The Concurrency Column

by

Luca Aceto

BRICS, Department of Computer Science
Aalborg University, 9220 Aalborg Ø, Denmark
luca@cs.auc.dk, http://www.cs.auc.dk/~luca/BEATCS

In this installment of the Concurrency Column, Patricia Bouyer and Fabrice Chevalier offer a survey of results on the control of timed and hybrid systems. This is a very active area of research that is both the source of interesting theoretical problems and, in light of the ubiquituous presence of embedded systems, has the potential for wide practical applications.

I hope that readers of this survey will enjoy it and will be enticed to work on this difficult, but rewarding, area. I have no doubt that Fabrice and Patricia will continue to be at the forefront of the developments in this field of concurrency theory, and look forward to their future work on this topic.

On the Control of Timed and Hybrid Systems*

Patricia Bouyer and Fabrice Chevalier
LSV, CNRS & ENS de Cachan, France
{bouyer,chevalie}@lsv.ens-cachan.fr

Abstract

In this paper, we survey some of the results which have been obtained the last ten years on the control of hybrid and timed systems.

1 Introduction

Timed and hybrid systems. Timed automata are a well-established and widely used model for representing real-time systems. Since their definition by Alur & Dill in the 90's [3], many works have investigated this model, both from a theoretical and an algorithmic point of view. Several tools have been developed for model-checking timed automata [12, 27] and have been used for verifying industrial case studies.

Hybrid automata can be seen as an extension of timed automata, but they have in fact been defined and studied roughly at the same time [19]. Though most verification problems for this model are undecidable, many works are devoted to its study, by providing for instance decidable subclasses or approximation algorithms.

Control of timed and hybrid systems. To deal with *open* systems, *i.e.* systems interacting with an environment (which is the case for most embedded systems), model-checking may not be sufficient, and we better need to *control* (or guide) the system so that it satisfies the specification, whatever the environment does. More formally, the *control problem* asks, given a system $\mathcal{S}$ and a specification φ, whether there exists a controller C such that $\mathcal{S}$ guided by C satisfies φ (see [30, 31] for initial papers on the control of discrete event systems). Since the mid-90's, work on the control of real-time systems is flourishing (see all references mentioned along this paper).

*Work partly supported by ACI Cortos, a program of the French ministry of research.

Positioning of this survey. In the literature, several formalisations of the control problem have been proposed, some of them are based on a two-player game formulation where the "controller" plays against the "environment". Mostly, results in one framework can be translated into another framework, but care needs however to be taken. In this paper we focus on "control games", an asymmetric formulation where the "environment" player is somehow more powerful than the "controller" player (the controller has to fix his strategy, and this strategy has to be winning whatever the environment does). This is a framework related to the one considered for instance in [15, 16, 9, 8]. A very close framework based on control maps is considered also in [4, 21]. In the literature, we can find more "concurrent" (and thus symmetric) formulations where both players independently choose a delay and an action to be performed after that delay, and action corresponding to the shortest delay is done [1], or a joint play is obtained with these two choices [20].

Outline of the paper. In Section 2, we will present basic notions on timed systems. In Section 3, we define the problem of control for timed systems we will consider, and explain how the basic safety and reachability control problems can be solved for the class of timed automata. We also shortly explain for which other external specifications languages (like timed automata, temporal logics, etc.) we can solve the control problem. In the previous section, there is an (implicit) hypothesis that the controller has complete information on the system. This is a restrictive hypothesis, which we relax in Section 4: under a partial observability assumption, the control problem becomes very difficult, and even the simplest control problems (reachability and safety) become undecidable, except if we restrict the resources of the controller. In Section 5, we briefly give some results concerning the control of two subclasses of hybrid systems, namely rectangular hybrid automata and o-minimal hybrid automata. We finally give some conclusions and mention some current challenges in Section 6.

2 Timed Automata

2.1 Preliminaries

Timed words. Let $\mathbb{R}_{\geq 0}$ be the set of non-negative reals and $\mathbb{Q}_{\geq 0}$ be the set of non-negative rational numbers. Let Σ be a finite alphabet. A *timed word* over Σ is a (possibly infinite) sequence $\sigma = (a_1, \tau_1)(a_2, \tau_2) \ldots$ over $\Sigma \times \mathbb{R}_{\geq 0}$ such that $\tau_i \leq \tau_{i+1}$ for every $1 \leq i < |\sigma|$ (where $|\sigma|$ denotes the (possibly infinite) length of σ). If σ is infinite, it is *non-Zeno* if the sequence $\{\tau_i\}_{i \in \mathbb{N}}$ is unbounded.

Clocks, operations on clocks. We consider a finite set X of variables, called *clocks*. A *valuation* over X is a mapping $v : X \to \mathbb{R}_{\geq 0}$ which assigns to each clock a time value in $\mathbb{R}_{\geq 0}$. We note $\mathcal{V}_X$ (or $\mathcal{V}$ when it is clear from the context) the set of valuations over X. We use $\vec{0}$ to denote the valuation which sets each clock $x \in X$ to 0. If $t \in \mathbb{R}_{\geq 0}$, the valuation $v + t$ is defined as $(v + t)(x) = v(x) + t$ for all $x \in X$. If Y is a subset of X, the valuation $v[Y \leftarrow 0]$ is defined as: for each clock x, $(v[Y \leftarrow 0])(x) = 0$ if $x \in Y$ and $(v[Y \leftarrow 0])(x) = v(x)$ otherwise.

