On Model Checking for Real-Time Properties with Durations*

Ahmed Bouajjani[†§] Rachid Echahed[‡§] Joseph Sifakis[†§]

Abstract

We address the verification problem of real-time properties involving duration constraints (predicates). The duration of a state property, along an interval of a computation sequence of a real-time system, is the time the property is true. In particular, the global time spent in such an interval is the duration of the formula `true`. We extend the real-time logic TCTL to a duration logic called SDTL where duration constraints can be expressed. Then, we investigate the problem of the verification of SDTL formulas w.r.t. a class of timed models of reactive systems. We propose new model checking procedures for the most significant properties expressible in SDTL including eventuality and invariance properties. We provide such results in the two cases of discrete and dense time.

1 Introduction

Real-time temporal logics have been introduced recently as extensions of the classical temporal logics like PTL [Pnu77] and CTL [CES83]. These logics allow to express hard real-time specifications of reactive systems. A typical example of such properties is the *bounded response* property asserting for example that during the execution of the system, whenever a state formula P_0 holds, eventually a state formula P_1 will hold before 5 time units.

Several real-time logics have been proposed in the last few years. These proposals consider either a discrete or a dense time domain as well as different models of real-time systems [AH89], [AH90], [ACD90], [HLP90], [HNSY92] ; see also [AH91] for a survey.

Recently, the interesting concept of *duration* have been introduced in the Calculus of Durations (CoD) [CHR91]. It allows to reason about the durations of state properties (formulas). That is to say, given a finite interval on a run of the system, the duration of some state property in this interval is the time during which the property has been true. Particularly, the global time spent by the system in a run interval is simply the duration of the formula `true`. In this sense, the concept of duration allows to extend the expressiveness of real-time logics. For example we may consider a variant of the bounded response property given above by reasoning about a time which is the duration of some state property instead of the global time. More generally, it is interesting to consider properties involving predicates (constraints) on the durations of several formulas. As an example of such properties, we may consider invariance properties such as :

1. *For any execution interval where the system spends more than 5 time units, the duration of P_0 does not exceed the half of the duration of P_1.*

Other important properties which may be considered are *eventuality* properties. The combination of these properties with invariance properties allows to express generalized *response properties* (always eventually) and *persistence properties* (eventually always). An example of a response property is

2. *From some point in the execution, whenever the duration of P_0 exceeds the half of the duration of P_1, from now on, eventually on any possible execution, the duration of P_2 will be strictly greater than the third of the duration of P_3.*

These properties can be expressed in a natural way in extended real-time logics where duration constraints can be considered. The crucial problem then is to verify automatically that a real-time system satisfies a duration property expressed in such a logic. In this context, the existing results concern only the class of bounded reachability properties which are dual of invariance properties [KPSY92, ACH]. Our aim in this paper is to extend the model checking techniques [QS82, CES83] to the other classes of duration properties, in particular those involving eventualities.

The verification problem of the duration properties has four parameters. The time domain (discrete or dense), the models of the real-time systems, the set of the logical operators and finally, the language of the constraints on the durations. The decidability of this

*Partially supported by the ESPRIT-BRA project REACT.
†VERIMAG, BP53X, 38041 Grenoble cedex, France.
‡LGI-IMAG, BP53X, 38041 Grenoble cedex, France.
§bouajjan@imag.fr, echahed@imag.fr, sifakis@imag.fr

problem depends on the choices we make for these four parameters.

In this paper we consider the two cases of discrete and dense time domain. As models for the real-time systems, we consider *Simple Timed Graphs* which are particular timed graphs according to the semantics defined in [NSY91]. They consist of finite transition systems where any transition between two control nodes q and q' is labelled by an interval of time. Such an interval defines a lower and an upper bound on the time spent by the system at q before reaching q'. The system is allowed to stay at any node as long as there exists an executable transition from this node.

Despite their relative restricted expressiveness w.r.t. general timed graphs, the simple timed graphs can model an important class of real-time systems.

As language of formulas, we define a branching-time logic called SDTL which is an extension of the real-time logic TCTL [ACD90]. In SDTL, we have duration variables that can be associated with state formulas and then used to express constraints on their durations. For example, in the formula $[x : P].\phi$ we associate the state formula P with the duration variable x. This variable denotes in ϕ the duration of P starting from the state where the association has been done. Using this notation, the properties (1) and (2) can be expressed by

1. $[x, y, z : \mathtt{true}, P_0, P_1].\forall\Box((x \geq 5) \Rightarrow (y \leq 2z))$,
2. $[x, y : P_0, P_1]$.
   $\forall\Box((2x \geq y) \Rightarrow [z, s : P_2, P_3].\forall\Diamond(3z > s))$.

The duration constraints we consider are in the general form $e(x_0, \cdots, x_n) \leq \beta$ where β is an integer and $e(x_0, \cdots, x_n)$ is a diverging expression on the duration variables x_i. In case of a discrete time domain we require that e corresponds to a recursive function and in case of a dense time domain we take e to be continuous, time-monotonic (which is a technical notion introduced latter). Examples of such functions are the linear functions such as $2x+y-3z$, but also non linear ones like $x^3 + 2y + 2^z$.

Our decision techniques are essentially based on a forward analysis of the state graph of the system. Such graph, when augmented by the information on the durations is in general infinite.

In a first part of the paper, we consider the case of a discrete time domain (naturals). We show in particular that the verification of eventuality properties is decidable when we consider linear duration constraints. Furthermore, we show that under some syntactic conditions we can handle general duration constraints both in invariance and eventuality properties.

Then, in a second part of the paper we show how the techniques developed in the discrete time case can be extended to the case of dense time. This introduces some restrictions on the occurrences of the duration constraints in the formulas. However, we can prove the decidability of the verification problem for a non trivial class of properties including response properties like (2) and persistence properties.

The paper is organized as follows. In section 2, we give the definition of the Simple Timed Graphs. In section 3, we introduce the Duration Temporal Logic (DTL) and define SDTL as a fragment of this logic. In section 4, we provide model checking techniques for the discrete time case. Section 5 is devoted to the model checking in the dense time case. We conclude the paper by mentioning some extensions of our results and by presenting some future work directions.

2 Models for Real-Time Systems

We introduce a class of models for real-time systems called Simple Timed Graphs. These models are a special case of timed graphs according to the definition given in [NSY91]. They are finite transition systems where any transition, say from node q to node q', is labelled by an interval of time. Such an interval defines a lower and an upper bound on the time spent by the system at node q, so that the considered transition may be executed.

Simple timed graphs correspond to timed graphs with a single clock which is reset at each transition. Despite this relative restriction w.r.t. general timed graphs, the simple timed graphs represent the most commonly used models for real-time systems.

Simple timed graphs are defined by considering a time domain $\mathcal{D}$ which can be either discrete or dense. Without loss of generality, we take $\mathcal{D} = \mathbb{N}$ (naturals) in the discrete case and $\mathcal{D} = \mathbb{R}^+$ (positive reals) in the dense case.

We call *transition constraint* any interval (n, m) where n and m are rationals and the parentheses belong to $\{[,]\}$. For example, we denote by $[n, m[$ a left closed right open interval. We denote by $\mathcal{I}$ the set of intervals.

Definition 2.1 *(Simple Timed Graphs) Let $\mathcal{P}$ be a set of atomic propositions. A Simple Timed Graph (STG for short) is a tuple $M = (Q, \Pi, R)$ where Q is a finite set of* control nodes, *$\Pi : Q \rightarrow 2^{\mathcal{P}}$ is a labelling mapping and $R \subseteq Q \times \mathcal{I} \times Q$ is a* time constrained transition relation.

A computation sequence of a model M may stay at a node q by letting time progress as long as at least one transition from q is executable. Thus, a computation sequence can be seen as an infinite sequence composed of two kinds of alterning transitions: *time progression transitions* and *moves between nodes*. The latter are considered to be instantaneous. We denote by D_q the interval of time the system may stay at node q, that is, the interval $[0, B_q) \cap \mathcal{D}$ where B_q is $max\{n \mid \exists (q, (m, n), q') \in R\}$. D_q is right closed iff B_q corresponds to a right closed interval. In the discrete time case D_q consists of the naturals less (or equal) than B_q. A *state* of a system $M = (Q, \Pi, R)$ is a pair $\langle q, t\rangle$ where $q \in Q$ and $t \in D_q$. We denote by S_M the set of the states in M, i.e. $S_M = \{\langle q, t\rangle \mid q \in Q$ and $t \in D_q\}$. For $s = \langle q, t\rangle$, we denote by ν and τ the two following projections on states: $\nu(s) = q$ and $\tau(s) = t$.

