INFORMATION AND COMPUTATION **104**, 2–34 (1993)

Model-Checking in Dense Real-Time

RAJEEV ALUR[1]

Department of Computer Science, Stanford University, Stanford, California 94305

COSTAS COURCOUBETIS[2]

Computer Science Department, University of Crete, and Institute of Computer Science, FORTH, Heraklion, Crete 71110, Greece

AND

DAVID DILL[3]

Department of Computer Science, Stanford University, Stanford, California 94305

Model-checking is a method of verifying concurrent systems in which a state-transition graph model of the system behavior is compared with a temporal logic formula. This paper extends model-checking for the branching-time logic CTL 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 $\exists\Diamond_{<5}$, meaning "possibly within 5 time units." The formulas of the resulting logic, *Timed CTL* (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 *timed graphs*—state-transition graphs annotated with timing constraints. As our main result, we develop an algorithm for model-checking, 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 significantly blow up the complexity of the model-checking problem. On the negative side, we show that the denseness of the underlying time domain makes the validity problem for TCTL Π_1^1-hard. The question of deciding whether there exists a timed graph satisfying a TCTL-formula is also undecidable. © 1993 Academic Press, Inc.

[1] Supported by NSF Grant CCR-8812595, by DARPA Contract N00039-84-C-0211, and by the USAF Office of Scientific Research under Contracts 88-0281 and 90-0057. Currently with AT & T Bell Laboratories, Murray Hill.

[2] Supported by ESPRIT BRA Project SPEC.

[3] Supported by the NSF under Grant MIP-8858807.

0890-5401/93 $5.00

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 temporal reasoning about reactive systems (34, 33, 16, 30) abstract away from quantitative time, preserving only qualitative properties (such as "eventually p holds"). In this paper, we extend these qualitative techniques to reason about quantitative timing behavior as well.

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 state-transition graph and the size of the formula, when the logic is the branching-time temporal logic CTL (computation tree logic) (12). CTL model-checking has been used to prove 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. There are three basic approaches to modeling real-time systems. *Discrete-time* models use the domain of integers to model time (e.g., 1, 23, 18). This approach accurately describes the behavior of *synchronous* systems, where all components are driven by a common global clock. However, to model *asynchronous* systems it becomes necessary to discretize continuous time 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. For instance, Brzozowski and Seger show that the reachability problem for asynchronous circuits with bounded delays cannot be solved correctly using the discrete-time model (9). Also 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., 32, 6, 10, 19). This model allows arbitrarily many transitions of any process between two successive *tick* transitions. (The discrete-time model can be viewed as a special case where the events happen only in lockstep with the *ticks*.) The timing delay between two events is measured by counting the number of *ticks* between them. When we require that there be k *ticks* between two transitions, we can only infer that the delay between them is at least $(k-1)$ time units and at most $(k+1)$ 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."

The third approach to modeling real-time behavior models time, more realistically, as a continuous quantity (e.g., 25, 29, 27, 14). We prefer to use this *dense-time* model (see (2) for some advantages of the dense-time model over the discrete models). 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 the discrete-time model 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 exact 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 interpret formulas over continuous computation trees. We model systems with delays 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 significantly affect the complexity of the model-checking problem. On the other hand, we show that questions such as satisfiability and finite satisfiability (satisfiability with respect to some timed graph) are undecidable for TCTL.

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 transitions of a system. We imagine that the system is equipped with a finite set of real-valued *clocks*. The clocks can be reset to 0 (independently of each other) with the state-transitions of the system, and keep track of the time elapsed since the last reset. To express the timing delays of the system, we associate with each transition an enabling condition which puts certain constraints on the values of the clocks, and require that the transition may be taken only if the current values of the clocks satisfy this condition. 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 a 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 problem of determining the truth of a TCTL-formula with respect to a timed graph. The main difficulty in developing such an algorithm is that a timed graph has infinitely many states due to its real-valued clock variables. The basic 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 only a *finite* number of equivalence classes under this relation, and this enables us to obtain an ordinary finite 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 various ways to extend our results.

Related Work

There have been several temporal logics with quantitative time (see (7) for a survey). These include linear-time logics with discrete semantics (23, 32, 6, 19), linear-time logics with dense semantics (8, 25), and branching-time logics with discrete semantics (18). The syntax of our logic is very similar to that of the *Real-Time CTL* of (18). Verification algorithms have been developed for logics with discrete semantics, but in the case of dense-time models the only previously known result is an undecidability result: in (6) it is shown that the satisfiability problem for a real-time extension of the linear-time temporal logic PTL is undecidable (Σ_1^1-hard) in the dense-time model.

Different ways of incorporating timing constraints in the state-transition graph model of a system have been studied recently. Perhaps the most standard way of introducing timing information in a process model is by associating lower and upper bounds with transitions. Examples of these are timed transition systems (32, 20), timed I/O automata (29), and

Modecharts (24). Our definition of timed graphs is based on the formalism of *timed automata* (14, 4). Timed automata accept *timed traces*—sequences of events in which each element has an associated real-valued time of occurrence. The semantics of timed automata is linear-time and event-based, whereas the semantics of timed graphs is branching-time and state-based. An automata-theoretic approach to verification of timing requirements of real-time systems has been developed using timed automata (4).

A model similar to ours was independently proposed and studied by Lewis (27). He defines *state-diagrams*, and gives a way of translating a circuit description to a state-diagram. A state-diagram is a finite-state machine where every edge is annotated with a matrix of intervals constraining various delays. Both the formalisms, state-diagrams and timed graphs, have the same expressiveness. In (28), Lewis defines a branching-time logic similar to TCTL: the syntax extends CTL with time-bounded versions of temporal operators, and the formulas are interpreted over the state-diagrams. Lewis presents an algorithm for model-checking for a special class of state-diagrams; the ones in which only a bounded number of transitions can happen in a time interval of unit length. Our algorithm does not need the latter assumption, and has a better worst-case complexity. We note that the decidability and lower bound results presented here carry over to his formalism also.

2. Computation Tree Logic

Computation tree logic (CTL) was introduced by Emerson and Clarke (16) as a specification language for finite-state systems. Let us briefly review its syntax and semantics.

Let AP be a set of atomic propositions. The formulas of CTL are inductively defined as

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

where $p \in \mathrm{AP}$, 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 Kripke structure $\mathcal{M} = (\mathcal{S}, \mu, \mathrm{E})$, where $\mathcal{S}$ is a countable set of states, $\mu: \mathcal{S} \to 2^{\mathrm{AP}}$ gives an assignment of truth values to propositions in each state, and E is a binary relation over $\mathcal{S}$ giving the possible transitions. A *path* is an infinite sequence of states $(s_0, s_1, \ldots)$ such that $\langle s_i, s_{i+1} \rangle \in \mathrm{E}$ for all $i \geqslant 0$. 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 as follows (since the structure is fixed, we abbreviate $(\mathcal{M}, s) \models \phi$ to $s \models \phi$):

$s \models p$ iff $p \in \mu(s)$.
$s \not\models$ **false**.
$s \models \phi_1 \to \phi_2$ iff $s \not\models \phi_1$ or $s \models \phi_2$.
$s \models \exists \bigcirc \phi$ iff $t \models \phi$ for some state $t \in \mathcal{S}$ such that $\langle s, t \rangle \in \mathrm{E}$.
$s \models \exists(\phi_1 \mathcal{U} \phi_2)$ iff for some path $(s_0, s_1, \ldots)$ with $s = s_0$, for some $i \geqslant 0$, $s_i \models \phi_2$ and $s_j \models \phi_1$ for $0 \leqslant j < i$.
$s \models \forall(\phi_1 \mathcal{U} \phi_2)$ iff for every path $(s_0, s_1, \ldots)$ with $s = s_0$, for some $i \geqslant 0$, $s_i \models \phi_2$ and $s_j \models \phi_1$ for $0 \leqslant j < i$.

A CTL-formula ϕ is called *satisfiable* iff there are a Kripke 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) (17). Consequently, the satisfiability question for CTL is decidable. It has been shown that it is deterministic exponential-time complete. However, the model-checking problem is in PTIME—given a CTL-formula ϕ and a Kripke structure $\mathcal{M} = (\mathcal{S}, \mu, \mathrm{E})$ together with a state $s \in \mathcal{S}$, there is an algorithm for deciding whether or not $(\mathcal{M}, s) \models \phi$ which runs in time $O[|\phi| \cdot (|\mathcal{S}| + |\mathrm{E}|)]$ (16).

Finite-state concurrent systems can be modeled as finite Kripke structures. Given a system modeled as a Kripke 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 (26). For example, in a system with two processes, we may wish to consider only those computation sequences in which each process executes infinitely often.

Since CTL cannot express correctness along fair paths, Clarke *et al.* have defined CTL^F by changing the semantics of CTL (12). The syntax of CTL^F is the 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 only over F-fair paths. The model-checking algorithm for CTL can be modified to handle this extension of CTL at a cost of a factor 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. A natural and straightforward extension is to put subscripts on the temporal operators to limit their scope in time (25, 18). For example, we can write $\exists\Diamond_{<5}p$ to say that along some computation path p becomes true within 5 time units. We use this approach to introduce explicit time in the syntax of TCTL.

