On the Synthesis of Discrete Controllers for Timed Systems (An Extended Abstract).

Conference Paper · March 1995

DOI: 10.1007/3-540-59042-0_76 · Source: DBLP

3 authors, including:

Oded Maler
French National Centre for Scientific Research
236 PUBLICATIONS **10,826** CITATIONS

SEE PROFILE

Joseph Sifakis
Université Grenoble Alpes
295 PUBLICATIONS **16,741** CITATIONS

SEE PROFILE

On the Synthesis of Discrete Controllers for Timed Systems*
(An Extended Abstract)

Oded Maler[1] Amir Pnueli[2] Joseph Sifakis[1]

[1] Spectre – Verimag, Miniparc-zirst, 38330 Montbonnot, France,
Oded.Maler@imag.fr
[2] Dept. of Computer Science, Weizmann Inst. Rehovot 76100, Israel,
amir@wisdom.weizmann.ac.il

Abstract. This paper presents algorithms for the automatic synthesis of real-time controllers by finding a winning strategy for certain games defined by the timed-automata of Alur and Dill. In such games, the outcome depends on the players' actions as well as on their *timing*. We believe that these results will pave the way for the application of program synthesis techniques to the construction of real-time embedded systems from their specifications.

1 Introduction

Consider a dynamical system P whose presentation describes all its possible behaviors. This system can be viewed as a plant to be controlled. A subset of the plant's behaviors, satisfying some criterion is defined as good (or acceptable). If the plant is, e.g., an airplane, this subset might consist of all behaviors which start at the departure point and end in the destination within some temporal interval (and with the number of living passengers kept constant). A controller C is another system which can interact with P in a certain manner by observing the state of P and by issuing control actions that influence the behavior of P, hopefully restricting it to be included in the subset of good behaviors. Carrying on with the airplane example, the controller might be in charge for turning the engines on and off, increasing the fuel consumption, etc. The synthesis problem is then, to find out whether, for a given P, there exists a realizable controller C such that their interaction will produce only good behaviors.

There are many variants of the formulation of this problem, differing from each other in the kind of dynamics considered and in the way the system and the goodness criteria are specified. The two most extreme examples are *reactive program synthesis* and *classical control theory*. In the former, the models are based

* This research was supported in part by the France-Israel project for cooperation in Computer Science and by the European Community ESPRIT Basic Research Action Projects REACT (6021). Verimag is a joint laboratory of cnrs, inpg, ujf and verilog sa. spectre is a project of inria.

on discrete transition-systems (automata). The plant P represents a combination of the environment actions and the specification of the desired interaction between the synthesized program C and the environment. The dynamics for this case is defined by a non-deterministic automaton, whose acceptance condition distinguishes good behaviors (interactions) from bad ones. The program has control over some of the transitions, and the problem is to find a strategy, that is, an effective rule for selecting at each state one among the possible transitions, such that bad behaviors are excluded. In classical control theory, the plant is a continuous dynamical system defined by a non-autonomous (that is, with input) differential equation. The input serves to express both the non-determinism of the environment (disturbances) and the effect of the controller actions. The controller synthesis problem in this context is to define a *feed-back law*, which *continuously* determines the controller's input to the plant, such that behavioral specification are met.

In this paper we are concerned with *real-time systems* where discrete state-transitions interact with the continuous passage of time. In this setting we model the plant as a *timed automaton*, that is, an automaton equipped with clocks that grow continuously in time while the automaton is in any of its states. The values of the clocks may interfere with the transitions by appearing in *guards*, which are the enabling conditions of the transitions. Thus, a transition may take place, for example, only if some clock value has passed a certain threshold. Transitions may as well reset clocks.

We show that the control synthesis problem is solvable when the plant specification is given by a timed automaton. This means that another timed automaton can be synthesized (when possible) such that its interaction with the environment will introduce only good timed traces. We arrive to these results by providing at first a simple and intuitive (if not new) solution to the discrete version of the problem and then adapt it to timed automata defined over a *dense* time domains. Technically, the solution is obtained by solving fixed-point equations involving both discrete transition relations and linear inequalities.

The rest of the paper is organized as follows: in section 2 we give a motivating toy example of a real-time controller synthesis problem. In section 3 we treat the discrete case and the real-time case is solved in section 4. The last section discusses some relevant past and future work.

2 An Example of a Real-time Control Synthesis problem

Consider the following game depicted in figure 1. Player P_1 starts running from the initial position marked by a circle at the left of the figure. It takes her e_1 seconds to reach the junction. At the junction she can either wait and do nothing or choose to run – either to the left or to the right. After having run in the chosen direction for e_2 seconds she reaches the corresponding bridge, and if the latter is not blocked (see below) she can run and reach the end position within e_3 seconds. If this final position is reached within less than c seconds since the game started, Player P_1 wins. The other player P_2 starts the game located between the two

bridges and within d seconds she *must* choose between blocking the left or the right bridge (which she does immediately upon making the decision). The whole

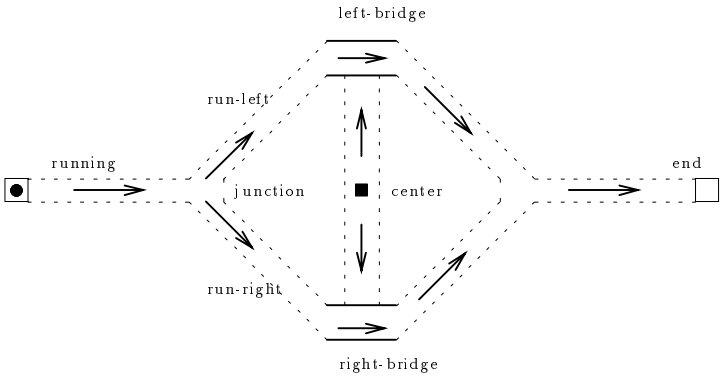

Fig. 1. A pursuit game.

system can be described as a product of two timed-automata having two clocks. The first clock x measures the amount of time elapsed since the beginning of the game. The second clock, y is used to impose velocity constraints on the behavior of P_1. These two automata are depicted[3] in figure 2. The product of these two automata is the timed-automaton of figure 3. One can see that there is a strategy to win from state AJ iff $\max(d, e_1) + e_2 + e_3 < c$. If this inequality holds, the strategy for Player P_1 is to stay at BJ until $x = \max(d, e_1)$. Then, after Player P_2 takes the system to either BI or BK, Player P_1 takes the transition to CI or DK, respectively and reaches the goal on time.

