopcroft and Ullman, 1979].

(1) 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 \( \phi \)-path of the form stated in Lemma 3. In trying to construct such a \( \phi \)-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 \( \phi \), it follows that this nondeterministic procedure requires only exponential space.

(2) EXPSPACE-hardness. Consider a deterministic \( 2^n \)-space-bounded Turing machine \( M \). For each input \( X \) of length \( n \), we construct a TPTL-formula \( \phi_X \) 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 \( \phi \) of length \( I \) takes space \( S(I) \geq 2^{cI/\log I} \) infinitely often. Thus, it suffices to construct, given the initial tape contents \( X \),

1. a sufficiently succinct formula \( \phi_X \) that describes the (unique) computation of \( M \) on \( X \) as an infinite sequence of propositions, and
2. a sufficiently succinct formula \( \phi_{\text{ACCEPT}} \) that characterizes the computation of \( M \) on \( X \) as accepting.
Then the implication 
\[ \phi_X \rightarrow \phi_{\text{ACCEPT}} \]
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:

\[
\hat{p}_i: \quad p_i \land \bigvee_{i' \neq i} \neg p_{i'} \land \bigvee_{j} \neg q_j,
\]
\[
r_{i,j}: \quad p_i \land q_j \land \bigvee_{i' \neq i} \neg p_{i'} \land \bigvee_{j' \neq j} \neg q_{j'},
\]
\[
s: \quad \land_{i} \neg p_i \land \land_{j} \neg q_j.
\]

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 \( \phi_X \) to consist of \( \square x. \circ y.y = x + 1 \), forcing time to resemble a state counter, and the following two conjuncts, which correspond to the requirements (1) and (2), respectively:

\[
\phi_{\text{INITIAL}}: \quad s \land \bigcirc r_{Xi,0} \land x. \land_{1 \leq i \leq X} \square y. (y = x + 1 \rightarrow \hat{p}_X) \land x. \square y(y = x + n < y \leq x + 2^n \rightarrow \hat{p}_0),
\]
\[
\phi_{\text{MOVE}}: \quad \square x. (s \rightarrow \bigcirc y(y = x + 2^n + 1 \land s)) \land \land_{P, Q, R} \square x. (P \land \bigcirc Q \land \bigcirc R \rightarrow \bigcirc y(y = x + 2^n + 2 \land f_M(P, Q, R))).
\]

Here \( P, Q, \) and \( R \) each range over the proposition \( \hat{p}_i, r_{i,j}, \) 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 \( f_M(\hat{p}_i, r_{i,j}, \hat{p}_{j'}) = \hat{p}_k \) and \( f_M(r_{i,j}, \hat{p}_{j'}, \hat{p}_{j''}) = r_{i', j''} \).

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

\[
\phi_{\text{ACCEPT}}: \quad \bigcirc \bigvee_{i,F} r_{i,F}.
\]

The lengths of \( \phi_{\text{INITIAL}}, \phi_{\text{MOVE}}, \) and \( \phi_{\text{ACCEPT}} \) 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 \( \phi_X \).

3.3. REAL-TIME VERIFICATION. Researchers have proposed a variety of different languages for defining real-time systems [Alur and Henzinger, 1992]. 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,
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 $\phi$ of a linear temporal logic. The verification problem asks if all possible runs of the system $S$ satisfy the specification $\phi$.

In the case of PTL, the tableau construction can be used to solve the verification problem [Lichtenstein and Pnueli, 1984]. 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 $\phi$ 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 $\phi$ is satisfied by all paths in a given structure is EXPSPACE-complete, and thus, in general, equally hard as deciding if $\phi$ is satisfied by all timed state sequences. The practitioner will note that the 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.

### 3.3.1. 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 $\text{Prev}_{\delta}$ or $\text{Prev}_{>\delta}$, which indicates that the time difference from the predecessor location is either exactly $\delta$ time units or unspecified, respectively.

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

- a finite set $L_0$ of locations;
- a state labeling function $\mu: L \rightarrow 2^P$ that labels every location $l \in L$ with a state $\mu_l \subseteq P$;
- a time labeling function $\nu$ that labels every location $l \in L$ with a time-difference proposition $\nu_l$, which is either $\text{Prev}_{\delta}$ for some $\delta \in \mathbb{N}$, or $\text{Prev}_{>\delta}$;
- a set $L_0 \subseteq L$ of initial locations;
- a set $\mathcal{T} \subseteq L^2$ of transitions.