The set of *(clock) constraints* (or *guards*) over a set of clocks X, denoted $\mathcal{G}(X)$, is given by the syntax "$g \ ::= \ x \sim c \ \mid \ g \wedge g$" where $x \in X$, $c \in \mathbb{Q}_{\geq 0}$ and $\sim$ is one of the comparison operators $<$, $\leq$, $=$, $\geq$, or $>$. We write $v \models g$ if the valuation v satisfies the clock constraint g, and is given by $v \models x \sim c$ if $v(x) \sim c$ and $v \models g_1 \wedge g_2$ if $v \models g_1$ and $v \models g_2$. The set of valuations over X which satisfy a guard $g \in \mathcal{G}(X)$ is denoted by $[\![g]\!]_X$, or simply $[\![g]\!]$ when X is clear from the context.

Timed automata [2, 3]. A *timed automaton* (TA for short) is a 6-tuple $\mathcal{A} = (\Sigma, X, Q, q_0, \longrightarrow, Inv)$ where Σ is a finite alphabet of actions, X is a finite set of clocks, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\longrightarrow \subseteq Q \times \mathcal{G}(X) \times \Sigma \times 2^X \times Q$ is a finite set of transitions, and $Inv : Q \to \mathcal{G}(X)$ assigns an invariant to every state. The timed automaton $\mathcal{A}$ is said to be *deterministic* if (q, g_1, a, Y_1, q_1), $(q, g_2, a, Y_2, q_2) \in \longrightarrow$ implies $[\![g_1]\!] \cap [\![g_2]\!] = \emptyset$. The class of deterministic timed automata is denoted DTA.

A configuration of $\mathcal{A}$ is a pair (q, v) where $q \in Q$ and v si a valuation over the set of clocks X. The timed automaton $\mathcal{A}$ defines a timed transition system, made of timed and discrete transitions. A *timed transition* in $\mathcal{A}$ is a transition of the form $(q, v) \xrightarrow{t} (q, v+t)$ where for every $0 \leq t' \leq t$, $v+t' \models Inv(q)$. A *discrete transition* in $\mathcal{A}$ is a transition of the form $(q, v) \xrightarrow{a} (q', v')$ when there exists $(q, g, a, Y, q') \in \longrightarrow$ such that $v \models g \wedge Inv(q)$, $v' = v[Y \leftarrow 0]$, and $v' \models Inv(q')$.

A run of $\mathcal{A}$ starting in (q_1, v_1) is a finite or infinite sequence of transitions $\rho = (q_1, v_1) \xrightarrow{t_1} (q'_1, v'_1) \xrightarrow{a_1} (q_2, v_2) \xrightarrow{t_2} \cdots$, which alternates between timed and discrete transitions. We will sometimes equivalently write it $\rho = (q_1, v_1) \xrightarrow{a_1, t_1} (q_2, v_2) \xrightarrow{a_2, t_2} \cdots$. We write $tw(\rho)$ the timed word associated to the run ρ, that is the finite or infinite sequence $(a_1, t_1)(a_2, t_2) \cdots$. If ρ is finite and ends in (q_n, v_n) we write $\mathsf{Last}(\rho) = (q_n, v_n)$. We write $\mathsf{Runs}(\mathcal{A})$ (resp. $\mathsf{Runs}_f(\mathcal{A})$) the set of runs (resp. finite runs) in $\mathcal{A}$. We say that $a \in \Sigma$ is *enabled* in (q, v) if there exists (q', v') such that $(q, v) \xrightarrow{a} (q', v')$. We say that λ (a symbol used to express time elapsing) is enabled in (q, v) if there exists (q', v') and $t > 0$ such that $(q, v) \xrightarrow{t} (q', v')$. We write $\mathsf{Enabled}((q, v))$ the subset of $\Sigma \cup \{\lambda\}$ of actions or λ enabled in (q, v).

The scrupulous reader may have noticed that we did not include an accepting condition to timed automata. This is for simplicity, but when necessary (for instance in Subsection 3.3), we will assume that there is an accepting condition in timed automata, for instance a set F of accepting states for finite words, or a set of R of repeated states for defining a Büchi condition for infinite words (and we could consider more general accepting conditions like Muller, Rabin or parity conditions). Acceptance of a timed word is defined classically and we write $L(\mathcal{A})$ the set of timed words accepted by $\mathcal{A}$.

2.2 Region automaton

The *region automaton* $\mathcal{R}(\mathcal{A})$ of a timed automaton $\mathcal{A}$ is a finite automaton, which abstracts timed behaviours of a timed automaton into (untimed) behaviours. This abstraction has been proposed by Alur & Dill in [2, 3] for deciding language emptiness of timed automata. Indeed, each state of the region automaton is a class[1] of an equivalence relation $\equiv_{\mathcal{A}}$ (of finite index) over configurations which is a *time-abstract bisimulation*: if $(q_1, v_1) \equiv_{\mathcal{A}} (q_2, v_2)$ and $(q_1, v_1) \xrightarrow{a} (q_1', v_1')$ with $a \in \Sigma$, then there exists $(q_2, v_2) \xrightarrow{a} (q_2', v_2')$ such that $(q_1', v_1') \equiv_{\mathcal{A}} (q_2', v_2')$; if $(q_1, v_1) \equiv_{\mathcal{A}} (q_2, v_2)$ and $(q_1, v_1) \xrightarrow{t_1} (q_1', v_1')$ for some $t_1 > 0$, then there exists $t_2 > 0$ such that $(q_2, v_2) \xrightarrow{t_2} (q_2', v_2')$ and $(q_1', v_1') \equiv_{\mathcal{A}} (q_2', v_2')$. We will not enter into more details and better refer to [3] for the description of the region automaton construction. However it is worth mentioning that this construction is the core of numerous decidability results for the model of timed automata.