3 The Discrete Case

Definition 1 (Plant). *A plant automaton is a tuple $\mathcal{P} = (Q, \Sigma_c, \delta, q_0)$, where Q is a finite set of states, Σ_c is a set of controller commands, $\delta : Q \times \Sigma_c \mapsto 2^Q$ is the transition function and $q_0 \in Q$ is an initial state.*

For each controller command $\sigma \in \Sigma_c$ at some state $q \in Q$ there are several possible consequences denoted by $\delta(q, \sigma)$. This set indicates the possible reactions

[3] In order to economize the use of arrows we employ a Statechart-like notation, i.e., an arrow originating from a super-state (dashed rectangle) represents identical arrows coming out of all the states contained in that super-state. We write transition guards above the arrows, clock resettings below and invariants inside states (as in state J of the first automaton). Note that the right automaton "observes" both clocks as well as the state of the left automaton.

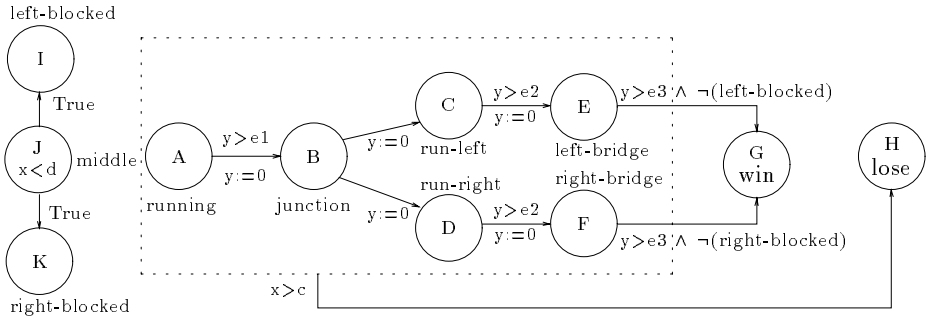

Fig. 2. The game as two interacting timed-automata.

of the plant to the controller's command, some of which may reflect a possible interference by the environment.[4] Choosing a command $\sigma \in \Sigma_c$ is what the controller can do at every stage of the game. A good choice is supposed to work for all possible continuations indicated by $\delta(q, \sigma)$.

Definition2 (Controllers). *A controller (strategy) for a plant specified by* $\mathcal{P} = (Q, \Sigma_c, \delta, q_0)$ *is a function* $C : Q^+ \mapsto \Sigma_c$. *A* simple *controller is a controller that can be written*[5] *as a function* $C : Q \mapsto \Sigma_c$.

According to this definition, a general controller may base its selection of the next command upon the complete history of states visited up to this point. This general definition does not impose any bound on the amount of memory the controller might need in order to implement the strategy. We are interested in the simpler cases of controllers that base their decisions on a finite memory, a special case of which is the simple controller where the decision is based on the last visited automaton state. Such controllers need only observe the current state of the plant.

For an infinite sequence of states $\alpha : q[0], q[1], \ldots$ and a natural number n, we denote by $\alpha[0..n]$ the finite sequence $q[0], q[1], \ldots, q[n]$.

Definition3 (Trajectories). *Let* $\mathcal{P}$ *be a plant and let* $C : Q^+ \mapsto \Sigma_c$ *be a controller. An infinite sequence of states* $\alpha : q[0], q[1], \ldots$ *such that* $q[0] = q_0$ *is called a trajectory of* $\mathcal{P}$ *if*

$$q[i+1] \in \bigcup_{\sigma \in \Sigma_c} \delta(q[i], \sigma)$$

[4] Unlike other formulation of 2-person games where there is an explicit description of the transition function of both players, here we represent the response of the second player (the environment) as a non-deterministic choice among the transitions labeled by the same σ.

[5] Which means that for every $q \in Q, w, w' \in Q^*$, $C(wq) = C(w'q)$.

