Control of Timed Systems

Franck Cassez, Nicolas Markey

Chapter 3

Control of Timed Systems

In this Chapter we address the problem of controller synthesis for timed systems. By timed systems we refer to systems which are subject to *quantitative* (hard) real-time constraints. We assume the reader is familiar with the basics of Timed Automata theory, or has read Chapter 1 and Chapter 2 in this book.

3.1. Introduction

It is not always possible to use *discrete time* to specify and model timing constraints. This is why *dense-time* specification formalisms, like timed automata [ALU 94] or time Petri nets [MER 74] have been introduced some years ago. Once a system is modelled by a timed automaton or a time Petri net, it can checked (Chapter 2) for quantitative properties. If a property is not satisfied, we still might be able to act on the system and *control* some of its behaviours: for instance by adding a component, a *controller*. How to compute such a timed controller is the purpose of this Chapter in which we review some algorithms for controller synthesis for timed automata.

3.1.1. *Verification of Timed Systems*

A standard approach to the verification of real-time systems, called *model checking* [SCH 99], consists in 3 phases: (1) build a complete model S for the system (*e.g.* a finite automaton); (2) specify the qualitative correctness property to be satisfied in a non ambiguous logical language (*e.g.* temporal logics) as a formula ψ and (3) check

Chapter written by Franck CASSEZ and Nicolas MARKEY.

algorithmically that S is a *model* of ψ, denoted $S \models \psi$: this is the *model checking* algorithm. Model cheking techniques for discrete event systems and qualitative properties, have been studied for quite a long time and are mature enough to have spread in industry. Nevertheless, for certain types of systems, the correctness property is formulated in terms of *quantitative* requirements that cannot be captured in a discrete time or logical time model. A typical example of such system is the *scheduling* of tasks where the durations of the tasks must be taken into account. It is sometimes possible to use discrete time as an abstraction for dense time but it may turn out to be of limited applicability: if the timing parameters of the system range from 1 to $10\,000$, using a discrete clock that ticks every time unit produces a prohibitive number of states, and *model checking* algorithms (even if symbolic) will not be able to handle such state spaces. Another disadvantage of using discrete time is that it is not easy to *compose* two systems specified with different time scales. Finally, we may have some limited knowledge of the durations of the tasks in the system. To take into account this uncertainty (durations of the tasks, of the communications), we can use dense time formalisms, for instance *timed automata (TA)* [ALU 94] which are finite automata extended with real-time *clocks*. Model checking algorithms for discrete event systems (finite automata, temporal logics like CTL or LTL) have been extended to timed automata and timed logics and these algorithms are implemented in a variety of tools like UPPAAL [AMN 01], KRONOS [YOV 97], CMC [LAR 98, LAR 05], PHAVER [FRE 05] (see Chapter 7 in this book) and HYTECH [HEN 97].

3.1.2. *The Controller Synthesis Problem*

To use a *model checking* algorithm, we must have a complete model describing all the behaviours of the system: thus we assume we have a model consisting of the *environment* of the system, the *sensors* and the *controller*. Such systems are called *closed* systems. On the contrary, in the framework of controller synthesis, we assume we have a model of the system which is *open*: if S is a model the system to be controlled, we have to find a controller C to build a closed system $C(S)$ "S controlled or supervised by C" which represents the complete model.

If the correctness property is ϕ, the *model checking* approach answers to questions of the form "Does $C(S)$ satisfy ϕ?". Hence the *model checking* problem can be formally defined by:

$$\text{given } S, C \text{ and } \phi, \text{ does } C(S) \models \phi \text{ hold ?} \qquad \text{[MCP]}$$

The *model checking* approach thus requires to first build (maybe by "hand") a controller C to be able to obtain a closed system $C(S)$. This can be a very difficult task for complex systems or when the correctness property involves timing constraints. Moreover, if using our handmade controller C, ϕ does not hold on $C(S)$, we have to iterate and modify C until we finally end up with a satisfactory controller[1]. An ideal solution would be to *compute* (we say *synthesize* in the sequel) a controller (if there is one) such that the correctness property is satisfied. This is the *controller synthesis problem (CSP)* for open systems. Before building a controller we still have to check if one exists, and this is the *control problem (CP)* formally defined by:

$$\text{given } S \text{ and } \phi, \text{ is there any } C \text{ such that } C(S) \models \phi \text{ ?} \qquad \text{[CP]}$$

To solve problem [CP], we often have to narrow the class of controllers we will consider. For instance, we can look for finite state controllers (with bounded memory) or timed controllers (that can be represented by timed automata). In any case, once we have checked that a controller exists, another task is to build a witness, which is the *controller synthesis problem (CSP)*:

$$\text{if the answer to [CP] is "yes", build a witness controller } C. \qquad \text{[CSP]}$$

These problems have been introduced and solved for finite state systems (discrete event systems) in the pioneering work of Ramadge and Wonham [RAM 87, RAM 89]. An alternative framework to study the control problem is the framework of *game theory* which we introduce in the next section.

3.1.3. *From Control to Game*

A control problem can be formulated as a two-player *game problem* (see [ARN 03, MCN 93, RIE 03, THO 95]): the model of the system is a transition system in which one of two players, controller or environment, is responsible for state change. In this framework, the controller has to find a *strategy* to ensure a particular property, and this, whatever the environment does.

An example of a two-player game is given in Fig. 3.1. In this game, round shaped states are Player 1' states in which Player 1 chooses the next move, and the square shaped states are Player 2' states.

Assume Player 1 plays the following strategy: when the game is in state ℓ_0, he chooses to go to ℓ_2, and when the game is in ℓ_3, he chooses to go to FINAL. If Player 1 plays this strategy, for any strategy of Player 2, the game is bound to end in FINAL. Player 1's strategy is said to be *winning* if the objective of the game (for Player 1) is to reach FINAL (this is called a *reachability objective*).

[1] This may take a while in the case no such controller exists ...

On the contrary, if the objective of the game for Player 1 is to avoid FINAL (this is called a *safety objective*), he still has a winning strategy: always choose to go to state ℓ_1.

Finally, if Player 1's objective is to visit ℓ_3 infinitely often (*repeated reachability objective* or *Büchi objective*), Player 1 has no winning strategy (he has to play a move and cannot refuse to play): if from ℓ_0 he chooses to go to ℓ_2, Player 2 can choose to go to FINAL; if he chooses to got to ℓ_1, Player 2 chooses to go back to ℓ_0.

The type of strategies Player 1 has to follow is also an important parameter for a game. If Player 1 wants to play one of the strategies we have defined previously, he just has to observe the *current* state of the game. Such strategies are called *memoryless* or *positional* strategies. Player 1 could also consider waiting a bit before enforcing state FINAL and for instance choose to go to ℓ_3 the first time the game reaches ℓ_2 and the second time go to FINAL. He will then have to remember that ℓ_2 has been reached once which can be done using one bit of memory. More generally, Player 1 could choose to delay the winning moves n times (choosing to go to ℓ_3 for the first n times the game enters ℓ_2) and at the $n+1$-th times the game enters ℓ_2, he can choose to go to FINAL. All these strategies, when n is fixed are *bounded memory* strategies. There are also games and objectives that cannot be won with bounded memory strategies and in this case Player 1 needs to record the entire history of the game from the beginning, to be able to choose a correct move.

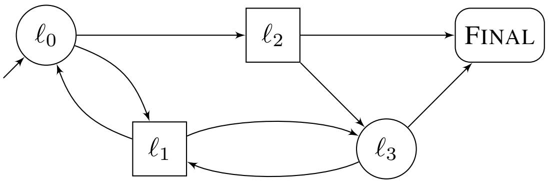


Figure 3.1. *A game example*

To solve a control problem using a game theory approach, it suffices to let Player 1 be the controller and Player 2 be the system to be controlled (the environment). The control problem is then: is there a *winning strategy* for Player 1 ? A winning strategy is an actual controller to supervise the system.

3.1.4. *Game Objectives*

Given a game G, a control objective ϕ, solving (G, ϕ) stands for the following control problem: "Is there a strategy C for Player 1 in G to ensure $C(G) \models \phi$?". If ϕ is a safety objective, and in this case we assume it is given as a set of states to stay in or to avoid, we have a *safety control problem (SafCP)*. If ϕ is a reachability objective, we have a *reachability control problem (RCP)*. We can define more complex objectives (repeated reachability, Büchi or co-Büchi) or even formulate them as (timed) temporal logic formulas.

3.1.5. *Varieties of Untimed Games*

A game as the one in Fig. 3.1 is a *turn-based* game. As for Chess or Draughts board games, players' moves alternate. A generalization of turn-based games can be obtained by tagging the states of the game with a shape either circle or square: in circle shaped states, Player 1 moves, and in square shaped states, Player 2 moves. In this case, the type of the state determines whose turn it is, but the turn does not always change in each round. This games are called *state-based* games and can be transformed in "equivalent" turn-based games. If the two players can play at the same time as in "Scissors Paper Stone", we have a *concurrent* game. We will briefly mention timed concurrent games in section 3.10.

Turn-based (finite) games enjoy nice properties. For instance they are *determined* for a large class, say $\mathcal{B}$ (that includes Büchi objectives), of control objectives [MAR 75]. Being determined means that in each state of the game, either Player 1 can win (G, ϕ) (ϕ taken in $\mathcal{B}$), or Player 2 can win $(G, \neg\phi)$ where $\neg\phi$ is the complement of ϕ (this implies that $\mathcal{B}$ is closed under complement). Concurrent games do not have this determinacy property.

Moreover, for the types of control objectives we have mentioned earlier (safety, reachability, repeated reachability, ...), turn-based and concurrent (untimed) games satisfy the following properties [THO 95]:

– it is decidable in polynomial time whether Player 1 has a winning strategy for (G, ϕ);

– if Player 1 has a winning strategy for (G, ϕ), he has a *memoryless* or *positional* winning strategy;

– if Player 1 has a winning strategy for (G, ϕ), there is a "most permissive" strategy which is memoryless as well.

The remainder of this chapter is organised as follows. Section 3.2 introduces the basic notions of timed games and their formal definitions, together with strategies. In Section 3.3, we give algorithms to solve the safety control problem for timed automata. The following sections consider more advanced topics related to the control of timed systems. Section 3.4 points out one limitation (related to the practical implementation of controllers) of the previous algorithms, and in Section 3.5, we focus on the notion of *implementable* controllers. In the following Section 3.6, we review some results on more complex control objectives. Section 3.7 addresses the problem of *optimal control* for reachability timed games. Efficient on-the-fly algorithms for controller synthesis are described in Section 3.8; a good complement to this section can be found at the end of Chapter 6. Finally, Section 3.10 considers an extended model of timed games introduced in Section 3.2.

3.2. Timed Games

The aim of this first section is to define the formal framework of timed games and related notions. The definition of *timed games* used in the sequel has been introduced in [ASA 98, MAL 95]: it is an extension of timed automata in which the set of actions is split between both players. Besides these discrete actions, each player can also decide to wait in the current location: in this case, time will elapse until one of the player possibly decides to perform an action.

3.2.1. *Timed Game Automata*

DEFINITION 3.1 (TIMED GAME AUTOMATON, TGA).– A *timed game automaton* (or simply *timed game*) is a 7-tuple $\mathcal{G} = (L, \ell_0, X, \Sigma_1, \Sigma_2, E, \mathit{Inv})$ satisfying the following conditions:

– Σ_1 and Σ_2 are disjoint alphabets;

– $(L, \ell_0, X, \Sigma_1 \cup \Sigma_2, E, \mathit{Inv})$ is a timed automaton (following Definition 2.2 of Chapter 2).