3 Control of Timed Automata

3.1 The control problem

Several formulations of the control problem can be found in the literature. It is sometimes defined as a game between a *controller* and a (possibly nasty) *environment*. The formulation we consider in this paper is an asymmetric game where the environment is more powerful than the controller.

We define the control problem for a timed system given as a timed automaton, but it could be easily generalized to other kinds of systems. A *plant* is a timed automaton $\mathcal{A} = (\Sigma, X, Q, q_0, \longrightarrow, Inv)$ where the alphabet Σ is partitioned into two subsets Σ_c and Σ_u corresponding respectively to controllable and uncontrollable actions. Intuitively, the controller will be able to perform controllable actions, whereas the environment will be able to perform uncontrollable actions.

A *controller strategy* (or simply a *strategy*) is a partial function f from the set $\mathsf{Runs}_f(\mathcal{A})$ to $2^{\Sigma_c \cup \{\lambda\}}$ such that for every finite run ρ such that $f(\rho)$ is defined,

[1] called a *region*.

$f(\rho) \subseteq \mathsf{Enabled}(\mathsf{Last}(\rho))$ and $f(\rho) \neq \emptyset$. The strategy f tells precisely what the controller has to do: if $f(\rho) \subseteq \Sigma_c$, then a discrete controllable action of $f(\rho)$ has to be done, whereas if $\lambda \in f(\rho)$, then it is possible to delay (or a discrete action of $f(\rho)$ can be done as well). Note that a strategy is not deterministic as it may propose a set of actions in $\Sigma_c \cup \{\lambda\}$. In the literature, several other notions of strategies can be found.

A run $\rho = (q_1, v_1) \xrightarrow{a_1,t_1} (q_2, v_2) \xrightarrow{a_2,t_2} \cdots$ is said *compatible with a strategy* f if for all i, writing $\rho_i = (q_1, v_1) \xrightarrow{a_1,t_1} \cdots \xrightarrow{a_{i-1},t_{i-1}} (q_i, v_i)$ we have that for all $0 \leq t' \leq t_i$, $\lambda \in f(\rho_i \xrightarrow{t'} (q_i, v_i + t'))$, and if $a_i \in \Sigma_c$, $a_i \in f(\rho_i \xrightarrow{t_i} (q_i, v_i + t_i))$. A *maximal run w.r.t a strategy* f (or simply *maximal run* when f is clear from the context) is either an infinite run or a run which satisfies: for all $t \geq 0$, if $\rho \xrightarrow{t} (q, v)$ then $\lambda \in f(\rho \xrightarrow{t} (q, v))$.

A strategy f is said *realizable* if for every finite run ρ such that $\lambda \in f(\rho)$, there exists $\delta > 0$ such that for all $0 \leq t < \delta$ if $\rho \xrightarrow{t} (q, v)$ then $\lambda \in f(\rho \xrightarrow{t} (q, v))$. This notion of realizability has been defined in [8] to avoid strategies which have no compatible run, but is relevant for most control problems.

Given a plant $\mathcal{A}$, a *specification* $\mathcal{S}$ for $\mathcal{A}$ is a subset of $\mathsf{Runs}(\mathcal{A})$. Intuitively it corresponds to the desired behaviours of the plant. In the following, we will consider special cases of specifications such as reachability objectives, safety objectives, or external specifications.

If $\mathcal{A}$ is a plant and $\mathcal{S}$ a specification for $\mathcal{A}$, a strategy f is *winning from a configuration (q,v)* if all maximal runs starting in (q, v) compatible with f are in the set $\mathcal{S}$. A configuration (q, v) is said *winning* if there is a realizable winning strategy from (q, v).

We can now formally define the control problem:

Problem (Control problem). Given a plant $\mathcal{A}$, a specification $\mathcal{S}$ and an initial configuration (q, v), determine if there is a realizable winning strategy from (q, v).

A natural further question is to (effectively) construct winning strategies, if one exists. In the rest of the paper, we will not devote much time to that subject, though it is sometimes a non-trivial one.

3.2 Reachability and safety control

In this subsection, we consider two natural types of specifications: reachability and safety properties. Intuitively, for a reachability specification, the goal for the controller is to reach a set of good states, whereas for a safety specification, the controller has to avoid a set of bad states.