To define the semantics, first we need to generalize the notion of a computation path. Recall that in the qualitative case, a path is simply an ω-sequence of states. In the quantitative case, 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[4] to states of the system. Formally,

Definition (s-Path). For a set $\mathscr{S}$ of states and $s \in \mathscr{S}$, an *s-path* through $\mathscr{S}$ is a map ρ from R to $\mathscr{S}$ 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; that is, by a binary relation over the set of states. In dense time, there is always a third state between any pair of states on a path, so another characterization of trees must be used. An alternative way to describe a tree is by specifying a collection of computation paths satisfying certain closure conditions (15). We use this approach to define the structures for TCTL. To state the precise definition of a "dense" tree consisting of "dense" paths, we need some auxiliary definitions.

Let $\mathscr{S}$ be a set of states, ρ be an s-path through $\mathscr{S}$, and $t \in \mathrm{R}$ be any time. The *prefix* of ρ up to time t, denoted by ρ_t, is a map from $[0, t)$ to $\mathscr{S}$ obtained by restricting the domain of ρ. The *suffix* of ρ at time t, denoted by ρ^t, is a $\rho(t)$-path defined by $\rho^t(t') = \rho(t + t')$ for every $t' \in \mathrm{R}$. Furthermore, if ρ' is some map from $[0, t)$ to $\mathscr{S}$, then its concatenation with ρ, denoted by $\rho' \cdot \rho$, is defined by:

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

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

Now we can define the notion of a TCTL-structure. Let AP be a set of atomic propositions.

[4] 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.

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.
- $\mu : \mathcal{S} \to 2^{\mathrm{AP}}$ 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 $\mathcal{S}$. f satisfies the following closure properties:

1. Suffix closure:

$$\forall s \in \mathcal{S} . \forall \rho \in f(s) . \forall t \in \mathbf{R} . \rho^t \in f[\rho(t)].$$

2. Fusion closure:

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

The fusion-closure condition says that the behavior of the system does not depend on the past but only on the current state. If a state s appears along a path at time t, the set of all possible computation paths can be obtained by concatenating the prefix up to time t with all the computations in $f(s)$. This requirement ensures that the reachability relation over $\mathcal{S}$ induced by f is transitive.

Since the states of a TCTL-structure are labeled with propositions, every computation path ρ has an associated map $(\mu \cdot \rho)$ from R to subsets of AP. 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). The computation paths of a real-time system should satisfy the *finite variability* condition: along a computation path the truth value of a proposition should change at most ω number of times. One can include this requirement in the definition of a TCTL-structure, but this choice does not change the complexity results to be presented.

Now we proceed to define the syntax and semantics of TCTL. Let N be the set of constants $\{0, 1, 2, ...\}$ denoting the natural numbers.[5] Throughout the paper we use $\sim$ to mean one of the binary relations $<$, $\leqslant$, $=$, $\geqslant$, or $>$.

DEFINITION (Syntax). The formulas ϕ of TCTL are inductively defined as follows,

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

where $p \in \mathrm{AP}$ and $c \in \mathrm{N}$.

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

Informally, $\exists(\phi_1 \mathcal{U}_{<c} \phi_2)$ means that for some computation path, there exists an initial prefix of time duration 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. 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:

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

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

The other logical connectives can be defined as usual. We define abbreviations $\exists\Diamond_{\sim c}\phi$ for $\exists(\mathbf{true}\,\mathcal{U}_{\sim c}\phi)$, $\forall\Diamond_{\sim c}\phi$ for $\forall(\mathbf{true}\,\mathcal{U}_{\sim c}\phi)$, $\exists\Box_{\sim c}\phi$ 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 $\geqslant 0$. For example, $\exists\Diamond\phi$ corresponds to $\exists\Diamond_{\geqslant 0}\phi$. 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.

The denseness of the underlying time domain allows us to encode computations of 2-counter machines in TCTL-formulas. Consequently, unlike other logics with similar syntax but discrete-time semantics (e.g., RTCTL of (18)), the satisfiability question for TCTL is undecidable—Σ_1^1-hard.[6] The proof is similar to the proof of undecidability of a linear-time dense real-time logic (6).

THEOREM (Undecidability of TCTL). *The satisfiability question for TCTL is Σ_1^1-hard.*

[6] The class Σ_1^1 consists of highly undecidable problems, including some nonarithmetical sets. See, for instance, (35) for an exposition of the analytical hierarchy.

Proof. The problem of deciding whether a nondeterministic 2-counter machine has a computation in which the starting location is visited infinitely often is known to be Σ_1^1-hard. We reduce this problem to TCTL-satisfiability.

Let M be a 2-counter machine with counters α and β, and n program instructions. Each instruction may increment or decrement one of the counters, or may be a conditional jump depending on one of them being zero, or may be a nondeterministic jump selecting between two locations.

A configuration of the machine is represented by a triple $\langle i, c, d\rangle$, where i gives the instruction to be executed next, and c and d give the contents of the counters α and β, respectively. A computation of the machine is an infinite sequence of triples starting with $\langle 1, 0, 0\rangle$.

We encode the computations of M in our logic using the propositions p_α, p_β, and $p_1, ..., p_n$. Let us say that a path ρ in a given structure encodes a configuration $\langle i, c, d\rangle$ over the interval $[a, b)$, with $a, b \in \mathrm{R}$ and $a < b$, iff the following hold:

- The proposition p_α changes its value exactly c times in the interval $[a, b)$ along ρ.
- The proposition p_β changes its value exactly d times in the interval $[a, b)$ along ρ.
- Each proposition p_j, $j \neq i$, is false everywhere in the interval $[a, b)$ along ρ.
- The proposition p_i is true everywhere in the interval $[a, b)$ along ρ.

We construct a TCTL-formula ϕ such that for all structures $\mathcal{M}$ and states s, $(\mathcal{M}, s) \models \phi$ iff there exists a computation $\{\langle i_j, c_j, d_j\rangle : j \geq 0\}$ of M such that every s-path ρ in $\mathcal{M}$ encodes the jth configuration over the interval $[j, j+1)$ for all $j \geq 0$.

The nature of the initial configuration can be easily expressed in TCTL. To relate one configuration to the next, we establish a one-to-one correspondence between the states separated by distance 1. The desired formula ϕ has one conjunct for each program instruction. For instance, if the second instruction is to increment β (and proceed to instruction 3), ϕ has the following conjunct:

$$\forall\Box\left[\Box_{[0,1)}p_2 \rightarrow \begin{pmatrix} \forall\Box_{[1,2)}\Big(p_3 \wedge \bigwedge_{1 \leq i \leq n, i \neq 3} \neg p_i\Big) \wedge \\ \forall\Box_{[0,1)}\mathit{copy}(p_\alpha) \wedge \\ \forall[\mathit{copy}(p_\beta)\,\mathcal{U}_{<1}\,\forall(\mathit{toggle}(p_\beta)\,\mathcal{U}_{>0}\,p_3)] \end{pmatrix}\right].$$

The abbreviation $\mathit{copy}(p)$ stands for the conjunction

$$(p \rightarrow \forall\Diamond_{=1}p) \wedge (\neg p \rightarrow \forall\Diamond_{=1}\neg p),$$

and *toggle*(p) stands for the conjunction

$$(p \rightarrow \forall \Diamond_{=1} \neg p) \wedge (\neg p \rightarrow \forall \Diamond_{=1} p).$$

The first conjunct requires the propositions representing the instruction counter change according to the desired scheme. The second conjunct makes sure that the number of times the value of p_α changes is the same in the next configuration as in the current one. The third conjunct holds at time t, iff the number of changes in the value of p_β in the interval $[t+1, t+2)$ is precisely one greater than the corresponding number for the interval $[t, t+1)$.

The conjunction of ϕ with the recurrence requirement $\forall \Box \forall \Diamond p_1$ is satisfiable iff M has a computation with the location counter equal to 1 infinitely often. Consequently, TCTL-satisfiability is Σ_1^1-hard. ▮

4. Timed Graphs

In this section we introduce *timed graphs* to model finite-state real-time systems. A timed graph operates with finite control—a finite set of *nodes* and a finite set of real-valued *clocks*. All clocks proceed at the same rate and measure the amount of time that has elapsed since they were started (or reset). Each transition of the system may reset some of the clocks. With each transition we associate an enabling condition which puts certain constraints on the values of the clocks, and require that the transition may be taken only if the current values of the clocks satisfy this condition.

For example, Fig. 1 shows a system with four nodes and two clocks, x and y. The clock x is reset 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 delay between the transition from s_0 to s_1 and the transition from s_2 to s_3 has lower bound 1 and upper bound 2. Similarly, the clock y constrains the transition from s_3 to s_0 to occur at least 2 time units later than the s_1 to s_2 transition.

Thus to express a bound on the delay between two transitions, we reset

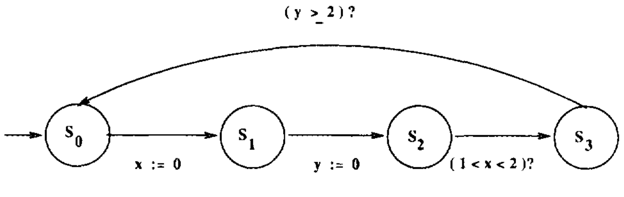

FIG. 1. Timed graph.

a clock with the first transition, and associate an enabling condition with the other transition. Note that different clocks can be started at different times, and there is no lower bound on the difference between their readings. Thus having multiple clocks allows modeling of multiple concurrent delays.