A timed game is thus a timed automaton the actions of which are associated with players: transitions that belong to the controller are called *controllable*, and the other ones belong to the environment and are called *uncontrollable*.

Informally, a *play* in a timed game proceeds as follows: it starts in an initial location ℓ_0 with all clocks set to 0, and is made of a sequence of action and delay transitions. Delay transitions are the "default" transitions, and occur as long as no player decides (or can) play an action transition. In the special case of turn-based games, the delay in each location is decided by the player controlling that location.

EXAMPLE 3.1 .– An example of a timed game is depicted on Fig. 3.2 [CAS 05]. Dashed transitions are uncontrollable (labelled with Σ_2), and solid transitions are controllable ones (labelled with Σ_1). This rule will be used all along this chapter.

In this example, all invariants are true, so that it is always possible to let time elapse. Assume that the controller decides to play action c_1 when $x = 1$. If the environment does not play action u_2 in the meantime, then the transition corresponding to action c_1 will be fired, and the game will end up in location ℓ_1 with $x = 1$. The controller will then have the opportunity to wait for an extra time unit, and then play action c_2, leading to his GOAL state. Notice that, when in location ℓ_1 with $x \geq 1$, the environment cannot play any action. He has to wait until the controller decides to play an action.

Of course, given the strategy "*play action* c_1 *when* $x = 1$" of the controller in the initial location, the environment could also have decided to play an action at an earlier time instant, for instance by playing u_2 when $x = 0.5$ (notice that u_1 is only available when $x > 1$). The play would then have moved to $(\ell_2, x = 0)$ (since the transition from ℓ_0 to ℓ_2 resets clock x).

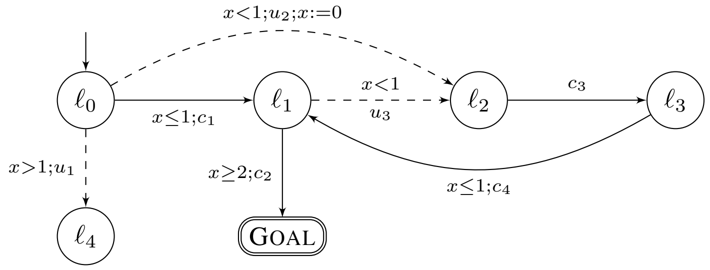


Figure 3.2. *An example of a timed game*

Those examples informally illustrate the intuition behind timed games. We now define the corresponding mathematical framework, which will allow us to formally check properties of those games.

In the sequel, we fix a game $\mathcal{G} = (L, \ell_0, X, \Sigma_1, \Sigma_2, E, \mathit{Inv})$.

3.2.2. *Strategies and Course of the Game*

Several different semantics have been proposed for timed games: time indeed requires to fix the "rules of the game". We begin here with a natural definition of the semantics of timed games as proposed in [MAL 95]. Section 3.10 will stress the drawbacks of this semantics (especially regarding strategies consisting in "stopping time") and propose another approach to remedy this problem [ALF 03].

3.2.2.1. *The Course of a Timed Game*

The definition of a run of a timed game is that of the underlying timed automaton; following Definition 3.1, a timed game is a timed automaton whose actions are partitioned between controllable and uncontrollable actions. Given a timed game $\mathcal{G} = (L, \ell_0, X, \Sigma_1, \Sigma_2, E, \mathit{Inv})$, we write $\mathcal{T}_\mathcal{G}$ for the timed transition system which gives the semantics of $\mathcal{G}$ (see Chapter 2). A run of the timed game $\mathcal{G}$ is a run of the timed transition system $\mathcal{T}_\mathcal{G}$: it is a sequence

$$\rho = s_0 \xrightarrow{d_1} s'_0 \xrightarrow{a_1} s_1 \xrightarrow{d_2} s'_1 \xrightarrow{a_2} \cdots \xrightarrow{d_{n-1}} s'_n \xrightarrow{a_n} s_n \cdots$$

where $a_i \in \Sigma_1 \cup \Sigma_2$, $d_i \in \mathbb{R}_{\geq 0}$, and in which configurations s_i and s'_i are pairs (ℓ, v) with $\ell \in L$ and v is a clock valuation. Delay and action transitions thus alternate (two consecutive delay transitions can be merged, and the last delay transition of a finite run can have infinite duration).

The set of runs of the timed game $\mathcal{G}$ is written $\mathrm{Exec}_\mathcal{G}$. We also define the subset of *maximal runs* as being the set fo runs that cannot be extended. This includes several kinds of runs:

– runs having infinitely many action transitions;

– finite runs with infinite duration;

– finite runs with finite duration from which no delay nor action transition is possible.

We write $\mathrm{Exec}^m_\mathcal{G}$ for the set of maximal runs of $\mathcal{G}$, and $\mathrm{Exec}^f_\mathcal{G}$ for the set of its finite runs. The length of a finite run is the number of its transitions (both delay and action transitions) along that run.

3.2.2.2. *Strategies*

Strategies are the central concept in game theory. A strategy tells the player how to play, depending on the history of the game. In the case of timed games, players are allowed to wait and stay idle (this is the λ transition in the definition below). As will be seen in the sequel, both players have to agree on playing that special action in order to let time elapse. As soon as one of the players decides to play one of his available actions, time will stop elapsing and the action will be played.

DEFINITION 3.2 (STRATEGY).– Let $j \in \{1, 2\}$ be a player of $\mathcal{G}$. A *strategy* for Player j is a mapping $s\colon \mathrm{Exec}^f_\mathcal{G} \to \Sigma_j \cup \{\lambda\}$ satisfying the following conditions: given a finite run ρ ending in state $q = (\ell, v)$,

– if $s(\rho) = a \in \Sigma_j$, then there must exist a transition $q \xrightarrow{a} q'$ in $\mathcal{T}_\mathcal{G}$;

– if $s(\rho) = \lambda$, then there must exist a positive delay $d > 0$ such that $q \xrightarrow{d} q'$ in $\mathcal{T}_\mathcal{G}$.

These conditions encode the fact that any action (including the λ action) proposed by a strategy must be allowed in the game $\mathcal{G}$. The set of strategies of Player j in $\mathcal{G}$ is denoted by $\mathrm{Strat}_j(\mathcal{G})$.

When a player plays according to a given strategy, he narrows the set of possible runs of the game to runs that comply with that strategy.

DEFINITION 3.3 (CONTROLLED GAME).– Let $q = (\ell, v)$ be a state of $\mathcal{G}$, and s be a strategy for Player 1 (the case of Player 2 would be symmetric). The set of *finite outcomes of s from* q, denoted by $\mathrm{Out}^f(q, s)$, is defined inductively as follows:

– q is the only zero-length run in $\mathrm{Out}^f(q, s)$;

– any finite run of $\mathcal{G}$ of the form $\rho' = \rho \xrightarrow{\sigma} q'$ (hence having positive length) belongs to $\mathrm{Out}^f(q, s)$ if, and only if, ρ belongs to $\mathrm{Out}^f(q, s)$ and one of the following three conditions holds:

- $\sigma \in \Sigma_2$;
- $\sigma \in \Sigma_1$ and $\sigma = s(\rho)$;
- $\sigma \in \mathbb{R}_{>0}$ and for any d such that $0 \leq d < \sigma$, it holds $s(\rho \xrightarrow{d} q'') = \lambda$.

The set of maximal outcomes of s from q, denoted with $\mathrm{Out}^m(q, s)$, is the union of the following two set of outcomes:

– the set of maximal finite runs in $\mathrm{Out}^f(q, s)$;

– the set of infinite runs all of whose prefixes are in $\mathrm{Out}^f(q, s)$.

It is understood in this definition that a player can remember the whole history of the play for applying his strategy. However, some strategies have the special feature of being independent of the history and only depending on the current state of the system. Those strategies are called *positional* or *memoryless* strategies. In those special cases, in order to alleviate notations, we will see those strategies as mapping from the set of states to the set of actions of the corresponding player. Below is an example of a positional strategy.

EXAMPLE 3.2 .– We stick to the example of Fig. 3.2, and consider the following memoryless strategy $\tilde{s}$:

– from states of the form (ℓ_0, v) with $v(x) < 1$, Player 1 (the controller) plays action λ. When $v(x) = 1$, he plays c_1. If ever $v(x) > 1$, he again decides to play λ (as this is his only allowed action). Formally, $\tilde{s}(\ell_0, x < 1 \vee x > 1) = \lambda$ and $\tilde{s}(\ell_0, x = 1) = c_1$;

– from (ℓ_1, v), he plays λ as long as $v(x) < 2$, and c_2 when $v(x) \geq 2$, *i.e.*, $\tilde{s}(\ell_1, x < 2) = \lambda$ and $\tilde{s}(\ell_1, x \geq 2) = c_2$;

– from (ℓ_2, v), he plays c_3, hence $\tilde{s}(\ell_2, v) = c_3$;

– from (ℓ_3, v), if $v(x) < 1$, he plays λ, and plays c_4 when $v(x) = 1$. When $v(x) > 1$, he has to play λ. Hence $\tilde{s}(\ell_3, x < 1 \vee x > 1) = \lambda$ and $\tilde{s}(\ell_3, x = 1) = c_4$;

– from ℓ_4 and GOAL, Player 1 has no choice and only has to wait: $\tilde{s}(\ell_4, v) = \tilde{s}(\text{GOAL}, v) = \lambda$.

Among the outcomes of this strategy from $(\ell_0, 0)$, some directly go to ℓ_1, some others go via ℓ_2 and ℓ_3 (in case Player 2 plays u_2), but one is easily convinced that no outcome can ever reach ℓ_4, and that actually all the outcomes eventually reach GOAL.

Let Ω be a set of runs (intended to represent the set of winning plays) defining the control objective. In the sequel, we write $(\mathcal{G}, \Omega)$ for the game $\mathcal{G}$ under the winning objective Ω (for Player 1).

DEFINITION 3.4 (WINNING STATE, WINNING STRATEGY).– Let $s \in \text{Strat}_j(\mathcal{G})$ be a strategy for Player 1. We say that s is *winning* from state q for Ω if $\text{Out}^m(q, s) \subseteq \Omega$, *i.e.,* , if all the maximal outcomes meet the objective.

A state q is *winning* for Player 1 if he has a winning strategy from q. We write $\mathsf{WinState}_1(\mathcal{G}, \Omega)$ for the set of states of $\mathcal{G}$ that are winning for Player 1.

Finally, a strategy s is said to be winning in $(\mathcal{G}, \Omega)$ if it is winning from the initial state: $\text{Out}^m((\ell_0, v_0), s) \subseteq \Omega$. We write $\mathsf{WinStrat}_j(\mathcal{G}, \Omega)$ for the set of winning strategies of Player j. In the sequel, we focus on winning strategies for Player 1 (the controller). We thus write $\mathsf{WinStrat}(\mathcal{G}, \Omega)$, being understood that it refers to player 1. We also sometimes omit to mention Ω when it is clear from the context.

EXAMPLE 3.3 .– Back to the example of Fig. 3.2, location GOAL is the reachability objective of the controller. The set Ω is then the set of (maximal) runs visiting that location. The strategy $\tilde{s}$ defined above is then winning from the initial state.

