Model-Checking for Real-Time Systems

Rajeev Alur[1] * Costas Courcoubetis[2] David Dill[1] †

[1]Department of Computer Science, Stanford University.
[2]AT&T Bell Laboratories, Murray Hill.

Abstract. Model-checking is a method of verifying concurrent systems in which a state-graph model of the system behavior is compared with a temporal logic formula. This paper extends CTL model-checking to the analysis of *real-time* systems, whose correctness depends on the magnitudes of the timing delays. For specifications, we extend the syntax of CTL to allow quantitative temporal operators such as $\exists\Diamond_{<5}$. The formulas of the resulting logic, TCTL, are interpreted over *continuous computation trees*, trees in which paths are maps from the set of nonnegative reals to system states. To model finite-state systems we introduce the notion of *timed graphs* — state-transition graphs extended with a mechanism that allows the expression of constant bounds on the delays between the state transitions.

As our main result, we develop an algorithm for model-checking, that is, for determining the truth of a TCTL-formula with respect to a timed graph. We argue that choosing a dense domain, instead of a discrete domain, to model time, does not blow up the complexity of the model-checking problem. On the negative side, we show that the denseness of the underlying time domain makes TCTL Π^1_1-hard. The question of deciding whether a given TCTL-formula is implementable by a timed graph is also undecidable.

1 Introduction

As digital systems become smaller and cheaper, their use to control and interact with physical processes will inevitably increase. These systems must meet hard real-time constraints: it is not sufficient for landing gear to descend "eventually" — there are lower and upper bounds on when the event can occur relative to other events. Since bugs in these systems can be subtle and expensive (possibly even life-threatening), correctness is critical. However, traditional methods for reasoning about reactive systems ([Pn77], [OL82], [EC82], [MP89]) abstract away from quantitative time preserving only qualitative properties (such as "eventually p holds"). In this paper, we extend these techniques to reason about quantitative as well as qualitative timing behavior.

One of the most successful techniques for *automatic verification* of finite-state systems has been *model-checking*: a property is given as a formula of a propositional temporal logic and automatically compared with a state-transition graph representing the actual behavior of the system. One of the advantages of this method is its efficiency: model-checking is linear in the product of the size of the structure and the size of the formula, when the logic is the branching-time temporal logic CTL (computation tree logic) ([CES86]). CTL model checking has been used for proving the correctness of concurrent systems such as circuits and communication protocols. Our approach is to augment both the state-transition graph and logical formulas with quantitative timing information.

First, we incorporate time explicitly in the underlying formal semantics for processes. Previous work on modeling real-time systems has used two basic approaches. *Discrete-time* models use the domain of integers to model time (e.g. [EMSS89]). This approach requires that continuous time be approximated by choosing some fixed quantum *a priori*, which limits the accuracy with which the system can be modeled. In theory, this allows the possibility that interesting behaviors (e.g. bugs) will be overlooked. In practice, the choice of a sufficiently small time quantum to be reasonably safe may blow up the state space to the point where verification is no longer feasible.

The *fictitious-clock* approach introduces a special *tick* transition in the model. Here time is viewed as a global state variable that ranges over the domain of natural numbers, and is incremented by one with every *tick* transition (e.g. [Ku83], [Os87], [AH89], [Bu89]). This model allows arbitrarily many transitions of any process between two successive *tick* transitions. The timing delay between two events is measured by counting the number of *tick*s between them. When we require that there be k *tick*s between two transitions, we

*Supported by the NSF grant CCR-8812595, by the DARPA contract N00039-84-C-0211, and by the USAF Office of Scientific Research under contracts 88-0281 and 90-0057.
†Supported by NSF under grant MIP-8858807.

CH2897-7/90/0000/0414$01.00 © 1990 IEEE

can only infer that the delay between them is larger than $k-1$ time units and smaller than $k+1$ time units. Consequently, it is impossible to state precisely certain simple requirements on the delays such as "the delay between two transitions equals 2 seconds."

These models are used, despite their apparent inaccuracy, because they are straightforward extensions of existing temporal models. Unlike this previous work, we model time, more realistically, as a continuous quantity: with each transition we associate a time value chosen from the set of nonnegative reals (R). We regard computations as continuous trees, in which the paths are maps from the time domain (R) to states of the system. This model differs from discrete-time models because it allows an unbounded number of environment events to happen between two successive system transitions. It differs from the fictitious-clock approach because the bounds on the actual delays between the transitions can be expressed.

We propose a real-time extension of CTL, which we call *timed* CTL (TCTL), and define a semantics based on continuous computation trees. We model the delays in a system using *timed graphs*. The main result of the paper is a model-checking algorithm for determining the truth of a TCTL-formula with respect to a timed graph. Our results show that the choice of a dense domain to model time, instead of a discrete one, does not affect the complexity of the model-checking problem. On the other hand, we show that the questions such as satisfiability and finite-satisfiability (satisfiability with respect to some timed graph) are undecidable for TCTL. These results highlight the theoretical differences underlying the various models of real-time.

Outline : In section 2, we review the definition and the relevant results about the branching-time logic CTL. In section 3, we define the logic TCTL. We extend the syntax of CTL to allow subscripts of time constants on the temporal operators, limiting their scope in time. For instance, we write $\exists\Diamond_{<5}p$ to mean "there is a p-state along some computation path within 5 time units." To define the semantics of the logic, we generalize the notion of computation paths from ω-sequences of states to maps from R to states. We show that because of the denseness of the underlying domain, the satisfiability question for TCTL is Σ^1_1-hard.

In section 4, we propose the concept of a timed graph to model a finite-state real-time system. Timed graphs can express constant bounds on the delays between the system transitions. We imagine that the system is equipped with a finite set of *clocks* which record the time elapsed since they were reset. A constant bound on the delay between two transitions can be expressed by resetting a clock on the first transition, and associating with the second transition an enabling condition which compares the clock value with the bound. To interpret TCTL-formulas over a timed graph we can associate with it a continuous computation tree over the states of the system: a state gives the node in the timed graph, and an assignment of time values to all its clocks. We prove that the problem of deciding whether a given TCTL-formula is satisfiable with respect to some timed graph is unsolvable.

In section 5, we develop an algorithm for model-checking. The main idea behind the algorithm is to define an equivalence relation over the states of the system such that any two equivalent states are indistinguishable by TCTL-formulas. There are a finite number of equivalence classes under this relation, which enables us to obtain an ordinary Kripke structure from a timed graph.

The complexity of the model-checking algorithm is exponential in the number of clocks and the length of the timing constraints, but linear in the size of the state-transition graph and the length of the formula. We show the problem to be PSPACE-complete.

Finally, in section 6, we discuss the possible ways to extend our formalism.

Related work : There have been several temporal logics with quantitative time. ([JM86], [Os87], [Ko89], EMSS89], [AH89]). As stated earlier, these formalisms are based on either discrete-time or fictitious-clock models.

The syntax of our logic is similar to real-time logics studied by Koymans ([Ko89]) and Emerson et.al. ([EMSS89]). The undecidability result for TCTL is similar to the undecidability result for a linear-time real-time logic based on a dense time domain ([AH89]).

The automata defined by Dill ([Di89]) accept *timed traces*, sequences of events in which each element has an associated real-valued time of occurrence. However, that was a *linear-time*, not branching-time, model. Alur and Dill ([AD90]) have studied the properties of the class of the languages of timed traces definable using these automata. None of this work considered temporal logics.

Lewis' work is the most similar to ours of any. In [Le89] he proposed *state-diagrams* to represent the behavior of asynchronous circuits in a dense-time model. In ([Le90]), he defines a branching-time logic similar to TCTL: CTL formulas are extended by subscripting the temporal operators with time intervals, and the formulas are interpreted over the state-diagrams. There is an algorithm for model-checking. Compared to our model, his method has some restrictions (at least in theory), for instance, it requires positive lower bounds on all delays. Also he does not investigate the kind of

decidability and lower bound results presented here.