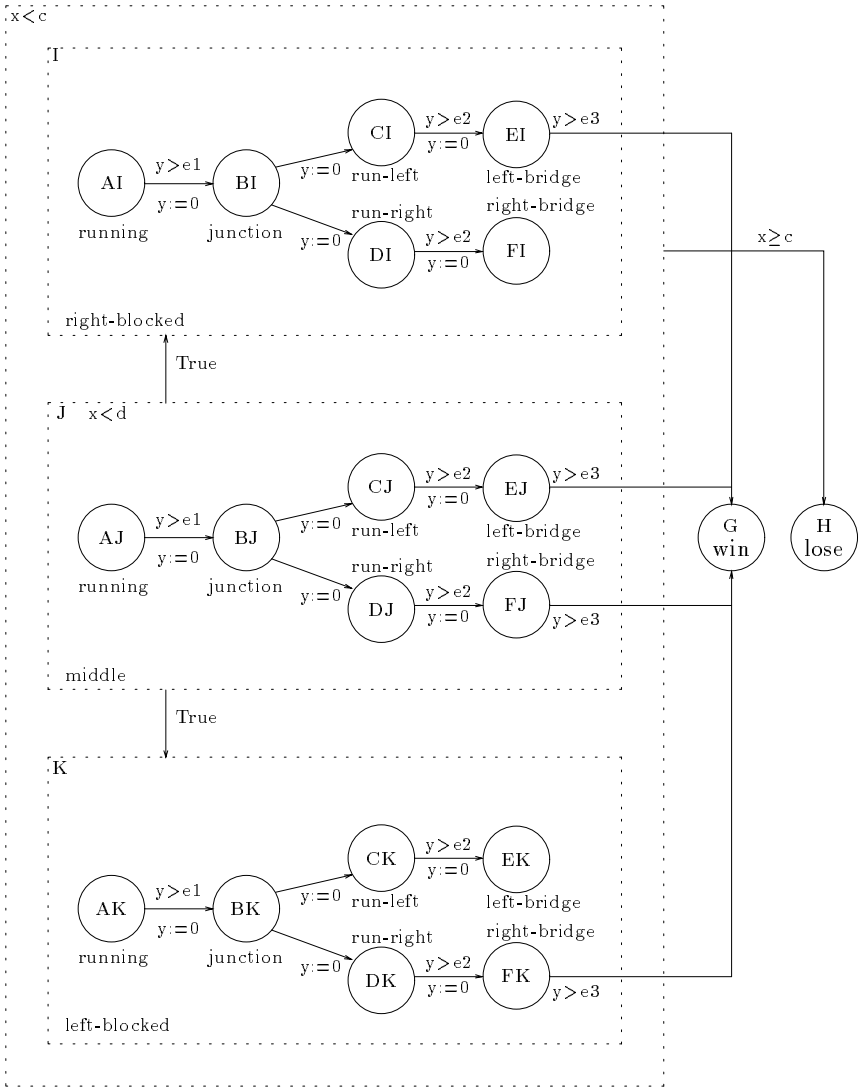

Fig. 3. The game described globally.

and a C-trajectory if $q[i+1] \in \delta(q[i], C(\alpha[0..i]))$ for every $i \geq 0$. The corresponding sets of trajectories are denoted by $L(\mathcal{P})$ and $L_C(\mathcal{P})$.

Clearly, every C-trajectory is a trajectory and $L_C(\mathcal{P}) \subseteq L(\mathcal{P})$. What remains is to define acceptance conditions, that is, a criterion to classify trajectories as good or bad. For every infinite trajectory $\alpha \in L(\mathcal{P})$, we let $Vis(\alpha)$ denote the set of all states appearing in α and let $Inf(\alpha)$ denote the set of all states appearing in α infinitely many times.

Definition 4 (Acceptance Condition). *Let $\mathcal{P} = (Q, \Sigma_c, \delta, q_0)$ be a plant. An acceptance condition for $\mathcal{P}$ is*

$$\Omega \in \{(F, \Diamond), (F, \Box), (F, \Diamond\Box), (F, \Box\Diamond), (\mathcal{F}, \mathcal{R}_n)\}$$

where $\mathcal{F} = \{(F_i, G_i)\}_{i=1}^{n}$ (Rabin condition) *and F, F_i and G_i are certain subsets of Q referred as the* good states. *The set of sequences of $\mathcal{P}$ that are accepted according to Ω is defined as follows:*

$L(\mathcal{P}, F, \Box)$	$\{\alpha \in L(\mathcal{P}) : Vis(\alpha) \subseteq F\}$	*α always remains in F*
$L(\mathcal{P}, F, \Diamond)$	$\{\alpha \in L(\mathcal{P}) : Vis(\alpha) \cap F \neq \emptyset\}$	*α eventually visits F*
$L(\mathcal{P}, F, \Diamond\Box)$	$\{\alpha \in L(\mathcal{P}) : Inf(\alpha) \subseteq F\}$	*α eventually remains in F*
$L(\mathcal{P}, F, \Box\Diamond)$	$\{\alpha \in L(\mathcal{P}) : Inf(\alpha) \cap F \neq \emptyset\}$	*α visits F infinitely often*
$L(\mathcal{P}, \mathcal{F}, \mathcal{R}_n)$	$\{\alpha \in L(\mathcal{P}) : \exists i.$ $\alpha \in L(\mathcal{P}, F_i, \Box\Diamond) \cap L(\mathcal{P}, G_i, \Diamond\Box)\}$	*α visits F_i infinitely often and eventually stays in G_i*

Definition 5 (Controller Synthesis Problem). *For a plant $\mathcal{P}$ and an acceptance condition Ω, the problem* **Synth**$(\mathcal{P}, \Omega)$ *is: Find a controller C such that $L_C(\mathcal{P}) \subseteq L(\mathcal{P}, \Omega)$ or otherwise show that such a controller does not exist.*

In case the answer is positive we say that $(\mathcal{P}, \Omega)$ is controllable.