A timed state sequence $p = (\sigma, \tau)$ is a computation of the timed state graph $T$ if there is an infinite path $l_0l_1l_2 \cdots$ through $T$ such that for all $i \geq 0$, $\sigma_i = \mu_{l_i}$ and if $\nu_i = \text{Prev}_{\delta}$, then $\tau_{i+1} = \tau_i + \delta$.

A TPTL-formula $\phi$ is satisfied (valid) in the timed state graph $T$ if some (all) computations of $T$ are models of $\phi$. The problem of model checking is to determine if a formula is valid in a timed state graph.

### 3.3.2. Model Checking

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

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

— The state information in \( l \) and \( \Phi \) is compatible; that is, \( p \in \mu_l \) iff \( z.p \in \Phi \) for all propositions \( p \in P \).
— The time information in \( l \) and \( \Phi \) is compatible; that is, either \( v_l = \text{Prev}_{\geq 0} \), or \( v_l = \delta_\theta \), or \( \delta_\theta = \text{Prev}_{k'} \) and \( v_l = \text{Prev}_k \) for some \( k' \geq k \).

The product \( T \) contains an edge from the vertex \((l_1, \Phi_1)\) to the vertex \((l_2, \Phi_2)\) iff \( T_\delta \) contains an edge from \( l_1 \) to \( l_2 \) and \( T(\phi) \) contains an edge from \( \Phi_1 \) to \( \Phi_2 \).

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

An infinite path through the product \( T_\delta \times T(\phi) \) is a \( \phi \)-path if its second projection is a \( \Phi \)-path through \( T(\phi) \); it is an initialized \( \Phi \)-path if, in addition, it starts at a vertex whose first projection is an initial vertex of \( T_\delta \). 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 \( T_\delta \) satisfies the TPTL-formula \( \phi \) iff the product \( T_\delta \times T(\neg \phi) \) contains an initialized \( \Phi \)-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 \( \phi \):

1. Construct the timed state graph \( T_\delta \) for \( S \).
2. Construct the initial tableau \( T(\neg \phi) \) for the negated formula \( \neg \phi \).
3. Construct the tableau product \( T = T_\delta \times T(\neg \phi) \).
4. Check if \( T \) contains an initialized \( \neg \phi \)-path. The system \( S \) meets the specification \( \phi \) 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 [Lichtenstein and Pnueli, 1985].

Since a structure can be checked for \( \phi \)-paths in polynomial time, the running time of the model checking algorithm is determined by the size of the tableau product \( T \), which contains \( O(|T_\delta| \cdot |T(\phi)|) \) vertices. The size of \( T_\delta \) typically is exponentially larger than the description of \( S \) itself [Henzinger, 1991]. We have seen that the size of \( T(\phi) \) can be two exponentials larger than \( \phi \).

**Theorem 3 (Model Checking).** The problem if a TPTL-formula \( \phi \) 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 \( \phi \).

3.3.3. 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 \( \phi \) iff the product \( T \times T(\neg \phi) \) contains an initial-
ized $\neg \phi$-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 problem for TPTL to model checking, it suffices to give a timed state graph $T$ of constant size such that formula $\phi$ is valid iff $\phi$ 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 $Prev_{\geq 0}$. \hfill \square

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 $\Sigma_1^1$-complete, by reducing a $\Sigma_1^1$-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 [Rogers, 1967]).

4.1. A $\Sigma_1^1$-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 nonjump instruction, $M$ proceeds nondeterministically to one of two specified instructions. We represent the configurations of $M$ by triples $(i, c, d)$, 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 $(0, 0, 0)$. 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 $\Sigma_1^1$-complete [Harel et al., 1983]. 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 $\Sigma_1^1$-hard.

**Proof of Lemma 8.** Every $\Sigma_1^1$-formula is equivalent to a $\Sigma_1^1$-formula $\chi$ of the form

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

for a recursive predicate $g$ [Harel et al., 1983]. For any such $\chi$, we can construct a nondeterministic 2-counter machine $M$ that has a recurring computation iff $\chi$ 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 $\Sigma^1_1$-complete. First, we observe that the satisfiability of a formula $\phi$ can, in all cases, be phrased as a $\Sigma^1_1$-sentence, asserting the existence of a model for $\phi$. Any timed state sequence $\rho$ for $\phi$ can be encoded, in first-order arithmetic, by finitely many infinite sets of natural numbers; say, one for each proposition $p$ in $\phi$, 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 $\phi$ holds in $\rho$. We conclude that satisfiability is in $\Sigma^1_1$.