2 Computation Tree Logic

Computation tree logic (CTL) was introduced by Emerson and Clarke ([EC82]) as a specification language for finite-state systems. Let us briefly review its syntax and semantics.

Let P be a set of atomic propositions. The formulas of CTL are inductively defined as follows:

$$\phi := p \mid \mathbf{false} \mid \phi_1 \rightarrow \phi_2 \mid \exists \bigcirc \phi_1 \mid \exists \phi_1 \mathcal{U} \phi_2 \mid \forall \phi_1 \mathcal{U} \phi_2$$

where $p \in P$, and ϕ_1, ϕ_2 are CTL-formulas.

$\exists \bigcirc \phi$ intuitively means that there is an immediate successor state, reachable by executing one step, in which ϕ holds. $\exists \phi_1 \mathcal{U} \phi_2$ means that for some computation path, there exists an initial prefix of the path such that ϕ_2 holds at the last state of the prefix and ϕ_1 holds at all the intermediate states. $\forall \phi_1 \mathcal{U} \phi_2$ means that for every computation path the above property holds.

Some of the commonly used abbreviations are: $\exists\Diamond\phi$ for $\exists \mathbf{true}\, \mathcal{U}\, \phi$, $\forall\Diamond\phi$ for $\forall \mathbf{true}\, \mathcal{U}\, \phi$, $\exists\Box\phi$ for $\neg\forall\Diamond\neg\phi$, and $\forall\Box\phi$ for $\neg\exists\Diamond\neg\phi$.

Formally, the semantics of CTL is defined with respect to a structure $\mathcal{M} = (\mathcal{S}, \mu, \mathrm{E})$, where $\mathcal{S}$ is a countable set of states, $\mu : \mathcal{S} \rightarrow 2^P$ gives an assignment of truth values to propositions in each state, and E is a total relation over $\mathcal{S}$ giving the possible transitions. A *path* is an infinite sequence of states $(s_0, s_1 \ldots) \in \mathcal{S}^\omega$ such that for each $i \geq 0$, $\langle s_i, s_{i+1} \rangle \in \mathrm{E}$. Given a CTL-formula ϕ, and a state $s \in \mathcal{S}$, the satisfaction relation $(\mathcal{M}, s) \models \phi$ (meaning ϕ is true in $\mathcal{M}$ at s) is defined inductively. For example,

$(\mathcal{M}, s) \models \exists \phi_1 \mathcal{U} \phi_2$ iff
for some path $(s_0, s_1 \ldots)$ with $s_0 = s$,
$\exists i \geq 0. [s_i \models \phi_2 \ \wedge\ \forall j : 0 \leq j < i. s_j \models \phi_1]$.

A CTL-formula ϕ is called *satisfiable* iff there is a structure $\mathcal{M}$ and a state s of it such that $(\mathcal{M}, s) \models \phi$. CTL has the *finite-model property*: if a CTL-formula is satisfiable, then it is satisfiable in a structure with a finite set of states (in fact, in a structure of size exponential in the size of the formula). Consequently, the satisfiability question for CTL is decidable. It has been shown that it is EXPTIME-complete. However, the model-checking problem is in PTIME — given a CTL-formula ϕ, a structure $\mathcal{M} = (\mathcal{S}, \mu, \mathrm{E})$, and a state $s_0 \in \mathcal{S}$, there is an algorithm for deciding whether or not $(\mathcal{M}, s_0) \models \phi$ which runs in time $O[|\phi| \cdot (|\mathcal{S}| + |\mathrm{E}|)]$ ([CES86]).

Finite-state concurrent systems can be modeled as finite CTL-structures. Given a system modeled as a CTL-structure, and the specification as a CTL-formula, the model-checking algorithm can be used to decide whether or not the implementation satisfies the specification.

In verifying concurrent systems, we are generally interested only in correctness along the *fair* computation paths. For example, in a system with two processes, we may wish to consider only those computation sequences in which each process executes infinitely often. Various notions of fairness exist in literature ([LPS81]); the basic one being that of a state from a designated set appearing infinitely often along the computation path.

Since CTL cannot express correctness along fair paths, Clarke et.al. have defined CTL^F by changing the semantics of CTL ([CES86]). The syntax of CTL^F is same as that of CTL, however, a CTL^F-structure $\mathcal{M}$ has an additional component $F \subseteq 2^{\mathcal{S}}$. A path $(s_0, s_1, \ldots)$ is called *F-fair* if for each $\alpha \in F$, there are infinitely many i such that $s_i \in \alpha$. Given a CTL^F-formula ϕ, we change the meaning of $(\mathcal{M}, s) \models \phi$ so that the path-quantifiers range over only F-fair paths. The model-checking algorithm for CTL can be modified to handle this extension of CTL at a cost of multiplicative factor of the cardinality of F.

3 TCTL: Syntax and semantics

In CTL we can write a formula $\exists\Diamond p$, which says along some computation path, p eventually becomes true. CTL does not provide a way to put a bound on the time at which p will become true. An obvious extension is to put subscripts on the temporal operators to limit their scope in time ([Ko89], [EMSS89]). For example, we can write $\exists\Diamond_{<5} p$ to say that along some computation path p becomes true within 5 time units.

Let P be a set of propositions, and N be the set of constants $\{0, 1, 2, \ldots\}$ denoting the natural numbers [1].

Definition [Syntax]. The formulas ϕ of TCTL are inductively defined as follows:

$$\phi := p \mid \mathbf{false} \mid \phi_1 \rightarrow \phi_2 \mid \exists \phi_1 \mathcal{U}_{\sim c} \phi_2 \mid \forall \phi_1 \mathcal{U}_{\sim c} \phi_2$$

where $p \in P$, $c \in \mathrm{N}$, and $\sim$ stands for one of the binary relations $<$, $\leq$, $=$, $\geq$, or $>$. ■

Informally, $\exists \phi_1 \mathcal{U}_{<c} \phi_2$ means that for some computation path, there exists an initial prefix of time length less than c such that ϕ_2 holds at the last state of the prefix, and ϕ_1 holds at all its intermediate states. $\forall \phi_1 \mathcal{U}_{<c} \phi_2$ means that for every computation path, there is an initial prefix with the above property.

[1] Instead of N we can choose a set which has a constant symbol corresponding to each rational number. With trivial modifications the model-checking algorithm can still be used.

The other logical connectives can be defined as usual. Throughout the paper we will use $\sim$ to mean one of the binary relations $<, \leq, =, \geq$, or $>$. We define abbreviations $\exists\Diamond_{\sim c}\phi$ for $\exists$ **true** $\mathcal{U}_{\sim c}\phi$, $\forall\Diamond_{\sim c}\phi$ for $\forall$ **true** $\mathcal{U}_{\sim c}\phi$, $\exists\Box_{\sim c}\phi$ for $\neg\forall\Diamond_{\sim c}\neg\phi$, and $\forall\Box_{\sim c}\phi$ for $\neg\exists\Diamond_{\sim c}\neg\phi$. The unrestricted temporal operators correspond to TCTL operators subscripted with ≥ 0. For example, $\exists\Diamond\phi$ corresponds to $\exists\Diamond_{\geq 0}\phi$. In TCTL we can also define temporal operators subscripted with time intervals. For instance, $\exists\Diamond_{(a,b)}\phi$, which says that "$\phi$ holds at least once during the time interval (a, b) along some computation path," can be written as $\exists\Diamond_{=a}\exists\Diamond_{<(b-a)}\phi$. Note that TCTL has no *next-time* operator $\bigcirc$, because if time is dense then, by definition, there is no unique next time.