Definition 6 (Controllable Predecessors). *Let $\mathcal{P} = (Q, \Sigma_c, \delta, q_0)$ be a plant. We define a function $\pi : 2^Q \mapsto 2^Q$, mapping a set of states $P \subseteq Q$ into the set of its* controllable predecessors, *i.e., the set of states from which the controller can "force" the plant into P in one step:*

$$\pi(P) = \{q \ : \ \exists \sigma \in \Sigma_c \ . \delta(q, \sigma) \subseteq P\}$$

Theorem 1. *For every $\Omega \in \{(F, \Diamond), (F, \Box), (F, \Diamond\Box), (F, \Box\Diamond), (\mathcal{F}, \mathcal{R}_1)\}$ the problem* **Synth**$(\mathcal{P}, \Omega)$ *is solvable. Moreover, if $(\mathcal{P}, \Omega)$ is controllable then it is controllable by a simple controller.*[6]

Sketch of Proof: Let us denote by W the set of *winning* states, namely, the set of states from which a controller can enforce good behaviors (according to criterion Ω). They can be characterized by the following fixed-point expressions:

[6] A first variant of this theorem has been proved by Büchi and Landweber in [BL69] with respect to the more general Muller acceptance condition. In that case the controller is finite-state but not necessarily simple. The fact that games defined by Rabin condition can be won using a simple (memory-less) strategies is implicit in various papers on decision problems for tree-automata [V94]. In a more extensive version of the paper, we hope to present an alternative and more systematic treatment of winning strategies with respect to Boolean combinations of acceptance conditions as well as a fixed-point characterization of $\mathcal{R}_n$ for a general n. Impatient or pessimistic readers may look at such a characterization in [TW94a].

$$\Box : \nu W \Big(F \cap \pi(W) \Big) \tag{1}$$

$$\Diamond : \mu W \Big(F \cup \pi(W) \Big) \tag{2}$$

$$\Diamond\Box : \mu W \nu H \Big(\pi(H) \cap \big(F \cup \pi(W) \big) \Big) \tag{3}$$

$$\Box\Diamond : \nu W \mu H \Big(\pi(H) \cup \big(F \cap \pi(W) \big) \Big) \tag{4}$$

$$\mathcal{R}_1 : \mu W \Big\{ \pi(W) \ \cup \ \nu Y \mu H \, . W \cup G \cap \Big(\pi(H) \cup \big(F \cap \pi(Y) \big) \Big) \Big\} \tag{5}$$

The plant is controllable iff $q_0 \in W$.

For a given plant and the induced controllable predecessor function π, it is straightforward to calculate the sets W defined by the fixed-point expressions (1)–(5) and check whether $q_0 \in W$. However, it may be illuminating to review the iterative process by which these sets are calculated.

Consider the cases $\Diamond$ and $\Box$: the iteration processes can be described, respectively, by the following schemes:

$\Diamond$:
$W_0 := \emptyset$
for $i = 0, 1, \ldots,$ **repeat**
  $W_{i+1} := F \cup \pi(W_i)$
until $W_{i+1} = W_i$

$\Box$:
$W_0 := Q$
for $i = 0, 1, \ldots,$ **repeat**
  $W_{i+1} := F \cap \pi(W_i)$
until $W_{i+1} = W_i$

For the $\Diamond$-case, each W_i contains the states from which a visit to F can be enforced after at most i steps and in the $\Box$-case, it consists of the states from which the plant can be kept in F for at least i steps. The sequences $\{W_i\}$ are monotone over a finite domain and hence convergence is guaranteed.

This procedure is constructive in the following sense: in the process of calculating W_{i+1}, whenever we add a state q to W_i (or do not remove it from W_i in the $\Box$-case) there must be at least one action $\sigma \in \Sigma_c$ such that $\delta(q, \sigma) \subseteq W_i$. So we define the controller at q as $C(q) = \sigma$. When the process terminates the controller is synthesized for all the winning states. It can be seen that if the process fails, that is, $q_0 \notin W$, then for every controller command there is a possibly bad consequence that will put the system outside F, and no controller, even an infinite-state one, can prevent this.

Consider now the case $\Box\Diamond$: this case calls for nested iterations; an external major iteration varying the values of the set W, and a nested minor iteration varying the values of the set H. This double iteration is described by the following scheme:

$W_0 := Q$
for $i = 0, 1, \ldots,$ **repeat**
$\quad\left[\begin{array}{l} H_0 := \emptyset \\ \textbf{for } j = 0, 1, \ldots, \textbf{ repeat} \\ \quad\quad H_{j+1} := \pi(H_j) \cup (F \cap \pi(W_i)) \\ \textbf{until } H_{j+1} = H_j \\ W_{i+1} := H_j \end{array}\right]$
until $W_{i+1} = W_i$

The major iteration starts with $W_0 = Q$. For that value of W, we iterate on H to compute the set of states from which a single visit to F can be enforced. For the next major iteration, we take W_1 to be this set. We now recompute H with respect to W_1 which yields the set of states from which one can enforce a visit to F followed by a visit to W_1 (and hence two visits to F). This process constructs a decreasing sequence $W_0, W_1, \ldots$, where each W_i is the set of states from which i visits to F can be enforced. This sequence converges to the set of states from which the plant can be driven to F infinitely many times. The strategy is extracted within the last execution of the inner loop as in the $\Diamond$-case. Formulation of the iterative processes for $\Diamond\Box$ and $\mathcal{R}_1$ is left as an exercise. ∎

4 The Real Time Case

Timed automata ([AD94]) are automata equipped with clocks whose values grow continuously. The clocks interact with the transitions by participating in pre-conditions (guards) for certain transitions and they are possibly reset when some transitions are taken. We let T denote $\mathbb{R}^+$ (the non-negative reals) and let $X = T^d$ (the clock space). We write elements of X as $\mathbf{x} = (x_1, \ldots, x_d)$ and denote the d-dimensional unit vector by $\mathbf{1} = (\underbrace{1, \ldots, 1}_{d})$. Since X is infinite and non-countable, we need a language to express certain subsets of X as well as operations on these subsets.