Definition 2.2 *(Transition Relations, Abstract State Graph) Let $M = (Q, \Pi, R)$ be an STG. We define the following transition relations.*

- *Time progression transitions* ($\xrightarrow{\delta}$)
  $\forall q \in Q.\ \forall t, t' \in D_q.\ \forall \delta \in \mathcal{D}.$
  $\langle q, t\rangle \xrightarrow{\delta} \langle q, t'\rangle$ *iff* $t' = t + \delta$.
- *Moves between nodes* ($\rhd$)
  $\forall q_1, q_2 \in Q.\ \forall t \in D_{q_1}.$
  $\langle q_1, t\rangle \rhd \langle q_2, 0\rangle$ *iff* $\exists (q_1, I, q_2) \in R.\ t \in I.$

For every $\delta \in \mathcal{D}$, let $\overset{\delta}{\Longrightarrow} = \xrightarrow{\delta} \circ \rhd$. We define the abstract state graph to be the graph $ASG_M = (S_M, \bigcup_{\delta \in \mathcal{D}} \overset{\delta}{\Longrightarrow})$.

We call computation sequence any infinite path in the abstract state graph where the time diverges. The formal definition is given below.

Definition 2.3 *(Computation Sequence) Let M be an STG. A computation sequence from a state $\langle q_0, t_0\rangle$ is an infinite path $\langle q_0, t_0\rangle \overset{\delta_0}{\Longrightarrow} \langle q_1, 0\rangle \overset{\delta_1}{\Longrightarrow} \cdots \langle q_i, 0\rangle \overset{\delta_i}{\Longrightarrow} \cdots$ in ASG_M such that $\sum \delta_i = \infty$.*
We denote by $\Sigma_{\langle q,t\rangle}$ the set of computation sequences starting from the state $\langle q, t\rangle$.
A digital computation sequence is a computation sequence where every duration δ_i is a multiple of some given rational u.

3 Duration Temporal Logic

We define in this section the logic DTL (Duration Temporal Logic) and its sublogic SDTL (Simple DTL). The logic DTL is a branching-time temporal logic where duration constraints can be expressed. It can be seen as an extension of the real-time logic TCTL [Alu91]. Informally, the duration of a formula φ along some computation path is the time during which φ holds. In DTL, duration variables are explicitly associated with formulas and then used to assert constraints on the duration of these formulas. For example, in the formula $\varphi_1 = [x : \varphi_2].\exists\Diamond(-2x \leq -5)$, the duration variable x is associated with φ_2. The formula φ_1 holds at some state s if there exists a computation path from s to a state s' such that the duration of φ_2 is greater than $5/2$. We give hereafter the syntax and the semantics of DTL.

Let $\mathcal{P}$ be a set of atomic propositions and $\mathcal{X}$be a set of variables. The language of DTL formulas $\mathcal{F}$ is a sub-language of the language $\mathcal{F}^*$ given by the following grammar :

- $\varphi ::= P \in \mathcal{P} \mid \xi \mid \neg\varphi \mid \varphi \vee \varphi \mid [x : \varphi].\varphi \mid \exists\psi$
- $\psi ::= \varphi \mathcal{U} \varphi \mid \neg\psi$
- $\xi ::= e(x_0, \cdots, x_n) \leq \beta$

where $P \in \mathcal{P}, n \in I\!N, \beta \in \mathbb{Z}, x_i \in \mathcal{X}$ and e denotes a function from $\mathcal{D}^{n+1}$ to $\mathcal{D}$. Elements in ξ are called duration constraints. We will write e instead of the expression $e(x_0, \cdots, x_n)$ when it is clear from the context.

In a formula $[x : \varphi_1].\varphi_2$, "$[x : \varphi_1]$" is seen as a quantification that associates φ_1 with x and binds the variable x in φ_2. Thus, every variable in a formula φ in $\mathcal{F}^*$ is *bound* or *free*. A formula φ is *closed* if all the duration variables occurring in it are bound. φ is *open* if it is not closed. The set of DTL formulas $\mathcal{F}$ consists of formulas φ such that duration variables are associated with closed formulas.

We use in the sequel the following standard abbreviations and notations: the boolean connectives $\wedge$ and $\Rightarrow$, $\texttt{true} = P \vee \neg P$, $\texttt{false} = \neg\texttt{true}$, $e < \beta = \neg(-e \leq -\beta)$, the universal path quantification $\forall\psi = \neg\exists\neg\psi$ and the classical temporal operator $\Diamond\varphi = \texttt{true}\ \mathcal{U}\varphi$, $\Box\varphi = \neg\Diamond\neg\varphi$. Notice that we have $\forall\Diamond\varphi = \neg\exists\Box\neg\varphi$ and $\forall\Box\varphi = \neg\exists\Diamond\neg\varphi$. We write $[\vec{x} : \vec{\varphi}].\phi$ or $[x_1, \cdots, x_n : \varphi_1, \cdots, \varphi_n].\phi$ to indicate several consecutive bindings such as $[x_1 : \varphi_1]. \cdots .[x_n : \varphi_n].\phi$.

The satisfaction relation of DTL relates states to formulas. In the case of open formulas, the satisfaction relation is defined w.r.t. an *environment* and a *formula association*.

An environment E yields an interpretation of duration constraints. It is defined as a mapping $E : \mathcal{X} \to \mathcal{D}$. We denote by $\mathcal{E}$ the set of all possible environments. The "empty" environment is denoted by ϵ ($\forall x \in \mathcal{X}, \epsilon(x) = 0$). Let ξ be the constraint $e(x_0, \cdots, x_n) \leq \beta$. We write $E(e(x_0, \cdots, x_n))$ for the valuation of $e(x_0, \cdots, x_n)$ in the environment E. Along a computation sequence, the environment changes according to the satisfiability, at the visited states, of the formulas associated with duration variables. A *formula association* is a function $\gamma : \mathcal{X} \rightarrow \mathcal{F}$ such that $\gamma(x)$ denotes a formula associated with the variable x. We denote by Γ the set of formula associations. A formula association γ is called *atomic* if it associates atomic propositions with variables.

Let $M = (Q, \Pi, R)$ be an STG. The satisfaction relation $\models$ is a subset of $S_M \times \mathcal{E} \times \Gamma \times \mathcal{F}$. We write $M : s \models_{(E,\gamma)} \varphi$ instead of $(s, E, \gamma, \varphi) \in \models$, that is to say φ holds at the state s under the environment E and the formula association γ.

Before giving a formal definition of the satisfaction relation, we show how an environment evolves with computation sequences. We also define what means for a formula of the form $\varphi_1 \mathcal{U} \varphi_2$ (or its negation) to be satisfied by a computation sequence σ of M, with an initial environment E and a formula association γ. We express this fact by $M : \sigma \models_{(E,\gamma)} \psi$.

Let σ be $\langle q_0, t_0 \rangle \stackrel{\delta_0}{\Longrightarrow} \cdots \langle q_i, t_i \rangle \stackrel{\delta_i}{\Longrightarrow} \cdots$ where $\forall i > 0$, $t_i = 0$. We define the environment E_i^δ at each state $\langle q_i, \delta \rangle$ in the sequence σ as follows :

$$\begin{aligned} E_0^{t+\delta}(x) &= E(x) + \int_t^{t+\delta} f_x^0(t)dt \\ E_{i+1}^{\delta}(x) &= E_i^{\delta_i}(x) + \int_0^{\delta} f_x^{i+1}(t)dt \end{aligned}$$

where $f_x^i(t) =$ if $\langle q_i, t \rangle \models_{(\epsilon,\gamma)} \gamma(x)$ then 1 else 0.

We define the relation $\models_{(E,\gamma)}$ as follows:

$M : \sigma \models_{(E,\gamma)} \neg\psi$ iff $M : \sigma \not\models_{(E,\gamma)} \psi$
$M : \sigma \models_{(E,\gamma)} \varphi_1 \mathcal{U} \varphi_2$ iff
$\exists n \geq 0.\ \exists \delta \in [t_n, t_n + \delta_n]$ such that :
(i) $\langle q_n, \delta \rangle \models_{(E_n^\delta,\gamma)} \varphi_2$,
(ii) $\forall \tau.\ t_n \leq \tau < \delta.\ \langle q_n, \tau \rangle \models_{(E_n^\tau,\gamma)} \varphi_1$ and
(iii) $\forall i.\ 0 \leq i \leq n-1.\ \forall \tau.\ t_i \leq \tau \leq \delta_i.$
$\langle q_i, \tau \rangle \models_{(E_i^\tau,\gamma)} \varphi_1$.