If we assume that along any particular computation path there is a unique state at every instant of time, then we can view the computation paths as maps from the time domain R to states of the system. Since we need to give meaning to notions such as "within time 5," we need the primitives of linear order $\prec$, addition $+$, and constants 0 and 1. To define the semantics of TCTL we choose the set of nonnegative reals, R, with the usual primitives to model time [2].

Let us first generalize the notion of a computation path from an ω-sequence of states to a map from R to states. For a set $\mathcal{S}$ and $s \in \mathcal{S}$, an *s-path* through $\mathcal{S}$ is a map ρ from R to $\mathcal{S}$ satisfying $\rho(0) = s$.

A structure for a branching-time logic should specify a set of labeled states and should associate a computation tree with each state. In discrete time, a structure can be described by associating a set of "next" states with each state. In dense time, there is always a third state between any pair of states on a path, so another characterization of trees must be found.

A dense tree can be considered to be a map from every state to a set of (dense) paths starting at that state. However, not every collection of paths describes a reasonable tree, so some additional properties are needed. To give the definition in detail, let us define some other concepts. Let $\mathcal{S}$ be any set, ρ be an s-path through $\mathcal{S}$, and t be any time. The *prefix* of ρ up to time t, denoted by ρ_t, is a map from $[0, t)$ to $\mathcal{S}$ obtained by restricting the domain of ρ. Furthermore, if ρ' is some map from $[0, t)$ to $\mathcal{S}$, then its concatenation with ρ, denoted by $\rho' \cdot \rho$, is defined by:

$$\text{for } t' \in \mathrm{R}\text{: } (\rho' \cdot \rho)(t') = \begin{cases} \rho'(t') & \text{if } t' < t \\ \rho(t' - t) & \text{otherwise} \end{cases}$$

If Π is a set of s-paths, then $\rho' \cdot \Pi$ is defined to be the set $\{\rho' \cdot \rho : \rho \in \Pi\}$.

[2] Instead of R we can choose the set of rational numbers to model time. Our results rely on the fact that the underlying domain is a dense linear order.

Now we can define the notion of a TCTL-structure.

Definition [TCTL-**structure**]. A structure for TCTL is a triple $\mathcal{M} = \langle \mathcal{S}, \mu, f \rangle$, where

- $\mathcal{S}$ is a set of states.
- $\mu : \mathcal{S} \to 2^P$ is a labeling function which assigns to each state the set of atomic propositions true in that state.
- f is a map giving for each $s \in \mathcal{S}$ a set of s-paths through S. f satisfies the following *tree constraint*:

$$\forall s \in \mathcal{S}.\, \forall \rho \in f(s).\, \forall t \in \mathrm{R}.\, \rho_t \cdot f[\rho(t)] \subseteq f(s)$$

■

Note that the definition of a TCTL-structure admits any variation of truth values of propositions along a computation path (for instance, one may define a path along which p is true at all rational times, and false at all irrational times). We could also add restrictions on paths to model the intuitive concept of *well-behavedness*, but that would not affect the undecidability result to be proved shortly.

We define below what it means for a TCTL-formula to be true in a state of a TCTL-structure.

Definition [**Satisfiability**]. For a TCTL-structure $\mathcal{M} = \langle \mathcal{S}, \mu, f \rangle$, a state $s \in \mathcal{S}$, and a TCTL-formula ϕ, the satisfaction relation $(\mathcal{M}, s) \models \phi$ is defined inductively as follows ($(\mathcal{M}, s) \models \phi$ is written as $s \models \phi$ as the structure is fixed):

- $s \models p$ iff $p \in \mu(s)$.
- $s \not\models$ **false**.
- $s \models (\phi_1 \to \phi_2)$ iff $s \not\models \phi_1$ or $s \models \phi_2$.
- $s \models \exists\, \phi_1\, \mathcal{U}_{\sim c}\phi_2$ iff for some $\rho \in f(s)$, for some $t \sim c$, $\rho(t) \models \phi_2$, and for all $0 \leq t' < t$, $\rho(t') \models \phi_1$.
- $s \models \forall\, \phi_1\, \mathcal{U}_{\sim c}\phi_2$ iff for every $\rho \in f(s)$, for some $t \sim c$, $\rho(t) \models \phi_2$, and for all $0 \leq t' < t$, $\rho(t') \models \phi_1$.

A TCTL-formula ϕ is called *satisfiable* iff there is a TCTL-structure $\mathcal{M}$ and a state s of $\mathcal{M}$, such that $(\mathcal{M}, s) \models \phi$. ■

It is interesting to note that TCTL is insensitive to *stuttering* (repetition of states with same labeling of propositions).

The denseness of the underlying time domain allows us to encode Turing machine computations in TCTL-formulas. Consequently, unlike other logics with similar syntax but discrete-time semantics, the satisfiability question for TCTL is undecidable — Σ_1^1-hard. The class Σ_1^1 consists of highly undecidable problems, including some non-arithmetical sets. The next theorem

is proved using 2-counter machines instead of Turing machines; the proof is similar to the proof of undecidability of a linear-time dense real-time logic ([AH89]).

Theorem [Undecidability of TCTL]. The satisfiability question for TCTL is Σ_1^1-hard. ■

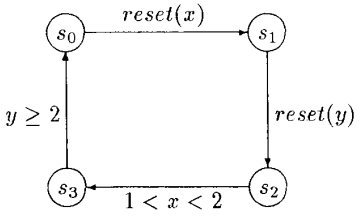
Proof. We reduce the problem of deciding whether a nondeterministic 2-counter machine has a computation in which the starting state is visited infinitely often to TCTL-satisfiability.

Let M be a nondeterministic 2-counter machine with counters α and β, and a program with n instructions. We use propositions $p_\alpha, p_\beta, p_1, \ldots p_n$. We say that a path ρ encodes a computation of M, if the following holds: the number of points in the interval $[j, j+1)$ at which p_α (p_β) is true along ρ equals the value of counter α (β) at j-th step, and if i_j is the instruction to be executed at j-th step then over the interval $[j, j+1)$, of the propositions $\{p_1, \ldots p_n\}$, only p_{i_j} is true at $\rho(j)$. The denseness of the time domain allows the counter values to get arbitrarily large. It is possible to write a TCTL-formula ϕ such that whenever $(\mathcal{M}, s) \models \phi$, every s-path in $\mathcal{M}$ encodes, according to the above scheme, a computation of M in which the start state repeats infinitely often. ■

4 Timed graphs

In this section we introduce *timed graphs* to model finite-state real-time systems. Each system has a finite set of *nodes* and a finite set of (real-valued) *clocks*. A clock can be reset simultaneously with any transition of the system. At any instant, the reading of a clock equals the time elapsed since the last time it was reset. The timing constraints are expressed by associating enabling conditions with transitions.

For example, the following graph represents a system with four nodes and two clocks x, and y.



The clock x gets initialized on the transition from s_0 to s_1. At any instant, the value of x equals the time elapsed since the last time this transition was taken. The enabling condition associated with the s_2 to s_3 transition expresses the following timing constraint: the transition from s_2 to s_3 always happens more than one second but less than two seconds after the transition from s_0 to s_1. Similarly, the clock y constrains the transition from s_3 to s_0 to occur at least two seconds later than the s_1 to s_2 transition.

Thus to express a bound on the delay between two transitions, we reset a clock with the first transition, and associate an enabling condition with the other transition. Note that clocks can be set asynchronously of each other. Also, arbitrarily many events can occur in a finite interval of time. There are only finitely many clocks because the future behavior of a process depends on only a finite number of delays.