Let $\mathcal{A} = (\Sigma, X, Q, q_0, \longrightarrow, Inv)$ be a plant, and let **Good** and **Bad** be two subsets of Q, we define the two following specifications:

$$\begin{cases} \mathcal{S}_{\mathbf{Good}} = \{\rho = (q_1, v_1) \xrightarrow{a_1,t_1} (q_2, v_2) \xrightarrow{a_2,t_2} \cdots \mid \exists i\ q_i \in \mathbf{Good}\} \\ \mathcal{S}_{\mathbf{Bad}} = \{\rho = (q_1, v_1) \xrightarrow{a_1,t_1} (q_2, v_2) \xrightarrow{a_2,t_2} \cdots \mid \forall i\ q_i \notin \mathbf{Bad}\} \end{cases}$$

Problem (Reachability control problem). Given a plant $\mathcal{A}$, a set of states **Good** and an initial configuration (q, v), determine if there is a realizable winning strategy from (q, v) for the specification $\mathcal{S}_{\mathbf{Good}}$.

Problem (Safety control problem). Given a plant $\mathcal{A}$, a set of states **Bad** and an initial configuration (q, v), determine if there is a realizable winning strategy from (q, v) for the specification $\mathcal{S}_{\mathbf{Bad}}$.

We now explain how to solve reachability and safety control problems in the timed framework [4]. A way of computing winning states for these specifications is to compute the *attractor* of goal states by iterating a *controllable predecessor* operator. This is a classical method in the theory of classical (untimed) games for computing winning states [18].

Let $\mathcal{A}$ be a plant. We define the controllable and uncontrollable discrete predecessors of a set of configurations $A \subseteq Q \times \mathcal{V}$ as follows:

$$\mathsf{cPred}(A) = \left\{ (q, v) \in Q \times \mathcal{V} \;\middle|\; \begin{array}{l} \exists c \in \Sigma_c,\ \text{c is enabled in } (q, v), \\ \text{and } \forall (q', v') \in Q \times \mathcal{V}, \\ (q, v) \xrightarrow{c} (q', v') \Rightarrow (q', v') \in A \end{array} \right\}$$

$$\mathsf{uPred}(A) = \left\{ (q, v) \in Q \times \mathcal{V} \;\middle|\; \begin{array}{l} \exists u \in \Sigma_u,\ \exists (q', v') \in Q \times \mathcal{V} \text{ s.t.} \\ (q, v) \xrightarrow{u} (q', v') \text{ and } (q', v') \in A \end{array} \right\}$$

The set $\mathsf{cPred}(A)$ is the set of configurations from which we can enforce a configuration of A by doing a controllable action. The set $\mathsf{uPred}(A)$ is the set of configurations from which the environment can do an uncontrollable action which leads to a configuration in A. In the untimed (finite-state) framework, these two operators are sufficient to define the set of configurations from which we can enforce the set A in one step, this is $\pi(A) = \mathsf{cPred}(A) \setminus \mathsf{uPred}(\overline{A})$. Then, starting from the set of configurations which are good, and iterating the operator π, we can compute the set of configurations from which we can enforce the set of states **Good**. Similarly (or dually), we can also compute the set of configurations from which we can avoid the set of states **Bad**. Note that in the untimed framework, these two problems are dual.

In the timed framework, these two discrete controllable and uncontrollable predecessors are not sufficient, and we need to define a time controllable predecessor operator of a set A of configurations: a state (q, v) is in $\pi(A)$ if and only if

(1) it is possible to let t time units elapse for some $t \geq 0$ and use a controllable action to reach A and no uncontrollable action played before or at t leads outside A; or (2) A can be reached by just letting time elapse and no uncontrollable action leads outside A. Formally the operator π is defined as follows:

$$\pi(A) = \left\{ (q,v) \in Q \times \mathcal{V} \;\middle|\; \begin{array}{l} \quad \exists t\ (q,v) \xrightarrow{t} (q',v'),\ (q',v') \in \mathsf{cPred}(A), \\ \quad \text{and } \mathsf{Post}_{[0,t]}(q,v) \cap \mathsf{uPred}(\overline{A}) = \emptyset; \\ \text{or } \exists t\ \mathsf{Post}_{[t,+\infty)}(q,v) \subseteq A, \\ \quad \text{and } \mathsf{Post}_{[0,+\infty)}(q,v) \cap \mathsf{uPred}(\overline{A}) = \emptyset \end{array} \right\}$$

where $\mathsf{Post}_I(q,v) = \{(q, v+t) \in Inv(q) \ \mid\ t \in I\}$ with I interval of $\mathbb{R}_{\geq 0}$.

We now contruct an increasing and a decreasing version of π to solve respectively reachability and safety control: $\pi_{\text{reach}}(A) = A \cup \pi(A)$ and $\pi_{\text{safe}}(A) = A \cap \pi(A)$. We note $\pi^*_{\text{reach}}(\mathbf{Good}) = \lim\limits_{k \to +\infty} \pi^k_{\text{reach}}(\mathbf{Good})$ and $\pi^*_{\text{safe}}(\mathbf{Bad}) = \lim\limits_{k \to +\infty} \pi^k_{\text{safe}}(\mathbf{Bad})$.