Notation: We write $E[x \leftarrow v]$ (resp. $\gamma[x \leftarrow \varphi]$) to denote the environment (resp. the formula association) that maps v (resp. φ) to x and coincides with E (resp. with γ) on the other variables.

Definition 3.1 *(Satisfiability) Let $M = (Q, \Pi, R)$ be an STG, γ a formula association, E an environment, s a state in S_M and φ a DTL formula. The relation $M : s \models_{(E,\gamma)} \varphi$ is inductively defined by :*

$M : s \models_{(E,\gamma)} P$ *iff* $P \in \Pi(\nu(s))$
$M : s \models_{(E,\gamma)} e \leq \beta$ *iff* $E(e) \leq \beta$ *is true.*
$M : s \models_{(E,\gamma)} \neg\varphi$ *iff* $M : s \not\models_{(E,\gamma)} \varphi$.
$M : s \models_{(E,\gamma)} \varphi_1 \vee \varphi_2$ *iff*
$(M : s \models_{(E,\gamma)} \varphi_1)$ *or* $(M : s \models_{(E,\gamma)} \varphi_2)$.
$M : s \models_{(E,\gamma)} [x : \varphi_1].\varphi_2$ *iff* $M : s \models_{(E',\gamma')} \varphi_2$
where $E' = E[x \leftarrow 0]$ *and* $\gamma' = \gamma[x \leftarrow \varphi_1]$.
$M : s \models_{(E,\gamma)} \exists\psi$ *iff* $\exists \sigma \in \Sigma_s.\ M : \sigma \models_{(E,\gamma)} \psi$.

In the sequel we focus on a fragment of DTL, called SDTL (Simple DTL), which is an extension of the initial version of TCTL defined in [ACD90]. In this version of TCTL the time constraints are expressed using bounded temporal operators like $\forall\Diamond_{\leq 5}$. This corresponds to a restriction on the use of the duration variables which leads to the definition of the logic SDTL.

Definition 3.2 *(SDTL formulas)*
An SDTL formula is any closed DTL formula such that between any occurrence of a variable x in some duration constraint and its binding, i.e. $[x : \varphi]$, there exists one and only one path quantifier.

Example: (of an SDTL formula)
The formula below expresses a *bounded response property* which says that at any state where the duration of P_0 is greater than 1, the system eventually reaches within 5 time units some state where the durations of P_1, say y, and P_2, say z, satisfy the constraint $3y - z \leq -1$.

$[u : P_0].$
$\forall\Box((-u \leq -1) \Rightarrow$
$[x, y, z : true, P_1, P_2].\forall\Diamond(x \leq 5) \wedge (3y - z \leq -1))$

We introduce now a normal form for SDTL formulas which is useful for the verification techniques given in the next sections.

Definition 3.3 *(Disjunctive Normal Form) The formulas in disjunctive normal forms (DNFfor short) are defined by the following grammar :*

- $\varphi ::= \bigvee \bigwedge \phi$
- $\phi ::= P \in \mathcal{P} \mid \neg\phi \mid \xi \mid [\vec{x} : \vec{\varphi}].\psi \mid \psi$
- $\psi ::= \exists\Box\varphi \mid \exists\varphi\mathcal{U}\zeta$
- $\zeta ::= \bigwedge \phi$
- $\xi ::= e(x_0, \cdots, x_n) \leq \beta$

As an example, the disjunctive normal form of the formula given in the previous example is:

$\neg[u : P_0].$
$\exists\Diamond((-u \leq -1) \wedge$
$[x, y, z : true, P_1, P_2].$
$\exists\Box(\neg(x \leq 5) \vee \neg(3y - z \leq -1)))$

Proposition 3.1 *Any SDTL formula has a disjunctive normal form.*

The proof of this proposition follows from the standard properties of boolean connectives and the following facts :

$$\exists\varphi_1\mathcal{U}(\varphi_2 \vee \varphi_3) = (\exists\varphi_1\mathcal{U}\varphi_2) \vee (\exists\varphi_1\mathcal{U}\varphi_3),$$
$$\exists\neg(\varphi_1\mathcal{U}\varphi_2) = (\exists\Box\neg\varphi_2) \vee (\exists\neg\varphi_2\mathcal{U}(\neg\varphi_1 \wedge \neg\varphi_2)).$$

Notice that SDTL formulas are obtained as a boolean combination of the three kinds of formulas defined below.

Definition 3.4 *(Trajectory Existence, Conditional State Reachability, State Reachability)*
An SDTL formula is a

- trajectory existence formula *if it is of the form:* $[\vec{x} : \vec{\varphi}].\exists\Box\phi$.
- state reachability formula *if it is of the form:* $[\vec{x} : \vec{\varphi}].\exists\Diamond\phi$.
- conditional state reachability formula *if it is of the form:* $[\vec{x} : \vec{\varphi}].\exists\phi_1\mathcal{U}\phi_2$.

Finally, as a consequence of the definition of SDTL formulas, we introduce the notion of top-level duration constraints in formulas of the form $[\vec{x} : \vec{\varphi}].\exists\psi$.

Definition 3.5 *(Top-level constraints in a formula)*
Let ϕ be a closed SDTL formula of the form $[\vec{x} : \vec{\varphi}].\exists\psi$. The top-level constraints of ϕ are the constraints appearing in ψ defined on the variables in $\vec{x}$.

The word top-level is suggested by the definition of SDTL formulas in which variable occurrences and their bindings are separated by only one path quantifier.

4 Model Checking : The Discrete Time Case

In this section we consider the verification problem of SDTL formulas in the case of simple timed graphs with discrete time domain $\mathcal{D} = I\!N$. In the three following subsections, we investigate separately the verification techniques for each of the three classes of SDTL formulas (i.e. state reachability, trajectory existence and conditional state reachability). These techniques are illustrated for formulas which are not nested. Nested formulas are considered in subsection 4.5.

Following the form of the formulas mentioned above, to prove $M : s \models_{(E,\gamma)} \exists\varphi$, it amounts to find a computation sequence σ starting from s such that $M : \sigma \models_{(E,\gamma)} \varphi$. The technique proposed in this paper consists in finding out such a sequence by finite exploration of an infinite tree called *labelled computation tree*.

We assume from now on that all the expressions $e(x_0, \cdots, x_n)$ involved in the duration constraints are *diverging* w.r.t. each variable x_i, that is to say,

$$\forall i.\ 0 \leq i \leq n.\ |\lim_{x_i \to \infty} e(x_0, \cdots, x_n)| = \infty$$

4.1 Labelled Computation Tree

Given a model M, we define SG_M to be the state graph $(S_M, \xrightarrow{1} \cup \rhd)$. Notice that SG_M is a subgraph of ASG_M. Any transition $\langle q, t\rangle \xrightarrow{n} \langle q, t+n\rangle$ in ASG_M is represented in SG_M by the path $\langle q, t\rangle \xrightarrow{1} \langle q, t+1\rangle \cdots \xrightarrow{1} \langle q, t+n\rangle$. A labelled computation tree is defined for a state $s_0 \in S_M$ as an "unfolding" of the graph SG_M starting from s_0.

Definition 4.1 *(Labelled Computation Tree)*
Let M be an STG and SG_M be its state graph. Given a state $s_0 \in S_M$, an environment E_0 and an atomic variable association γ, we define the labelled computation tree $\mathcal{T}(s_0, E_0)$, as follows:

- *the nodes are tuples (s, E, T) where $s \in S_M$, $E \in \mathcal{E}$ and $T \in I\!N$,*
- *the root is $(s_0, E_0, 0)$,*
- *(s', E', T') is a successor of (s, E, T) iff*
  - *either $s \rhd s'$ and $E = E'$ and $T = T'$*
  - *or $s \xrightarrow{1} s'$ and $T' = T + 1$ and $\forall x \in \mathcal{X}.\ E'(x) = E(x) + \Delta(x)$*

where $\Delta(x) =$ if $\gamma(x) \in \Pi(\nu(s))$ then 1 else 0.

Let $\pi = (s_0, E_0, T_0), \cdots, (s_i, E_i, T_i) \cdots$ be a path in $\mathcal{T}(s_0, E_0)$. Informally π represents a computation sequence of M. The sequence $s_\pi = s_0, s_1, \cdots, s_i, \cdots$ represents the consecutive states of this computation sequence, E_i stands for the environment at state s_i and T_i represents the global time elapsed from state s_0 to state s_i. We write E_π to denote the sequence $E_0, E_1, \cdots$ and T_π to denote the sequence $T_0, T_1, \cdots$. We say that π is time divergent if $T_\infty = \infty$.