Definition [Timed graph]. A timed graph is a tuple $\mathrm{G} = \langle \mathrm{S}, \mu, s_0, \mathrm{E}, \mathrm{C}, \pi, \tau \rangle$, where

- S is a finite set of nodes.
- $\mu : \mathrm{S} \to 2^P$ is a labeling function assigning to each node the set of atomic propositions true in that node.
- $s_0 \in \mathrm{S}$ is the start node.
- $\mathrm{E} \subseteq \mathrm{S} \times \mathrm{S}$ is a set of edges.
- C is a finite set of clocks.
- $\pi : \mathrm{E} \to 2^{\mathrm{C}}$ tells which clocks should be reset with each edge.
- τ is a function labeling each edge with an enabling condition built using the boolean connectives over the atomic formulas of the form $x \leq c$ or $c \leq x$, where $x \in \mathrm{C}$, and $c \in \mathrm{N}$ [3].

■

The system starts at node s_0 with all its clocks initialized to 0. The values of all the clocks increase uniformly with time. At any point in time, the system can make a transition, if the associated condition is enabled by the current values of the clocks. The transitions are instantaneous. With each transition e, the clocks in $\pi(e)$ get reset to 0 and start counting time with respect to that transition. At any instant, the state of the system can fully be described by specifying the current node and the values of all its clocks. Any behavior of the system gives a map from R to such states.

Let $\Gamma(\mathrm{G})$ denote $[\mathrm{C} \to \mathrm{R}]$, the set of *time assignments* for the clocks of G. A state of the system is of the form $\langle s, \nu \rangle$, where $s \in \mathrm{S}$, and $\nu \in \Gamma(\mathrm{G})$.

Some notation: Let $\nu \in \Gamma(\mathrm{G})$ and $t \in \mathrm{R}$. $\nu + t$ denotes the time assignment for C, which assigns to

[3] We could generalize the enabling conditions to allow all quantifier-free formulas built using the primitives of linear order, constants, and addition by constants, but allowing addition of variables makes model-checking undecidable.

each y the value $\nu(y) + t$. $[x \mapsto t]\nu$ denotes the time assignment for $C \cup \{x\}$ which assigns t to x and agrees with ν on the values of the rest of the clocks.

An $\langle s, \nu \rangle$-run records the states at time instants at which the transitions occur along a possible computation of the system started in $\langle s, \nu \rangle$.

Definition [$\langle s, \nu \rangle$-run]. Given a state $\langle s, \nu \rangle$ of G, an $\langle s, \nu \rangle$-run of G is an infinite sequence of triples $(\langle s, \nu, 0 \rangle = \langle s_1, \nu_1, t_1 \rangle, \langle s_2, \nu_2, t_2 \rangle, \ldots) \in (S \times \Gamma(G) \times R)^\omega$ satisfying the following constraints for each $i \geq 1$:

- Time of $(i+1)$-th transition is strictly greater than the time of i-th transition: $t_{i+1} > t_i$
- $e_i = \langle s_i, s_{i+1} \rangle$ is an edge in E.
- The time assignment ν_{i+1} at time t_{i+1} equals $[\pi(e_i) \mapsto 0](\nu_i + t_{i+1} - t_i)$.
- The time assignment $(\nu_i + t_{i+1} - t_i)$ satisfies the enabling condition $\tau(e_i)$.
- Every time value is eventually reached, that is, for any $t \in R$, there exists some j such that $t_j \geq t$.

■

An $\langle s, \nu \rangle$-run gives a path ρ as a map from R to the states of the system as follows: for $t \in R$ such that $t_j \leq t < t_{j+1}$, $\rho(t) = \langle s_j, \nu_j + t - t_j \rangle$.

The last restriction in the above definition is referred to as the *progress* condition on time. It rules out the runs in which an infinite number of transitions occur in a bounded interval of time.

The paths generated by a timed graph have certain noteworthy properties. Firstly, the state $\langle s_i, (\nu_i + t_{i+1} - t_i) \rangle$ is the left limit of ρ as time approaches t_{i+1}. At the transition point there is a possible discontinuity because of resetting of some of the clocks. The time assignment at time t_{i+1} shows the values of the clocks in $\pi(e_i)$ as 0.

Secondly, if we view ρ as a map giving the truth assignment to propositions at every time instant, then ρ is right-continuous. The values of all the propositions remain the same over the time interval $[t_i, t_{i+1})$. Furthermore, the value of any proposition changes at most ω times along ρ.

Now that we have defined the computation paths generated by a timed graph, we can interpret TCTL-formulas over it.

Definition [Finite Satisfiability]. Given a timed graph $G = \langle S, \mu, s_0, E, C, \pi, \tau \rangle$ the corresponding TCTL-structure is $\mathcal{M}_G = \langle S \times \Gamma(G), \mu', f \rangle$, where the labeling function is defined by $\mu'(\langle s, \nu \rangle) = \mu(s)$, and $f(\langle s, \nu \rangle)$ consists of precisely the paths corresponding to the $\langle s, \nu \rangle$-runs of G.

For a TCTL-formula ϕ, $G \models \phi$ iff $(\mathcal{M}_G, \langle s_0, \nu_0 \rangle) \models \phi$, where ν_0 is defined by $\nu_0(x) = 0$, for each $x \in C$.

A TCTL-formula ϕ is called *finitely-satisfiable* if and only if there exists a timed graph G such that $G \models \phi$.

■

It turns out that the finite satisfiability question is also undecidable. Consequently, automatic synthesis of a synchronization skeleton meeting the constraints specified by a TCTL-formula is not possible.

Theorem [Undecidability of finite satisfiability]. The set of finitely-satisfiable TCTL-formulas is not recursive. ■

Proof. We reduce the halting problem for 2-counter machines to the finite satisfiability question. As in the proof of the undecidability of TCTL-satisfiability, given a deterministic 2-counter machine M, we construct a TCTL-formula ϕ such that the paths in the models of ϕ encode the halting computation of M. If M does not halt, then ϕ is not satisfiable. If M halts, we can construct a timed graph with just one run, which corresponds to the encoding of the finite computation of M. ■

5 Model-checking

In this section we will develop an algorithm for deciding whether a finite-state real-time system presented as a timed graph meets its specification given as a TCTL-formula. We will also study the complexity of the model-checking problem.

Suppose the system is described by a timed graph $G = \langle S, \mu, s_0, E, C, \pi, \tau \rangle$. The system has infinitely many states, and the TCTL-structure corresponding to it is infinite. However, not all of these states are distinguishable by our logic. If two states corresponding to the same node agree on the integral parts of all the clock values, and also on the ordering of the fractional parts of all the clocks, then the computation trees rooted at these two states cannot be distinguished by TCTL-formulas. The integral parts of the clocks in a state are needed to determine whether or not a particular enabling condition is met, whereas the ordering of the fractional parts is needed to decide which clock will change its integral part first. For example, if two clocks x and y are between 0 and 1 in a state, then a transition with enabling condition $x = 1$ can be followed by a transition with enabling condition $y = 1$, depending on whether or not the state satisfies $x < y$.

The integral readings of the clocks can get arbitrarily large. But if a clock x is never compared with a constant greater than c, then its actual value, once it exceeds c, is of no consequence in deciding the allowed paths.

Now we formalize this notion and prove our claim. For each $x \in C$, let c_x be the largest constant that x is compared with in any enabling condition. For any $t \in R$, $fract(t)$ denotes the fractional part of t, and $\lfloor t \rfloor$ denotes the integral part of t.

Definition [Equivalence of time assignments]. Given $\nu, \nu' \in \Gamma(G)$, $\nu \cong \nu'$ iff the following conditions are met:

- For each $x \in C$, either $\lfloor \nu(x) \rfloor$ and $\lfloor \nu'(x) \rfloor$ are the same, or both $\nu(x)$ and $\nu'(x)$ are greater than c_x.
- For every $x, y \in C$ such that $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $fract(\nu(x)) \leq fract(\nu(y))$ iff $fract(\nu'(x)) \leq fract(\nu'(y))$.