3.3. Computation of Winning States and Strategies

Controller synthesis algorithms for timed automata are given in [ALF 01, ALF 03, ASA 98, MAL 95]. They compute fix-points (as for untimed games): the set of states which are winning in n steps is computed inductively. When n tends to infinity, all the winning states are collected. These algorithms terminate because the region equivalence relation on clock valuations introduced in Chapter 2 is also a good equivalence relation for timed games.

In this section, we use a running example depicted in Fig. 3.3 where the control objective is to avoid location BAD. The results of the synthesis algorithms for this example are given at the end of this section.

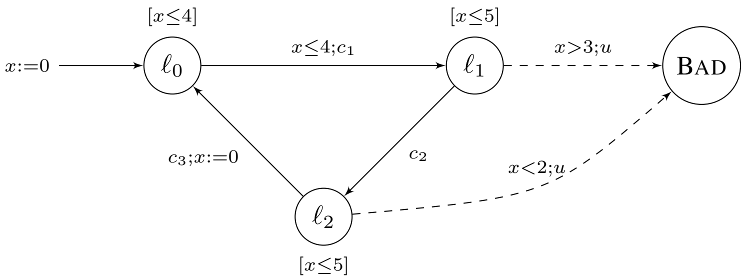


Figure 3.3. *A Timed Game Automaton S*

3.3.1. *Controllable Predecessors*

In order to implement our inductive fix-point algorithm, we define the *controllable predecessors* of a set of states: indeed, to win a game in at most $n + 1$ steps, the controller must have a strategy to force in one step, a state from which it can win in at most n steps.

Let $\mathcal{G}$ be a timed game, and $\mathcal{T}_\mathcal{G}$ the timed transition system which is the semantics of $\mathcal{G}$. For a given subset X of S (set of states of $\mathcal{T}_\mathcal{G}$), we define the sets $\mathit{CPre}(X)$ and $\mathit{UPre}(X)$ by:

$$\mathit{CPre}(X) = \{q \in S \mid \exists c \in \Sigma_c \text{ such that } q \xrightarrow{c} q' \text{ et } q' \in X\};$$

$$\mathit{UPre}(X) = \{q \in S \mid \exists u \in \Sigma_u \text{ such that } q \xrightarrow{u} q' \text{ et } q' \in X\}.$$

These two sets are respectively the set from which the controller can force, with a controllable transition, the game into the set X, and the set from which the environment can force, with an uncontrollable transition, the game into the set X. To complete the definition we have to take into account the time elapsing action λ.

Fig. 3.4 depicts the conditions for a state q to be a *controllable predecessor* of a set of states X: the controller must be able to *force* the game into a state of X, first by time elapsing (λ) and then by firing a controllable transition c; and from each state q_t encountered during the time elapsing phase, the environment must not be able to fire an uncontrollable transition leading outside of X (denoted $\overline{X}$).

In Fig. 3.4, state q' is not a controllable predecessor of X, because during the time elapsing phase, the game reaches a state in $\mathit{UPre}(\overline{X})$, from where the environment can force the game in $\overline{X}$. On the contrary q is a controllable predecessor of X.

DEFINITION 3.5 (CONTROLLABLE PREDECESSOR).– A state q is a controllable predecessor of X iff:

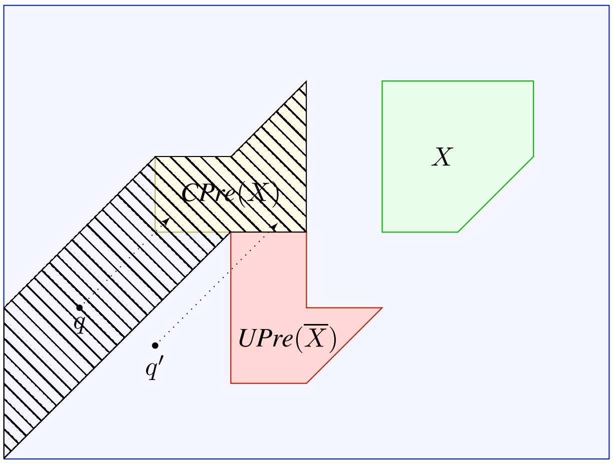


Figure 3.4. *Controllable Predecessors*

1) there is some $\delta \geq 0$, such that $q \xrightarrow{\delta} q'$ and $q' \in CPre(X)$;

2) for all $0 \leq t \leq \delta$ such that $q \xrightarrow{t} q_t$, we have $q_t \notin UPre(\overline{X})$.

The set of controllable predecessors of X is denoted $\pi(X)$.

In Fig. 3.4, the black lined polygon corresponds to the controllable predecessors of the set X. In the example of Figure 3.3, the state $(\ell_1, x = 1)$ is a controllable predecessor of $(\ell_2, x = 2)$: a controller can let time elapse (1 time unit) until $(\ell_1, x = 2)$ and then fire c_2 from ℓ_1 to ℓ_2. During the time elapsing phase no uncontrollable transition can be fired.

REMARK 3.1 .– This definition of controllable predecessors is an extension of the one given for discrete games. It does not take into account the special features of timed games, like liveness enforced by invariants. For instance, for a reachability game with control objective GOAL, and one uncontrolable transition $(0, x \leq 1, u, \varnothing, \text{GOAL})$ with $Inv(0) \equiv x \leq 1$, the environment must play action u before the time point 1. Thus if we would take this into account, a controller could win just by waiting.

It is possible to take into account the cases where the environment is bound to play by modifying the previous definitions. The results we give in this section are still valid in this case.

3.3.2. *Symbolic Operators*

The semantics of a timed game automaton is a timed transition system, and as for *model checking* of timed automata, the set of states is infinite. Thus we must develop symbolic techniques, gathering states in "*families*" which have the same properties.

We will use the notion of equivalence, *regions*, between states introduced in paragraph 2.4.2 of Chapter 2. To prove the correctness of this approach, we must show that two states in the same region have the same properties regarding their winning status. If this holds, we can can compute the controllable predecessors of a region. The following Lemma summarizes these two claims:

LEMMA 3.1 .– *If Z is a (finite) union of regions (we say a* zone*), then:*
$\mathbf{P_1}$*: CPre(Z), UPre(Z) and $\pi(Z)$ are unions of regions (thus zones),*
$\mathbf{P_2}$*: CPre(Z), UPre(Z) et $\pi(Z)$ can be effectively computed.*

Proving the previous Lemma requires the introduction of an intermediate symbolic operator which computes the states from which a set of states A can be reached by time elapsing without encountering a state in a set B. This operator is given by:

$$\mathit{Pre}_t(A, B) = \{s \in S \mid \exists \delta \geq 0.\ s \xrightarrow{\delta} s' \wedge s' \in A \wedge \\ \forall 0 \leq t \leq \delta.\ [(s \xrightarrow{t} s'') \Rightarrow (s'' \notin B)]\}.$$

Taking $A = \mathit{CPre}(X)$ and $B = \mathit{UPre}(\overline{X})$ allows us to define a symbolic version of the controllable predecessors operator π using Pre_t: $\pi(X) = \mathit{Pre}_t(\mathit{CPre}(X), \mathit{UPre}(\overline{X}))$.

EXAMPLE 3.4 .– For the automaton of Fig. 3.3, we have $\mathit{CPre}(\ell_1, x \leq 3) = (\ell_0, x \leq 3)$ and if $Z = (\ell_0, x \leq 4) \cup (\ell_1, x \geq 0) \cup (\ell_2, x \geq 0)$ then $\pi(Z) = Z'$ avec $Z' = (\ell_0, x \leq 3) \cup (\ell_1, 0 \leq x \leq 3) \cup (\ell_2, x \geq 2)$.

3.3.3. *Symbolic Computation of Winning States*

We can now proceed to the symbolic computation of winning states of a timed game. A *symbolic state* of a TGA is given as a finite union of pairs (ℓ, Z) where ℓ is a location of the TGA and Z a zone. We assume we have to solve a safety game, and the set of bad states to avoid, is given by BAD which is a symbolic state. The set of winning states of such games is the greatest fix-point of the mapping $h\colon X \mapsto \overline{\text{BAD}} \cap \pi(X)$. As h is a monotonic (decreasing) mapping, this is well defined. To compute this fix-point, we start by computing $h(S)$ (where S is the set of states of the game), then $h(h(S))$, and so on. This algorithm produces a decreasing sequence of symbolic

X_i	ℓ_0	ℓ_1	ℓ_2
0	$0 \leq x \leq 4$	$0 \leq x \leq 5$	$0 \leq x \leq 5$
1	$0 \leq x \leq 4$	$0 \leq x \leq 3$	$2 \leq x \leq 5$
2	$0 \leq x \leq 3$	–	–

(a) Winning States

	ℓ_0	ℓ_1	ℓ_2
λ	$x < 3$	$x < 3$	$x < 5$
c	$x \leq 3$	$2 \leq x$	$x \leq 5$

(b) Most Permissive Strategy f^*

Table 3.1. *Iterative Computation of h^* for the Example of Figure 3.3*

state and as there is a finite number of such states, this computation terminates in a finite number of steps. When it stabilizes, the greatest fix-point h^* of h is computed: it is a subset of $\overline{\text{BAD}}$, and from each state in h^* the controller has a strategy to stay within h^* by time elapsing and then firing a discrete transition. The set h^* is also the largest subset of $\overline{\text{BAD}}$ that the controller can enforce.

EXAMPLE 3.5 .– For the example of Fig. 3.3, the iterative computation of h^* is given in Table 3.1(a): the winning zone for each location ℓ_i appears in the corresponding column.

REMARK 3.2 .– For reachability games, the goal is to enforce a symbolic state GOAL. Thus the set of winning states is the least fix-point of the mapping $h\colon X \mapsto \text{GOAL} \cup \pi(X)$. Again, this least fix-point can computed iteratively by computing $h(\varnothing), h(h(\varnothing))$ and so on. Termination is ensured because the sequence of symbolic states which is computed is monotonic. For more complex control objectives (repeated reachability, ...), nested fix-points are required (see [MAL 95, ASA 98, ALF 01, ALF 03]).

As we can effectively decide whether the initial state of a TGA belongs to the symbolic representation h^* we can decide whether there is a winning strategy using h^* and moreover:

THEOREM 3.1 ([ASA 98, HEN 99B]).– *The control problems SafCP and RCP are decidable for TGA and EXPTIME-complete.*

3.3.4. *Synthesis of Winning Strategies*

To solve the Control Synthesis Problem for a safety game, we can define a *most liberal* or *most permissive* strategy. Actually, this strategy is not a strategy in the sense of section 3.2.2.2 as it maps states to *sets* of controllable actions and not to a single controllable action. The most permissive strategy f^* associates with each run ρ a set of actions from $\Sigma_c \cup \{\lambda\}$ which are safe: every non-blocking strategy f (in the sense of section 3.2.2.2) such that $f(\rho) \in f^*(\rho)$ is winning and conversely a strategy f is winning only if $f(\rho) \in f^*(\rho)$. This is why f^* is called the most permissive strategy. For the synthesis problem we have the following result:

THEOREM 3.2 ([ASA 98]).– *If $\mathcal{G}$ is a timed game automaton such that the initial state of $\mathcal{G}$ is winning for a safety objective, then there is a positional winning most permissive strategy f^*.*

To compute this most permissive strategy, we begin by strengthening the guards of the controllable transitions such that, from a winning state s, firing the transition with the strengthened guard leads to a winning state. This amounts to computing a most liberal pre-condition. In our case, we strengthen (the guard of) transition c_1 by $x \leq 3$, c_2 by $x \geq 2$ (which ensures that c_2 is not fired too early). The most permissive strategy is then defined by: wait in each winning state s such that there is some $\delta > 0$ and $s \xrightarrow{\delta} s'$ with s' winning; fire a controllable transition c_i if the strenghtened guard is satisfied. For the example of Figure 3.3 the most permissive strategy is given in Table 3.1(b). Notice that to obtain a *non-Zeno* strategy, one should not choose infinitely often λ in ℓ_0 (or ℓ_1, ℓ_2), but at some point in time we should fire c_1. Using the previous method we can compute a timed automaton C (given in Fig. 3.5) which is a representation of the most permissive strategy f^*, and such that *Reach*$(C(G)) \subseteq \phi$ with $\phi =$ *Reach*$(G) \setminus \{\text{BAD}\}$. For reachability games, computing a winning strategy may be a bit more complicated. A correct algorithm is given in [BOU 04b] for this case.