For any formula ϕ which is a boolean combination of atomic propositions and constraints, we say that a node (s, E, T) satisfies ϕ if and only if $M : s \models_{(E,\gamma)} \phi$.

Definition 4.2 *(Circuits)*
Let $\pi = (s_1, E_1, T_1), \cdots, (s_m, E_m, T_m)$ be a finite path in $\mathcal{T}(s_0, E_0)$. We say that π is a circuit if and only if $s_1 = s_m$. Furthermore, let $\{e_1, \cdots, e_n\}$ be a set of expressions. We say that a circuit π is decreasing (resp. strictly decreasing, constant) for $\{e_1, \cdots, e_n\}$ if $(E_1(e_1), \cdots, E_1(e_n)) \sim (E_m(e_1), \cdots, E_m(e_n))$ where $\sim$

denotes respectively $\leq$, $<$ and $=$ for the usual product ordering. Finally, we say that a circuit π is time strictly increasing (resp. time constant) if and only if $T_m > T_1$ (resp. $T_m = T_1$).

4.2 State Reachability

We tackle in this section the verification problem of state reachability formulas of the form :

$$[\vec{x} : \vec{P}].\exists\Diamond \bigwedge_{i=1}^{n} P_i \wedge (e_i \leq \beta_i)$$

These properties are dual of invariance properties. In the case where the e_i's are linear, i.e., $e_i = \sum_{j=1}^{m} \alpha_j.x_j$ where the α_j's are integers ($\in \mathbb{Z}$), the verification of state reachability formulas may be done using the results in [KPSY92].

We consider now the verification of formulas of the form $[\vec{x} : \vec{P}].\exists\Diamond(P \wedge (e \leq \beta))$ where e is any recursive function. An example of such formula is

$$[x, y, z : P_0, P_1, P_2].\exists\Diamond(x^2 - 3.y + 2^z \leq -5)$$

Theorem 4.1 *(Deciding state reachability for the single general constraint case)*
Let M be an STG, s be a state in S_M and ϕ be an SDTL formula such that $\phi = [\vec{x} : \vec{P}].\exists\Diamond(P \wedge (e \leq \beta))$ where e is recursive. The problem $M : s \models \phi$ is decidable.

Proof: We explore the tree $\mathcal{T}(s, \epsilon)$ from its root, stopping the construction at each node satisfying one of the following conditions :

1. the node satisfies the constraint $e \leq \beta$,
2. a circuit is detected.

The answer is *yes* if either we stop with the first condition or if we have detected a circuit which is strictly decreasing for the expression e and from which P is reachable.

Let us prove the correctness of this procedure: First, the procedure obviously terminates because the tree is finite branching. It is consistent since in case we have detected some strictly decreasing circuit for e, by repeating this circuit, since e is a diverging, eventually $e \leq \beta$ will be satisfied and we can get arbitrarily low values for e in it. As from this circuit we can reach some state s' satisfying the atomic proposition P (this can obviously be verified since the state graph is finite) there exists a number of iterations of the circuit which allows to reach s' with the constraint $e \leq \beta$ satisfied. As $\Diamond(P \wedge (e \leq \beta))$ must be satisfied by a time divergent path, we verify that from state s' there exists some reachable time strictly increasing circuit. Finally the procedure is complete since for any path where $e \leq \beta$ is satisfied there exists a path without the rejected circuits where $e \leq \beta$ is satisfied. $\square$

The theorem 4.1 can be extended easily to consider, in addition to a general duration constraint, any constraint involving an increasing function of (global) time. For example we can verify the following formula where z represents time:

$$[x, y, z : P_0, P_1, \mathrm{true}].\exists\Diamond((2z^3 \leq 30) \wedge (x^2 - 3y \leq -5))$$

4.3 Existence of Trajectories

We consider now the verification problem of trajectory existence formulas, i.e. formulas of the form $[\vec{x} : \vec{\varphi}].\exists\Box\phi$. These formulas are dual of *eventuality* properties which are involved in interesting classes of properties like *response* properties. We investigate the verification of formulas of the two following forms:

$$[\vec{x} : \vec{P}].\exists\Box(P \wedge \bigwedge_{i=1}^{m}(e_i \leq \beta_i))$$

where the e_i's are recursive functions, and

$$[\vec{x} : \vec{P}].\exists\Box \bigvee_{j=1}^{l} P^j \wedge \bigwedge_{i=1}^{m_j}(e_i^j \leq \beta_i^j)$$

where the e_i^j's are linear functions. Intuitively, for such formulas the verification procedure consists in a search of some path in the labelled computation tree which is ultimately periodic, i.e. repeats infinitely many times some "satisfactory circuit".

Definition 4.3 *(Satisfactory circuits)*
Let $\phi = \bigvee_{j=1}^{l} P^j \wedge \bigwedge_{i=1}^{m_j}(e_i^j \leq \beta_i^j)$. We say that a circuit π is satisfactory for ϕ iff it is time strictly increasing and there exists $J = \{j_1, \cdots, j_k\} \subseteq \{1, \cdots, l\}$ such that π is decreasing for $\{e_i^j : j \in J, i \in \{1, \cdots, m_j\}\}$ and the formula $\bigvee_{j \in J} P^j \wedge \bigwedge_{i=1}^{m_j}(e_i^j \leq \beta_i^j)$ is true along π.

The proof of the decidability of trajectory existence problem is based on the following key lemma.

Lemma 4.1 *Let π be a divergent path of $\mathcal{T}(s_0, E_0)$ and let $\{e_1, \cdots, e_n\}$ be a set of expressions. If for any $i \in \{1, \cdots, n\}$, the sequence $E_\pi(e_i)$ is upper bounded by an integer β_i, then the path π contains an infinite sequence of time strictly increasing circuits which are decreasing for $\{e_1, \cdots, e_n\}$.*

Proof: Let $\pi = (s_0, E_0, T_0), (s_1, E_1, T_1), \cdots$. The proof is by induction on the number n of the expressions.

Let $n = 1$. Let e be the unique expression to consider. As the number of states is finite, the path π must repeat infinitely some state s. Let $J = \{j_0, j_1, \cdots\}$ be the indices of the occurrences of s in π. Let $M_0 = \max \{E_j(e) \ : \ j \in J\}$ and let u_0 be an index in J such that $M_0 = E_{u_0}(e)$. M_0 exists because the sequence $E_\pi(e)$ is upper bounded. As π is time divergent, there exists some index $k \in J$ such that $k > u_0$ and $T_k > T_{u_0}$. Let $M_1 = \max \{E_j(e) \ : \ j \in J \text{ and } j \geq k\}$ and let u_1 be an index in J greater than k such that $M_1 = E_{u_1}(e)$. The subpath of π from index u_0 to index u_1 is a time strictly increasing circuit which is decreasing for the expression e. Now, we can iterate this reasoning and construct an infinite sequence of indices $U = u_0, u_1, \cdots$ which corresponds to an infinite sequence of circuits which are time strictly increasing and decreasing for e.

Let $n > 1$. Assume that the property is true for the expressions $\{e_1, \cdots, e_{n-1}\}$ and show that it remains true when we add the expression e_n. Let $U = u_0, u_1, \cdots$ be the sequence of indices delimiting the infinite sequence of circuits which have been found for $\{e_1, \cdots, e_{n-1}\}$. We can make the same reasoning as in the case of a single expression presented above by taking $e = e_n$, $J = U$ and constructing a new sequence of indices U' for the set $\{e_1, \cdots, e_n\}$. $\Box$

Proposition 4.1 *(Conjunctions)*
Let π be an infinite time divergent path from the root of $\mathcal{T}(s_0, E_0)$. If $s_\pi \models_{(E_0,\gamma)} \Box(P \wedge \bigwedge_{i=1}^n e_i \leq \beta_i)$ where the e_i's are recursive , then π contains a satisfactory circuit for ϕ.

The proof of this proposition follows from lemma 4.1.

Proposition 4.2 *(Disjunctions)*
Let π be an infinite time divergent path from the root of $\mathcal{T}(s_0, E_0)$. If $s_\pi \models_{(E_0,\gamma)} \Box(P^j \wedge \bigvee_{j=1}^l \bigwedge_{i=1}^{m_j}(e_i^j \leq \beta_i^j))$ where the e_i^j's are linear, then π contains a satisfactory circuit for ϕ.

The proof of this proposition is by induction on l. It consists in giving an upper bound on the length of π so that π necessarily includes a satisfactory circuit. See the details in [BES93].