■

$[\nu]$ denotes the equivalence class of $\Gamma(G)$ to which ν belongs.

Lemma. Let $s \in S$, and $\nu, \nu' \in \Gamma(G)$ with $\nu \cong \nu'$. For every TCTL-formula ϕ, $(\mathcal{M}_G, \langle s, \nu \rangle) \models \phi$ iff $(\mathcal{M}_G, \langle s, \nu' \rangle) \models \phi$. ■

Proof. Before we prove the lemma we need to show that the runs starting at $\langle s, \nu \rangle$ and $\langle s, \nu' \rangle$ are similar in the following sense: Let r be any $\langle s, \nu \rangle$-run $\{\langle s_i, \nu_i, t_i \rangle : i \geq 1\}$. We will say that the i-th element $\langle s_i, \nu_i, t_i \rangle$ is "similar" to another element $\langle s_i, \nu'_i, t'_i \rangle$, if the following conditions hold: $\nu_i \cong \nu'_i$, $\lfloor t_i \rfloor = \lfloor t'_i \rfloor$, and for each $x \in C$, $fract(\nu_i(x)) \sim fract(t_i)$ iff $fract(\nu'_i(x)) \sim fract(t'_i)$. The claim is that we can find an $\langle s, \nu' \rangle$-run $r' : \{\langle s_i, \nu'_i, t'_i \rangle : i \geq 1\}$ such that for each $i \geq 1$, i-th elements of r and r' are similar.

Given r we construct the required run r' step by step. The base case follows from $\nu \cong \nu'$. Suppose we have constructed r' upto i steps. Let $\delta = t_{i+1} - t_i$, the delay between the $(i+1)$-th and i-th transition of r. We try to find δ' such that it is possible to extend r' by s_i to s_{i+1} transition at time $t'_i + \delta'$ such that $(i+1)$-th elements of the two runs are similar.

Because of the similarity of the i-th elements, the necessary and sufficient requirement of the desired δ' is that $\nu'_i(y)$, for $y \in C$, and t'_i should cross the same number of integer boundaries due to the addition of δ', as $\nu_i(y)$ and t_i cross due to the addition of δ.

For example, let $C = \{x, y\}$. Let $\nu_i = [0.3, 1.7]$, $t_i = t'_i = 2$, $\delta = 1.4$, and $\nu'_i = [0.8, 1.9]$. In this case δ' should satisfy: $\delta' \in (0.2, 1.2)$, $\delta' \in (1.1, 2.1)$, and $\delta' \in (1, 2)$. So choosing $\delta' \in (1.1, 1.2)$ serves the purpose.

The reader can convince himself that the existence of δ' meeting the constraints depends only on the ordering of the fractional parts of t'_i, and the clocks in ν'_i. Since the i-th elements of r and r' agree on this ordering of the fractional parts, the existence of δ guarantees the existence of δ'.

The proof of the lemma uses an induction argument over the structure of the formulas. We can show that if a path of an $\langle s, \nu \rangle$-run r satisfies the path formula $\phi_1 \mathcal{U}_{\sim c} \phi_2$, then the path of the $\langle s, \nu' \rangle$-run corresponding to r, constructed as above, also satisfies the same formula. ■

A *region* is a pair $\langle s, [\nu] \rangle$, where $s \in S$, and $\nu \in \Gamma(G)$.

Let ϕ be any TCTL-formula. We want to label the regions with all the subformulas of ϕ such that $\langle s, [\nu] \rangle$ is labeled with ψ iff $(\mathcal{M}_G, \langle s, \nu \rangle) \models \psi$.

Suppose we want to determine whether $\langle s, [\nu] \rangle$ should be labeled with $\exists\, \phi_1 \mathcal{U}_{<c} \phi_2$. We try to find a run starting at $\langle s, \nu \rangle$ such that the associated path satisfies $\phi_1 \mathcal{U}_{<c} \phi_2$. As time progresses, the state of the system changes, but the truth of the subformulas ϕ_1 and ϕ_2 can change only when the state moves to a new region. Hence, instead of the desired run we search for a finite sequence of regions starting at $\langle s, [\nu] \rangle$ such that each region can be reached from the previous one by a state-transition or increase in the values of clocks. Furthermore, for the desired sequence, ϕ_2 should be true over its last region, ϕ_1 should be true over all the intermediate regions, and the total time elapsed in traversing it must be less than c.

We could define an edge relation over the regions, and label each region with the subformulas of ϕ using the above ideas. However, the complexity of the algorithm can be improved using the following trick: To keep track of the time elapsed in traversing a sequence of regions, we introduce an extra clock $x \notin C$. Let $C^* = C \cup \{x\}$. Let $c(\phi)$ denote the largest constant appearing in ϕ. We define an equivalence relation $\cong^*$ on $[C^* \to R]$ similar to the relation $\cong$ on $\Gamma(G)$ with $c_x = c(\phi)$. Furthermore, for $\nu \in [C^* \to R]$, let $[\nu]^*$ denote its equivalence class with respect to $\cong^*$.

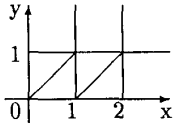
Let us call an element of the form $\langle s, [\nu]^* \rangle$, where $s \in S$, and $\nu \in [C^* \to R]$, an *augmented region*. Note that each region is refined into many augmented regions. Now to find out whether there exists a path from a region v_1 to a region v_2 that can be traversed within time c, we try to find a path from the region v_1 augmented with $x = 0$ to an augmented region which is a refinement of v_2 and satisfies $x < c$. The new clock x is never reset, and is updated consistently with the other clocks. The value of x is of no importance once it crosses $c(\phi)$, and hence while defining $\cong^*$ over $[C^* \to R]$ we take $c_x = c(\phi)$.

The edge relation between augmented regions captures two different types of events: (1) transitions in G and (2) moving into a new equivalence class of states because of the passage of time. We define a *successor* function over equivalence classes of time assignments to capture the second type of transitions.

Let α, β be distinct equivalence classes of $[C^* \to R]$.

$succ(\alpha) = \beta$ iff for each $\nu \in \alpha$, there exists a positive $t \in \mathrm{R}$ such that $\nu+t \in \beta$, and for all $t' < t$, $\nu+t' \in \alpha \cup \beta$.

Let us consider an example with two clocks x and y with $c_x = 2$ and $c_y = 1$. The equivalence classes are shown in the following figure: Corner points (e.g. $(1,1)$), open line segments (e.g. $\{(x,y) : (0 < x < 1) \wedge (x = y)\}$, $\{(x,1) : x > 2\}$), and open regions (eg. $\{(x,y) : 0 < x < y < 1\}$, $\{(x,y) : (1 < x < 2) \wedge (y > 1)\}$) represent equivalence classes.



The successor of any class α is the class to be hit first by a line drawn from some point in α in the diagonally upwards direction. For example, the successor of the class $\{(1,0)\}$ is the class $\{(x,y) : (1 < x < 2) \wedge (y = x - 1)\}$. The successor of $\{(x,y) : 0 < y < x < 1\}$ is the class $\{(1,y) : 0 < y < 1\}$.

Let us call an equivalence class α a *boundary class*, if for each $\nu \in \alpha$ and for any positive t, ν and $\nu+t$ are not equivalent. In the above example, the classes which lie on either horizontal or vertical lines are boundary classes.

Now we can construct the region graph as follows:

Definition [Region graph]. The region graph $\mathrm{R}(\mathrm{G}, \phi)$ is defined to be the graph $\langle V_1, E_1 \rangle$. The vertex set V_1 is the set of all augmented regions. The edge set E_1 consists of two types of edges:

- Edges representing the passage of time: Each vertex $\langle s, \alpha \rangle$ has an edge to $\langle s, succ(\alpha) \rangle$.
- Edges representing the transitions of G: Each vertex $\langle s, \alpha \rangle$, for each edge $e = \langle s, s' \rangle \in \mathrm{E}$, has an edge to $\langle s', [[\pi(e) \mapsto 0]\nu]^* \rangle$, provided ν satisfies the enabling condition $\tau(e)$, and either $succ(\alpha) = [\nu]^*$, or $\nu' \in \alpha$ and α is not a boundary class.