Definition 7 (k-polyhedral sets). *Let k be a positive integer constant. We associate with k three subsets of 2^X:*

- *$\mathcal{H}_k$ – the set of half-spaces consisting of all sets having one of the following forms: X, $\emptyset$, $\{\mathbf{x} \in \mathbb{R}^+ : x_i \# c\}$, $\{\mathbf{x} \in \mathbb{R}^+ : x_i - x_j \# c\}$, for some $\# \in \{<, \leq, >, \geq\}$, $c \in \{0, \ldots, k\}$.*
- *$\mathcal{H}_k^{\cap}$ – the set of convex sets consisting of intersections of elements of $\mathcal{H}_k$.*
- *$\mathcal{H}_k^*$ – the set of k-polyhedral sets containing all sets obtained ¿from $\mathcal{H}_k$ via union, intersection and complementation.*

Clearly, for every k, $\mathcal{H}_k^*$ has a finite number of elements, each of which can be written as a finite union of convex sets. Some authors call the elements of $\mathcal{H}_k^*$ *regions*.

Definition 8 (Reset Functions). *Let $F(X)$ denote the class of functions $f : X \mapsto X$ that can be written in the form $f(x_1, \ldots, x_d) = (f_1, \ldots f_d)$ where each f_i is either x_i or 0.*

Definition 9 (Timed Automata). *A timed automaton is a tuple $\mathcal{T} = (Q, X, \Sigma, I, R, q_0)$ consisting of Q, a finite set of discrete states,[7] a clock domain $X = (\mathbb{R}^+)^d$ for some $d > 0$, an input alphabet Σ ($\Sigma = \Sigma_c \cup \{e\}$ including the controller actions Σ_c and a single environment action e), $I : Q \mapsto \mathcal{H}_k^{\cap}$ is the state invariance function* (we denote $I(q)$ by I_q) *and $R \subseteq Q \times \Sigma \times \mathcal{H}_k^{\cap} \times F(X) \times Q$ is a set of transition relations, each of the form $\langle q, \sigma, g, f, q' \rangle$, where $q, q' \in Q$, $\sigma \in \Sigma$, $g \in \mathcal{H}_k^{\cap}$, and $f \in F(X)$, $q_0 \in Q$ is the initial state of $\mathcal{T}$.*

A *configuration* of $\mathcal{T}$ is a pair $(q, \mathbf{x}) \in Q \times X$ denoting a discrete state and the values of the clocks. When in a configuration $(q, \mathbf{x})$ such that $\mathbf{x} \in I_q$, the automaton can "let" time progress, i.e., remain in q and let the values of the clocks increase uniformly as long as $\mathbf{x}$ is still in I_q. Whenever $\mathcal{T}$ is in $(q, \mathbf{x})$ such that for some $r = \langle q, \sigma, g, f, q' \rangle \in R$, $\mathbf{x} \in g$ (the "guard"of r is satisfied), the automaton can respond to a σ input and move to $(q', f(\mathbf{x}))$. Sometimes $I_q \cap g \neq \emptyset$ and both options are possible, namely at some configurations we can either stay in q and let $\mathbf{x}$ increase with time or take a transition.

Without loss of generality, we assume that for every $q \in Q$ and every $\mathbf{x} \in X$, there exists some $t \in T$ such that $\mathbf{x} + \mathbf{1}t \notin I_q$. That is, the automaton cannot stay in any of its discrete states forever.

Definition 10 (Steps and Trajectories). *A step of $\mathcal{T}$ is a pair of configurations $((q, \mathbf{x}), (q', \mathbf{x}'))$ such that either:*

1. *$q = q'$ and for some $t \in T$, $\mathbf{x}' = \mathbf{x} + \mathbf{1}t$, $\mathbf{x} \in I_q$ and $\mathbf{x}' \in I_q$.* In this case we say that $(q', \mathbf{x}')$ is a t-successor of $(q, \mathbf{x})$ and that $((q, \mathbf{x}), (q', \mathbf{x}'))$ is a *t-step.*
2. *There is some $r = \langle q, \sigma, g, f, q' \rangle \in R$ such that $\mathbf{x} \in g$ and $\mathbf{x}' = f(\mathbf{x})$.* In this case we say that $(q', \mathbf{x}')$ is a σ-successor of $(q, \mathbf{x})$ and that $((q, \mathbf{x}), (q', \mathbf{x}'))$ is a *σ-step.*

A trajectory of $\mathcal{T}$ is a sequence $\beta = (q[0], \mathbf{x}[0]), (q[1], \mathbf{x}[1]), \ldots$ of configurations such that for every i, $((q[i], \mathbf{x}[i]), (q[i+1], \mathbf{x}[i+1]))$ is a step.

A trajectory is *non-Zeno* if it has infinitely many t-steps and the sum of the corresponding t's diverges. We denote the set of all non-Zeno trajectories that $\mathcal{T}$ can generate by $L(\mathcal{T})$. Given a trajectory β we can define $Vis(\beta)$ and $Inf(\beta)$ as in the discrete case by referring to the projection of β on Q and use $L(\mathcal{T}, \Omega)$ to denote acceptable trajectories as in definition 4.

Definition 11 (Real-time Controllers). *A simple real-time controller is a function $C : Q \times X \mapsto \Sigma_c \cup \{\perp\}$.*

[7] Called "locations" by some authors, in order to distinguish them from global states that include the clock values – we will use "configurations" for the latter.

According to this function the controller chooses at any configuration $(q, \mathbf{x})$ whether to issue some enabled transition σ or to do nothing and let time go by. We denote by $\Sigma_c^\perp$ the range of controller commands $\Sigma_c \cup \{\perp\}$. We also require that the controller is k-polyhedral, i.e, for every $\sigma \in \Sigma_c^\perp$, $C^{-1}(\sigma)$ is a k-polyhedral set.