Now we show that the trajectory existence problem is decidable for a conjunction of arbitrary duration constraints as well as for formulas of the form $\bigvee_{j=1}^l P^j \wedge \bigwedge_{i=1}^{m_j}(e_i^j \leq \beta_i^j)$ where the e_i's are linear.

Theorem 4.2 *(Deciding existence of trajectories)*
Let M be an STG, s be a state in S_M and $\varphi = [\vec{x} : \vec{P}].\exists\Box\phi$. The problem $M : s \models_{(E,\gamma)} \varphi$ is decidable either when $\phi = P \wedge \bigwedge_{i=1}^m (e_i \leq \beta_i)$ where the e_i's are recursive , or $\phi = \bigvee_{j=1}^l P^j \wedge \bigwedge_{i=1}^{m_j}(e_i^j \leq \beta_i^j)$ where the e_i^j's are linear.

Proof: We explore the tree $\mathcal{T}(s, \epsilon)$ from its root, stopping the construction at each node satisfying one of the following conditions :

1. the formula ϕ is falsified by the node,
2. a time constant circuit is detected,
3. a satisfactory circuit for ϕ is detected.

The answer is *yes* if and only if a satisfactory circuit for ϕ is detected.

Now, let us show the correctness of this procedure: First, it is consistent since a satisfactory circuit can be repeated infinitely many and thus there exists a path satisfying $\Box\phi$. Conversely, the procedure is complete. Indeed, any path satisfying a formula $\Box\phi$ which contains a time constant circuit still satisfies the formula after removing the time constant circuit from the path. On the other hand, for any path without time constant circuits which satisfies $\Box\phi$, there exists, by propositions 4.1 and 4.2, a satisfactory circuit for ϕ. Finally, the procedure stops as $\mathcal{T}(s, \epsilon)$ is finite branching and any path in it either falsifies ϕ at some node or contains a satisfactory circuit for ϕ. $\Box$

4.4 Conditional State Reachability

We consider now closed SDTL formulas in DNF of the form $[\vec{x} : \vec{\varphi}].\exists\phi_1\mathcal{U}\phi_2$ satisfying a syntactic condition called CSR (for Conditional State Reachability).

Definition 4.4 *(The condition CSR)*
An SDTL formula in DNF $\phi = [\vec{x} : \vec{\varphi}].\exists\phi_1\mathcal{U}\phi_2$ is CSR if it satisfies one of the following conditions, where φ, φ_1 and φ_2 stand for SDTL formulas in DNF :

1: ϕ_1 is closed and either ϕ_2 has at most one top-level constraint using a recursive function or all its top-level constraints are linear,

2: $\phi_1 = \varphi_1 \wedge (\bigwedge_{i=1}^n (e_i \leq \beta_i)) \vee \varphi_2$ where the e_i's are recursive functions, and $\phi_2 = \{\phi_1 \wedge\}\varphi$,

3: $\phi_1 = \varphi_1 \wedge (\bigvee_{i=1}^n (e_i \leq \beta_i)) \vee \varphi_2$ where the e_i's are linear, and $\phi_2 = \{\phi_1 \wedge\}\varphi$,

4: $\phi_1 = \varphi_1 \wedge (\bigwedge_{i=1}^n (\beta_i^1 \leq e_i \leq \beta_i^2)) \vee \varphi_2$ where the e_i's are recursive functions, and $\phi_2 = \{\phi_1 \wedge\}\varphi'$ with $\varphi' = (e \leq \beta) \wedge \varphi$

As we have mentioned above in subsection 4.2, we can relax the conditions (1) and (2) of CSR without affecting our results by allowing the addition to the single general duration constraint any set of constraints involving increasing functions of (global) time.

Theorem 4.3 *(Decidability of the conditional state reachability problem under the condition CSR)*
Let M be an STG, s be a state in S_M and $\phi = [\vec{x} : \vec{P}].\exists\phi_1 \mathcal{U} \phi_2$ be an SDTL formula in DNF, without nested path quantifiers, which satisfies the condition CSR. Then the problem $M : s \models_{(E,\gamma)} \phi$ is decidable.

The proof of this theorem may be found in [BES93].

4.5 Model-checking for a fragment of SDTL

This subsection aims to combine the results of the three previous ones. That is why we define the notion of *legal* formulas as SDTL formulas whose subformulas (including formulas associated to duration variables) have a "same shape" as formulas considered above.

Definition 4.5 *(Legal SDTL formulas)*
We say that an SDTL formula in DNF is legal if all its $\mathcal{U}$-subformulas are CSR and all its $\Box$-subformulas are of the two following forms : $\Box\phi \wedge \bigwedge_{i=1}^{m}(e_i \leq \beta_i)$ where the e_i's are recursive functions and ϕ is a legal formula, or $\Box \bigvee_{j=1}^{l} \phi^j \wedge \bigwedge_{i=1}^{m_j} \phi_i^j \wedge (e_i^j \leq \beta_i^j)$ where the e_i^j's are linear functions and the ϕ^j's are legal formulas.

Theorem 4.4 *The verification problem is decidable for legal SDTL formulas.*

Proof: Let φ be a legal formula. We denote by $cl(\varphi)$ the set of its closed subformulas and their negations. The algorithm proceeds as the classical model-checking algorithm of CTL [CES83] by defining a sequence of labelling functions $\Lambda_i : S_M \rightarrow 2^{cl(\varphi)}$ associating to each state the formulas of height i satisfied by this state. The height of a formula corresponds to the number of nested closed formulas in it. These labelling functions are defined as follows: Take $\Lambda_0 = \Pi \circ \nu$, and the function Λ_{i+1} is obtained by checking for each state whether it satisfies the formulas of height $i+1$, using the techniques given by theorems 4.1, 4.2 and 4.3 in addition to those developed in [KPSY92] for state reachability formulas involving linear constraints. At the $(i+1)^{th}$ step, the formulas of height less or equal than i are considered as atomic propositions which are valuated at each state by the function Λ_i. $\Box$

5 Model Checking : The Dense Time Case

We consider now that the time ranges over the set of positive reals ($\mathbb{R}^+$). Given a model $M = (Q, \Pi, R)$, a node $q \in Q$ and a formula ϕ, we are interested in the verification problem $M : \langle q, 0\rangle \models \phi$. We show that this problem can be solved using the techniques developed for the discrete time case by considering a suitable rational time unit u. This unit depends on the model M and the formula ϕ. Actually, we construct a model M_u by *slicing* M w.r.t. the unit u and then, we apply our verification procedures to M_u by considering its nodes as single states.

For this aim, we restrict ourselves to formulas such that all their subformulas have at most one top-level duration constraint $e \sim \beta$ where $\sim \in \{\leq, <\}$ such that e corresponds to a continuous function which is monotonic at each node of the considered model M. We introduce a requirement on the functions e, we call *time monotonicity*, which insures that e is monotonic at each node of any model.

Definition 5.1 *(Time-monotonicity)*
Let e be a function on $\mathbb{R}^n$. We say that e is time-monotonic *if and only if $\forall J \subseteq \{1, \cdots, n\}$, $\forall x_1, \cdots, x_n \in \mathbb{R}$, either*

- $\forall\delta \in \mathbb{R}^+ .\ e(x_1, \cdots, x_n) \leq e(x'_1, \cdots, x'_n)$ *or*
- $\forall\delta \in \mathbb{R}^+ .\ e(x_1, \cdots, x_n) \geq e(x'_1, \cdots, x'_n)$

where $x'_j = ($if $j \in J$ then $x_j + \delta$ else $x_j)$.

Notice that monotonic functions are time-monotonic but the converse is not true in general. For example, the function e such that $e(x, y) = x - y$ is not monotonic but it is time-monotonic. An important class of time-monotonic functions is the class of linear functions, i.e., functions defined by $e(x_1, \cdots, x_n) = \sum_{i=1}^{n} \alpha_i x_i$ where $\alpha_i \in \mathbb{Z}$. Another example of a time-monotonic function is $e(x, y, z) = x^3 + 5y^2 + 2^z$.

The restriction to a single top-level duration constraint $e \sim \beta$ allows, while exploring the model M_u, to consider for any transition (q, I, q') the time in the interval I which minimizes e in the node q. Furthermore, the monotonicity of e makes this time correspond to either the left or the right bound of I depending on whether e is increasing or decreasing in q (if I is open, this time can be taken arbitrarily close to the bounds). This reasoning obviously fails if we consider two constraints since they are not necessarily increasing or decreasing in the same nodes.

Finally, we consider from now on that the duration variables are associated with atomic propositions.