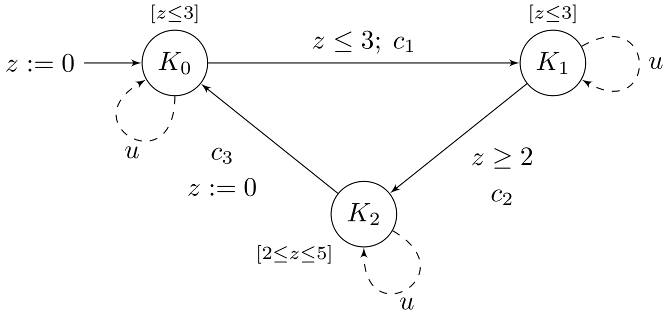


Figure 3.5. *Most Permissive Controller C*

3.4. Zeno Strategies

Fix-point algorithms presented in Section 3.3 compute the set of winning states, and the corresponding winning strategies. However, we did not take into account the so-called "Zenoness" problem: it could be the case —and is the case in the example of

Fig. 3.6 where the aim is to always avoid the BADstate— that a state is winning thanks to non-realistic strategies. In the example of Fig. 3.6, the strategy of the controller consists in firing transition c infinitely many times during the first time unit, thus preventing clock x to reach value 1. Such a controller is called a *Zeno* controller and is not realistic because physical time cannot be stopped.

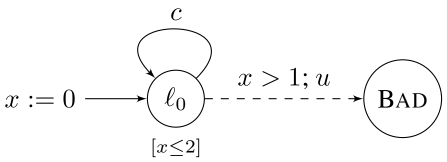


Figure 3.6. *A safety game having a winning Zeno strategy*

Symmetrically, there is no strategy for the environment to reach the BAD state in this game, since any strategy of the environment has to "wait" as long as $x \leq 1$. Hence the controller can fire transition c at dates $1 - 1/n$, which yields an infinite run never reaching BAD.

As will be seen in Section 3.10, a (rather technical) solution exists to rule out those non-realistic strategies. More generally, control problems (in particular for safety games) have been extended to generate *implementable* controllers (thus in particular non-Zeno). This extension is presented in the following section.

3.5. Implementability

Variants of the safety control problem SafCP have been proposed in many papers [HEN 99a, HEN 99b, ASA 98, MAL 95].

In the paper [CAS 02], we can find a classification of these problems according to the following criteria:

– discrete time control: can we control the system with a controller that reacts every β time units? This is the *Known Sampling Rate* (KSR) *control problem*. A more general question is ask for the computation of a value β such that the system is controllable. This is the *Unknown Sampling Rate* (USR) *control problem*.

– dense time control: in this setting the controller can either (i) control time elapsing, which is the setting of the seminal papers [MAL 95, ASA 98]; in this case this is the *Unknown Switch Condition* (USC) *control problem*; or (ii) cannot control time elapsing and in this case this is the *Known Switch Condition* (KSC) *control problem*.

The decidability status of the standard safety control problem SafCP also depends on the expressive power of the formalism used to specify timed games. Before giving the decidability status of the previous problems we review some well-known classes of extensions of timed automata.

3.5.1. *Hybrid Automata*

The more general class that is usually considered for the control problem SafCP is the class of *Linear Hybrid Automata* (LHA) [HEN 96]. The reader is refered to Chapter 7 for a detailed presentation of LHA. This class extends timed automata in the following manner: (i) the evolution rate of the variables[2] are given by intervals of the form $1 \leq \dot{x} \leq 3$; (ii) guards and resets are linear functions like $2x + 3y \leq 4$ et $x := 3y - 7z$.

An interesting subclass of LHA is the class of *Rectangular Automata* (RA) introduced in [HEN 99b]. For RA, the guards and resets cannot involve two distinct variables: they are of the form $x \geq 1 \wedge y < 2$ and for resets $x :=]2, 3]$ to assign a value v with $2 < v \leq 3$ to x. This class has interesting and surprising decidability results: the reachability problem is decidable for RA [HEN 99b]. A subclass of RA is the class of *Initialised Rectangular Automata* (IRA): if a derivative of a variable changes on a transition from ℓ to ℓ', then the variable must be reset on this transition.

The reachability problem is decidable for RA but for any subclass if becomes undecidable [HEN 98]. Another interesting class of hybrid systems with good decidability properties, is the class of *O-minimal Automata*, (OminA) [LAF 00]: in this model, the dynamics can be non-linear (exponential or any o-minimal theory) but all the variables must be reset when firing a discrete transition.

Decidability results for the safety control problem on LHA are summarised in Table 3.5.1: The meaning of abbreviations are: KSC = *Known Switch Conditions controller*, USC = *Unknown Switch Conditions controller*, KSR = *Known Sampling Rate controller*, and USR = *Unknown Sampling Rate controller*. $\surd$ = decidable, $\times$ = undecidable.

A recent result by Patricia Bouyer, Thomas Brihaye et Fabrice Chevalier [BOU 06b] is the decidability of the Reachability control problem, USC variant, for o-minimal automata.

3.5.2. *On the Existence of Non-Implementable Continuous Controllers*

In [CAS 02] it is proved that the USR-CP is undecidable, even for the class of timed automata. Another result of the paper is that, there are systems that can be controlled with a dense time controller (USC), but by no sampling controller (USR). We

[2] We note $\dot{x}$ the derivative of x.

System Given by:	KSC	USC	KSR	USR
Timed Auto.	√ [MAL 95]	√ [MAL 95]	√ [HOF 92]	× [CAS 02]
Initialised Rect. Auto.	√ [HEN 99a]	× [HEN 98]	√ [HEN 99b]	× [CAS 02]
Rect. Auto.	× [HEN 99a]	× [HEN 99a]	√ [HEN 99b]	× [CAS 02]

Table 3.2. *Safety Control Problem: Decidability Results.*

have given an example of such a system, given by the timed automaton of Figure 3.7. This timed automaton can be controlled with a non-Zeno controller. In this example, all the transitions are controllable. To avoid location BAD, it suffices to avoid firing a d-transition. This can be achieved by computing a most permissive controller: this controller C only requires to leave ℓ_2 when $x < 1$, and thus we add the invariant $[x < 1]$ to location ℓ_2. The controller C has to do the sequence of actions $(a.b.c)^\omega$, but the time elapsed between the $n+1$-th a and the $n+1$-th b must be smaller than for the previous round n. More precisely, if the time elapsed between the n-th a and the n-th b is δ_n, the controller has to ensure that $\sum_{i=1}^{+\infty} \delta_i \leq K$ where $K \leq 1$. Because of this, no sampling controller can control the system: given a sampling rate β, after a finite number of rounds m, the sum $\sum_{i=1}^{m} \delta_i$ is beyond 1. The controller C is non-zeno (and non-blocking) because the n-th a action occurs at time n. This shows dense time controllability does not imply sampling controllability. In other words, if the answer to the USC-CP is "yes", the most permissive controller is not always *implementable* on a digital controller.

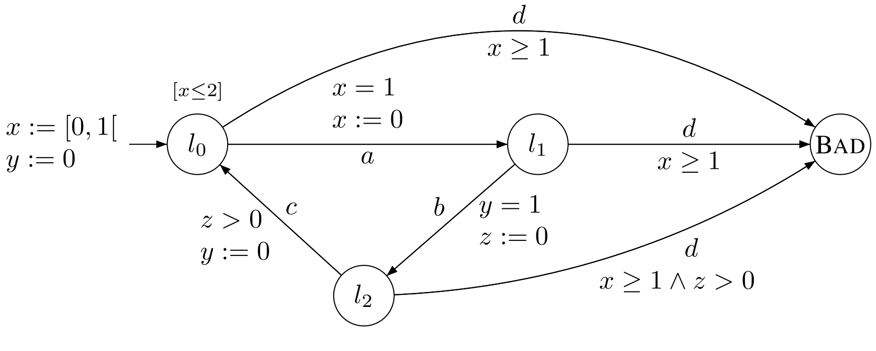


Figure 3.7. *A system that cannot be controlled with a sampling controller*

3.5.3. *Recent Results and Open Problems*

The main undecidability result of [CAS 02] can be extended to intervals of the form $[\alpha, \beta]$ *i.e.,* we impose that a controllable action occurs within $[\alpha, \beta]$ after the preceeding controllable action. The proof of this result is given in [DOY 07].

The previous result show that the notion of *implementable controllers* is important and must be addressed in the control problem. This aspects of the control problem have been extensively studied by Jean-François Raskin and his group in [DEW 05a, DEW 05b, DEW 04a, DEW 04b] and a synthesis of this work (in French) is given in [ALT 05].

An open problem is the direct synthesis of implementable or robust controllers which woudl extend the corresponding results on robust *model checking* [BOU 06f, BOU 08b].

3.6. Specification of Control Objectives

In the previous section, we have considered safety (or reachability) control objectives. In this section, we model the systems by timed automata and focus on more general types of control objectives.

In [ALF 01] the authors give an algorithm to deal with general ω-regular control objectives. They consider ω-regular objectives on TGA with a "surprise" semantics (see section 3.10 in this Chapter). They show that the CP with this "surprise" semantics is EXPTIME-complete. It is to be noticed that those control objectives are often called *internal* in the sense that they refer to the state properties (and clocks) of the system to be controlled. The safety and reachability properties that we have considered previously are examples of internal property. In the case of timed systems they only refer to the untimed sequences of states of the system and thus have a restrictive expressiveness: it is possible to specify a property like "from a state satisfying p we must reach a state satisfying q" but nothing like "from a state satisfying p we must reach a state satistying q within d time units" (*bounded liveness*). In [FAE 02a], the authors show that the control problem is 2EXPTIME-complete for specifications given by LTL formulae[3].

A control objective is a property of a closed system, and it is natural to write properties using temporal logics or automata, *i.e.,* , to give the set of timed words the controlled system has to generate. The language[4] $\mathcal{L}(A)$ accepted by a timed automaton

[3] This result does not contradict the complexity result of [ALF 03] because the specification is given as an LTL formula and not by the automaton that recognises the valid sequences.