■

There is a simple correspondence between the runs of G and infinite paths through $\mathrm{R}(\mathrm{G}, \phi)$. Let $r : \{\langle s_i, \nu_i, t_i \rangle : i \geq 1\}$ be any run of G with the corresponding path ρ in $\mathcal{M}_\mathrm{G}$. For each $i \geq 1$, we can find a finite path β_i in $\mathrm{R}(\mathrm{G}, \phi)$ joining $\langle s_i, [[x \mapsto t_i]\nu_i]^* \rangle$ to $\langle s_{i+1}, [[x \mapsto t_{i+1}]\nu_{i+1}]^* \rangle$ by taking successor classes of $[[x \mapsto t_i]\nu_i]^*$ as time increases from t_i to t_{i+1}. For any $t : 0 \leq t < (t_{i+1} - t_i)$, the state on ρ at time $t_i + t$ is $\langle s_i, \nu_i + t \rangle$, and $\langle s_i, [[x \mapsto t](\nu_i + t)]^* \rangle$ appears on β_i. On the other hand, if $\langle s_i, \alpha \rangle$ appears on β_i, then there is some t such that $0 \leq t < (t_{i+1} - t_i)$ and $[x \mapsto t_i + t](\nu_i + t) \in \alpha$. Thus the path β obtained by concatenating the segments β_i corresponds to the run r in a natural way. Similarly, given an infinite path through $\mathrm{R}(\mathrm{G}, \phi)$ we can construct a corresponding run of G.

Since time progresses without bound along r, every clock $y \in \mathrm{C}^*$ is either reset infinitely often or eventually it always increases. Hence, for each $y \in \mathrm{C}^*$, along β, infinitely many augmented regions satisfy either $y = 0$, or $y > c_y$.

Let us denote the set $\{\langle s, [\nu]^* \rangle : s \in \mathrm{S} \wedge (\nu(y) = 0 \vee \nu(y) > c_y)\}$ by F_y, for $y \in \mathrm{C}^*$. If we treat the region graph as a structure for CTL^F with the fairness family given by $F_p = \{F_y : y \in \mathrm{C}^*\}$, then along any F_p-*fair* path, for any $y \in \mathrm{C}^*$, either y gets reset infinitely often, or eventually its value exceeds c_y, and never decreases. Conversely, for any F_p-*fair* path in the region graph, we can find a corresponding run along which time increases without bound.

Labeling Algorithm : We label the regions with subformulas of ϕ starting from the subformulas of length 1, then of length 2, and so on. Though we are labeling the regions, and not the augmented regions, in the following description assume that the labels of the regions carry over to the augmented regions in the following way: the augmented region $\langle s, [\nu]^* \rangle$ gets the labels of the region $\langle s, [\nu|_\mathrm{C}] \rangle$.

We also label the augmented regions with special propositions as follows: Let $p_{\sim c}$ be a new proposition corresponding to every subscript $\sim c$ appearing in ϕ. Label a vertex $\langle s, [\nu]^* \rangle$ with $p_{\sim c}$ if $\nu \models x \sim c$, else label it with $\neg p_{\sim c}$. Furthermore, let p_b be a new proposition which is true at a vertex $\langle s, \alpha \rangle$ if α is a boundary class.

Let ψ be a subformula of ϕ. Assume that the regions are already labeled with each subformula of ψ. Let $v = \langle s, [\nu] \rangle$ be any region.

Case $\psi \in P$: If $\psi \in \mu(s)$ then label v with ψ else with $\neg\psi$.

Case $\psi =$ **false** : Label v with $\neg\psi$.

Case $\psi = \phi_1 \rightarrow \phi_2$: If v is labeled with $\neg\phi_1$ or with ϕ_2 then label it with ψ else with $\neg\psi$.

Case $\psi = Q\,\phi_1\,\mathcal{U}_{\sim c}\phi_2$, where Q is either an existential or universal quantifier: The vertices of $\mathrm{R}(\mathrm{G}, \phi)$ are labeled with ϕ_1 or its negation, and ϕ_2 or its negation. v should be labeled with ψ, if some (or, every, depending upon Q) F_p-*fair* path through $\mathrm{R}(\mathrm{G}, \phi)$ starting at $\langle s, [[x \mapsto 0]\nu]^* \rangle$, has a prefix $v_1, v_2, \ldots v_n = \langle s_n, [\nu_n]^* \rangle$, such that each v_i, $1 \leq i < n$, is labeled with ϕ_1, v_n is labeled with ϕ_2, and $\nu_n \models x \sim c$. Furthermore, if $[\nu_n]^*$ is not a boundary region, then it should also be labeled with ϕ_1.

This condition can be tested using conventional model-checking for CTL^F ([CES86]). Let $\psi' = Q\,\phi_1\,\mathcal{U}\,(\phi_2 \wedge p_{\sim c} \wedge (p_b \vee \phi_1))$. Label the vertices of $R(G,\phi)$ with ψ' or its negation by checking it over F_p-*fair* paths in $R(G,\phi)$.

If $\langle s, [[x \mapsto 0]\nu]^* \rangle$ is labeled with ψ' then label v with ψ, else with $\neg\psi$. ■

The following lemma states the correctness of the above labeling procedure.

Lemma [Correctness of the labeling algorithm]. Let ψ be a subformula or the negation of a subformula of ϕ. If the above labeling algorithm labels $\langle s, [\nu] \rangle$ with ψ then $(\mathcal{M}_G, \langle s, \nu \rangle) \models \psi$. ■

This suggests a decision procedure for model-checking: Given a timed graph G and a TCTL-formula ϕ, first construct the region graph $R(G,\phi)$. Then label all the regions with the subformulas of ϕ using the labeling procedure. By the above lemma, $G \models \phi$ iff $\langle s_0, \nu_0 \rangle$ is labeled with ϕ.

Complexity of the Algorithm : Using the ideas discussed above, one can implement an algorithm for model-checking which runs in time linear in the qualitative part, and exponential in the timing part of the input. Before we can analyze the complexity of the algorithm, we prove the following lemma:

Lemma [Number of equivalence classes]. The number of equivalence classes of $[C^* \to R]$ induced by $\cong^*$ is bounded by $|C^*|! \cdot 2^{|C^*|} \cdot \Pi_{y \in C^*}(2 \cdot c_y + 2)$. ■

Proof. An equivalence class $[\nu]^*$ of $[C^* \to R]$ can be described by a pair of arrays $\langle \alpha, \beta \rangle$ as follows: For each $y \in C^*$, the array α tells to which of the intervals $\{[0,0], (0,1), [1,1], \ldots (c_y - 1, c_y), [c_y, c_y], (c_y, \infty)\}$ the value $\nu(y)$ belongs. β encodes the ordering of the fractional parts of the clocks in C^*. One component of β gives a permutation Π of C^* such that $\Pi(x_1) < \Pi(x_2)$ iff $\nu(x_1) \leq \nu(x_2)$. The other component of β tells, for each $y \in C^*$, whether or not the fractional part of $\nu(y)$ equals the fractional part of its Π-predecessor.