DEFINITION (Timed Graph). A timed graph is a tuple $G = \langle S, \mu, s_{\text{init}}, E, C, \pi, \tau \rangle$, where

- S is a finite set of nodes.
- $\mu: S \to 2^{AP}$ is a labeling function assigning to each node the set of atomic propositions true in that node.
- $s_{\text{init}} \in S$ is the start node.
- $E \subseteq S \times S$ is a set of edges.
- C is a finite set of clocks.
- $\pi: E \to 2^C$ 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 from the atomic formulas of the form $x \leqslant c$ or $c \leqslant x$, where $x \in C$ and $c \in N$.[7]

The system starts at node s_{init} 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 satisfied by the current values of the clocks. The transitions are instantaneous. With each transition e, the clocks in $\pi(e)$ are 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.

A *clock assignment* for a set of clocks C is a function from C to R. Let $\Gamma(G)$ denote the set of all clock assignments for the clocks of G. A *state* of the system is of the form $\langle s, v \rangle$, where $s \in S$ and $v \in \Gamma(G)$.

Some notation: Let $v \in \Gamma(G)$ and $t \in R$. Then $v + t$ denotes the clock assignment which assigns to each clock y the value $v(y) + t$, and $[x \mapsto t]\, v$ denotes the clock assignment for $C \cup \{x\}$ which assigns t to x and agrees with v on the values of the rest of the clocks.

Now we define a *run* of a timed graph to formally capture its behavior. An $\langle s, v \rangle$-run records the states at time instants at which the transitions occur along a possible computation of the system started in $\langle s, v \rangle$.

[7] We could generalize the enabling conditions to allow all quantifier-free formulas built using the primitive of linear order, constants, and addition by constants, but allowing addition of variables makes model-checking undecidable (2).

Definition ($\langle s, v\rangle$-Run). Given a state $\langle s, v\rangle$ of G, an $\langle s, v\rangle$-run of G is an infinite sequence

$$(\langle s_0, v_0, t_0\rangle, \langle s_1, v_1, t_1\rangle, \langle s_2, v_2, t_2\rangle, ...)$$

of nodes $s_i \in \mathrm{S}$, clock assignments $v_i \in \Gamma(\mathrm{G})$, and times $t_i \in \mathrm{R}$ satisfying the following constraints:

• *Initialization*: The run starts in state $\langle s, v\rangle$ at time 0: $s_0 = s$, $v_0 = v$, and $t_0 = 0$.

• *Consecution*: For each $i \geqslant 0$:

— The time of the $(i+1)$th transition is strictly greater than the time of the ith transition: $t_{i+1} > t_i$.

— $e_i = \langle s_i, s_{i+1}\rangle$ is an edge in E.

— The clock assignment v_{i+1} at time t_{i+1} equals $[\pi(e_i) \mapsto 0]$ $(v_i + t_{i+1} - t_i)$.

— The clock assignment $(v_i + t_{i+1} - t_i)$ satisfies the enabling condition $\tau(e_i)$.

• *Progress of time*: Every time value is eventually reached; that is, for any $t \in \mathrm{R}$, there exists some j such that $t_j \geqslant t$.

The last restriction in the above definition rules out the behaviors in which an infinite number of transitions occur in a bounded interval of time.

An $\langle s, v\rangle$-run gives an $\langle s, v\rangle$-path ρ as a map from R to the states of the system: for $t \in \mathrm{R}$, if $t_j \leqslant t < t_{j+1}$, then the state $\rho(t)$ at time t is $\langle s_j, v_j + t - t_j\rangle$.

The paths generated by a timed graph have certain noteworthy properties. First, the state $\langle s_i, (v_i + t_{i+1} - t_i)\rangle$ 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 clock assignment at time t_{i+1} shows the values of the clocks in $\pi(e_i)$ as 0. We can view ρ as a map giving the truth assignment to propositions at every time constant. Then the values of all the propositions remain the same over the time interval $[t_i, t_{i+1})$ (thus, ρ is right-continuous). Hence, 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 associate a TCTL-structure with it.

Definition (TCTL-Structure of a Timed Graph). Given a timed graph G, the corresponding TCTL-structure is $\mathcal{M}_{\mathrm{G}} = \langle \mathrm{S} \times \Gamma(\mathrm{G}), \mu', f\rangle$, where the labeling function is defined by $\mu'(\langle s, v\rangle) = \mu(s)$, and $f(\langle s, v\rangle)$ consists of the paths corresponding to the $\langle s, v\rangle$-runs of G.

It is straightforward to verify that the collection f of paths in the above definition satisfies suffix-closure and fusion-closure requirements. Now we can interpret TCTL-formulas over a timed graph.

DEFINITION (Finite Satisfiability). For a TCTL-formula ϕ and a timed graph G, we define $G \models \phi$ precisely when $(\mathcal{M}_G, \langle s_{\text{init}}, \nu_{\text{init}} \rangle) \models \phi$, where ν_{init} is defined by $\nu_{\text{init}}(x) = 0$ for all $x \in C$.

A TCTL-formula ϕ is called *finitely satisfiable* iff 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 timed graph meeting the constraints specified by a TCTL-formula is not possible in general.

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.

While proving the undecidability of TCTL-satisfiability, we encoded the computations of a given 2-counter machine M using a formula ϕ. Now let us assume that M is deterministic and its halting corresponds to the location counter taking a specific value, say, n. Let ψ be the conjunction of ϕ and the halting requirement $\forall \Diamond p_n$. If M does not halt, then ψ is not satisfiable. If M halts, then it does so in, say, m steps, with the counter value never exceeding m. In this case, we construct a timed graph with just one run, which corresponds to the encoding of the finite computation of M. Thus, ψ is finitely satisfiable iff M halts. ▌

5. MODEL-CHECKING

In this section we 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 also study the complexity of the model-checking problem.

5.1. *Equivalence of Clock Assignments*

Suppose the system is described by a timed graph $G = \langle S, \mu, s_{\text{init}}, E, C, \pi, \tau \rangle$. The timed graph 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 whether or not a transition with enabling condition $x=1$ can be followed by a transition with enabling condition $y=1$, depends 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 $x \in \mathrm{C}$, let c_x be the largest constant that x is compared with in any enabling condition. For $t \in \mathrm{R}$, $fract(t)$ denotes the fractional part of t, and $\lfloor t \rfloor$ denotes its integral part.

DEFINITION (Equivalence of Clock Assignments). For clock assignments v and v' in $\Gamma(\mathrm{G})$, $v \cong v'$ iff the following conditions are met:

- For each clock $x \in \mathrm{C}$, either $\lfloor v(x) \rfloor$ and $\lfloor v'(x) \rfloor$ are the same, or both $v(x)$ and $v'(x)$ are greater than c_x.
- For every pair of clocks x, $y \in \mathrm{C}$ such that $v(x) \leqslant c_x$ and $v(y) \leqslant c_y$,
  1. $fract(v(x)) \leqslant fract(v(y))$ iff $fract(v'(x)) \leqslant fract(v'(y))$, and
  2. $fract(v(x)) = 0$ iff $fract(v'(x)) = 0$.

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 Fig. 2. 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 (e.g., $\{(x, y): 0<x<y<1\}$, $\{(x, y): (1<x<2) \wedge (y>1)\}$) represent different equivalence classes.

We use $[v]$ to denote the equivalence class of $\Gamma(\mathrm{G})$ to which v belongs. A *region* is a pair $\langle s, [v] \rangle$, where $s \in \mathrm{S}$ and $v \in \Gamma(\mathrm{G})$. Observe that there are only finitely many such regions. The next lemma says that states belonging to the same region satisfy the same set of TCTL-formulas.

LEMMA (Equivalence of Clock Assignments). *Let* $s \in \mathrm{S}$, *and* $v, v' \in \Gamma(\mathrm{G})$

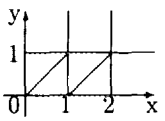

FIG. 2. Equivalence of clock assignments.

with $v \cong v'$. *For every TCTL-formula* ϕ, $(\mathcal{M}_G, \langle s, v\rangle) \models \phi$ *iff* $(\mathcal{M}_G, \langle s, v'\rangle) \models \phi$.

Proof. Before we prove the lemma we need to show that the runs starting at $\langle s, v\rangle$ and $\langle s, v'\rangle$ are similar in the following sense. Let r be an $\langle s, v\rangle$-run $\{\langle s_i, v_i, t_i\rangle : i \geqslant 0\}$. We show that there exists an $\langle s, v'\rangle$-run r' with the same sequence of state-transitions as r taken at "almost" the same times.

Let us extend the equivalence relation over clock assignments to pairs of the form $\langle v, t\rangle$, where v is a clock assignment and t is a time value. Define $\langle v, t\rangle \cong \langle v', t'\rangle$ iff the following conditions hold:

- The clock assignments v and v' are equivalent: $v \cong v'$.