Proposition 1 ([4])**.** *Let $\mathcal{A}$ be a plant, let **Good** (resp. **Bad**) a set of good (resp. bad) states. The set of winning states for the reachability specification $\mathcal{S}_{\mathbf{Good}}$ is $\pi^*_{reach}(\mathbf{Good})$, whereas the set of winning states for the safety specification $\mathcal{S}_{\mathbf{Bad}}$ is $\pi^*_{safe}(\mathbf{Bad})$.*

From this characterization of the set of winning states, we can deduce an algorithm for computing the set of winning states for reachability and safety specifications: indeed it is easy to show that when A is a union of regions (see Subsection 2.2), then $\pi(A)$ is a union of region and can be effectively computed. So the fixed points $\pi^*_{\text{reach}}(\mathbf{Good})$ and $\pi^*_{\text{safe}}(\mathbf{Bad})$ of π_{reach} and π_{safe} respectively can be computed after a finite number of iterations: the set of winning states can thus be effectively computed.

Example. We develop a short example to illustrate how the controllable predecessor operator acts. We assume $\Sigma_c = \{c\}$, $\Sigma_u = \{u\}$, and the following plant with $\mathbf{Good} = \{q_3\}$:

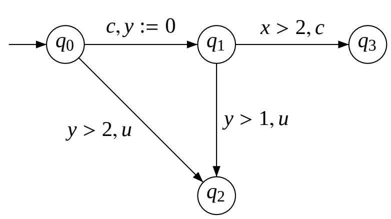

Figure 1: A plant to control

The computation of the fixed point is the following (we note G the set of good configurations we want to enforce, *i.e.* $G = \{(q_3, (x, y)) \mid x \geq 0 \text{ and } y \geq 0\}$):

$$\begin{cases} \pi_{reach}(G) &= G \cup \{(q_1, (x, y)) \mid y \leq 1 \text{ and } x - y > 1\} \\ \pi^2_{reach}(G) &= \pi_{reach}(G) \cup \{(q_0, (x, y)) \mid y \leq 2 \text{ and } y - x < 1\} \\ \pi^*_{reach}(G) &= \pi^2_{reach}(G) \end{cases}$$

The set of states from which the controller can enforce state q_3 is thus $\pi^*_{reach}(G)$, as described above. Thus, the initial configuration $(q_0, \vec{0})$ is winning. □

From the above fixed point computation, we can extract winning strategies (like in the untimed framework [18]). However, strategies which are extracted that way may not be realizable: this is for instance the case in the previous example where the extracted strategy says "λ" on $(q_1, (x = 2, y))$ (with $y < 1$) and "c" on $(q_1, (x > 2, y))$ (with $y \leq 1$). It is then necessary to make the strategy realizable [8].[2]

Moreover by adapting the PSPACE-hardness proof of the reachability problem in timed automata, one can show that reachability and safety control of timed automata are EXPTIME-hard [21]. As the above-mentioned fixed points can be computed in exponential time (because there is an exponential number of regions [3]), we have the following theorem:

Theorem 2. *Reachability and safety control problems in timed automata are EXPTIME-complete.*

3.3 External specifications

In the previous subsection we have investigated the control of timed automata for reachability and safety specifications; these are internal specifications as the winning condition is given on states of the plant itself. The control problem for various external specifications has also been considered.

External specifications given as timed automata. A natural external specification is one given by a timed automaton. Let $\mathcal{A}$ be a plant and $\mathcal{B}$ a timed automaton with an accepting condition for infinite words. We can see $\mathcal{B}$ as a specification for the plant $\mathcal{A}$ by defining $\mathcal{S}_{\mathcal{B}} = \{\rho \in \mathsf{Runs}(\mathcal{A}) \mid tw(\rho) \in L(\mathcal{B})\}$.[3]

For this kind of specifications, the following decidability results have been proved:

[2] Note that the new realizable strategy might not be expressible as a timed automaton, even if the former was definable as a timed automaton.

[3] Note that negative specifications like $\mathcal{S}^-_{\mathcal{B}} = \{\rho \in \mathsf{Runs}(\mathcal{A}) \mid tw(\rho) \notin L(\mathcal{B})\}$ have also been considered in [15].

Theorem 3 ([15]). *The control problem for specifications given as timed automata over infinite words is undecidable. The control problem for specifications given as deterministic timed automata over infinite words is decidable and can be solved in 2EXPTIME.*

The control problem for specifications given as (non-deterministic) timed automata is undecidable as inclusion of timed automata can be reduced to that problem. However a natural assumption when synthesizing timed systems is to restrict the power of the controller by looking for a controller which is a timed automaton using a fixed number of clocks and a fixed set of constants. In that case, we say that we fix the *granularity* of the controller. This assumption is done for instance for solving the satisfiability problem of the logic L_ν [26] or for various control problems [15, 9, 6].

A granularity is a triple $\mu = (k, m, K)$ where k, m, K are integers. A timed automaton is said of granularity μ if it uses k clocks and constants mentioned in its constraints are of the form $\frac{i}{m}$ with $0 \leq i \leq K$. Let $\mathcal{A}$ be a plant, a strategy f for $\mathcal{A}$ is said *μ-granular* if it can be represented by a deterministic timed automaton $\mathcal{B}$ of granularity μ. Formally if ρ is a run of $\mathcal{A}$ compatible with strategy f and ρ' is the unique corresponding run in $\mathcal{B}$ then $f(\rho) = \mathsf{Enabled}(\mathsf{Last}(\rho')) \cap (\Sigma_c \cup \{\lambda\})$.

The DTA$_\mu$-control problem for a specification given as a timed automaton over infinite words then asks, given a plant $\mathcal{A}$, a granularity μ, a timed automata over infinite words $\mathcal{B}$ and an initial configuration (q, v), whether there is a μ-granular realizable winning strategy from (q, v) for the specification $\mathcal{S}_\mathcal{B}$.