[4] We consider languages over infinite words *e.g.* ω-regular languages.

is a set of behaviours (timed words) and this can be considered to be a control objective: in this sense this is an *external* control objective. Notice that it can either specify good or bad behaviours. Assume we have two timed automata: A_d which defines the desired (permitted) behaviours of the system, and A_u which defines the undesired behaviours. Let S be the system we want to control. The *External Specification Control Problem*, (ESCP) is the following:

$$\text{is there any } C \text{ s.t. } (1)\ \mathcal{L}(C(S)) \cap \mathcal{L}(A_u) = \varnothing \textit{ et } (2)\ \mathcal{L}(C(S)) \subseteq \mathcal{L}(A_d)\ ? \quad \text{[ESCP]}$$

In the paper [DSO 02], the authors prove that:

1) the sub-problems ESCP.(1) and ESCP.(2) are undecidable when A_d and A_u are non-deterministic timed automata;

2) these problems become decidable if A_d and A_u are deterministic; deterministic timed automata can be complemented and in this case the ESCP can be reduced to a (internal) safety control problem;

3) the control problem ESCP.(2) is decidable and 2EXPTIME-complete if the resources (number of clocks, granularity and constants used in the specification) of the controller are bounded, and this even if A_u is non-deterministic. ESCP.(1) remains undecidable even for bounded resources.

Expressing control objectives using timed automata is a very powerful tool and there is a gap between the control objectives we considered previously (safety or reachability) and these very general class of ω-regular properties. Temporal logics are a less powerful yet often sufficient specification formalism for most properties needed in practise.

In [BOU 06a], the authors consider a timed extension of LTL namely MTL [KOY 90]. The control problem is undecidable for MTL specifications but become decidable if the resources of the controller are bounded (it is still non primitive recursive). In [FAE 02b], the authors consider TCTL specifications without equality and prove that the control problem (for timed automata) can be decided in exponential time for this type of specifications.

In [BOU 05b], control objectives are specified using the (timed) temporal logic L_ν (more precisely a fragment L_ν^{det} of L_ν) which is a fragment of the timed μ-calculus: it allows to specify *bounded safety* properties.

What is interesting is that the CP for L_ν^{det} control objectives can be reducec to *model checking* problem and this later problem is EXPTIME-complete.

Finally, there is a nice logic which was specifically designed to specify properties of open (discrete) systems: *Alternating-time Temporal Logic* (ATL) [ALU 02]. This logic is an extension of CTL where the path quantifiers are replaced by “strategies quantifiers". It is possible to write a property like "Player 1 has a strategy to reach a state from which Player 2 will not be able to reach a winning state".

Several recent papers have extended the definitions of this logocs to timed systems. One can then write a property specifying that a winning state must be reached "within five time units". The extended logic has been studied for time automata operating in discrete time [LAR 06], and PTIME algorithms are proposed for model-cheking the extended logic. For timed automata in dense time, some results are available [BRI 07, HEN 06a] but the semantics iof TGA s a bit different from the one we have introduced so far (see section 3.10).

3.7. Optimal Control

If we consider the reachability *Reachability Control Problem* (RCP),, we want to synthesise a controller which enforces a particular state or location q. Once solved, we can ask for the controller to be *optimal* *e.g.* the state q is reached as soon as possible with the controller. This version of optimal control called *time-optimal control* was solved in [ASA 99].

3.7.1. *Timed Automata with Costs*

A few years later in 2001, timed automata were extended with *costs* or *prices*: this extension was called *priced timed automata* (PTA) in [BEH 01b, BEH 01a] and *weighted timed automata* (WTA) in [ALU 01] (both models are exactly the same thing but they were proposed independently at the same time by two different groups.)

The notion of cost or price generalises the notion of elapsed time: the cost is a linear function of the elapsed time in each location, and is an integer value on each discrete transition. In the example of Figure 3.8, the costs in the PTA $\mathcal{A}$ are given by the keyword **cost**.

When the cost is associated with a location, it stands for the *cost per time unit* when time elapses in this location. When it is associated with a discrete transition, it stands for the cost of *firing* the transition. Assume δ time units elapse in location ℓ_0, and then c_1 is fired. The invariant on ℓ_1 prevents time from elapsing in ℓ_1. This location is a choice location in which we immediately go either to ℓ_2 or ℓ_3. Assume we choose ℓ_2 and let δ' time units elapse before firing c_2. In this case the total cost of this execution is $5 \cdot \delta + 10 \cdot \delta' + 1$. In PTA all the transitions are controllable.

The optimal-cost computation problem for PTA consists in computing the minimal cost to reach a particular location. The (smart) solutions to the optimal-cost computation problem for PTA were simultaneously proposed in [ALU 01, BEH 01b]. Notice

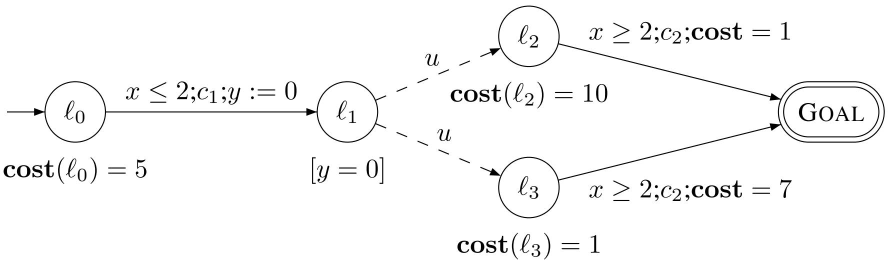

$x \leq 2; c_1; y := 0$
ℓ_0
$\mathbf{cost}(\ell_0) = 5$
ℓ_1
$[y = 0]$
u
u
ℓ_2
$\mathbf{cost}(\ell_2) = 10$
$x \geq 2; c_2; \mathbf{cost} = 1$
ℓ_3
$\mathbf{cost}(\ell_3) = 1$
$x \geq 2; c_2; \mathbf{cost} = 7$
GOAL

Figure 3.8. *A PTGA $\mathcal{A}$*

that in PTA, there is only one player and this is not the cost-optimal control problem. A symbolic algorithm [BEH 01a] has been implemented in an extended version of UPPAAL called UPPAAL CORA [BEH 07b].

If we consider PTA and a dual problem *i.e.,* keep the system in a set of states, we can define an optimal problem as well: here the model has *costs* and *rewards*, and the aim is to minimise the limit of the ration cost/reward on infinite runs. This latter problem was solved in [BOU 08a, BOU 04a].

A step further consists in defining a logic which can refer to cost values: such a logic, WCTL, has been defined in [BRI 04], and extends TCTL with cost variables. Inside this logic, it is possible to write a formula which which states "it is possible to reach a state q and this at a cost which is less than K." The model-checking problem for WCTL against timed automata was proved undecidable (even if the time domain is discrete). Nevertheless, for a subset of WCTL, there are some classes of timed automata for which model-checking is decidable. Recently in [BOU 07], the authors have proved that WCTL was decidable for o-minimal PTA. The formal definition of a priced timed game automaton is the follwoing:

DEFINITION 3.6 (PRICED TIMED GAME AUTOMATON).– A *priced timed game automaton* (PTGA) is a pair $(\mathcal{G}, \mathbf{cost})$ where $\mathcal{G} = (L, \ell_0, X, \Sigma_1, \Sigma_2, E, \mathit{Inv})$ is a timed game automaton and **cost** is a *cost mapping* such that: $\mathbf{cost} : L \cup E \rightarrow \mathbb{N}$, *i.e.,* **cost** assocaites with each loation a cost per time unit and a firing cost with each discrete transition. Let $\mathcal{T}_{\mathcal{G}} = (Q, Q_0, \Sigma_1, \Sigma_2, \Gamma, \delta)$ be the semantics of $\mathcal{G}$. If ρ is the run of $\mathcal{T}_{\mathcal{G}}$ defined by:

$$\rho = (\ell_0, v_0) \xrightarrow{(\delta_1, e_1)} (\ell_1, v_1) \xrightarrow{(\delta_2, e_2)} \cdots \xrightarrow{(\delta_n, e_n)} (\ell_n, v_n)$$

we can define the *cost* of ρ by:

$$\mathbf{Cost}(\rho) = \sum_{i=0}^{n-1} \big(\mathbf{cost}(\ell_i) \times \delta_{i+1} + \mathbf{cost}(\ell_i \xrightarrow{e_i} \ell_{i+1}) \big).$$

The cost of a run is thus the sum of the discrete costs and the delay costs.

3.7.2. *Optimal Cost in Timed Games*

We can now formulate the optimal-cost problem for timed games. In this case we start with a model which is a *Priced Timed Game Automaton* (PTGA) *i.e.,* a TGA enriched with costs. We consider here reachability games: the aim is to reach a location at an optimal cost, and this optimal cost is the best the controller can ensure, whatever the adversary (environment) does. The automaton $\mathcal{A}$ of Figure. 3.8 is an example of a PTGA. Uncontrollable transitions are labelled by u. Notice that in this game, there is a controller which can enforce location GOAL: fire c_1 and then whatever the environment chooses in ℓ_1, fire c_2. In the sequel, we assume that the reachability control problem has been solved and the answer was "yes" *i.e.,* there is a controller which can enforce GOAL. Our problem is now to compute the optimal cost.

This optimal cost is defined as the minimum value the controller can guarantee whatever the environment does. For example, in the PTGA $\mathcal{A}$ of Figure 3.8, there are two families of runs: the runs in which the environment chooses to go from ℓ_1 to ℓ_2 and the runs in which it chooses to go from ℓ_1 to ℓ_3. The only real decision the controller has to make is to choose how long to wait in ℓ_0 before firing c_1 (indeed when in ℓ_2 or ℓ_3 it chooses to fire c_2 as soon as $x \geq 2$.) Let δ be the amount of time the controller chooses to wait for in ℓ_0. The costs that can result from this choice depend on the choice of the environment: if it chooses $\ell_i, i = 2, 3$, the cost is respectively $\alpha_2(\delta) = 5 \cdot \delta + 10 \cdot (2 - \delta) + 1$ and $\alpha_3(\delta) = 5 \cdot \delta + 1 \cdot (2 - \delta) + 7$. Once the controller has fired c_1, the environment will try to maximise the cost of the current run: it will choose $\ell_i, i = 2, 3$ such that $\alpha_i(\delta)$ is maximal. The aim of the controller is to minimise this maximum over all paths.

Thus the optimal cost for the PTGA $\mathcal{A}$ is defined by:

$$\mathbf{OptCost} = \inf_{0 \leq \delta \leq 2} \max(5 \cdot \delta + 10 \cdot (2 - \delta) + 1, 5 \cdot \delta + 1 \cdot (2 - \delta) + 7).$$

For $\mathcal{A}$, this value is $\frac{43}{3}$ which means c_1 should be fired at the date $\frac{4}{3}$. The problem of computing automatically this optimal cost is the *Optimal Cost Reachability Control Problem* (OC-RCP).

Of course we can generalise the definition of the optimal cost given for $\mathcal{A}$ to PTGA. With the model of PTGA we have introduced, the *time optimal control* problem is a particular version: in each location, the cost rate is 1 and each discrete transition has a null cost.

Let $\mathsf{WinStrat}(q, \mathcal{G})$ be the set of winning strategies from q in $\mathcal{G}$. Given a strategy s for the contrller, we can define the cost of s from a state q of $\mathcal{T}_{\mathcal{G}}$ by:

$$\mathbf{Cost}(q, s) = \sup \{\mathbf{Cost}(\rho) \mid \rho \in \mathrm{Out}^m(q, f)\} .$$

DEFINITION 3.7 (OPTIMAL COST).– Let $(\mathcal{G}, \mathbf{cost})$ be a PTGA. We consider the RCP for $\mathcal{G}$ and the goal location is GOAL. The *set of possible costs*, $\mathbf{Cost}(q)$, from q in $\mathcal{G}$ when playing s is defined by:

$$\mathbf{Cost}(q) = \{\mathbf{Cost}(q, s) \mid s \in \mathsf{WinStrat}(q, \mathcal{G})\}.$$

The *optimal cost from* q is $\mathbf{OptCost}(q) = \inf \mathbf{Cost}(q)$. The *optimal cost for* $\mathcal{G}$, $\mathbf{OptCost}(\mathcal{G})$, is $\sup_{q \in Q_0} \mathbf{OptCost}(q)$.