- The time values t and t' agree on their integral parts: (i) $\lfloor t\rfloor = \lfloor t'\rfloor$, and (ii) $fract(t) = 0$ iff $fract(t') = 0$.

- The fractional part of the time value has the same relationship to that of the clocks in both elements: for each clock x, (i) $fract(v(x)) < fract(t)$ iff $fract(v'(x)) < fract(t')$, and (ii) $fract(v(x)) > fract(t)$ iff $fract(v'(x)) > fract(t')$.

We show that there is an $\langle s, v'\rangle$-run r': $\{\langle s_i, v'_i, t'_i\rangle : i \geqslant 0\}$ such that $\langle v_i, t_i\rangle \cong \langle v'_i, t'_i\rangle$ for all $i \geqslant 0$.

Given r we construct the desired run r' step by step. The base case follows since $v \cong v'$ and $t_0 = t'_0 = 0$. Now suppose we have constructed r' up to i steps. Let $\delta = t_{i+1} - t_i$ be the delay between the $(i+1)$th and ith transitions of r. We try to find δ' such that we can extend r' using the edge $e_i = \langle s_i, s_{i+1}\rangle$ at time $t'_i + \delta'$ so that equivalence is maintained at the $(i+1)$th step.

Let Δ be the set $\{t_i\} \cup \{v_i(x) : x \in C\}$. The set Δ' is defined similarly. Because of the equivalence at the ith step, the necessary and sufficient requirement of the desired δ' is that each element in Δ' should cross the same number of integer boundaries (due to the addition of δ') as the corresponding element from Δ crosses (due to the addition of δ).

For example, let $C = \{x, y\}$. Let $v_i = [0.3, 1.7]$, $t_i = t'_i = 2$, $v'_i = [0.8, 1.9]$, $t_{i+1} = 3.4$, and $v_{i+1} = [1.7, 3.1]$. In this case $\delta = 1.4$, and δ' should satisfy $0.8 + \delta' \in (1, 2)$, $1.9 + \delta' \in (3, 4)$, and $2 + \delta' \in (3, 4)$. 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 the elements in Δ', which is the same as the ordering of the fractional parts of the elements in Δ. Hence, the existence of δ guarantees the existence of δ'.

Now we proceed to prove the lemma. The proof is by induction on the structure of ϕ. The base case and the cases corresponding to the logical connectives follow immediately. We will consider the case $\phi = \exists(\phi_1 \mathcal{U}_{\sim c} \phi_2)$.

Suppose $\langle s, v\rangle \models \phi$. There exists an $\langle s, v\rangle$-run $r: \{\langle s_i, v_i, t_i\rangle : i \geqslant 0\}$ with the corresponding path ρ and time value t such that $t \sim c$, and $\rho(t) \models \phi_2$, and $\rho(u) \models \phi_1$ for all $u < t$.

Let $r': \{\langle s_i, v'_i, t'_i\rangle : i \geqslant 0\}$ be the corresponding $\langle s, v'\rangle$-run with the associated path ρ' constructed as above. Let $t_j \leqslant t < t_{j+1}$. We know that $\langle v_j, t_j\rangle$ and $\langle v'_j, t'_j\rangle$ are equivalent. By an argument identical to the one used in construction of r', we can find $t'_j \leqslant t' < t'_{j+1}$ such that $\langle v_j + t - t_j, t\rangle \cong \langle v'_j + t' - t'_j, t'\rangle$. Clearly, $t' \sim c$.

Note that $\rho(t) = \langle s_j, v_j + t - t_j\rangle$ and $\rho'(t') = \langle s_j, v'_j + t' - t'_j\rangle$. Hence, by the induction hypothesis and $\rho(t) \models \phi_2$, we obtain $\rho'(t') \models \phi_2$.

Let u' be such that $u' < t'$. Suppose $t'_k \leqslant u' < t'_{k+1}$, $k \leqslant j$. Now using the equivalence of $\langle v'_k, t'_k\rangle$ and $\langle v_k, t_k\rangle$, we can find u such that $\langle v'_k + u' - t'_k, u'\rangle \cong \langle v_k + u - t_k, u\rangle$. Note that $\rho(u) = \langle s_k, v_k + u - t_k\rangle$ and $\rho'(u') = \langle s_k, v'_k + u' - t'_k\rangle$. Using the fact that $\rho(u) \models \phi_1$ and the induction hypothesis, we get $\rho'(u') \models \phi_1$.

Thus ρ' fulfills $\phi_1 \mathcal{U}_{\sim c} \phi_2$, and hence, $\langle s, v'\rangle \models \phi$. ▮

5.2. *The Region Graph*

The basic idea of our algorithm is to construct a finite structure from the finitely many regions and then to employ the usual discrete-time model-checking techniques.

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

Suppose we want to determine whether $\langle s, [v]\rangle$ should be labeled with $\exists\Diamond_{<c}\psi$. We try to find a run starting at $\langle s, v\rangle$ such that the associated path satisfies $\Diamond_{<c}\psi$. As time progresses, the state of the system changes, but the truth of the subformula ψ 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 $\langle s, [v]\rangle$ such that each region can be reached from the previous one by a state-transition or increase in the values of clocks. Furthermore, ψ should be true in the last region in the desired sequence, 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 idea. 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 $x \notin \mathrm{C}$. Let $\mathrm{C}^* = \mathrm{C} \cup \{x\}$ and let $\Gamma^*(\mathrm{G})$ denote the set of all assignments from C^* to R. Let $c(\phi)$ denote the largest constant appearing in ϕ. We define an equivalence relation $\cong^*$ over $\Gamma^*(\mathrm{G})$ similar to the relation $\cong$ on $\Gamma(\mathrm{G})$ with $c_x = c(\phi)$. Furthermore, for a clock assignment $v \in \Gamma^*(\mathrm{G})$, let $[v]^*$ denote its equivalence class with respect to $\cong^*$.

Let us call an element of the form $\langle s, [v]^* \rangle$, where $s \in S$ and $v \in \Gamma^*(G)$, an *augmented region*. Note that each region is refined into many augmented regions. Now to determine the truth of a formula $\exists\Diamond_{<c}\psi$ in a region v, we try to find a region v' where ψ holds, and a path from the region v augmented with $x = 0$ to an augmented region which is a refinement of v' and satisfies $x < c$. The new clock x is never reset along this path, and is updated consistently with the other clocks. This ensures that the path can indeed be traversed within time c. If ψ itself is a temporal formula, say, $\exists\Diamond_{\geqslant d}\psi'$, we search for an appropriate path starting at the region v' augmented with $x = 0$. Thus the clock x gets reused for evaluating all subformulas. Note that the value of x is used to evaluate the subscripts on the temporal operators, and is of no importance once it crosses $c(\phi)$, and hence while defining $\cong^*$ over $\Gamma^*(G)$ 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 clock assignments because of the passage of time. We define a *successor* function over equivalence classes of clock assignments to capture the second type of transitions.

Definition (Successor Region). Let α and β be distinct equivalence classes of $\Gamma^*(G)$. The class β is said to be the successor of α, written $succ(\alpha) = \beta$, iff for each $v \in \alpha$, there exists a positive $t \in R$ such that $v + t \in \beta$, and $v + t' \in \alpha \cup \beta$ for all $t' < t$.

Successor is defined for every equivalence class except the *end class*—the equivalence class satisfying $x > c_x$ for all clocks x.

Let us consider the equivalence classes shown in Fig. 2 again. 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 $v \in \alpha$ and for any positive t, v and $v + t$ 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. Consider a vertex $\langle s, \alpha \rangle$. Suppose α is a boundary class. Then the only way a region change occurs is due to the passage of time. This is because the equivalence class changes before any transition of G can occur. Hence there is a single outgoing edge to the vertex $\langle s, succ(\alpha) \rangle$. Now suppose α is not a boundary class. As in the previous case we put an edge to $\langle s, succ(\alpha) \rangle$ representing passage of time (if α is the end-class, then it has no successor, and this edge does not exist). In addition, we put edges representing transitions of G also. Let $e = \langle s, s' \rangle$ be an edge of G. If the current clock values satisfy the enabling condition $\tau(e)$ then the transition e can be taken before the equivalence class changes. The new class in this case is α with the clocks in $\pi(e)$ reset to 0, and we put an edge to $\langle s', [\pi(e) \mapsto 0]\, \alpha \rangle$. Also the transition from s to s' can occur precisely at the instant when the current equivalence class changes to $succ(\alpha)$, provided this new class $succ(\alpha)$ satisfies the enabling condition $\tau(e)$. In this case, we put an edge to the vertex $\langle s', [\pi(e) \mapsto 0]\, succ(\alpha) \rangle$. The precise definition of the region graph is given below.

DEFINITION (Region Graph). The region graph $R(G, \phi)$ is defined to be the graph $\langle V^*, E^* \rangle$. The vertex set V^* is the set of all augmented regions. The edge set E^* consists of two types of edges:

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

There is a simple correspondence between the runs of G and infinite paths through $R(G, \phi)$. Let r: $\{\langle s_i, v_i, t_i \rangle : i \geq 0\}$ be any run of G. Later we show that we can find a path β in $R(G, \phi)$ by connecting augmented regions of the form $\langle s_i, [[x \mapsto t_i]\, v_i]^* \rangle$. Since time progresses without bound along r, either every clock $y \in C^*$ is reset infinitely often or eventually it always increases. Hence, for each $y \in C^*$, along β, infinitely many augmented regions satisfy either $y = 0$ or $y > c_y$.