5.1 The Slicing Transformation

We define in this subsection the slicing transformation of an STG with respect to a rational time unit. Let $M = (Q, \Pi, R)$ be an STG. We introduce first the set of time units U_M for which the slicing transformation is defined. We denote by $\mathcal{B}_M$ the set of the bounds of all the transition constraints of M. For any rational r represented by a fraction in its reduced form, we denote by $num(r)$ and $den(r)$ respectively the numerator and the denominator of r.

Let $u_M = 1/\kappa_M$ where $\kappa_M = lcm\{|den(r)| \ : \ r \in \mathcal{B}_M\}$. The set U_M is defined as the set of time units which are obtained by dividing u_M by some natural k, i.e., $U_M = \{u \in \mathbb{Q}^+ \ : \ \exists k \in \mathbb{N}.\ u = u_M/k\}$.

For $u \in U_M$, we denote by $u\mathbb{N}$ the set of the multiples of u, i.e., $u\mathbb{N} = \{ku \ : \ k \in \mathbb{N}\}$.

Definition 5.2 *(Sliced model)*
Given $u \in U_M$, we define the STG $M_u = (Q_u, \Pi_u, R_u)$ as follows

- $Q_u = \{(q,t) \in Q \times u\mathbb{N} \ : \ t \in D_q\}$,
- $\forall (q,t) \in Q_u.\ \Pi_u(q,t) = \Pi(q)$,
- $\forall q \in Q.\ \forall t \in u\mathbb{N}.\ ((q,t),[u,u],(q,t+u)) \in R_u$ *iff* $t+u \in D_q$,
- $\forall q,q' \in Q.\ \forall t \in u\mathbb{N}.\ ((q,t),[0,0],(q',0)) \in R_u$ *iff* $\exists (q,I,q') \in R$ *and* $t \in I$,
- $\forall q,q' \in Q.\ \forall t \in u\mathbb{N}.\ ((q,t),]0,u[,(q',0)) \in R_u$ *iff* $\exists (q,(n,m),q') \in R$ *and* $t \in (n,m)$ *and* $t < m$.

The elements of U_M allow to slice uniformly the nodes of M. Indeed, for any unit u in U_M, each node of M corresponds to a set of nodes in M_u and all the transitions are labelled either by $[0,0]$, $[u,u]$ or $]0,u[$. Notice that the state $\langle (q,t),u\rangle$ and $\langle (q,t+u),0\rangle$ of M_u correspond to the same state $\langle q, t+u\rangle$ of M. It is easy to see that this transformation preserves the satisfiability of SDTL formulas.

Proposition 5.1 *(Preservation of SDTL formulas)*
Let M be an STG, ϕ be an SDTL formula and $u \in U_M$. $M : \langle q,t\rangle \models \phi$ iff $M_u : \langle (q, u.\lfloor t/u\rfloor), t - u.\lfloor t/u\rfloor\rangle \models \phi$.

5.2 Formulas with a single duration constraint

We consider here the verification problem for formulas containing at most one duration constraint. Let us consider an STG $M = (Q, \Pi, R)$ and a formula ϕ to be checked at some state of M. Let γ be the atomic formula association defined by ϕ and $\xi = e \sim \beta$ be the constraint appearing in ϕ where $\sim \in \{\leq, <\}$. Similarly to the discrete time case, our decision techniques are based on finite explorations of a labelled computation tree. However, in the present case, the labelled computation tree is obtained by unfolding the model M_{u_M}. Any path from the root of this tree to some node q corresponds to an infinite (non countable) set of computation sequences. To each of these computation sequences corresponds a value of e at q (actually, for each node q, we consider the value of e at the state $\langle q, 0\rangle$). Indeed, for any transition (q_1, I, q_2) in M_{u_M} of the considered path, we can choose any time in I and may have a different value of e at q. Since e is continuous, the set of such possible values of e at q is an interval I_e. As we are interested in the existence of some values of e less than β, we consider only the lower bound of the interval I_e. The time-monotonicity of e guarantees that from the lower bound of I_e at q, we can compute the lower bounds at the successors of q by minimizing or maximizing the allowed duration at q depending on whether e is strictly increasing or strictly decreasing in q. If e is constant in q, we maximize the duration to guarantee the progress of time. We may know how the function e evolves in each node q by checking the sign of its weight $w_q(e)$ defined by $w_q(e) = E_q(e)$ where E_q is the environment such that for any variable x, $E_q(x) = if \gamma(x) \in \Pi(q)$ then 1 else 0. Thus, while unfolding the model, we maintain at each node the environment corresponding to the lower bound of e and the information saying whether this bound is large or strict. Furthermore, to be able to decide if a path is time divergent, we maintain also at each node q the value of the maximal global time spent from the root, which is consistent with the lower bound of e at q.

We give hereafter the formal definition of the labelled computation tree. First, let us introduce the following notation. Let E by an environment and $t \in \mathbb{Q}^+$. We denote by $E + t$ the environment such that for any variable x, $(E+t)(x) = E(x) + \Delta_t(x)$ where $\Delta_t(x) =$ if $\gamma(x) \in \Pi(q)$ then t else 0.

Definition 5.3 *(Labelled Computation Tree)*
Let $u = u_M$ and consider the sliced model $M_u = (Q_u, \Pi_u, R_u)$. Given a constraint $\xi = e \sim \beta$, a node $q_0 \in Q$ and an environment E_0, we define the tree $\mathcal{T}(s_0, E_0)$ as follows :

- *the nodes are tuples (q, E, b, T) where $q \in Q_u$, $E \in \mathcal{E}$, $b \in \{[,]\}$ and $T \in \mathbb{Q}^+$, $b = [$ (resp. $]$) means that the bound is large (resp. strict),*
- *the root is $(q_0, E_0, [, 0)$,*

- (q', E', b', T') *is a successor of* (q, E, b, T) *iff* $\exists I = (n, m) \in \mathcal{I}.\ (q, I, q') \in R_u$ *and*
  - *if* $w_q(e) = 0$ *then* $E' = E$, $b' = b$ *and* $T' = T + m$,
  - *if* $w_q(e) > 0$ *then* $E' = E + n$, $b' = b_l$ *and* $T' = T + n$,
  - *if* $w_q(e) < 0$ *then* $E' = E + m$, $b' = b_r$ *and* $T' = T + m$,

  where
  $b_l =$ (*if* $b = [$ *and* I *is left closed then* $[$ *else* $]$) *and*
  $b_r =$ (*if* $b = [$ *and* I *is right closed then* $[$ *else* $]$)

Now, for a node (q, E, b, T) of the labelled computation tree, consider the interval $I_e = (E(e), +\infty[$ where the left bound is b and let I_ξ be the interval $]-\infty, \beta)$ where the right bound is either $[$ or $]$ depending on whether $\sim$ is respectively $<$ or $\leq$. The constraint $\xi = e \sim \beta$ is satisfied at (q, E, b, T) if and only if the intersection of the intervals I_e and I_ξ is non empty.

The notion of satisfactory circuit for e can be defined exactly as in the discrete time case by reasoning on the sequences E_π and T_π for a given path π in the labelled computation tree. Notice that when e is constant at some node, we take the maximal allowed time. This insures that in a time-constant circuit, we cannot let time progress without increasing e.

The same decision techniques given in the discrete time case can be applied and we get the following theorem.

Theorem 5.1 *(Verification of single duration constraint formulas) Let M be an STG, e be a continuous time-monotonic function and $\sim \in \{<, \leq\}$. The problems*

- $M : \langle q, 0\rangle \models [\vec{x} : \vec{P}].\exists\Box(P_1 \vee P_2 \wedge (e \sim \beta))$,
- $M : \langle q, 0\rangle \models [\vec{x} : \vec{P}].\exists P_1 \mathcal{U}(P_2 \wedge (e \sim \beta))$,
- $M : \langle q, 0\rangle \models [\vec{x} : \vec{P}].\exists(P_1 \vee P_2 \wedge (e \sim \beta))\mathcal{U} P_3$

are decidable.