Definition12 (Controlled Trajectories). *Given a simple controller C, a pair $((q, \mathbf{x}), (q', \mathbf{x}'))$ of configurations is a C-step if it is either*

1. *an e-step, or*
2. *a σ-step such that $C(q, \mathbf{x}) = \sigma \in \Sigma_c$ or*
3. *a t-step for some $t \in T$ such that for every t', $t' \in [0, t)$, $C(q, \mathbf{x} + \mathbf{1}t') = \perp$.*

A C-trajectory is a trajectory consisting of C-steps. We denote the set of C-trajectories of $\mathcal{T}$ by $L_C(\mathcal{T})$.

Definition13 (Real-Time Controller Synthesis). *Given a timed automaton $\mathcal{T}$ and an acceptance condition Ω, the problem* **RT-Synth**$(\mathcal{T}, \Omega)$ *is: Construct a real-time controller C such that $L_C(\mathcal{T}) \subseteq L(\mathcal{T}, \Omega)$.*

In order to tackle the real-time controller synthesis problem we introduce the following definitions. For $t \in T$ and $\sigma \in \Sigma$, the configuration $(q', \mathbf{x}')$ is defined to be a (t, σ)-*successor* of the configuration $(q, \mathbf{x})$ if there exists an intermediate configuration $(\widehat{q}, \widehat{\mathbf{x}})$, such that $(\widehat{q}, \widehat{\mathbf{x}})$ is a t-successor of $(q, \mathbf{x})$ and $(q', \mathbf{x}')$ is a σ-successor of $(\widehat{q}, \widehat{\mathbf{x}})$.[8] Then we define function $\delta : (Q \times X) \times (T \times \Sigma_c^\perp) \mapsto 2^{Q \times X}$, where $\delta((q, \mathbf{x}), (t, \sigma))$ stands for all the possible consequences of the controller attempting to issue the command $\sigma \in \Sigma_c^\perp$ after waiting t time units starting at configuration $(q, \mathbf{x})$.

Definition14 (Extended Transition Function). *For every $t \in T$ and $\sigma \in \Sigma_c$, the set $\delta((q, \mathbf{x}), (t, \sigma))$ consists of all the configurations $(q', \mathbf{x}')$ such that either*

1. *$(q', \mathbf{x}')$ is a (t, σ)-successor of $(q, \mathbf{x})$, or*
2. *$(q', \mathbf{x}')$ is a (t', e)-successor of $(q, \mathbf{x})$ for some $t' \in [0, t]$.*

This definition covers successor configurations that are obtained in one of two possible ways. Some configurations result from the plant waiting patiently at state q for t time units, and then taking a σ-labeled transition according to the controller recommendation. The second possibility is of configurations obtained by taking an environment transition at any time $t' \leq t$. *This is in fact the crucial new feature of real-time games – there are no "turns" and the adversary need not wait for the player's next move.*

As in the discrete case, we define a predecessor function that indicates the configurations from which the controller can force the automaton into a given set of configurations.

[8] Note that this covers the case of $(q', \mathbf{x}')$ being simply a σ-successor of $(q, \mathbf{x})$ by viewing it as a $(0, \sigma)$-successor of $(q, \mathbf{x})$.

Definition 15 (Controllable Predecessors). *The controllable predecessor function $\pi : 2^Q \times 2^X \mapsto 2^Q \times 2^X$ is defined for every $K \subseteq Q \times X$ by*

$$\pi(K) = \{(q, \mathbf{x}) \ : \ \exists t \in T . \exists \sigma \in \Sigma_c . \delta((q, \mathbf{x}), (t, \sigma)) \subseteq K\}$$

As in the discrete case, the sets of winning configurations can be characterized by a fixed-point expressions similar to (1)–(5) over $2^Q \times 2^X$. Unlike the discrete case, the iteration is not over a finite domain, yet some nice properties of timed-automata (see [AD94], [ACD93], [HNSY94] for more detailed proofs) guarantee convergence. Assume that $Q = \{q_0, \ldots, q_m\}$. Clearly, any set of configurations can be written as $K = \{q_0\} \times P_0 \cup \cdots \cup \{q_m\} \times P_m$, where $P_0, \ldots, P_m$ are subsets of X. Thus, the set K can be uniquely represented by a *set tuple* $\mathcal{K} = \langle P_0, \ldots, P_m \rangle$ and we can view π as a transformation on set tuples.

A set tuple $\mathcal{K}$ is called *k-polyhedral* if each component P_i, $i = 0, \ldots, m$, belongs to $\mathcal{H}^*_k$. We will show that the function π always maps a k-polyhedral set tuple to another k-polyhedral set tuple. As a first step, we will represent the function π in terms of its action on components. Without loss of generality, we assume that for every $q \in Q$, $\sigma \in \Sigma_c$ there is at most one $r = \langle q, \sigma, g, f, q' \rangle \in R$. Let $\langle P'_0, \ldots, P'_m \rangle = \pi(\langle P_0, \ldots, P_m \rangle)$. Then, for each $i = 0, \ldots, m$, the set P'_i can be expressed as

$$P'_i = \bigcup_{\langle q_i, \sigma, g, f, q_j \rangle \in R} \{\mathbf{x} \ : \ \exists t \in T . \begin{pmatrix} \mathbf{x} \in I_{q_i} \wedge \mathbf{x} + \mathbf{1}t \in I_{q_i} \wedge \\ \mathbf{x} + \mathbf{1}t \in g \wedge f(\mathbf{x} + \mathbf{1}t) \in P_j \wedge \\ (\forall t' \leq t) \bigwedge_{\langle q_i, e, g', f', q_k \rangle \in R} \\ (\mathbf{x} + \mathbf{1}t' \in g' \ \rightarrow \ f(\mathbf{x} + \mathbf{1}t') \in P_k) \end{pmatrix}\} \quad (6)$$

This ugly-looking formula just states that $\mathbf{x}$ is in P'_i iff for some j, σ and t we can stay in q_i for t time units and then make a transition to some configuration in $\{q_j\} \times P_j$, while all other environment transitions that might be enabled between 0 and t will lead us to a configurations which are in some $\{p_k\} \times P_k$.