Let us denote the set

$$\{\langle s, [v]^* \rangle : s \in S \wedge (v(y) = 0 \vee v(y) > c_y)\}$$

by F_y, for $y \in C^*$. If we treat the region graph as a structure for CTL^F with the fairness family given by $F_p = \{F_y : y \in C^*\}$, then the paths corresponding to the runs of G are F_p-*fair*. Conversely, for any F_p-*fair* path in the region graph, we can find a corresponding run along which time increases without bound.

The construction of the region graph can be best understood through the example shown in Fig. 3. The timed graph has three states s_0, s_1, and s_2, and two clocks y and z. Looking at the enabling conditions we get $c_y = c_z = 1$. To construct the region graph we introduce a new clock x, and let $c_x = 1$ (this means that we can use the region graph for model-checking any TCTL-formula ϕ with $c_\phi = 1$). In the region graph, we have only shown those vertices that are reachable from the initial vertex $\langle s_0, \{x = y = z = 0\} \rangle$. Each vertex is labeled with the node of the timed

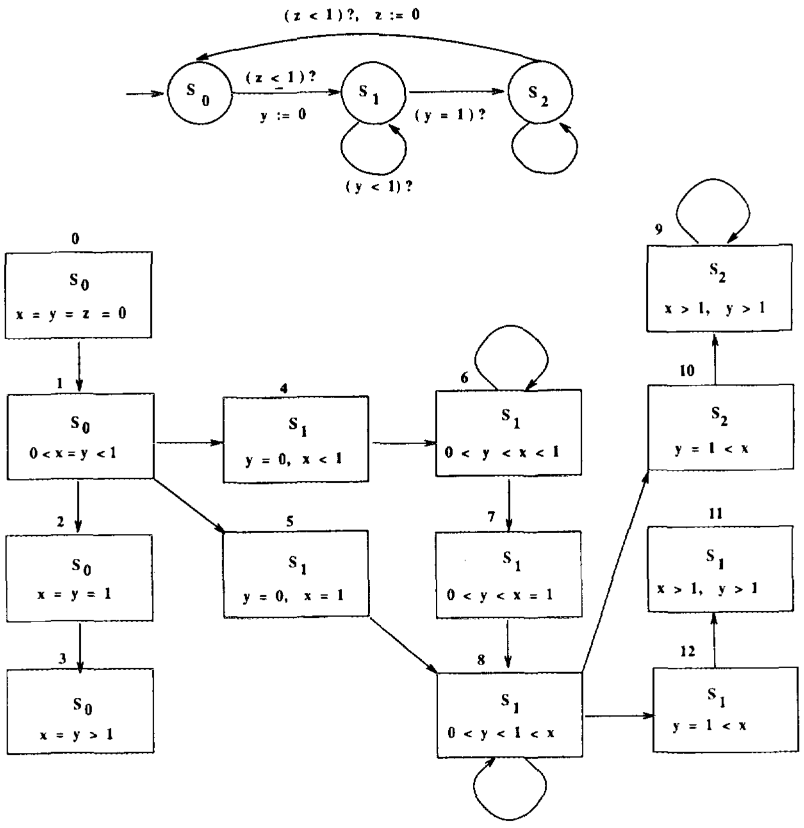

FIG. 3. Constructing the region graph.

graph, and the constraints describing the equivalence class of clock values (the condition $x = z$ holds at all vertices, but is omitted). Note that though the timed graph has an edge from s_2 to s_0, in the region graph there is no way to reach an s_0-vertex from the s_2-vertices. Thus the region graph helps in detecting timing inconsistencies along the paths in the original timed graph. Every infinite fair path starting at the initial vertex 0 ends in looping at the vertex 9. Observe that looping indefinitely at vertex 6 or 8 is disallowed by the fairness conditions.

5.3. *Labeling Algorithm*

We label the augmented regions with subformulas of ϕ, or their negations, starting from the subformulas of length 1, then of length 2, and so on. Initially, we also label the augmented regions with special propositions as follows. Every vertex is labeled with the formula **true**. For every subscript $\sim c$ appearing in ϕ, let there be a new proposition $p_{\sim c}$. Label a vertex $\langle s, [v]^*\rangle$ with $p_{\sim c}$ if $v \models x \sim c$, else label it with $\neg p_{\sim c}$. Furthermore, let p_b be a new proposition which is true at a vertex $\langle s, \alpha\rangle$ iff α is a boundary class.

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

- If ψ is an atomic proposition, then if $\psi \in \mu(s)$ then label v with ψ else with $\neg\psi$.
- If ψ is the implication $\phi_1 \rightarrow \phi_2$ then, if v is labeled with $\neg\phi_1$ or with ϕ_2, then label it with ψ, else with $\neg\psi$.
- Suppose ψ is the temporal formula $Q\ (\phi_1 \mathcal{U}_{\sim c} \phi_2)$, where Q is either an existential or a universal quantifier. Label v with ψ, if some (or every, depending upon Q) F_p-*fair* path through R(G, ϕ) starting at $\langle s, [[x \mapsto 0]\ v]^*\rangle$, has a prefix $v_1, v_2, ..., v_n$, such that each v_i, $1 \leqslant i < n$, is labeled with ϕ_1, and v_n is labeled with ϕ_2, and with $p_{\sim c}$, and with either p_b or ϕ_1. Otherwise label v with $\neg\psi$.

Let us consider the last clause carefully. First, note that, as mentioned earlier, we search for the paths starting at $\langle s, [[x \mapsto 0]\ v]^*\rangle$ instead of v itself. This ensures that the constraints satisfied by x properly reflect the time taken to traverse a path. Clearly, the last vertex v_n should satisfy the constraint $x \sim c$. Also it should be labeled with ϕ_2, and all the previous vertices v_i, $1 \leqslant i < n$, should be labeled with ϕ_1. The last requirement that v_n be labeled with either p_b or ϕ_1 requires some explanation. Suppose $v_n = \langle s_n, \alpha_n\rangle$. If α_n is not a boundary class then if the equivalence class along a computation path is α_n at time t, then it is α_n at some time $t' < t$ also. Since we want ϕ_1 to hold at all time instants preceding some point corresponding to v_n, we require, in this case, that ϕ_1 should hold at v_n also.

The condition required for computing a label from the labels of the subformulas can be tested using conventional model-checking algorithms for CTL^F (12).

As an example consider again the region graph shown in Fig. 3. Suppose there are two propositions p and q, such that the proposition p is true only at s_0 and the proposition q is true only at s_1. The vertices 0, 2, 4, 5, 7, 10, and 12 are labeled with p_b, and the other vertices are labeled with $\neg p_b$. Consider the formula $\phi = \exists(p \mathcal{U}_{\leqslant 1} q)$. The algorithm will label the vertex 0 with ϕ. For instance, the path 0, 1, 5, 8, 10, 9, 9, 9, ... is an infinite fair path starting at 0 such that all of q and p_b and $p_{\leqslant 1}$ hold at the vertex 5, and p holds at vertices 0 and 1. On the other hand, for $\psi = \exists\Diamond_{\geqslant 1} p$, the algorithm labels the vertex 0 with $\neg\psi$. This is because none of the vertices appearing on an infinite path starting at the vertex 0 is labeled with both p and $p_{\geqslant 1}$.

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

LEMMA (Correctness of the Labeling Algorithm). *Let ψ be a subformula of ϕ. The above labeling algorithm labels $\langle s, [v] \rangle$ with ψ iff $(\mathcal{M}_G, \langle s, v \rangle) \models \psi$.*

Proof. We assume that the CTL labeling algorithm works correctly. The proof is by induction on the structure of ψ. The cases $\psi \in AP$, $\psi = \mathbf{false}$, and $\psi = \phi_1 \to \phi_2$ are self-evident. We prove that $\langle s, v \rangle \models \psi$ iff the algorithm labels $\langle s, [v] \rangle$ with ψ, where $\psi = \exists(\phi_1 \mathcal{U}_{\sim c} \phi_2)$. The other case $\psi = \forall(\phi_1 \mathcal{U}_{\sim c} \phi_2)$ is similar.

Proof of $\Rightarrow$. First observe the following property which follows from the definition of the successor function. For any clock assignment v' and time values $t, t' \in R$, we can find a finite sequence of length j of equivalence classes $\{[v' + t_i]^* : 1 \leqslant i \leqslant j\}$ related by successor function (that is, $succ([v' + t_i]^*) = [v' + t_{i+1}]^*$), such that for all $t \leqslant t'' \leqslant t'$, for some $1 \leqslant k \leqslant j$, $(v' + t'') \cong^* (v' + t_k)$.