REMARK 3.3 .– Notice that we consider the supremum from the initial states of $\mathcal{G}$ and this means that we consider that the choice of the initial state is not controllable. If there is a single initial state (like in the following examples) we have $\mathbf{OptCost}(\mathcal{G}) = \mathbf{OptCost}(q_0)$.

The *Cost Optimal Reachability Control Problem* (CO-RCP) asks the following:

$$\text{Given } (\mathcal{G}, \mathbf{cost}), \text{ compute } \mathbf{OptCost}(\mathcal{G}). \qquad \text{[CO-RCP]}$$

3.7.3. *Computation of the Optimal Cost*

A first step towards a solution of the optimal cost reachability control problem was given for *acyclic* timed games in [LAT 02]. As the PTGA does not have any cycle, the computation of the optimal cost can be reduced to a linear optimisation problem.

If we consider PTGA which can contain cycles, it is not obvious that the algorithm proposed in [LAT 02] can be tuned to accommodate cycles. A solution to the OC-RCP was simultaneoulsy given by Alur *et al.* [ALU 04] and Bouyer *et al.* [BOU 04c, BOU 05a]. [ALU 04] gives some complexity results (lower bound) and in [BOU 04c] gives decidability results for PTGA as well as structural properties of the class of strategies (or controllers) that can win this type of games. The algorithm of [BOU 04c] was implemented in HYTECH [HEN 97] and this is described in [BOU 05a].

We detail here the algorithm of [BOU 04c] to solve the OC-RCP. The idea we propose to solve the optimal cost problem[5] for PTGA is the following: the OC-RCP

[5] As emphasised earlier we assume there is a strategy to win the game *i.e.*, to force GOAL. This ensures that the optimal cost exists and is an finite number.

can be rephrased as: “What is the minimum amount of resources (time, petrol, ...) I should start with, to be able to enforce GOAL, and when GOAL is reached I have not run out of resources?” Thus to solve the OC-RCP we can proceed as follows:

1) we start with a PTGA $\mathcal{A}$ similar to the one given on Figure 3.8;

2) then we build a game $H_{\mathcal{A}}$, which is a timed game with a special variable *rsrc*: we end up with a timed game which is a special kind of linear hybrid automata (LHA). This variable stores the amount of resources which is left at each point in the game. Its value will then decrease in a location ℓ of $H_{\mathcal{A}}$ with a rate which is the opposite of the rate of the cost in the corresponding location in $\mathcal{A}$.

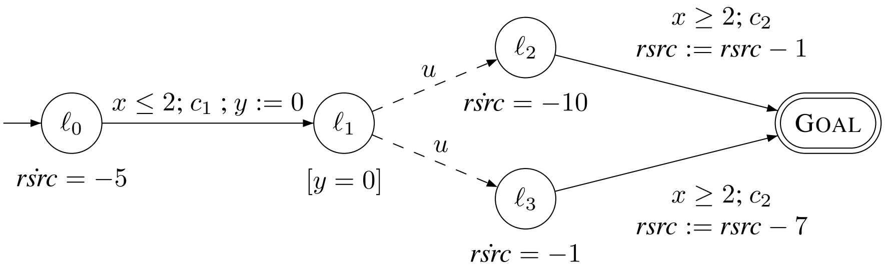


Figure 3.9. *The LHA $H_{\mathcal{A}}$ built from $\mathcal{A}$.*

Applied to $\mathcal{A}$ of Figure 3.8, this construction gives the automaton $H_{\mathcal{A}}$ on Figure 3.9 (the notation $\dot{rsrc}$ stands for the first derivative of *rsrc*). The OC-RCP for PTGA can then be reduced to a (standard) Reachability Control Problem for a *linear hybrid game*[6] (LHA): the control objective is to enforce GOAL and a value of *rsrc* which is larger or equal than zero *i.e.,* the control objective is $(\text{GOAL}, rsrc \geq 0)$. We know how to effectively compute the set of winning states for this type of (linear hybrid) games, and if the algorithm terminates we obtain a set of pairs of the form (ℓ, Z) where ℓ is a location and Z a polyhedron. In particular we may obtain a set of pairs (ℓ_0, Z) giving the set of winning states (values of the variables) in the initial location. By definition of the algorithm to solve linear hybrid games, Z is the maximal winning zone *i.e.,* if one starts outside Z, the game cannot be won.

The projection of the winning set Z in ℓ_0 on the variable *rsrc* gives the set of values of *rsrc* from which we can win the game *i.e.,* enforce $(\text{GOAL}, rsrc \geq 0)$. We can prove that this set is of the form $rsrc \bowtie k$ with $\bowtie \{>, \geq\}$ and this means that the optimal cost is k.

[6] See 3.5.1 in Chapter 7.

Decidability of the OC-RCP for PTGA is also reduced to the decidability of the RCP for the class of LHA that is obtained when we use our translation. Les résultats obtenus par cette démarche sont les suivants:

– for bounded PTGA (all the clocks are bounded) such that on each cycle, the cost is at least $\beta > 0$ (we call this class of PTGA, *cost non-Zeno PTGA*), the classical backward algorithm that computes the winning states of the associated linear hybrid game terminates;

– the optimal cost can be obtained by computing the projection of the set of initial winning states on the variable *rsrc*. When the previous algorithm terminates, this set is always of the form *rsrc* $\bowtie k$ with $\bowtie \{>, \geq\}$ and k is the optimal cost. We can decide whether there is a strategy that guarantees the optimal cost, *i.e.,* an optimal strategy: if the initial winning interval is of the form *rsrc* $\geq k$ the answer is "yes", otherwise it is of the form *rsrc* $> k$ and there is a family of strategies that can ensure a cost $k + \varepsilon$ for any $\varepsilon > 0$. Still there is no optimal strategy;

– optimal strategies when they exist, may need cost information *i.e.,* the accumulated amount of resources consumed since the game started. In this sense, the clocks of the PTGA are not always sufficient to decide what to do to ensure winning at optimal cost, and we may need the cost information.

3.7.4. *Recent Results and Open Problems*

Since these results were published, some new results about optimal cost for PTGA were obtained:

– in [BRI 05], the authors prove that the *Cost non-Zeno* the optimal cost computation problem is undecidable. This shows that the assumption we had made to prove termination of our algorithm is necessary. This proof was refined in [BOU 06c] and the most recent result is that for PTGA with 3 clocks the optimal cost computation problem is undecidable;

– in [BOU 06e], the authors prove that for turn-based PTGA with one clock, the optimal cost can be computed (3EXPTIME).

– finally, in [BOU 07], there is a proof that the optimal cost is computable for o-minimal automata.

An interesting open problem is the Cost Optimal Safety Control Problem where the goal is to minimize the cost over infinite runs. This can be viewed as the generalisation of the result in[BOU 08a, BOU 04a] (costs and rewards) to timed games with costs and rewards.

3.8. Efficient Algorithms for Controller Synthesis

In the last ten years, a lot of progress has been made in the design of efficient tools for the analysis (model-checking) of timed systems. Tools like KRONOS [YOV 97] or UPPAAL [AMN 07, AMN 01] have become very efficient and widely used to check properties of timed automata but still no real efficient counterpart had been designed for timed games.

One of the main reasons behind the efficiency of tools like UPPAAL is the *on-the-fly* generation of the state space (this is also due to the use of efficient data structures in the implemtation of the algorithms). Notice that *on-the-fly* algorithms were already crucial for model-checkers for finite communicating systems like SPIN [HOL 07].

The algorithm for computing controllers for safety timed games described in section 3.3.3 is based on *backwards* fix-point computations of the set of winning states [MAL 95, ASA 98, ALF 01]. We call this algorithm a *backward algorithm*. A drawback of a backward algorithm is that it does not take into account the fact that a state is reachable from the initial state or not. Backward computation may thus yield an enormous set of winning (symbolic) states most of which could be non unreachable. Moreover, at each iteration, we compute the : pi operator for these unreachable states.

A more difficult problem is that for simple extensions of timed automata, it is very difficult and sometimes impossible to compute the controllable predecessor operator π. For instance, if transitions are of the form $i := 2j + k$ (which is allowed in UPPAAL), computing the controllable predecessors of the symbolic state $(\ell, j \geq 0 \wedge k \geq 0)$ means collecting all the values of i such that $i = 2j + k$. This computation can be very expensive. In practise, if we start from a given initial state and given j and k, there might be a few reachable (admissible) values for j and k and consequently a few for i. Or a least, for each new value of i which is reached, we know the values of j and k that were used to obtain i.

A first step towards a solution is to compute the set of reachable states R. Then for each step of the iterative algorithm using π for computing winning states, take the intersection on the winning states and R. Even if these sets are represented with efficient data structures this can be very expensive and imply the computation of the whole state space of the system which may be unncessary to check whether a winning strategy exists.

3.8.1. *On-the-fly Algorithms*

To obtain a real *on-the-fly* algorithm for computing winning states, we should not have to compute the set R of reachable states before we compute the winning set of states but rather explore the symbolic state space as needed.

Regarding timed games, in [ALT 99, ALT 02], Karine Altisen and Stavros Tripakis have proposed a (partially) on-the-fly method for solving timed games. This algorthm consists in first computing an abstraction of the region graph (which can be huge) and second use an *on-the-fly* algorithm on the abstraction. In this sense, this is not a real *on-the-fly* algorithm on the timed automaton as we first have to compute a finite abstraction of the set of reachable states.

To design a truly *on-the-fly* algorithm for the computation of winning states for reachability timed game automata we start from the *on-the-fly* algorithm suggested by Liu & Smolka in [LIU 98] for linear-time model-checking of finite-state systems.

To obtain an on-the-fly algorithm for TGA, we can first extend the algorithm suggested by Liu & Smolka in [LIU 98] to the RCP for finite automata.

The idea for this on-the-fly algorithm for a finite game $\mathcal{A}$ is the following. We assume that some variables store two sets of transitions: `ToExplore` store the transitions that have been explored and `ToBackPropagate` store the transitions the target states of which has been declared winning. Moreover, the set of explored states R of $\mathcal{A}$ is also stored. For each state s of $\mathcal{A}$ already explored we keep track of its *statut* which is either *winning* (this is te initial value for state s in GOAL) or *unknown* (this is the initial value for the other states).

To perform a step of the on-the-fly algorithm, we pick a transition (s, a, s') in one of the two sets `ToExplore` or `ToBackPropagate` and we do step (A1) for a transition from `ToExplore` and (A2) for a transition from `ToBackPropagate`:

– **(A1)** we explore (s, a, s'):

– if s' is encoutered for the first time, we add the outgoing transitions of s' to `ToExplore`; if s' is a winning state in GOAL, we update the status of s' to *winning* and otherwise set it to *unknown*;

– if the status of s' is *winning*, we add (s, a, s') to the set `ToBackPropagate`; otherwise we just keep track of the fact that "the status of s depends on the status of s' via transition (s, a, s')";

– **(A2)** we backpropagate some information for the transition (s, a, s'). Actually, if it is in `ToBackPropagate`, this must be because the status of s' became *winning* recently and we have to update the status of s accordingly:

– either a is a controllable action, and in this case we update a counter $c(s)$. The meaning of $c(s)$ is "$c(s)$ is the number of controllable transitions from s leading to a winning state";

– or a is uncontrollable and we increment a counter $u(s)$. The meaning of $u(s)$ is: "$u(s)$ is the number of uncontrollable transitions from s leading to a winning state";

– Once $c(s)$ or $u(s)$ has been updated, the status of s may change. Let $U(s)$ be the total number of uncontrollable transitions originating from s in $\mathcal{A}$. If $c(s) \geq 1$ and $u(s) = U(s)$, s can be declared a *winning* state. The status of the states which depend on s has to be re-considered. Thus we add to `ToBackPropagate` all the transitions (s'', a, s) such that the status of s'' depends on s via (s'', a, s).