Theorem 4 ([15]). *The DTA$_\mu$-control problem for specifications given as timed automata over infinite words is 2EXPTIME-complete.*

This result can be proved by reducing this control problem to a parity game over a finite automaton (this automaton is obtained using some region construction).

External specifications given as formulas of (timed) temporal logics. Various (timed) temporal logics have been used as specification languages for control problems. We first focus on linear-time (timed) temporal logics. If ϕ is a formula of some linear-time (timed) temporal logic, we write $\mathcal{S}_\phi = \{\rho \in \mathsf{Runs}(\mathcal{A}) \mid tw(\rho) \in L(\phi)\}$ for the specification defined by ϕ (where $L(\phi)$ is the set of models of ϕ).

We assume the reader is familiar with the linear-time temporal logic LTL, and better refer to [29] for the definition of this logic. The control problem for specifications given as LTL formulas has been studied in [16]:

Theorem 5 ([16]). *The control problem over timed automata for specifications given as LTL formulas is 2EXPTIME-complete.*

The technique used to prove this result relies on the construction of a tree automaton (based on regions) which accepts all winning strategies.

The logic MTL [23] is a timed extension of LTL with time constraints on modalities. For instance, in this logic, we can write bounded response time properties like $\Box(p \rightarrow \Diamond_{\leq 5} q)$ which says that every time p holds, q has to hold within a time window of 5 time units. Similarly to specifications given as timed automata, the control problem is undecidable for specifications given as MTL formulas but becomes decidable if we restrict to μ-granular strategies. However in this case the complexity is much higher:

Theorem 6 ([6])**.** *The control problem for* MTL*-specifications is undecidable. The* DTA_μ*-control problem for* MTL*-specifications is decidable and has non-primitive recursive complexity.*

It is worth reminding that already model-checking of MTL is non-primitive recursive [28], and that the halting problem of a lossy-channel system can be simulated by an MTL model-checking problem. Roughly, using the power of the controller, we can simulate with a control problem the halting problem of a perfect channel system, which leads to undecidability. The decidability when fixing the granularity of the controller is not based on regions, but on a better-quasi-order.

Branching-time (timed) logics have also been considered in the literature. For lack of space, and because it does not perfectly fit our definition of specifications,, we do not enter into details and better point out the corresponding references: [16, 17]. Finally, games for specifications given as formulas of the μ-calculus have been studied in the rather different framework of symmetric timed games [1].

4 Partial Observability

In the previous section we have supposed that the controller has perfect information on the plant: at any time, the controller knows in which state the plant is. In this section, we consider the more general problem of control under partial observability: the controller has only partial information about the plant and should control it whatever is the behaviour (observable or not) of the environment.

In this section we suppose that Σ_u is partionned into two subsets Σ_u^{obs} and Σ_u^{unobs}. The actions of Σ_u^{obs} are uncontrollable but observable, whereas the actions of Σ_u^{unobs} are not observable (and thus not controllable). The controllable actions of Σ_c remains observable.

We define a consistent strategy for the controller as a strategy which depends only on observations of actions in Σ_c and Σ_u^{obs}. Let π_{obs} be the projection of timed words on $\Sigma_c \cup \Sigma_u^{obs}$ (which is defined in a natural way by erasing actions in Σ_u^{unobs}

and their corresponding dates), a strategy is said *consistent* if for all runs ρ and ρ' such that $\pi_{obs}(tw(\rho)) = \pi_{obs}(tw(\rho'))$, then $f(\rho)$ and $f(\rho')$ are simultaneously (un-)defined, and when they are defined, $f(\rho) = f(\rho')$.

The control problem under partial observability is thus the following:

Problem (Control problem under partial observability). Given a plant $\mathcal{A}$, a specification $\mathcal{S}$ and an initial configuration (q, v), determine if there is a consistent realizable winning strategy from (q, v).

If we consider specifications given as timed automata, the control under partial observability is undecidable even for deterministic specifications. On the other side, the decidability proof for the DTA$_\mu$ control problem can be extended to partial observability:

Theorem 7 ([9]). *The control problem under partial observability for specifications given as deterministic timed automata over infinite words is undecidable. The control problem under partial observability for specifications given as deterministic timed automata over infinite words is decidable and 2EXPTIME-complete.*

Indeed, techniques used in [9] can be easily extended to show that reachability control under partial observability is undecidable. However it cannot be applied to safety control under partial observability. We present here an original construction which shows that safety control under partial observability is undecidable when we consider non-Zeno controllers, *i.e.* controllers which generate only non-Zeno behaviours (the fact that the controller has to be non-Zeno will be encoded in the specification).

Given a plant $\mathcal{A}$ and a set **Bad** of bad states we consider the specification $\mathcal{S}^{\neg Zeno}_{\mathbf{Bad}} = \mathcal{S}_{\mathbf{Bad}} \cap \{\rho \mid tw(\rho) \text{ is non-Zeno}\}$.[4] The safety control problem under partial observability for non-Zeno controllers is given a plant $\mathcal{A}$, a set of states **Bad** and an initial configuration (q, v), determine if there is a realizable winning strategy from (q, v) for the specification $\mathcal{S}^{\neg Zeno}_{\mathbf{Bad}}$.