Now suppose $\langle s, v \rangle \models \psi$. Let $r: \{\langle s_i, v_i, t_i \rangle : i \geqslant 0\}$ be the $\langle s, v \rangle$-run which satisfies $\phi_1 \mathcal{U}_{\sim c} \phi_2$, and let ρ be the corresponding path. There exists $t \sim c$ such that $\rho(t) \models \phi_2$ and $\rho(t') \models \phi_1$ for all $t' < t$. We will construct a corresponding F_p-*fair* path in R(G, ϕ).

Let $v_{i0} = [x \mapsto t_i]\, v_i$ for $i \geqslant 0$. Using the above mentioned property, for each $i \geqslant 0$, we can find a path β_i through R(G, ϕ) of the form

$$\langle s_i, [v_{i0}]^* \rangle \to \langle s_i, [v_{i1}]^* \rangle \to \cdots \langle s_i, [v_{ij_i}]^* \rangle \to \langle s_{i+1}, [v_{(i+1)0}]^* \rangle,$$

for some j_i. This path is constructed by taking the successor classes of $[v_{i0}]^*$ as time increases from t_i to t_{i+1}, till we reach the class $[v_{i0} + (t_{i+1} - t_i)]^*$. We know for all $t_i \leqslant t' < t_{i+1}$, there is some $0 \leqslant k \leqslant j_i$, such that $(v_{i0} + t' - t_i) \cong^* v_{ik}$.

Let β be the concatenation of all such segments. Since time progresses along r, there is some i such that $t_i > c_x$ (and, for any $j \geqslant i$, $t_j > c_x$, since x never gets reset). Also for every clock $y \in C$, if it is not reset after a certain point i, then there exists a later point k such that for any $j \geqslant k$, $v_j(y) > c_y$. So from the construction of β it follows that it is F_p-fair.

Now we show that β satisfies the formula $\phi_1 \mathcal{U}(\phi_2 \wedge p_{\sim c} \wedge (p_b \vee \phi_1))$. We use v_{ij} to denote the vertex $\langle s_i, [v_{ij}]^* \rangle$. Let $t_k \leqslant t < t_{k+1}$.

The way we have constructed β_k, we can find $l \leqslant j_k$ such that $(v_{k0} + t - t_k) \cong^* v_{kl}$.

First, since $t \sim c$, we obtain $v_{kl}(x) \sim c$, and hence v_{kl} is labeled with $p_{\sim c}$.

Since $\rho(t) \models \phi_2$, and TCTL-formulas cannot distinguish between equivalent states, $\langle s_k, v_{kl} \rangle \models \phi_2$. So by the induction hypothesis v_{kl} is labeled with ϕ_2.

Let $v_{k'l'}$ be a node on β before v_{kl} (that is, either $k' < k$ or ($k' = k$ and $l' < l$)). The way we have constructed $\beta_{k'}$, there is $t_{k'} \leqslant t' < t_{k'+1}$ such that $v_{k'0} + (t' - t_{k'}) \cong^* v_{k'l'}$ and $t' < t$. Since $\rho(t') \models \phi_1$, we obtain $\langle s_{k'}, v_{k'l'} \rangle \models \phi_1$. So by the induction hypothesis $v_{k'l'}$ is labeled with ϕ_1.

If v_{kl} is not labeled with p_b, then v_{kl} is not a boundary class, and there exists a positive δ such that $v_{kl} - \delta \cong^* v_{kl}$. Since $\rho(t - \delta) \models \phi_1$, v_{kl} should be labeled with ϕ_1.

Hence the start vertex of β, that is $\langle s, [[x \mapsto 0]\, v]^* \rangle$, should be labeled with ψ.

Proof of $\Leftarrow$. Let us start with the following observation which follows from the construction of the region graph: Let $\langle s', \alpha_1 \rangle, ..., \langle s', \alpha_n \rangle$ be a path in R(G, ϕ) consisting solely of edges corresponding to the passage of time. Suppose there is edge from $\langle s', \alpha_n \rangle$ to $\langle s'', \alpha' \rangle$ corresponding to a state-transition. Now given that the state $\langle s', v' \rangle$ appears on a partially constructed run of G at time t such that $[x \mapsto t]\, v' \in \alpha_1$, we can find time t' such that the run can be extended by a transition from s' to s'' at time t'. Also $[\pi(\langle s', s'' \rangle) \mapsto 0](v' + t' - t) \in \alpha'$.

Now assume that the algorithm labels $\langle s, [v] \rangle$ with ψ. There exists an F_p-*fair* path in R(G, ϕ) of the form

$$\langle s, [[x \mapsto 0]\, v]^* \rangle \to v_1 \to v_2 \cdots \to v_n \to v_{n+1} \cdots$$

such that for $i < n$, v_i is labeled with ϕ_1, and if $v_n = \langle s_n, [v_n]^* \rangle$ then $v_n \models x \sim c$, v_n is labeled with ϕ_2, and either $[v_n]^*$ is a boundary class or v_n is labeled with ϕ_1.

Let $i_1, i_2, ...$ be the (infinite) sequence of integers such that the edge v_{i_j} to v_{i_j+1} corresponds to a state-transition. Since the edges representing passage of time do not form loops such an infinite sequence exists (recall that the end-class does not have a successor and, hence, no outgoing edge corresponding to the passage of time).

Let $s_0 = s$, and $v_0 = v$. Using the observation mentioned above, we can write $v_{i_j+1} = \langle s_j, [[x \mapsto t_j]\, v_j]^* \rangle$, $j \geqslant 1$, such that the sequence $\{\langle s_j, v_j, t_j \rangle : j \geqslant 0\}$ satisfies all the conditions to be qualified as $\langle s, v \rangle$-run of G, except possibly the progress condition. We show that we can always choose t_j's so that the sequence satisfies progress condition. Suppose there exists a clock y which gets reset infinitely often and infinitely many times satisfies $y = 1$; then the time sequence has to progress. The only problematic case is when after a certain transition point k, for each clock y, either it stays greater than c_y all the time, or it gets reset infinitely often without ever assuming the value 1. In this case a converging sequence of time values is possible. But in such a case, after the kth transition point, the only atomic conditions that are true are of the form $y < 1$ or $y > c_y$. It is easy to show that we can choose a progressing sequence of time values maintaining the truth of all enabling conditions along the path.

By an argument symmetric to the first part of the proof it can shown that this run satisfies ψ. ∎

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, ϕ). *Then label all the augmented regions with the subformulas of* ϕ *using the labeling procedure. The timed graph* G *satisfies the TCTL-specification* ϕ *iff* $\langle s_{\text{init}}, [v_{\text{init}}]^* \rangle$ *is labelled with* ϕ.

5.4. *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, let us consider how to effectively represent the equivalence classes.

We can represent an equivalence class $[v]^*$ of $\Gamma^*(G)$ induced by $\cong^*$ by a triple of arrays $\langle \alpha, \beta, \gamma \rangle$ as follows:

The array α is a C*-indexed array associating with each clock $y \in C^*$ one of the intervals from

$$\{[0, 0], (0, 1), [1, 1], ..., (c_y - 1, c_y), [c_y, c_y], (c_y, \infty)\}.$$

The array α represents a clock assignment v iff for each clock $y \in C^*$, $v(y) \in \alpha(y)$.

Let C^*_α be the set of clocks y such that $\alpha(y)$ is of the form $(i, i+1)$ for some $i \leqslant c_y$. Thus C^*_α is the set of clocks with nonzero fractional part. The array $\beta : C^*_\alpha \to \{1..|C^*_\alpha|\}$ is a permutation of C^*_α. It gives the ordering of the fractional parts of the clocks in C^*_α with respect to $\leqslant$. The array β represents a clock assignment v iff for each pair $y, z \in C^*_\alpha$, if $\beta(y) < \beta(z)$ then $fract(v(y)) \leqslant fract(v(z))$.

The array γ is a boolean C^*_α-indexed array, and is used to specify which clocks in C^*_α have the same fractional parts. For each clock y, $\gamma(y)$ tells whether or not the fractional part of $v(y)$ equals the fractional part of its β-predecessor. The array γ represents a clock assignment v iff for each $y \in C^*_\alpha$, $\gamma(y)$ equals 0 iff there is a clock $z \in C^*_\alpha$ such that $\beta(z) = \beta(y) + 1$ and $fract(v(y))$ equals $fract(v(z))$.

Thus α encodes the integral parts of the clock assignments, and β together with γ encodes the ordering of their fractional parts. The triple $\langle \alpha, \beta, \gamma \rangle$ represents the set of clock assignments v such that each of α, β, and γ represent v according to the above convention. It is easy to see that the sets represented by these triples are in fact the equivalence classes of $\cong^*$, and every equivalence class is represented by some triple. We use this representation to prove the following lemma:

LEMMA (Number of Equivalence Classes). *The number of equivalence classes of $\Gamma^*(G)$ induced by $\cong^*$ is bounded by* $|C^*|! \cdot 2^{|C^*|} \cdot \prod_{y \in C^*} (2 \cdot c_y + 2)$.