An on-the-fly algorithm for reachability games will process the sets `ToExplore` and `ToBackPropagate` in a random order. As soon as the status of the initial state of $\mathcal{A}$ is *winning*, we can stop the algorithm and the answer to the RCP is "yes" (moreover a winning strategy can be computed). This algorithm is linear in the size of $\mathcal{A}$. Indeed, each transition is processed at most twice: once in `ToExplore` and once in `ToBackPropagate`. To design the on-the-fly version for timed games, we consider *symbolic* states or zones which are unions of regions. Given a TGA and a symbolic state (ℓ, Z) we do a forward exploration of each transition from location ℓ to ℓ' by computing the discret succesors of Z and the time successors (contrained by ℓ' invariant). The status of the symbolic is then updated as for the untimed case. The on-the-fly algorithm for timed games together with the correctness proof are given in [CAS 05]. Actually, for timed games, the algorithm we give is not linear in the size of the region graph! We use zones to store symbolic states, and a region maybe included in several zones. Our algorithm is thus not optimal as there is an algorithm which is linear in the size of the region graph [ALT 99]. However, this algorithl is really on-the-fly, and in practise gives good results, as witnessed by the experiments carried out with the games version of UPPAAL called UPPAAL-TIGA [BEH 07a] which is presented in Chapter 6.

3.8.2. *Recent Results and Open Problems*

Using the previous algorothm, it is possible to compute *time optimal* controllers (cost associated with locations are all 1 and cost associated with discrete transitions are all 0). It suffices to give an upper bound on the maximum time to reach GOAL (see [CAS 05] for details). There is also a "safety games" version of the previous algorithm . In [CAS 07b], this on-the-fly algorithm has been adpated to games under *partial observation* (see next section 3.9). An ovserview of the on-the-fly algorithms for timed games us given in [CAS 07a].

Some open problems still remain:

– on-the-fly algorithm could be extended to more general control objectives like repeated reachability (Büchi objective). This is really useful for timed games as Büchi objectives allow to enforce a controller to be non-Zeno;

– anotehr direction is to extend the previous algorithm and find good data structures to solve the optimal cost reachability control problem (see section 3.7).

3.9. Partial observation

In the previous sections, we (tacitly) assumed that the controller had perfect knowledge of the environment, and that it could observe all the actions and states of the system. In practice, it might not always be the case: in some cases, the controller can only observe some of the actions of the environment (which are said to be *observable*), and can only have partial informations about the current state of the environment. Still, we would like to be able, as much as possible, to control the environment: this problem is known as *Control Under Partial Observation* (referred to as PO in the sequel).

In this case, we keep assuming that the set of events is partitionned into two sets —controllable (Σ_c) and uncontrollable (Σ_u)— but only a subset Σ_u^o of Σ_u is *observable*. The controller is only able to observe a timed trace $(a_0, t_0)(a_1, t_1) \cdots (a_n, t_n)$ in $((\Sigma_c \cup \Sigma_u^o) \times \mathbb{R}_{\geq 0})^*$, and based on this information, it must decide which actions to play.

As regards strategies, we say that a strategy f is *trace-based* if for any two runs ρ and ρ' yielding the same observation[7], we have $f(\rho) = f(\rho')$.

The control problem under partial observation (PO-CP) aims at deciding whether there exists a winning *trace-based* strategy f (*i.e.*, a *trace-based controller*, TBC). The PO-CP problem is then the following:

> given a partially observable system S and an objective ϕ, does there exists a TBC C such that $(S \parallel C) \models \phi$? [PO-CP]

A survey on partial observation in timed systems is given in [BOU 05c]. In case the specification is given as a deterministic timed automaton, the PO-CP is undecidable [BOU 03]. Even if we restrict to the *safety* fragment and if we are only interested in synthesizing non-Zeno controllers (Non-Zeno-PO-SCP), the problem remains undecidable [BOU 06d].

However, by fixing in advance the resources available to the controller, the proof of [DSO 02] on control with external specifications (see Sect. 3.6) can be extended to the partial observation setting. The result is then the following: if the resources available to the controller are fixed and if the control objective ϕ is given as a deterministic finite automaton, then the PO-CP is EXPTIME-complete [BOU 03].

Recent works [CHA 06, DEW 06] focus on partial observation of the *states* of a finite-state system. A timed extension of those works is given in [CAS 07b].

[7] *I.e.*, having the same projection on $((\Sigma_c \cup \Sigma_u^o) \times \mathbb{R}_{\geq 0})^*$.

3.10. Changing game rules ...

As mentionned above when defining TGAs, several semantics have been defined for timed games. We present below an alternative semantics, based on [ALF 03], and whose main advantage is that it can handle Zeno strategies.

Informally[8], the main difference in the rules is the following: instead of choosing between waiting and performing an action, each player concurrently choose an action and *how long* they want to wait before performing that action. The rest is identical: the player with the shortest delay is selected, and his action is performed after delaying for the amount of time this player had chosen.

Though this might look like a minor change, this semantics has one important advantage: at each step, we know which player has chosen the shortest delay. That way, along Zeno runs, we are able to detect which player has been unfair by choosing converging delays. The other player is declared the winner in this case[9]

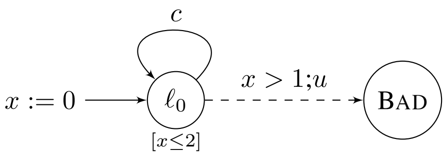


Figure 3.10. *A (safety) game in which winning requires Zenoness*

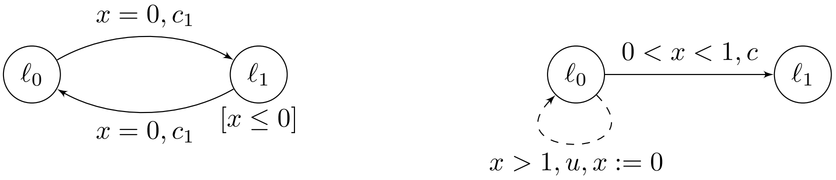
Let us go back to our example of Section 3.4, depicted on Fig. 3.10. Consider the following strategy: in state (ℓ_0, x) with $x < 1$, the controller proposes a positive delay δ such that, after this delay, the value of x is of the form $1 - 1/n$ (this is always possible when $x < 1$). Together with this value, the controller decides to perform action c. With our new semantics, this strategy is not winning: it suffices for the environment to play the strategy "*wait until* $x = 1$ *and perform action* u" to win, since the controller will be disqualified.

This setting has drawbacks unfortunately. For instance, even for simple reachability objectives, winning strategies might need memory: from state $(\ell_0, 0)$ of the game depicted on Fig. 3.11(a), the controller has to play c_1 at date 0 (in order to satisfy the reachability condition). The play will then come back to the same state $(\ell_0, 0)$ after the second move. If the controller is memoryless, he will keep applying this strategy, which results in a Zeno run.

[8] We won't give the precise definition of those games in this chapter, and refer the eager reader to [ALF 03, BRI 07, HEN 06b], where the details can be found.

[9] It could be the case that both players are disqualified. There is no winner in this case.

Even though the players cannot rely on Zeno behaviours to win, there are examples where it is necessary to apply a "Zeno strategy" for a finite number of steps. Figure 3.11(b) depicts such an example: the objective in this game being to reach state ℓ_1, the controller will play action c at dates $1/2$, $3/4$, $7/8$, etc., always letting $1/2^{n+1}$ time units elapse between the n-th and $n+1$-st step. This strategy is Zeno, but it is intended to be applied only a finite number of times. Plays that are winning for the environment under that strategy are plays in which the environment itself always plays smaller delays. In that case, the environment loses.



(a) strategy requiring memory

(b) "surprise" strategy

Figure 3.11. *Examples of timed games*

Under such rules, most of the results presented in this chapter still hold. In particular, the notion of *controllable predecessors* can be extended to this setting, and zone-based arguments are still valid; that way, fixpoint techniques can be developped, still resulting in exponential time algorithms.

Recent extensions of those models have been proposed, where all the players can act upon the action to be performed even if they don't choose the smallest delay. Those games extend classical *concurrent games* [ALU 02] with clocks: in this setting, each player will not only tell his preferred delay and action, but also the actions he would like to perform is a smaller delay were to be chosen by some other player. The resulting action would then depend on all the choices of all the players, even if their preferred delays were too large. In [BRI 07], the techniques and results presented above have been adapted to this framework.

3.11. Bibliography

[ALF 01] DE ALFARO L., HENZINGER T.A., MAJUMDAR R., "Symbolic Algorithms for Infinite-State Games", *Proc. 12th International Conference on Concurrency Theory (CONCUR'01)*, vol. 2154 of *Lecture Notes in Computer Science*, p. 536-550, Springer, 2001.

[ALF 03] DE ALFARO L., FAËLLA M., HENZINGER T.A., MAJUMDAR R., STOELINGA M., "The Element of Surprise in Timed Games", *Proc. 14th International Conference on Concurrency Theory (CONCUR'03)*, vol. 2761 of *Lecture Notes in Computer Science*, p. 142-156, Springer, 2003.

[ALT 99] ALTISEN K., TRIPAKIS S., "On-the-Fly Controller Synthesis for Discrete and Dense-Time Systems", *Proc. World Congress on Formal Methods in the Development of Computing System (FM'99)*, vol. 1708 of *Lecture Notes in Computer Science*, p. 233-252, Springer, 1999.

[ALT 02] ALTISEN K., TRIPAKIS S., "Tools for Controller Synthesis of Timed Systems", *Proc. 2nd Workshop on Real-Time Tools (RT-TOOLS'02)*, 2002, Proc. published as Technical Report 2002-025, Uppsala University, Sweden.

[ALT 05] ALTISEN K., MARKEY N., REYNIER P.A., TRIPAKIS S., "Implémentabilité des automates temporisés", *Actes du 5ème Colloque sur la Modélisation des Systèmes Réactifs (MSR'05)*, p. 395-406, Autrans, France, Hermès, October 2005.

[ALU 94] ALUR R., DILL D., "A Theory of Timed Automata", *Theoretical Computer Science*, vol. 126, num. 2, p. 183-235, Elsevier Science, 1994.

[ALU 01] ALUR R., LA TORRE S., PAPPAS G.J., "Optimal Paths in Weighted Timed Automata", *Proc. 4th International Workshop Hybrid Systems, Computation and Control (HSCC'01)*, vol. 2034 of *Lecture Notes in Computer Science*, p. 49-62, Springer, 2001.

[ALU 02] ALUR R., HENZINGER T.A., KUPFERMAN O., "Alternating-time temporal logic", *Journal of the ACM*, vol. 49, p. 672-713, ACM Press, 2002.