5.3 Formulas with several duration constraints

We consider now the verification problem of formulas containing several duration constraints. Recall that we have supposed in the beginning of this section that we consider only formulas having at most one top-level duration constraint at each subformula. We consider the following restrictions on the models and the formulas. We suppose that the models are *largely constrained*, i.e., all their transition constraints are large (closed intervals). On the other hand, we restrict ourselves to formulas such that in their disjunctive normal form, all the duration constraints are linear and large (use $\leq$) and all the negations occurring in the scope of a (existential) path quantifier apply to atomic propositions. We call the set of such formulas SDTL$^+$. The disjunctive normal form of SDTL$^+$ formulas are of the form φ in the following grammar

$$
\begin{array}{lcl}
\varphi & ::= & \phi \mid \neg\varphi \mid \varphi \vee \varphi \\
\phi & ::= & \bigvee \bigwedge \eta \\
\eta & ::= & P \mid \neg P \mid [\vec{x} : \vec{P}].\exists\zeta \\
\zeta & ::= & \Diamond\psi \mid \Box\psi \mid \psi\mathcal{U}\phi \mid \phi\mathcal{U}\psi \\
\psi & ::= & ((e \leq \beta) \wedge \phi) \vee \phi
\end{array}
$$

Despite the restriction, SDTL$^+$ formulas allow to express interesting properties such as:

- **Response Properties** like
  $[x, y : P_0, P_1]$.
  $\forall\Box((x \geq 2y) \Rightarrow [z, t : P_2, P_3].\forall\Diamond(3z < t))$
- **Persistence Properties** like
  $[x, y : P_0, P_1]$.
  $\forall\Diamond((x > 2y) \wedge [z, t : P_2, P_3].\forall\Box(3z < t))$

Without loss of generality, we focus in the sequel on the formulas of the form ϕ in the above grammar because any SDTL$^+$ formula is a boolean combination of such formulas.

Given a largely constrained STG M and an SDTL$^+$ formula ϕ, we define the time unit $u_M(\phi) \in U_M$ (see the definition of U_M in Section 5.1) such that the satisfiability of ϕ in M is shown to be equivalent to its satisfiability in a discrete time STG obtained from the slicing of M w.r.t. $u_M(\phi)$.

Definition 5.4 *(The time unit $u_M(\phi)$)*
We define inductively the time unit $u_M(\phi)$ for any SDTL$^+$ formula ϕ by $u_M(\phi) = 1/\kappa_M(\phi)$ where

$\kappa_M(\phi_1 \vee \phi_2) = \kappa_M(\phi_1 \wedge \phi_2) = lcm(\kappa_M(\phi_1), \kappa_M(\phi_2))$,
$\kappa_M(P) = \kappa_M(\neg P) = 1/u_M$,
$\kappa_M(\exists\Diamond\psi) = \kappa_M(\exists\Box\psi) = \kappa_M(\psi)$,
$\kappa_M(\exists\psi\mathcal{U}\phi) = \kappa_M(\exists\phi\mathcal{U}\psi) = lcm(\kappa_M(\phi_1), \kappa_M(\phi_2))$,
$\kappa_M((e \leq \beta) \wedge \phi_1 \vee \phi_2) =$
$\quad lcm(\{|w_q(e)| \ : \ q \in Q\}).\ lcm(\kappa_M(\phi_1), \kappa_M(\phi_2))$.

Remark 5.1 *If ϕ' is a subformula of ϕ then there exists some $k \in I\!N$ such that $u_M(\phi) = u_M(\phi')/k$. This means that the slicing of M w.r.t. $u_M(\phi)$ is a refinement of the slicing of M w.r.t. $u_M(\phi')$.*

Let $M_u = (Q_u, \Pi_u, R_u)$ be a sliced STG. Let DR_u be the transition relation obtained by removing in R_u

the transitions labelled by $]0, u[$. We denote by DM_u the STG (Q_u, Π_u, DR_u) with the discrete time domain $\mathcal{D} = u\mathbb{N}$.

Proposition 5.2 *(Satisfiability by digital computation sequences) Let M be a largely constrained STG, ϕ an SDTL$^+$ formula and $u = u_M(\phi)$.*

(a) $M_u : \langle (q_0, t_0), 0\rangle \models \phi$ *implies* $DM_u : \langle (q_0, t_0), 0\rangle \models \phi$,

(b) $(\exists t \in]0, u[.\ M_u : \langle (q_0, t_0), t\rangle \models \phi)$ *implies* $(\forall t' \in [0, u].\ M_u : \langle (q_0, t_0), t'\rangle \models \phi)$.

The point (a) of the proposition above says that if the formula ϕ is satisfied in the dense time model M_u then it is satisfied in the discrete time model DM_u.

The point (b) means that the *internal* states of the node (q_0, t_0) of M_u are equivalent w.r.t. the formula ϕ and thus, from Remark 5.1, they are also equivalent w.r.t. all the subformulas of ϕ. Furthermore, the point (b) says that when ϕ is satisfied by an internal state in some node, it is necessarily satisfied by all the states of the node, including the states with time 0 and u.

Proof: (Hints) The two points (a) and (b) are proved simultaneously by induction on the structure of the formulas. We illustrate the proof on the significant cases. For details, see [BES93].

Consider first the point (a) and take for example $\phi = [\vec{x} : \vec{P}] : \exists\Box(\phi' \wedge (e \leq \beta))$. Let $\sigma = \langle (q_0, t_0), 0\rangle \xRightarrow{\delta_0} \cdots \langle (q_i, t_i), 0\rangle \xRightarrow{\delta_i} \cdots$ be a computation sequence that satisfies $\psi = \Box(\phi' \wedge (e \leq \beta))$. Consider $i \geq 0$ such that $\delta_i \in]0, u[$. If $w_{q_i}(e) > 0$ then we replace δ_i by 0. Otherwise, we take in M_u the transitions $\langle (q_i, t_i), 0\rangle \xRightarrow{u} \langle (q_i, t_i + u), 0\rangle \xRightarrow{0} \langle (q_{i+1}, t_{i+1}), 0\rangle$. In this fashion, we obtain a computation sequence σ' which takes at each transition either 0 or u time. The value of e at any state $\langle (q_i, t_i), 0\rangle$ in σ' is less or equal than its value at the same state in σ. Furthermore, σ' is time-divergent since the original sequence σ is time-divergent and thus, since it satisfies ψ, there are necessarily infinitely many nodes q_i visited by σ such that $w_{q_i}(e) \leq 0$. Finally, the satisfaction of the subformula ϕ' at each state of σ' comes from the point (b) by induction hypothesis. Thus, σ' satisfies ψ in M_u and then in DM_u.

Consider now the point (b). Take for example $\phi = [\vec{x} : \vec{P}] : \exists\Diamond(\phi' \wedge (e \leq \beta))$ and let $\sigma = \langle (q_0, t_0), 0\rangle \xRightarrow{\delta_0} \cdots \xRightarrow{\delta_{n-1}} \langle (q_n, t_n), 0\rangle \xrightarrow{\delta_n} \langle (q_n, t_n), \delta_n\rangle$ be a computation sequence such that $\langle (q_n, t_n), \delta_n\rangle$ satisfies $\phi' \wedge (e \leq \beta)$. Let $u' = u_M(\phi')$. By Definition 5.4, we have $u = u'/\kappa$ where $\kappa = lcm\{w_q(e) : q \in Q\}$. Thus, each node of $M_{u'}$ can be sliced into κ nodes of M_u. Let ν_0 and ν_n be the nodes of $M_{u'}$ which contain respectively the nodes (q_0, t_0) and (q_n, t_n) of M_u. Suppose without loss of generality that $w_{q_0}(e)$ and $w_{q_n}(e)$ are not equal to zero. We slice the nodes ν_0 and ν_n w.r.t. to the units $u_0 = 1/w_{q_0}(e)$ and $u_n = 1/w_{q_n}(e)$ respectively. Consider, for j equals to 0 or n, the times $\{i.u_j : i \in \{0, 1, \cdots, w_{q_j}(e)\}\}$. By definition of κ, any interval $[i.u_j, (i+1).u_j]$ covers an exact number of nodes of M_u. Let $I_0 = [i_0.u_0, (i_0+1).u_0]$ and $I_n = [i_n.u_n, (i_n+1).u_n]$ be the intervals in ν_0 and ν_n which cover the nodes (q_0, t_0) and (q_n, t_n).

Let us suppose that both $w_{q_0}(e)$ and $w_{q_n}(e)$ are positive (the other cases are similar). Let s and s' be the states which correspond respectively to the times $(i_0 + 1).u_0$ and $i_n.u_n$ in the nodes ν_0 and ν_n. Since $w_{q_n}(e) > 0$, necessarily s' satisfies $(e \leq \beta)$ and by induction hyphothesis, the point (b) says that ϕ' is satisfied by any state in ν_n, in particular by s'. Furthermore, since $w_{q_0}(e) > 0$, using the same computation sequence σ and starting from the state s, we satisfy necessarily the formula $\phi' \wedge (e \leq \beta)$ at s'.