Proof. The number of equivalence classes is bounded by the number of triples $\langle \alpha, \beta, \gamma \rangle$ of the desired form. The number of ways to choose α is $\prod_{y \in C^*} (2 \cdot c_y + 2)$. For a given α, the number of ways to choose β is bounded by the number of permutations over C^*_α, which is bounded by $|C^*|!$, and the number of ways to choose γ is bounded by the number of boolean arrays over C^*_α, which is bounded by $2^{|C^*|}$. ∎

Now from the definition of the region graph, and from the above lemma, it follows that

$$|V^*| = O\left[c(\phi) \cdot |S| \cdot |C|! \cdot \prod_{y \in C} c_y\right].$$

The first clause in the definition of E^* contributes at most one edge for every vertex, and the second clause contributes at most two edges for an edge in E and an equivalence class of $\Gamma^*(G)$. Hence,

$$|E^*| = O\left[c(\phi) \cdot (|S| + |E|) \cdot |C|! \cdot \prod_{y \in C} c_y\right].$$

Thus the size of the region graph is (i) exponential in the number of clocks, (ii) exponential in the length of timing constraints assuming binary encoding for the constants, (iii) linear in the size of the node-transition graph, (iv) linear in the number of operators in ϕ, and (v) exponential in the length of the subscripts in ϕ.

THEOREM (Model-Checking for TCTL). *Given a timed graph* G *and a TCTL-formula ϕ, there is a decision procedure for checking whether or not* G *satisfies ϕ which runs in time* $O[c(\phi) \cdot |\phi| \cdot (|S| + |E|) \cdot |C|! \cdot \prod_{y \in C} c_y]$.

Proof. First construct the region graph $R(G, \phi) = \langle V^*, E^* \rangle$. Using the above mentioned representation the successor class of any class can be computed in time $O[|C^*|]$. Hence, $R(G, \phi)$ can be constructed in time $O[|V^*| + |E^*|]$. 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 $O[(|V^*| + |E^*|) \cdot |C^*|]$, assuming they are already marked with the subformulas of ψ, using the labeling algorithm for CTL^F (12). So the labeling algorithm takes time $O[|\phi| \cdot |C^*| \cdot (|V^*| + |E^*|)]$. The complexity follows from the bounds on the sizes of V^* and E^*. ▮

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

COROLLARY (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 satisfiable by a timed graph.

5.5. *Model-Checking is PSPACE-Complete*

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 (22). We reduce this problem to TCTL model-checking.

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

We construct a timed graph G as shown in Fig. 4. The set of clocks C is $\{x, x_1, ..., x_n\}$. The start node is s_0. The enabling condition b 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 at the node s_{n+1} and false everywhere else.

For every $\langle s_0, \nu_{\mathrm{init}} \rangle$-run, the node s_i is reached at time i for $1 \leqslant i \leqslant n$. Furthermore, $\langle s_n, \nu, n \rangle$ appears on the run, where $\nu(x) = n$, and for $1 \leqslant i \leqslant n$, $\nu(x_i)$ equals n or $n - i$ depending on which of the two paths

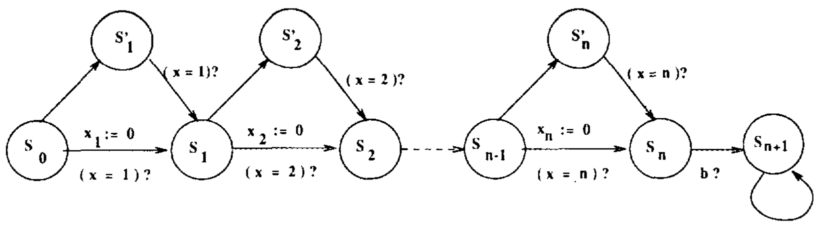

FIG. 4. Timed graph encoding QBF.

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 $\langle s_{n+1}, v+1, n+1 \rangle$ appears on this path if β is true with respect to the truth assignment I defined as: if $v(x_i) = n - i$ then $I(x_i) = \textbf{true}$ else $I(x_i) = \textbf{false}$. Let ϕ be the TCTL-formula, $Q_1 \diamond_{=1} . Q_2 \diamond_{=1} \cdots Q_n \diamond_{=1} . \exists \diamond_{=1} p$.

Now, it is a straightforward exercise to show that G $\models \phi$ iff α is true. Note that the graph G has $O[n]$ nodes and edges, $O[n]$ clocks, and the length of its timing constraints is $O[n \cdot log\, n]$. The lemma follows. ∎

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 trick involved is fairly standard.

The main procedure of the algorithm is a recursive procedure *label*(v, ψ), which returns *true* if v should be labeled with ψ else returns *false*. First, note that every vertex can be represented in space $O[l]$, and its outgoing edges can be computed using the same amount of space. Let n be the maximum depth of the nesting of the path-quantifiers in ψ. We claim that a nondeterministic 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 $\psi \in AP$, $\psi = \textbf{false}$, 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, ϕ_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 $\psi = \forall(\phi_1 \mathscr{U}_{\sim c} \phi_2)$. The negation of ψ can be written using existential path-quantifiers. For instance, the following equivalence holds:

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

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

By Savitch's theorem the deterministic version can be implemented in space polynomial in $[|\phi| \cdot l]$. ▌

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

We have presented a theory for modeling, specifying, and verifying finite-state real-time systems. The main contribution of the paper is working out the mathematics of temporal reasoning in the dense-time model. 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 communication protocols, asynchronous circuits, and real-time control systems. In this section we discuss the complexity of our method in comparison with others, some extensions, and some of the more recent results.

Complexity of Reasoning about Real-Time

Let us analyze the increase in the complexity of temporal reasoning due to the introduction of real-time in temporal logics. Recall that 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 state-transition graph. In TCTL, adding time constants in the formulas contributes another multiplicative factor equal to the largest constant in the formula, while adding clocks to the state-transition graph contributes a factor equal to the product of the constants in the timing constraints and the factorial of the number of clocks. Thus one has to pay a cost of an extra exponential for introducing real-time. This blow-up by an exponential is observed for other real-time formalisms also. For instance, the satisfiability problem for the linear-time temporal logic PTL is PSPACE-complete, whereas the satisfiability problem for its real-time extension TPTL is EXPSPACE-complete (6). The satisfiability problem for CTL is hard for deterministic exponential time, but the satisfiability problem for its real-time extension RTCTL is hard for deterministic doubly-exponential time (18).

We should also compare the cost of real-time reasoning in the discrete models to that in the dense-time model. We claim that the added cost of the dense-time model over discrete models is small. To be specific let us consider the model-checking problem for TCTL in the discrete-time model. That is, we keep the syntax of TCTL-formulas and the timed graphs unchanged, but change the time domain from R to N (or, alternatively, require $t_{i+1} = t_i + 1$ for all $i \geqslant 0$, in the definition of the run). The problem is PSPACE-complete even in this new setting. We can modify our algorithm so that its worst-case time complexity is $O[|\phi| \cdot (|S| + |E|) \cdot \prod_{x \in C} c_x]$. Thus, the only extra cost of choosing the domain R is the multiplicative factor of $|C|!$.

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 now a timed graph is augmented with a set of fairness constraints $F \subseteq 2^S$. An $\langle s, v \rangle$-run $\{\langle s_i, v_i, t_i \rangle : i \geqslant 0\}$ 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 the F-fair runs of G.

For model-checking, we construct the region graph R(G, ϕ) as before. The fairness family F' for R(G, ϕ) is obtained by replacing each α in F by the set $\{\langle s, [v]^* \rangle : s \in \alpha\}$. The labeling algorithm is as before, with the modification that while checking for the path-formulas, we restrict attention to $(F' \cup F_p)$-fair paths in R(G, ϕ). The complexity of the algorithm increases by a multiplicative factor equal to the cardinality of F.

TCTL *with Freeze Quantifiers*

For writing real-time specifications the linear-time logic TPTL (6) uses a syntax different from the usual bounded temporal operators. In this logic the bounded response property that "every p-state is followed by some q-state within 5 time units," is written as

$$\Box x.(p \rightarrow \Diamond y.(q \wedge y \leqslant x + 5)).$$

The *time quantifier* "$x.$" binds the associated variable x to the "current" time: $x.\phi(x)$ holds at time t iff $\phi(t)$ does. Read the above formula as "in every state with time x, if p holds then there is a later state with time y such that both q and $(y \leqslant x+5)$ hold". A similar extension of the syntax is possible for TCTL also, and has been studied in (2); let us call this extension TCTL$_q$. The syntax of TCTL$_q$ uses the time quantifiers which allow references to the times of states, and admits the addition of timing constraints; that is, atomic formulas that relate the times of different states. As timing constraints, we permit comparisons of clock values, possibly with addition of constants. The formulas of TCTL$_q$ are built from propositions and timing constraints by logical connectives, CTL temporal operators, and time quantifiers. Note that the logic TCTL$_q$ is strictly more expressive than TCTL. 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." No formula of TCTL specifies this property. The model-checking algorithm can be modified to handle formulas of TCTL$_q$ (2). This is done by introducing additional clocks, one per each temporal operator in the TCTL$_q$-formula, while constructing the region graph. Recall that, in contrast, only one additional clock suffices to check any TCTL-formula.

Recent Results

Since we presented our results in the "5th IEEE Symposium on Logic in Computer Science" in June 1990, researchers have solved many other problems in the context of reasoning about real-time systems modeled as timed graphs. We briefly survey these results.