Theorem 8. *The safety control problem under partial observability for non-Zeno controllers is undecidable.*

The proof consists in reducing the halting problem of a 2-counter machine and uses roughly the same encoding as the undecidability of the universality of timed automata [3]. A configuration of a 2-counter machine with n states is encoded by a timed word over the alphabet $\{b_1, \cdots, b_n, a_1, a_2, a_3\}$. We will encode the first configuration of the 2-counter machine within the time interval $[0, 1[$, and the time interval $[i, i+1[$ will contain the encoding of the i^{th} configuration of the

[4] recall that $\mathcal{S}_{\mathbf{Bad}}$ is the set of runs which avoid **Bad**.

execution of the 2-counter machine. If the 2-counter machine is in state j and counter values are c and d then the corresponding timed word should contain the actions $b_j a_1^c a_2^d a_3^e$ between time i and $i+1$ (the use of a_3 is explained later). To express for example that the value of counter 1 does not change between the i^{th} and the $(i+1)^{\text{th}}$ configurations we require that every a_1 of the interval $[i, i+1[$ has a matching a_1 one time unit later and that there is no a_1 in $[i+1, i+2[$ without a matching a_1 one time unit earlier. Similarly, to express that the first counter has decreased by one, we check that all a_1's apart the last one in the interval $[i, i+1[$ has a matching one time unit later in $[i+1, i+2[$, and every a_1 in the interval $[i+1, i+2[$ is matched one time unit before by another a_1.

The use of action a_3 is as follows: we require that the first configuration is encoded by $b_1 a_3^e$ for some $e \in \mathbb{N}$; then, after each configuration the number of a_3 must be decreased by one.

The plant $\mathcal{A}$ is the universal timed automaton. The set of controllable actions is $\Sigma_c = \{b_1, \cdots, b_n, a_1, a_2, a_3\}$: the controller will play an encoding of the execution of the 2-counter machine freely. We use a single unobservable action (we take $\Sigma_u^{obs} = \emptyset$ and $\Sigma_u^{unobs} = \{u\}$) which can be used by the environment to check whether the encoding played by the controller is correct. Non-deterministically and in an unobservable way, the environment launches a *test* to check that the controller has simulated the 2-counter machine correctly (that is it checks that all actions are matched correctly one time unit later or earlier, as explained above).

If the environment discovers that the encoding is incorrect it goes to a bad state and the controller loses. So the controller has to play a correct encoding for winning. Moreover if the number of a_3 played during an interval $[i, i+1[$ is null and the goal state of the 2-counter machine has not been reached, then the plant also goes into a bad state. So to control the plant properly, during the first time unit the controller must play at least n a_3 where n is the number of steps needed to reach q. Then it just has to play a correct encoding leading to q to win the control game.

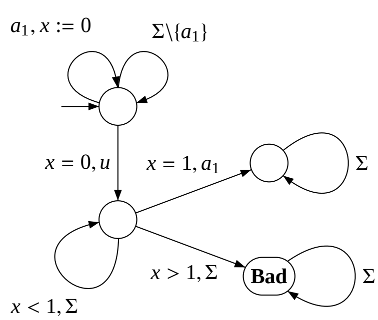

Figure 2: Environment checking that every a_1 is matched one t.u. later by an a_1

On Figure 2, we give an example of how the environment can check that every a_1 is matched one time unit later by another a_1. Note that this construction works only because u is not observable: as the controller cannot know when it is played it has to always simulate correctly the 2-counter machine.

Note that this proof shows that both control by general strategies and control by strategies defined by DTA (with non-fixed granularity) are undecidable. In fact to show that control under partial observability by general strategies a slighty simpler proof not involving a_3 actions can be done.

Recently, another framework for control under partial observability has been developed [14], where observations are based on the visited states. Until now, this work applies only to finite-state automata, and discrete-time timed (and even hybrid) automata.

5 Control of Hybrid Systems

Hybrid automata are an extension of timed automata in which variables are not clocks but are more general: they can for instance follow rules given by differential equations. This model, though immediately undecidable, is widely used and studied (this is the topic of the conference HSCC which takes place every year since 1998). The literature on that subject is substantial, we can thus not give an overview of all results on that subject, we thus select two kinds of models for which decidability results can be obtained.

5.1 Rectangular hybrid automata

The model of rectangular hybrid automata is an extension of the model of timed automata. It thus extends finite automata with real-valued variables, the enabling condition for each discrete move is a cartesian product of intervals, and the first derivative of each continuous variable is bounded by constants from below and above. Checking reachability properties in that model is undecidable unless we assume that they are initialized [22], which means that every transition changing the slope of a variable has to reset it.

In that context, the control problem for safety specifications is undecidable for rectangular timed automata [22]. Several simpler control problems have thus been considered.

Sampling control. In this framework, the controller performs actions every time unit (or every δ if δ is the sampling rate of the controller). When the sampling rate is known in advance, the safety sampling control problem for rectangular hybrid automata is decidable [21]. When the sampling rate is not known *a priori*, the problems becomes undecidable [11]. Recently a new notion of sampling has been proposed [24], and it would be interesting to check whether the control problem can become decidable in this (more natural) framework.