To show that the satisfiability problem of a logic is $\Sigma^1_1$-hard, it suffices, given a nondeterministic 2-counter machine $M$, to construct a formula $\phi_M$ such that $\phi_M$ 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 $\Sigma^1_1$-complete.

PROOF OF THEOREM 5. We encode a computation $\Gamma$ of $M$ by the "time" sequence $\tau$ 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 $\Gamma$. The sequence $\tau$ satisfies the progress condition, but not the monotonicity condition.

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

$$\phi_{\text{INIT}}: \quad x \cdot x = 0 \land \bigcirc x \cdot x = n \land \bigcirc^2 x \cdot x = n \land \bigcirc^3 x \cdot x = n$$

(we abbreviate a sequence of $m$ next operators by $\bigcirc^m$). Then, ensure proper consecution by adding a $\Box$-conjunct $\phi_i$ 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

$$\phi_1: \quad \Box x. \begin{cases} \bigcirc^4 y . (y = 2 \lor y = 3) \\ \land \bigcirc y \cdot \bigcirc^4 z . z = y + 1 \\ \land \bigcirc^2 y \cdot \bigcirc^4 z . z = y \\ \land \bigcirc^3 y \cdot \bigcirc^4 z . z = y + 1 \end{cases}.$$ 

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

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

Clearly, the conjunction $\phi_M$ 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 the
first-order temporal logic with a single state variable ranging over the natural
numbers is \( \Pi^1_1 \)-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 \( \Sigma^1_1 \)-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 \( r_1, r_2, \ldots \) of states.

The initial configuration \( \langle p_0 \rangle \) 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 (\( \Box x. \diamond y. y > x \)), to ensure that every state is uniquely identifi-
able by its time. Then, we can copy groups of \( r \)-states by establishing a
one-to-one correspondence of \( r \)-states \( (j = 1, 2) \) at time \( t \) and time \( 2t \); clearly,
there are enough time gaps to accommodate an additional \( r \)-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:

\[
\Box x.(p_1 \rightarrow (\psi_1 \land \psi_2 \land \psi_3(r_1) \land \psi_3(r_2) \land \psi_4)),
\]

where

\begin{align*}
\psi_1 : & \quad \diamond z. (z = 2x \land (p_2 \lor p_3)), \\
\psi_2 : & \quad \Box y_1. \Box y_2. (y_2 < 2x \rightarrow \diamond z_1, (z_1 = 2y_1 \land \Box z_2, z_2 = 2y_2)), \\
\psi_3(r) : & \quad \Box y_3. (y < 2x \land r \rightarrow \diamond z_3, (z = 2y \land r)), \\
\psi_4 : & \quad \Box y_4. (y_2 = 2x \rightarrow \diamond z_1, (z_1 = 2y_1 \land \Box r_1 \land \Box z_2, z_2 = 2y_2)).
\end{align*}

The first conjunct \( \psi_1 \) ensures the proper progression to one of the two
specified instructions, 2 or 3; the second one \( \psi_2 \) establishes a one-to-one
correspondence between states in successive intervals representing configura-
tions, while the formula \( \psi_3(r) \) copies \( r \)-states. The last conjunct \( \psi_4 \) adds,
finally, an \( r \)-state at the end of the successor configuration, as required by the
increment operation. \( \square \)

We can modify this proof by reducing time to a state counter (\( \Box x. \Box y. y = x + 1 \)), and letting all propositions be false in the resulting additional (padding)
states. Thus, the satisfiability problem for TPTL with multiplication by 2 is
$\Sigma^1_1$-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 $\Pi^1_1$-complete. A similar result has been obtained independently in [Halpern, 1991], where it is shown that Presburger arithmetic becomes $\Pi^1_1$-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 [Jahanian and Mok, 1986], GCTL [Harel, 1988], RTTL [Ostroff, 1990], and MTL [Koymans, 1990]. All of these formalisms admit addition over time as a primitive, which renders them undecidable [Alur and Henzinger, 1990]).

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., $T = \mathbb{Q}$), the satisfiability problem becomes $\Sigma^1_1$-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 $\phi_M$. □

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

\[ \forall x. x < S(x), \]
\[ \forall x, y. (x < y \rightarrow S(x) < S(y)). \]

To show that, for arbitrary dense time domains, the satisfiability problem is in $\Sigma^1_1$, a standard Löwenheim–Skolem argument is necessary to infer the existence of countable models.

**Acknowledgments.** 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 $\Sigma^1_1$-completeness of a problem on Turing machines.

**References**


RECEIVED JANUARY 1990; REVISED SEPTEMBER 1992; ACCEPTED OCTOBER 1992