Alur *et al.* have developed an algorithm for checking specifications in the linear-time logic MITL against a timed graph (5). The logic MITL extends the syntax of the linear-time temporal logic PTL by allowing subscripts on the temporal operators. The subscript on a temporal operator may be any *nonsingular* interval of R with integer boundaries. It is interesting that allowing singular intervals as subscripts, that is, operators such as $\Diamond_{=3}$, makes the model-checking problem undecidable!

Courcoubetis and Yannakakis use timed graphs to solve certain minimum and maximum delay problems for real-time systems (13). For instance, they show how to compute the earliest and the latest time a target state can appear along the runs of a timed graph given an initial state and a clock assignment.

The research on timed graphs has been linked up to the research on real-time process algebras also. Sifakis and co-workers show how to translate a term of the real-time process algebra ATP to a timed graph (31).

Another extension of the work reported here is incorporation of probabilistic information in a timed graph. We add probabilities to our model by associating fixed distributions with the delays. Now we can express constraints like "the delay between the request and the response is distributed uniformly between 2 to 4 seconds." This extension makes a timed graph a *generalized semi-Markov process* (GSMP). While defining the semantics of TCTL-formulas in this probabilistic model, the existential quantifier is interpreted as "with positive probability," and the universal quantifier means "with probability 1." In (3), we present an algorithm for checking whether a GSMP satisfies its TCTL-specification. That algorithm combines model-checking algorithm of this paper with model-checking for discrete-time Markov chains.

To apply our algorithm to verify any practical system, we must devise ways to cope with the PSPACE complexity of the problem. Recently heuristic to implement CTL model-checking without explicitly enumerating all the states have been proposed. For instance, Burch *et al.* (11) propose the use of binary decision diagrams to represent large state sets symbolically. Henzinger *et al.* (21) have shown how to extend these symbolic methods to model-checking of TCTL formulas.

ACKNOWLEDGMENTS

Pierre Wolper suggested looking at real-time in a branching-time, instead of linear-time, model. We thank him as well as Thomas Henzinger, Harry Lewis, Zohar Manna, and Moshe Vardi for helpful discussions.

RECEIVED September 24, 1990; FINAL MANUSCRIPT RECEIVED February 3, 1992

REFERENCES

1. S. AGGARWAL AND R. KURSHAN (1983). Modeling elapsed time in protocol specification, *in* "Protocol Specification, Testing, and Verification" (H. Rudin and C. West, Eds.), Vol. III, pp. 51–62.
2. R. ALUR (1991). "Techniques for Automatic Verification of Real-Time Systems," Ph.D. thesis, Stanford University.
3. R. ALUR, C. COURCOUBETIS, AND D. DILL (1991). Model-checking for probabilistic real-time systems, *in* "Automata, Languages and Programming: Proceedings of the 18th ICALP," Lecture Notes in Computer Science, Vol. 510, pp. 115–136, Springer-Verlag, Berlin/New York.
4. R. ALUR AND D. DILL (1990). Automata for modeling real-time systems, *in* "Automata, Languages and Programming: Proceedings of the 17th ICALP," Lecture Notes in Computer Science, Vol. 443, pp. 322–335, Springer-Verlag, Berlin/New York.
5. R. ALUR, T. FEDER, AND T. HENZINGER (1991). The benefits of relaxing punctuality, *in* "Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing," pp. 139–152.

6. R. Alur and T. Henzinger (1989). A really temporal logic, *in* "Proceedings of the 30th IEEE Symposium on Foundations of Computer Science," pp. 164–169.
7. R. Alur and T. Henzinger (1992). Logics and models of real time: A survey, *in* "Proceedings of REX workshop 'Real-Time: Theory in Practice,'" Lecture Notes in Computer Science, Vol. 600, pp. 74–106, Springer-Verlag, Berlin/New York.
8. A. Bernstein and P. Harter (1981). Proving real-time properties of programs with temporal logic, *in* "Proceedings of the Eighth ACM Symposium on Operating System Principles," pp. 164–176.
9. J. Brzozowski and C. Seger (1991). Advances in asynchronous circuit theory. Part II. Bounded inertial delay models, MOS circuits, design techniques, *in* "Bulletin of the European Association for Theoretical Computer Science," Vol. 43, pp. 199–263.
10. J. Burch (1989). Combining CTL, trace theory and timing models, *in* "Automatic Verification Methods for Finite State Systems: Proceedings of the First CAV," Lecture Notes in Computer Science, Vol. 407, pp. 197–212, Springer-Verlag, Berlin/New York.
11. J. Burch, E. Clarke, D. Dill, L. Hwang, and K. L. McMillan (1990). Symbolic model checking: 10^{20} states and beyond, *in* "Proceedings of the Fifth IEEE Symposium on Logic in Computer Science," pp. 428–439.
12. E. Clarke, E. A. Emerson, and A. P. Sistla (1986). Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Trans. Programming Languages Systems* **8**, No. 2, 244–263.
13. C. Courcoubetis and M. Yannakakis (1991). Minimum and maximum delay problems in real-time systems, *in* "Proceedings of the Third Workshop on Computer-Aided Verification, Aalborg University, Denmark," Lecture Notes in Computer Science, Vol. 575, pp. 399–409, Springer-Verlag, Berlin/New York.
14. D. Dill (1989). Timing assumptions and verification of finite-state concurrent systems, *in* "Automatic Verification Methods for Finite State Systems" (J. Sifakis, Ed.), Lecture Notes in Computer Science, Vol. 407, Springer-Verlag, Berlin/New York.
15. E. A. Emerson (1983). Alternative semantics for temporal logics, *Theoret. Comput. Sci.* **26**, 121–130 (1983).
16. E. A. Emerson and E. M. Clarke (1982). Using branching-time temporal logic to synthesize synchronization skeletons, *Sci. Comput. Programming* **2**, 241–266.
17. E. A. Emerson and J. Halpern (1982). Decision procedures and expressiveness in the temporal logic of branching time, *in* "Proceedings of the 9th ACM Symposium on Principles of Programming Languages," pp. 169–179.
18. E. A. Emerson, A. Mok, A. P. Sistla, and J. Srinivasan (1989). Quantitative temporal reasoning, presented at the First Workshop on Computer-aided Verification, Grenoble, France.
19. E. Harel, O. Lichtenstein, and A. Pnueli (1990). Explicit-clock temporal logic, *in* "Proceedings of the Fifth IEEE Symposium on Logic in Computer Science," pp. 402–413.
20. T. Henzinger, Z. Manna, and A. Pnueli (1991). Temporal proof methodologies for real-time systems, *in* "Proceedings of the 18th ACM Symposium on Principles of Programming Languages," pp. 353–366.
21. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine (1992). Symbolic model-checking for real-time systems, *in* "Proceedings of the Seventh IEEE Symposium on Logic in Computer Science," pp. 394–406.
22. J. Hopcroft and J. Ullman (1979). "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, MA.
23. F. Jahanian and A. Mok (1986), Safety analysis of timing properties in real-time systems, *IEEE Trans. Software Engr.* **SE-12**, No. 9, 890–904.
24. F. Jahanian and A. Mok (1987). A graph-theoretic approach for timing analysis and its implementation, *IEEE Trans. Comput.* **C-36**, 961–975.

25. R. Koymans (1990). Specifying real-time properties with metric temporal logic, *J. Real-Time Systems* **2**, 255–299.
26. D. Lehman, A. Pnueli, and J. Stavi (1982). Impartiality, justice, and fairness: The ethics of concurrent termination, *in* "Automata, Languages and Programming: Proceedings of the Ninth ICALP," Lecture Notes in Computer Science, Vol. 115, pp. 264–277, Springer-Verlag, Berlin/New York.
27. H. Lewis (1989), "Finite-State Analysis of Asynchronous Circuits with Bounded Temporal Uncertainty," Technical Report TR-15-89, Harvard University.
28. H. Lewis (1990). A logic of concrete time intervals, *in* "Proceedings of the Fifth IEEE Symposium on Logic in Computer Science," pp. 380–389.
29. N. Lynch and H. Attiya (1990). Using mappings to prove timing properties, *in* "Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing," pp. 265–280.
30. Z. Manna and A. Pnueli (1989). The anchored version of the temporal framework, *in* "Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency," Lecture Notes in Computer Science, Vol. 354, Springer-Verlag, Berlin/New York.
31. X. Nicollin, J. Sifakis, and S. Yovine (1991). From ATP to timed graphs and hybrid systems, *in* "Proceedings of REX Workshop 'Real-Time: Theory in Practice,'" Lecture Notes in Computer Science, Vol. 600, pp. 549–572, Springer-Verlag, Berlin/New York.
32. J. Ostroff (1990). "Temporal Logic of Real-Time Systems," Research Studies Press.
33. S. Owicki and L. Lamport (1982). Proving liveness properties of concurrent programs, *ACM Trans. Programming Languages Systems* **4**, No. 3, 455–595.
34. A. Pnueli (1977). The temporal logic of programs, *in* "Proceedings of the 18th IEEE Symposium on Foundations of Computer Science," pp. 46–77.
35. H. Rogers (1967). "Theory of Recursive Functions and Effective Computability," McGraw–Hill, New York.