Time-abstract restriction. In these games, when the strategy is to wait, it is not possible to bound the time which is waited, and the environment chooses when the next discrete action happens. In this framework, the strategy to wait is viewed as a discrete action, and this somehow "*untimes*" the system. Under this time-abstract restriction, rectangular hybrid games are decidable [13], even for specifications given as LTL formulas [20].

5.2 O-minimal hybrid automata

O-minimal systems [25] are hybrid systems which have very rich continuous dynamics (for instance polynomial and exponential functions can be used), but have limited discrete behaviours (at each discrete step, all variables have to be reset).

The control of such systems is not always decidable as o-minimal structures (and thus reachability) are not necessarily decidable. However if we restrict to decidable structures the (safety and reachability) control of o-minimal hybrid systems is decidable [7].

The technique of the proof is standard as it uses a computation of controllable predecessors over a symbolic representation of the state-space. However there is a major difference with the case of timed automata: in o-minimal hybrid systems, time-abstract bisimulation is not the right tool to compute the set of winning states. A finer bisimulation called *suffix-partition* has to be used to prove decidability. Moreover if the system is controllable, a strategy can be defined in the underlying structure.

6 Conclusion

The control of timed and hybrid has been the core of much research these last ten years. In this paper, we have focused on a formulation of the control problem which is an asymmetric two-player game between the controller and the environment. We have given several results concerning the control problem, from the simplest reachability and safety control problems of timed automata to more involved control problems like the control for external specifications, under partial observability, or for more general systems like subclasses of hybrid automata.

These last years, an extension of timed automata with costs has been studied, which can be used for modeling resource consumption in timed systems. These automata are simple linear hybrid automata with a single hybrid variable (which is an "observer" variable), and can used in a formulation of timed games with an optimization criterion on the cost. Those kinds of games have direct applications for modeling for instance scheduling problems, and have been recently much studied. We refer to [5] for a recent survey on this model.

In 2005, the tool Uppaal TiGA has been developed, which solves the safety control problems of timed automata, symbolically and in a forward manner [10].

References

[1] L. d. Alfaro, M. Faella, Th. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *Proc. 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003.

[2] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier Science, 1998.

[5] P. Bouyer. Weighted timed automata: Model-checking and games. In *Proc. 22nd Conf. Mathematical Foundations of Programming Semantics (MFPS XXII)*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 3–17. Elsevier, 2006. Invited paper.

[6] P. Bouyer, L. Bozzelli, and F. Chevalier. Controller synthesis for MTL specifications. Submitted, 2006.

[7] P. Bouyer, Th. Brihaye, and F. Chevalier. Control in o-minimal hybrid systems. In *Proc. 21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*. IEEE Computer Society Press, 2006. To appear.

[8] P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In *Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'04)*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, 2004.

[9] P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In *Proc. 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2003.

[10] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *Proc. 16th International Conference on Concurrency Theory (CONCUR'2005)*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005.

[11] F. Cassez, Th. A. Henzinger, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In *Proc. 5th International Workshop on Hybrid Systems: Computation and Control (HSCC'02)*, volume 2289 of *LNCS*, pages 134–148. Springer, 2002.

[12] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Proc. Hybrid Systems III: Verification and Control (1995)*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1996.

[13] L. de Alfaro, Th. A. Henzinger, and R. Majumdar. Symbolic algorithms for infinite-state games. In *Proc. 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 536–550. Springer, 2001.

[14] M. De Wulf, L. Doyen, and J.-F. Raskin. A lattice theory for solving games of imperfect information. In *Proc. 8th International Workshop on Hybrid Systems: Computation and Control (HSCC'06)*, volume 3927 of *LNCS*, pages 153–168. Springer, 2006.

[15] D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *Proc. 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 571–582. Springer, 2002.

[16] M. Faella, S. La Torre, and A. Murano. Automata-theoretic decision of timed games. In *Proc. 3rd International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI'02)*, volume 2294 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2002.

[17] M. Faella, S. La Torre, and A. Murano. Dense real-time games. In *Proc. 17th Annual Symposium on Logic in Computer Science (LICS'02)*, pages 167–176. IEEE Computer Society Press, 2002.

[18] E. Grädel, W. Thomas, and Th. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[19] Th. A. Henzinger. The theory of hybrid automata. In *Proc. 11th Annual Symposim on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.

[20] Th. A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 1999.

[21] Th. A. Henzinger and P. W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221:369–392, 1999.

[22] Th. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

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

[24] P. Krčál and R. Pelánek. On sampled semantics of timed systems. In *Proc. 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 310–321. Springer, 2005.

[25] G. Lafferriere, G. J. Pappas, and S. Sastry. O-minimal hybrid systems. *Mathematics of Control, Signals, and Systems*, 13(1):1–21, 2000.

[26] F. Laroussinie, K. G. Larsen, and C. Weise. From timed automata to logic – and back. In *Proc. 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, volume 969 of *Lecture Notes in Computer Science*, pages 529–539. Springer, 1995.

[27] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

[28] J. Ouaknine and J. B. Worrell. On the decidability of metric temporal logic. In *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'05)*, pages 188–197. IEEE Computer Society Press, 2005.

[29] A. Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[30] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symposium on Principles of Programming Languages (POPL'89)*, pages 179–190. ACM, 1989.

[31] P. Ramadge and W. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77(1):81–98, 1989.