Now, consider the path in $M_{u'}$ starting from ν_0 and ending by ν_n which corresponds to the nodes that are visited by the computation sequence σ. As we have seen previously in the proof of the point (a), for any intermediate node between ν_0 and ν_n, we can choose a duration δ which is either 0 or u' in order to minimize e. If the weight of this node is w, the variation of e in this node is $w.\delta$ since e is linear. This makes the accumulated variations of e in these nodes a multiple of u'. Furthermore, the variation of e is exactly u' in any interval $[i.u_j, (i+1).u_j]$ in the nodes ν_j for $j \in \{0, n\}$.

Thus, there exists some computation sequence from s to s' such that the function e has some value e' at s' which satisfies the constraint $e \leq \beta$ and which is a multiple of u'. This sequence minimizes e at each step, thus e' is less or equal to the value of e we obtain at s' by the sequence σ. So, if we start from $\langle (q_0, t_0), t\rangle$ and follow at s this new computation sequence instead of σ, the constraint $e \leq \beta$ remains satisfied at s'. Moreover, the value of e obtained at s' is $e' + v$ for some v such that $0 < v < u'$ since the variation of e in the interval I_0 is exactly u'. Since e' is a multiple of u' and $e' + v \leq \beta$ then $e' + u' \leq \beta$. Thus, all the states in the interval I_0 satisfy ϕ. $\Box$

The following lemma allows to decide whether a formula ϕ is uniformly satisfied at some node of M_u (i.e. ϕ is satisfied by all the states in the node) when ϕ is satisfied by the *boundary* states of the node (i.e. the states with time 0 and u) w.r.t. the discrete time model DM_u.

Lemma 5.1 *(Uniform satisfiability)*

Let M be a largely constrained STG, ϕ an SDTL$^+$ formula and $u = u_M(\phi)$. Consider that the disjunctive normal form of ϕ can be written $\phi = \bigvee_{i=1}^{n} \phi_i$ where the ϕ_i's are not disjunctive formulas.
If $((q_0, t_0), [u, u], (q_0, t_0 + u)) \in DR_u$ and there exists some $i \in \{1, \cdots, n\}$ such that

1. *$DM_u : \langle (q_0, t_0), 0 \rangle \models \phi_i$ and*

2. *$DM_u : \langle (q_0, t_0 + u), 0 \rangle \models \phi_i$*

then $(\forall t \in [0, u].\ M_u : \langle (q_0, t_0), t \rangle \models \phi)$.

Proof: (Hints) The proof is by induction on the structure of the formulas. The uniform satisfiability for non-disjunctive formula can be shown using the monotonicity of the functions e in the duration constraints. Consider now a disjunctive formula as for example $\phi = \phi_1 \vee \phi_2$. Suppose that ϕ_1 is satisfied at $\langle (q_0, t_0), 0 \rangle$ but not at $\langle (q_0, t_0 + u), 0 \rangle$ and that the converse holds for ϕ_2. Then, since the state $\langle (q_0, t_0 + u), 0 \rangle$ is equivalent to the state $\langle (q_0, t_0), u \rangle$ and by Proposition 5.2, neither ϕ_1 nor ϕ_2 is satisfied at the states $\langle (q_0, t_0), t \rangle$ with $t \in]0, u[$. Thus, a disjunction is uniformly satisfied in some node if and only if one of its (non-disjunctive) members is uniformly satisfied in this node. $\square$

The following proposition says that if a formula ϕ is satisfied in the discrete time model DM_u then ϕ is satisfied in the dense time model M_u.

Proposition 5.3 *(Inferring "dense" satisfiability from "discrete" satisfiability)*
Let M be a largely constrained STG, ϕ an SDTL$^+$ formula and $u = u_M(\phi)$. Then
$DM_u : \langle (q_0, t_0), 0 \rangle \models \phi$ implies $M_u : \langle (q_0, t_0), 0 \rangle \models \phi$.

The proof of this proposition is by induction on the structure of the formulas. The case of trajectory existence formulas is solved using Lemma 5.1. The other cases are immediate.

Propositions 5.2(a) and 5.3 assert the completeness and the consistency of the discrete verification procedure when applied to the model DM_u with $u = u_M(\phi)$.

Theorem 5.2 *(Verification of SDTL$^+$ formulas)*
Let M be a largely constrained STG and φ be an SDTL$^+$ formula. The problem $M : \langle q, 0 \rangle \models \varphi$ is decidable.

6 Conclusion

We have investigated the verification problem of real-time systems against temporal formulas with duration constraints. We have shown that for the most common properties this problem is decidable. Several extensions of this work can be considered.

First, we have established our results for real-time models which can be seen as timed graphs with only one clock that is reset at each transition. Thus, an interesting question is whether our techniques can be extended to general timed graphs. Actually, this can be done in the discrete time case. All we need is to construct the state graph corresponding to the timed graph. This construction is a specialization of the region graph construction reported in [ACD90]. However, the question remains open in the case of dense time.

Many questions remain also open concerning the set of SDTL formulas that can be verified, especially in the dense time case. We have shown that for a fragment of dense time SDTL we can use the verification techniques developed for a discrete time domain. A crucial problem is to determine the class of dense time SDTL formulas that can be verified, in particular by using discrete time methods. For this, digitization techniques may be useful [HMP91],[KPSY92].

Another interesting question concerns the language of duration constraints that can be considered. We have shown that in some significant cases, one can consider constraints on some functions such as polynomials or exponentials. However, we are in general either restricted to linear functions or, when the time domain is dense, to time-monotonic functions. So, it would be interesting to find verification methods allowing more general constraints.

Finally, it is interesting to remark that in the discrete time case, the notion of duration can be used in particular to reason about the number of occurrences of some state or transition property in a finite graph. In the untimed classical framework, this allows to obtain straightforwardly an extension of the temporal logic CTL where it is possible to express properties involving comparisons between the numbers of event or state changes occurrences. Our verification techniques can then be applied to verify such properties. It is important to see that these properties are in general non regular, i.e., they can not be expressed by finite state (infinite tree) automata.

Acknowledgements:

We would like to thank Amir Pnueli for valuable suggestions and comments. We thank also Mihalis Yannakakis for a suggestive communication and finally, we thank Sergio Yovine for many interesting discussions.

References

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real time systems. In *Proc. 5th Symp. on Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.

[ACH] R. Alur, C. Courcoubetis, and T. A. Henziger. Computing accumulated delays in real-time systems.

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

[AH90] R. Alur and T. Henzinger. Real-time logics: complexity and expressiveness. In *Proc. 5th Symp. on Logic in Computer Science*, pages 390–401. IEEE Computer Society Press, 1990.

[AH91] R. Alur and T. Henzinger. Logics and models of real-time: a survey. In *Proc. REX Workshop "Real-Time: Theory in Practice"*, the Netherlands, June 1991. Lecture Notes in Computer Science 600, Springer-Verlag.

[Alu91] R. Alur. *Techniques for automatic verification of real-time systems.* PhD thesis, Department of Computer Science, Stanford University, August 1991.

[BES93] A. Bouajjani, R. Echahed, and J. Sifakis. On Model Checking for Real-Time Properties with Durations. RTC 42, IMAG, Grenoble, 1993.

[CES83] E. M. Clarke, E. A. Emerson, and E. Sistla. Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications: A Practical Approach. In *10th Symposium on Principles of Programming Languages (POPL 83).* ACM, 1983. Complete version published in ACM TOPLAS, 8(2):244–263, April 1986.

[CHR91] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–276, 1991.

[HLP90] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit-clock temporal logic. In *Proc. 5th Symp. on Logic in Computer Science*, pages 402–413. IEEE Computer Society Press, 1990.

[HMP91] T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proc. 18th Symp. on Principles of Programming Languages.* ACM Press, 1991.

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *Proc. 7th Symp. on Logic in Computer Science.* IEEE Computer Society Press, 1992.

[KPSY92] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: a class of decidable hybrid systems. In *Workshop on Theory of Hybrid Systems*, Lyngby, Denmark, October 1992.

[NSY91] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In *Proc. REX Workshop "Real-Time: Theory in Practice"*, the Netherlands, June 1991. Lecture Notes in Computer Science 600, Springer-Verlag.

[Pnu77] A. Pnueli. The Temporal Logic of Programs. In *18th Symposium on Foundations of Computer Science (FOCS 77).* IEEE, 1977. Revised version published in Theoretical Computer Science, 13:45–60, 1981.

[QS82] J-P. Queille and J. Sifakis. Specification and verification of concurrent systems is cesar. In *International Symposium on Programming, LNCS 137.* Springer Verlag, 1982.