The number of ways to choose α is $\Pi_{y \in C^*}(2 \cdot c_y + 2)$. The first component of β can be chosen in $|C^*|!$ ways; whereas the number of ways to choose its second component is bounded by $2^{|C^*|}$. The lemma follows. ■

Now from the definition of the region graph, it follows that $|V_1| = O[c(\phi) \cdot |S| \cdot |C|! \cdot \Pi_{y \in C} c_y]$. The first clause in the definition of E_1 contributes at most one edge for every vertex, and the second clause contributes at most two edges for any edge in E and an equivalence class of $[C^* \to R]$. Hence, $|E_1| = O[c(\phi) \cdot (|S| + |E|) \cdot |C|! \cdot \Pi_{y \in C} c_y]$.

Thus the size of the region graph is exponential in the length of timing constraints of the given timed graph, but linear in the size of the node-transition graph.

Theorem [Model-checking for TCTL]. Given a timed graph $G = \langle S, \mu, s_0, E, C, \pi, \tau \rangle$ and a TCTL-formula ϕ, there is a decision procedure for checking whether or not $G \models \phi$ which runs in time $O[c(\phi) \cdot |\phi| \cdot (|S| + |E|) \cdot |C|! \cdot \Pi_{y \in C} c_y]$. ■

Proof. First construct the region graph $R(G,\phi) = \langle V_1, E_1 \rangle$. Using the above representation the successor class of any class can be computed in time $O[|C^*|]$. Hence, $R(G,\phi)$ can be constructed in time $O[|V_1| + |E_1|]$. Then run the labeling algorithm on the subformulas of ϕ. The number of fairness constraints is $|C^*|$. The vertices of $R(G,\phi)$ can be marked with a formula ψ in time $O[(|V_1| + |E_1|) \cdot |C^*|]$, assuming they are already marked with the subformulas of ψ, using the labeling algorithm for CTL^F ([CES86]). So the labeling algorithm takes time $O[|\phi| \cdot |C^*| \cdot (|V_1| + |E_1|)]$. The complexity follows from the bounds on $|V_1|$ and $|E_1|$. ■

Since we have shown that the model-checking problem is decidable, we can also characterize the complexity class of deciding finite satisfiability of a TCTL-formula.

Corollory [Complexity of finite satisfiability]. The problem of deciding whether a given TCTL-formula is finitely-satisfiable is complete for the class of recursively enumerable problems (Σ_1-complete). ■

Proof. The set of timed graphs is enumerable. For any given timed graph G one can find whether or not G models the given TCTL-formula. Consequently, the set of TCTL-formulas for which there exists a timed graph model, is recursively enumerable. Σ_1-hardness was proved earlier. ■

Note that since the set of satisfiable TCTL-formulas is not recursively enumerable, there must be some formulas which are satisfiable but not finitely-satisfiable. Thus TCTL does not have the finite model property.

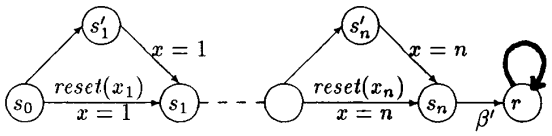
Complexity of Model-checking : The model-checking algorithm we considered, requires time exponential in the length of the timing constraints. Now we establish a lower bound for the problem, and show the problem to be PSPACE-complete.

Lemma [Model-checking is PSPACE-hard]. Given a timed graph G and a TCTL-formula ϕ, the problem of deciding whether or not $G \models \phi$ is PSPACE-hard. ■

Proof. The problem of deciding the truth of a quantified boolean formula (QBF) is known to be PSPACE-hard ([HU79]). We will reduce this problem to TCTL model-checking.

Let $\alpha = Q_1 p_1 . Q_2 p_2 \ldots Q_n p_n . \beta(p_1, \ldots p_n)$ be a QBF, where β is a formula of propositional logic over the propositions $p_1, \ldots p_n$, and each Q_i is either a universal or existential quantifier.

We construct a timed graph G as shown below:



The set of clocks C is $\{x, x_1, \ldots x_n\}$. s_0 is the start node. β' is obtained from β by replacing each p_i with the atomic formula $x_i = n + 1 - i$. There is just one proposition p, and it is true only in the state r.

For every $\langle s_0, \nu_0 \rangle$-run, the node s_i is reached at time i. Furthermore, $\langle s_n, \nu, n \rangle$ appears on the run, where $\nu(x) = n$, and for $1 \leq i \leq n$, $\nu(x_i)$ equals n or $n - i$ depending on which of the two paths between s_{i-1} and s_i was taken. There are 2^n different paths from s_0 to s_n, and each can be viewed as encoding a different truth assignment for the propositions. It is easy to see that $\langle r, \nu + 1, n + 1 \rangle$ appears on this path if β is true with respect to the truth assignment I defined as: if $\nu(x_i) = n - i$ then $I(x_i) = true$ else $I(x_i) = false$. Let ϕ be the TCTL-formula, $Q_1 \Diamond_{=1} . Q_2 \Diamond_{=1} \ldots . Q_n \Diamond_{=1} . \exists \Diamond_{=1} p$.

Now, it is a straightforward exercise to show that $G \models \phi$ if and only if α is true. ■

Lemma [Model-checking is in PSPACE]. Given a timed graph G and a TCTL-formula ϕ, the problem of deciding whether or not $G \models \phi$ can be solved using space polynomial in the length of the input. ■

Proof. The region graph has size exponential in the length l of the input, and hence if we construct it fully, and label its vertices with the subformulas of ϕ, the algorithm will need space exponential in l. We sketch out another version of the labeling algorithm which saves on the work-space, by computing the labels of the vertices as they are required.

The main procedure of the algorithm is $label(v, \psi)$, which returns *true* if v should be labeled with ψ else returns *false*. Let n be the maximum depth of the nesting of the path-quantifiers in ψ. We claim that a non-deterministic version of *label* can be implemented so as to use space $O[l \cdot n]$. This can be proved by an induction on the structure of ψ.

The cases $\psi \in P$, $\psi =$ **false**, and $\psi = \psi_1 \rightarrow \psi_2$ are straightforward.

For $\psi = \exists \phi_1 \mathcal{U}_{\sim c} \phi_2$, the procedure nondeterministically guesses a path $v \rightarrow v_1 \rightarrow \cdots v_m$. The path is guessed vertex by vertex, at each step checking that the newly guessed vertex is connected by an edge from the previous vertex. Furthermore, some F_p-*fair* path should be accessible from v_m. The procedure checks that $label(v_i, \phi_1)$ returns *true* for $i < m$, and also $label(v_m, \phi_2 \wedge p_{\sim c} \wedge (p_b \vee \phi_1))$ returns *true* (p_b, and $p_{\sim c}$ are as defined in the labeling algorithm, and for them *label* can be implemented in constant space). It just needs to remember the current guess and the previous guess. This does involve recursive calls to *label*, so the total space required is $O[l]$ plus the space for *label* when called with ϕ_1 or ϕ_2 as the second argument. The claim follows by the inductive hypothesis.

Now consider the case $\psi = \forall \phi_1 \mathcal{U}_{\sim c} \phi_2$. The negation of ψ can be written using existential path-quantifiers, and then we can proceed as in the previous case. For instance, the following equivalence holds:

$$\forall \phi_1 \mathcal{U}_{<c} \phi_2 \equiv \neg[\exists \neg\phi_2 \mathcal{U}_{<c} (\neg\phi_1 \wedge \neg\phi_2) \vee \exists \Box_{<c} \neg\phi_2]$$

label is called recursively on each subformula of the above translation. The case $\psi = \exists \Box_{\sim c} \phi'$ is handled similar to the previous case.

By Savitch's theorem the deterministic version can be implemented in space $O[l^2 \cdot n^2]$. So model-checking can be done in space $O[|\phi|^2 \cdot l^2]$. ■

The following theorem follows immediately from the above lemmas:

Theorem [Complexity of Model-checking]. Given a timed graph G and a TCTL-formula ϕ, the problem of deciding whether or not $G \models \phi$ is PSPACE-complete. ■

6 Discussion

In this section we discuss the complexity of our method in comparison with others, possible extensions, and the possible directions for further investigation.