Claim 2 (Closure of $\mathcal{H}^*_k$ under π). *If $\mathcal{K} = \langle P_0, \ldots, P_m \rangle$ is k-polyhedral so is $\pi(\mathcal{K}) = \langle P'_0, \ldots, P'_m \rangle$.*

Sketch of Proof: It can be verified that every P'_i can be written as a Boolean combinations of sets of the form:

$$I_{q_i} \cap \{\mathbf{x} \ : \ \exists t . \mathbf{x} + \mathbf{1}t \in I_{q_i} \cap g \cap f^{-1}(P_j) \wedge \forall t' \leq t . \mathbf{x} + \mathbf{1}t' \in \overline{g'} \cup f'^{-1}(P_k)\}$$

for some guards g, g' and reset functions f, f', where we use $f^{-1}(P)$ to denote $\{\mathbf{x} : f(\mathbf{x}) \in P\}$. Since timed reachability is distributive over union, i.e.,

$$\{\mathbf{x} : \exists t . \mathbf{x} + \mathbf{1}t \in S_1 \cup S_2\} = \{\mathbf{x} : \exists t . \mathbf{x} + \mathbf{1}t \in S_1\} \cup \{\mathbf{x} : \exists t . \mathbf{x} + \mathbf{1}t \in S_2\}$$

it is sufficient to prove the claim assuming k-convex polyhedral sets. Clearly, when f is a reset function $f^{-1}(S) = \{\mathbf{x} : f(\mathbf{x}) \in S\}$ is k-convex whenever S is. So what remains to show is that for any two k-convex sets S_1 and S_2, the set

$\pi_{t',t}(S_1, S_2)$, denoting all the points in S_1 from which we can reach S_2 (via time progress) without leaving S_1, and defined as

$$\pi_{t',t}(S_1, S_2) = \{\mathbf{x} : \exists t \,.\mathbf{x} + \mathbf{1}t \in S_2 \wedge \forall t' \leq t \,.\mathbf{x} + \mathbf{1}t' \in S_1\}$$

is also k-convex. Based on elementary linear algebra it can be shown that $\pi_{t',t}(S_1, S_2)$ is an intersection of some of the half-spaces defining S_1 and S_2, together with half-spaces of the form $x_i \geq 0$, and half-spaces of the form $\{\mathbf{x} : x_i - x_j \# c\}$ where c is an integer constant not larger than the maximal constant in the definitions of S_1 and S_2 (see figure 4 for intuition). ∎

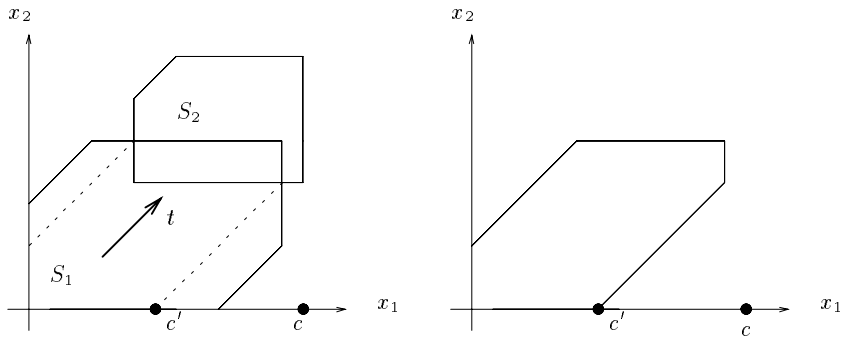

Fig. 4. The set $\pi_{t',t}(S_1, S_2)$ for the sets S_1, S_2 appearing in the right hand side appears in the left. One can see that the the integer constant c' in the new inequality $x_1 - x_2 < c'$ is an integer and is smaller than the constant c in the S_2 inequality $x_1 \leq c$.

Theorem 3 (Control Synthesis for Timed Systems). *Given a timed automaton $\mathcal{T}$ and an acceptance condition*

$$\Omega \in \{(F, \Diamond), (F, \Box), (F, \Diamond\Box), (F, \Box\Diamond), (\mathcal{F}, \mathcal{R}_1)\},$$

the problem **RT-Synth**$(\mathcal{T}, \Omega)$ *is solvable.*

Sketch of Proof: We have just shown that $2^Q \times \mathcal{H}_k^*$ is closed under π. Any of the iterative processes for the fixed-point equations (1)–(5) starts with an element of $2^Q \times \mathcal{H}_k^*$. For example, the iteration for $\Diamond$ starts with $W_0 = Q \times F$. Each iteration consists of applying Boolean set-theoretic operations and the predecessor operation, which implies that every W_i is also an element of $2^Q \times \mathcal{H}_k^*$ – a finite set. Thus, by monotonicity, a fixed-point is eventually reached. ∎

The strategy is extracted in a similar manner as in the discrete case. Whenever a configuration $(q, \mathbf{x})$ is added to W, it is due to one or more pairs of the form $([t_1, t_2], \sigma)$ indicating that within any t, $t_1 < t < t_2$, issuing σ after waiting t will lead to a winning position. Hence by letting $C(q, \mathbf{x}) = \perp$ when $t_1 > 0$ and $C(q, \mathbf{x}) = \sigma$ when $t_1 = 0$ we obtain a k-polyhedral controller.

5 Relation to Other Work

The problem of finding a winning strategy for finite and infinite games has a long history, and it lies in the intersection of Logic, Game Theory, Descriptive Set Theory and Program Synthesis. The pioneering constructive result in this area is due to Büchi and Landweber (see [BL69], [TB73] [R72]) where the winning strategy is extracted from the description of the game, in the case when the game is ω-regular. These games can be viewed as a special case of a more general type of games expressed using Rabin's *tree automata*, introduced by Gurevich and Harrington ([GH82]). Emerson and Jutla [EJ91] used fixed-point equations (over a richer language) to characterize winning states for games associated with such automata.

The idea that a reactive program can be viewed as a two-person game which the program plays against the environment, attempting to maintain a temporal specification, has been explored in [PR89] and [ALW89]. It has been realized there that finding a winning strategy for such a game is tantamount to synthesizing a program for the module that guarantees to maintain the specification against all environment inputs and their timing. For the finite-state case, algorithms for such synthesis were presented, based on checking emptiness of tree automata. The area of infinite games is still very active and mentioning all the results and contribution in this area is beyond the scope of this paper – [T94] is a good place to start.

Within the control community, Ramadge and Wonham ([RW87], [RW89]) have built an extensive automata-theoretic framework (RW) for defining and solving control synthesis problems for discrete-event systems. Thistle and Wonham [TW94a] have proposed a fixed-point characterization for the winning states in an automaton game, and their approach is very close to ours in what concerns the discrete part. A similar characterization of controllability has been suggested by Le Borgne [LB93] in the context of dynamical systems over finite fields.

As for real-time games, an extension of the RW framework for *discrete* timed systems has been proposed in [BW93]. Unlike this approach, we work in the timed automaton framework (suggested by Alur and Dill [AD94] and studied extensively by, e.g., [HNSY94], [ACD93]), where Time is a continuous entity, whose passage interacts with discrete transitions.

The only work within the RW framework on timed automata that we are aware of is that of Wong-Toi and Hoffman [WTH92]. Our work differs from theirs in the following aspects: They adhere to the language-oriented approach of [RW89], while we prefer to develop a state-oriented model, which we believe to be more adequate for real-time and hybrid systems because timed languages are not easy objects to work with. Secondly, they solve the control problem by completely "discretizing" the timed-automaton into a finite-state automaton (the "region graph") and then solve the discrete synthesis problem. This procedure can introduce an unnecessary blow-up in the size of the system. Our method, working *directly* on the timed automaton, makes only the discretizations necessary to solve the control problem.

The approach we have outlined can be extended immediately to treat *hybrid*

automata – a generalization of timed automata where the continuous variables can grow in different rates, and where the guards and invariants can be constructed from arbitrary linear inequalities. As in the the corresponding *analysis* problems for such systems, the fixed-point iteration might not converge, and thus we have only a semi-decision procedure.

Another interesting and important problem is that of partial observation: We assumed that the controller can precisely observe the the whole configuration of the plant, including the values of *all* the relevant clocks. In realistic situations the plant can be observed only up to some equivalence relation on its states, and the controller has to operate under some uncertainty.

References

[AD94] R. Alur and D.L. Dill, A Theory of Timed Automata, *Theoretical Computer Science* 126, 183–235, 1994.

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill, Model Checking in Dense Real Time, Information and Computation 104, 2–34, 1993.

[ALW89] M. Abadi, L. Lamport, and P. Wolper, Realizable and Unrealizable Concurrent Program Specifications. In *Proc. 16th Int. Colloq. Aut. Lang. Prog.*, volume 372 of *Lect. Notes in Comp. Sci.*, pages 1–17. Springer-Verlag, 1989.

[BW93] B.A. Brandin and W.M. Wonham, Supervisory Control of Timed Discrete-event Systems, *IEEE Transactions on Automatic Control*, 39, 329-342, 1994.

[BL69] J.R. Büchi and L.H. Landweber, Solving Sequential Conditions by Finite-state Operators, *Trans. of the AMS* 138, 295-311, 1969.

[EJ91] E.A. Emerson and C.J. Jutla, Tree Automata, μ-calculus and Determinacy, *Proc. 32nd FOCS*, 1991.

[GH82] Y. Gurevich and L. Harrington, Trees, Automata and Games, *Proc. 14th STOC*, 1982.

[HNSY94] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, Symbolic Model-checking for Real-time Systems, *Information and Computation* 111, 193–244, 1994.

[LB93] M. Le Borgne, *Dynamical Systems over Finite Fields*, Ph.D. thesis, Univ. Rennes 1, 1993 (In French).

[PR89] A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module, In *Proc. 16th ACM Symp. Princ. of Prog. Lang.*, pages 179–190, 1989.

[R72] M.O. Rabin, Automata on Infinite Objects and Church's Problem, AMS, 1972.

[RW87] P.J. Ramadge and W.M. Wonham, Supervisory Control of a Class of Discrete Event Processes, *SIAM J. of Control and Optimization* 25, 206-230, 1987.

[RW89] P.J. Ramadge and W.M. Wonham, The Control of Discrete Event Systems, *Proc. of the IEEE* 77, 81-98, 1989.

[TW94a] J.G. Thistle and W.M. Wonham, Control of Infinite Behavior of Finite Automata, *SIAM J. of Control and Optimization* 32, 1075-1097, 1994.

[T94] W. Thomas, On the Synthesis of Strategies in Infinite Games, *These proceedings.*

[TB73] B.A. Trakhtenbrot and Y.M. Barzdin, *Finite Automata: Behavior and Synthesis*, North-Holland, Amsterdam, 1973.

[V94] M.Y. Vardi, Personal communication.

[WTH92] H. Wong-Toi and G. Hoffmann, The Control of Dense Real-Time Discrete Event Systems, Technical report STAN-CS-92-1411, Stanford University, 1992.

This article was processed using the LaTeX macro package with LLNCS style