Complexity of Reasoning about Real-time: The complexity of the model-checking algorithm for CTL is linear in the product of the size of the formula and the size of the structure. In TCTL, adding times in the formulas contributes another multiplicative factor equal to the largest constant in the formula, while adding clocks to the structure contributes a factor equal to the product of the constants in the timing constraints and the factorial of the number of clocks.

What is the cost of the dense-time model over discrete time? We claim that it is small. Adding times to the formulas seems to cause an identical and unavoidable blowup in both cases. If we keep the syntax of TCTL-formulas and the timed

graphs unchanged, but change the semantics to discrete time, the problem is still PSPACE-complete, but model-checking can be speeded-up to run in time $O[c(\phi) \cdot |\phi| \cdot (|S| + |E|) \cdot \Pi_{x \in C} c_x]$. Thus, the only extra cost of choosing the domain R is the multiplicative factor of |C|!. The problem is still PSPACE-complete. Other procedures for reasoning about discrete-time have shown similar blowups ([EMSS89], [AH89]).

Introducing Fairness into TCTL: To handle *fairness* CTL was extended to CTL^F. A similar extension is possible for TCTL also. We call the resulting logic TCTL^F. The syntax is same as that of TCTL; but a timed graph is augmented with a set of fairness constraints $F \subseteq 2^S$. An $\langle s, \nu \rangle$-run $\{\langle s_i, \nu_i, t_i \rangle : i \geq 0\}$ of G is said to be F-fair iff for each $\alpha \in F$, there are infinitely many i such that $s_i \in \alpha$. Given a TCTL^F-formula ϕ, we change the meaning of $G \models \phi$ so that the path-quantifiers range over only those paths that correspond to F-fair runs of G.

For model-checking, we construct the region graph $R(G, \phi)$ as before. The fairness family F' for $R(G, \phi)$ is obtained by replacing each α in F by the set $\{\langle s, [\nu]^* \rangle : s \in \alpha\}$. The labeling algorithm is as before, except while checking for the path-formulas, we restrict attention to F'-fair paths in $R(G, \phi)$. This fairness requirement can be handled as in CTL^F. The complexity of the algorithm increases by a multiplicative factor of the cardinality of F.

Introducing Temporal Quantifiers into TCTL: Other ways have been proposed for extending the temporal logic notation to write timing properties. The language TPTL ([AH89]) uses temporal quantifiers of the form $\Diamond x$, meaning "eventually in a state with time x". For example, the response property, "every p-state is followed by a q-state within 5 time units," is written as $\Box x.(p \rightarrow \Diamond y.(q \wedge y < x + 5))$. The syntax allows quantifier-free formulas involving the time variables, and the primitives of addition by constants and comparisons. We can extend the syntax of TCTL in an analogous manner.

In discrete time, this extension is no more expressive than our current notation (though it can express certain properties more succinctly). However, in dense time, it seems to be strictly more expressive. For example, consider the formula:

$$\exists \Diamond x. [p \wedge \exists \Diamond (q \wedge \exists \Diamond z. (r \wedge z < x + 5))]$$

It says: "there exists a computation path with a p-state followed by a q-state, followed by an r-state, which is within 5 time units from the p-state." We believe that no formula of TCTL specifies this property. The model-checking algorithm can be modified to handle this extension of TCTL. This can be done by introducing additional clocks, one per each temporal operator in the formula, while constructing the region graph.

TCTL* cannot be model-checked: Note that one can extend the linear-time temporal logic PTL also with the subscripted operators to get a real-time specification language, and interpret it over dense computation paths. The resulting logic is undecidable. Model-checking is also undecidable, because in case of the linear-time temporal logics, satisfiability can be reduced to model-checking over a trivial structure which imposes no constraints on the paths generated. Consequently, if we attempt to extend TCTL to TCTL*, in the same way as CTL was extended to CTL* by disassociating the path quantifiers from the temporal operators ([CES86]), the model-checking problem for the resulting logic is undecidable.

Future Research : We have proposed a plausible notion of finite-state real-time systems, and associated semantics of dense computation paths with it. Alternative formulations possibly exist, and need to be studied. With the success of the CTL-based techniques in the automatic verification of finite-state concurrent systems, we feel hopeful that the model-checking algorithm developed in this paper can be used in practice for verifying moderately sized systems.

Timed graphs can be viewed as generating (or accepting) *dense words* (meaning, words viewed as maps from a dense linear order to an alphabet). Since there exist strong connections between various notions of ω-automata, the theory of ω-regular expressions, and linear temporal logics, to get more insights into the nature of timed graphs, certain theoretical aspects of the languages of dense words need to be understood.

Another promising direction for extending this research is to add probabilistic information to our models, and to develop a formalism for reasoning about time and reliability together.

Acknowledgements : Pierre Wolper suggested looking at real-time in a branching-time, instead of linear-time, model. Harry Lewis helped clarify the relation between this paper and his own work. We thank him as well as Thomas Henzinger, Zohar Manna, and Moshe Vardi for helpful discussions.

References

[AK83] S. Aggarwal, R.P. Kurshan, "Modeling elapsed time in protocol specification," *Protocol Specification, Testing and Verification*, III, 1983.

[AD90] R. Alur, D.L. Dill, "Automata for modeling real-time systems," 17th ICALP, 1990.

[AH89] R. Alur, T.A. Henzinger, "A really temporal logic," 30th IEEE FOCS, 1989.

[Bu89] J.R. Burch, "Combining CTL, trace theory and timing models," *Automatic Verification Methods for Finite State Systems*, *LNCS* **407**, 1989.

[CES86] E.M. Clarke, E.A. Emerson, A.P. Sistla, " Automatic verification of finite-state concurrent systems using temporal logic specifications," ACM *TOPLAS* **8(2)**, 1986.

[Di89] D.L. Dill, " Timing assumptions and verification of finite-state concurrent systems," *Automatic Verification Methods for Finite State Systems*, *LNCS* **407**, 1989.

[EC82] E.A. Emerson, E.C. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons," *Science of Computer Programming* **2**, 1982.

[EMSS89] E.A. Emerson, A.K. Mok, A.P. Sistla, J. Srinivasan, "Quantitative temporal reasoning," presented at the Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France, 1989.

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

[JM86] F. Jahanian, A.K. Mok, " Safety analysis of timing properties in real-time systems," IEEE *Trans. on Software engineering*, **12(9)**, 1986.

[Ko89] R. Koymans, "Specifying message passing and time-critical systems with temporal logic," Ph.D. Thesis, Eindhoven Univ. of Tech., 1989.

[Le89] H.R. Lewis, "Finite-state analysis of asynchronous circuits with bounded temporal uncertainty," Tech. Report TR-15-89, Harvard Univ., 1989.

[Le90] H.R. Lewis, "A logic of concrete time intervals," 5th IEEE LICS, 1990.

[LPS81] D. Lehman, A. Pnueli, J. Stavi, "Impartiality, justice, and fairness: The ethics of concurrent termination," In *Automata, Languages, and Programming*, Springer *LNCS* **115**, 1981.

[MP89] Z. Manna, A. Pnueli, "The anchored version of the temporal framework," *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency* (J.W. deBakker, W.-P. de Roever, and G. Rozenberg, eds.), Springer *LNCS* **354**, 1989.

[OL82] S. Owicki, L. Lamport, "Proving liveness properties of concurrent programs," ACM *TOPLAS* **4**, 1982.

[Os87] J.S. Ostroff, *Temporal Logic of Real-time Systems*, Ph.D. Thesis, Univ. of Toronto, 1987. (Also Research Studies Press, 1990.)

[Pn77] A. Pnueli, "The temporal logic of programs," 18th IEEE FOCS, 1977.