[ALU 04] ALUR R., BERNADSKY M., MADHUSUDAN P., "Optimal Reachability in Weighted Timed Games", *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, vol. 3142 of *Lecture Notes in Computer Science*, p. 122-133, Springer, 2004.

[AMN 01] AMNELL T., BEHRMANN G., BENGTSSON J., D'ARGENIO P.R., DAVID A., FEHNKER A., HUNE T., JEANNET B., LARSEN K.G., MÖLLER O., PETTERSSON P., WEISE C., YI W., "UPPAAL – Now, Next, and Future", *Proc. Modelling and Verification of Parallel Processes (*MOVEP*2k)*, vol. 2067 of *Lecture Notes in Computer Science*, p. 99-124, Springer, 2001.

[AMN 07] AMNELL T., BEHRMANN G., BENGTSSON J., D'ARGENIO P.R., DAVID A., FEHNKER A., HUNE T., JEANNET B., LARSEN K.G., MÖLLER O., PETTERSSON P., WEISE C., YI W., "UPPAAL", 2007, http://www.uppaal.com.

[ARN 03] ARNOLD A., VINCENT A., WALUKIEWICZ I., "Games for synthesis of controllers with partial observation", *Theoretical Computer Science*, vol. 1, num. 303, p. 7-34, Elsevier Science, 2003.

[ASA 98] ASARIN E., MALER O., PNUELI A., SIFAKIS J., "Controller Synthesis for Timed Automata", *Proc. IFAC Symposium on System Structure and Control*, p. 469-474, Elsevier Science, 1998.

[ASA 99] ASARIN E., MALER O., "As Soon as Possible: Time Optimal Control for Timed Automata", *Proc. 2nd International Workshop Hybrid Systems, Computation and Control (HSCC'99)*, vol. 1569 of *Lecture Notes in Computer Science*, p. 19-30, Springer, 1999.

[BEH 01a] BEHRMANN G., FEHNKER A., HUNE T., LARSEN K.G., PETTERSSON P., ROMIJN J., VAANDRAGER F., "Efficient Guiding Towards Cost-Optimality in UPPAAL", *Proc. 7th International Conference on Tools and Algorithms for the Construction and Analysis* *of Systems (TACAS'01)*, vol. 2031 of *Lecture Notes in Computer Science*, p. 174-188, Springer, 2001.

[BEH 01b] BEHRMANN G., FEHNKER A., HUNE T., LARSEN K.G., PETTERSSON P., ROMIJN J., VAANDRAGER F., "Minimum-Cost Reachability for Priced Timed Automata", *Proc. 4th International Workshop Hybrid Systems, Computation and Control (HSCC'01)*, vol. 2034 of *Lecture Notes in Computer Science*, p. 147-161, Springer, 2001.

[BEH 07a] BEHRMANN G., COUGNARD A., DAVID A., FLEURY E., LARSEN K.G., LIME D., "UPPAAL-TiGA", 2007, http://www.cs.aau.dk/~adavid/tiga/.

[BEH 07b] BEHRMANN G., LARSEN K.G., RASMUSSEN J.I., "UPPAAL-CORA", 2007, http://www.cs.aau.dk/~behrmann/cora/.

[BOU 03] BOUYER P., D'SOUZA D., MADHUSUDAN P., PETIT A., "Timed Control with Partial Observability", *Proc. 15th International Conference on Computer Aided Verification (CAV'03)*, vol. 2725 of *Lecture Notes in Computer Science*, p. 180-192, Boulder, États-Unis, Springer, July 2003.

[BOU 04a] BOUYER P., BRINKSMA E., LARSEN K.G., "Staying Alive As Cheaply As Possible", *Proc. 7th International Conference Hybrid Systems, Computation and Control (HSCC'04)*, vol. 2993 of *Lecture Notes in Computer Science*, p. 203-218, Philadelphie, États-Unis, Springer, March 2004.

[BOU 04b] BOUYER P., CASSEZ F., FLEURY E., LARSEN K.G., Optimal Strategies in Priced Timed Game Automata, BRICS Reports Series num. RS-04-0, BRICS, Aalborg, Danemark, February 2004, ISSN 0909-0878.

[BOU 04c] BOUYER P., CASSEZ F., FLEURY E., LARSEN K.G., "Optimal Strategies in Priced Timed Game Automata", *Proc. 24th Conference on Fundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, vol. 3328 of *Lecture Notes in Computer Science*, p. 148-160, Chennai, Inde, Springer, December 2004.

[BOU 05a] BOUYER P., CASSEZ F., FLEURY E., LARSEN K.G., "Synthesis of Optimal Strategies Using HyTech", *Proc. Workshop on Games in Design and Verification (GDV'04)*, vol. 119 of *Electronic Notes in Theoretical Computer Science*, p. 11-31, Boston, États-Unis, Elsevier Science, February 2005.

[BOU 05b] BOUYER P., CASSEZ F., LAROUSSINIE F., "Modal Logics for Timed Control", *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, vol. 3653 of *Lecture Notes in Computer Science*, p. 81-94, San Francisco, États-Unis, Springer, 2005.

[BOU 05c] BOUYER P., CHEVALIER F., KRICHEN M., TRIPAKIS S., "Observation partielle des systèmes temporisés", *Actes du 5ème Colloque sur la Modélisation des Systèmes Réactifs (MSR'05)*, p. 381-393, Autrans, France, Hermès, October 2005.

[BOU 06a] BOUYER P., BOZZELLI L., CHEVALIER F., "Controller Synthesis for MTL Specifications", *Proc. 17th International Conference on Concurrency Theory (CONCUR'06)*, vol. 4137 of *Lecture Notes in Computer Science*, p. 450-464, Bonn, Allemagne, Springer, August 2006.

[BOU 06b] BOUYER P., BRIHAYE T., CHEVALIER F., "Control in o-Minimal Hybrid Systems", *Proc. 21st IEEE Symposium on Logic in Computer Science (LICS'06)*, p. 367-378, Seattle, États-Unis, IEEE Computer Society Press, August 2006.

[BOU 06c] BOUYER P., BRIHAYE T., MARKEY N., “Improved Undecidability Results on Weighted Timed Automata”, *Information Processing Letters*, vol. 98, num. 5, p. 188-194, Elsevier Science, June 2006.

[BOU 06d] BOUYER P., CHEVALIER F., “On the Control of Timed and Hybrid Systems”, *EATCS Bulletin*, vol. 89, p. 79-96, European Association for Theoretical Computer Science, June 2006.

[BOU 06e] BOUYER P., LARSEN K.G., MARKEY N., RASMUSSEN J.I., “Almost Optimal Strategies in One-Clock Priced Timed Automata”, *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS’06)*, vol. 4337 of *Lecture Notes in Computer Science*, p. 345-356, Kolkata, Inde, Springer, December 2006.

[BOU 06f] BOUYER P., MARKEY N., REYNIER P.A., “Robust Model-Checking of Linear-Time Properties in Timed Automata”, *Proc. 7th Latin American Symposium on Theoretical Informatics (LATIN’06)*, vol. 3887 of *Lecture Notes in Computer Science*, p. 238-249, Valdivia, Chili, Springer, March 2006.

[BOU 07] BOUYER P., BRIHAYE T., CHEVALIER F., “Weighted O-Minimal Hybrid Systems are more Decidable than Weighted Timed Automata!”, *Proc. Symposium on Logical Foundations of Computer Science (LFCS’07)*, Lecture Notes in Computer Science, New-York, États-Unis, Springer, June 2007.

[BOU 08a] BOUYER P., BRINKSMA E., LARSEN K.G., “Optimal Infinite Scheduling for Multi-Priced Timed Automata”, *Formal Methods in System Design*, vol. 32, num. 1, p. 2-23, Kluwer Academic Publishers, February 2008.

[BOU 08b] BOUYER P., MARKEY N., REYNIER P.A., “Robust Analysis of Timed Automata *via* Channel Machines”, *Proc. 11th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS’08)*, vol. 4962 of *Lecture Notes in Computer Science*, p. 157-171, Budapest, Hongrie, Springer, March 2008.

[BRI 04] BRIHAYE T., BRUYÈRE V., RASKIN J.F., “Model-Checking for Weighted Timed Automata”, *Proc. Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS’04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT’04)*, vol. 3253 of *Lecture Notes in Computer Science*, p. 277-292, Springer, 2004.

[BRI 05] BRIHAYE T., BRUYÈRE V., RASKIN J.F., “On Optimal Timed Strategies”, *Proc. 3th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS’05)*, vol. 3829 of *Lecture Notes in Computer Science*, p. 49-64, Springer, 2005.

[BRI 07] BRIHAYE T., LAROUSSINIE F., MARKEY N., OREIBY G., “Timed Concurrent Game Structures”, *International Conference on Concurrency Theory (CONCUR’07)*, vol. 4703 of *Lecture Notes in Computer Science*, p. 445-459, Springer, 2007.

[CAS 02] CASSEZ F., HENZINGER T.A., RASKIN J.F., “A Comparison of Control Problems for Timed and Hybrid Systems”, *Proc. 5th International Workshop on Hybrid Systems, Computation and Control (HSCC’02)*, vol. 2289 of *Lecture Notes in Computer Science*, p. 134-148, Springer, 2002.

[CAS 05] CASSEZ F., DAVID A., FLEURY E., LARSEN K.G., LIME D., "Efficient On-The-Fly Algorithms for the Analysis of Timed Games", *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, vol. 3653 of *Lecture Notes in Computer Science*, p. 66-80, San Francisco, États-Unis, Springer, August 2005.

[CAS 07a] CASSEZ F., "Efficient On-the-Fly Algorithms for Partially Observable Timed Games", *Proc. 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, vol. 4763 of *Lecture Notes in Computer Science*, p. 5-24, Springer, 2007.

[CAS 07b] CASSEZ F., DAVID A., LARSEN K.G., LIME D., RASKIN J.F., "Timed Control with Observation Based and Stuttering Invariant Strategies", *Proc. 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, vol. 4762 of *Lecture Notes in Computer Science*, p. 307-321, Springer, 2007.

[CHA 06] CHATTERJEE K., DOYEN L., HENZINGER T.A., RASKIN J.F., "Algorithms for Omega-Regular Games with Imperfect Information", *Proc. 20th International Workshop on Computer Science Logic (CSL'06)*, p. 287-302, 2006.

[DEW 04a] DE WULF M., DOYEN L., MARKEY N., RASKIN J.F., "Robustness and Implementability of Timed Automata", *Proc. Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, vol. 3253 of *Lecture Notes in Computer Science*, p. 118-133, Springer, 2004.

[DEW 04b] DE WULF M., DOYEN L., RASKIN J.F., "Almost ASAP Semantics: From Timed Models to Timed Implementations", *Proc. 7th International Workshop Hybrid Systems, Computation and Control (HSCC'04)*, vol. 2993 of *Lecture Notes in Computer Science*, p. 296-310, Springer, 2004.

[DEW 05a] DE WULF M., DOYEN L., RASKIN J.F., "Almost ASAP Semantics: From Timed Models to Timed Implementations", *Formal Aspects of Computing*, vol. 17, num. 3, p. 319-341, Springer, 2005.

[DEW 05b] DE WULF M., DOYEN L., RASKIN J.F., "Systematic Implementation of Real-Time Models", *Proc. International Symposium of Formal Methods Europe (FM'05)*, vol. 3582 of *Lecture Notes in Computer Science*, p. 139-156, Springer, 2005.

[DEW 06] DE WULF M., DOYEN L., RASKIN J., "A Lattice Theory for Solving Games of Imperfect Information", *Proc. 9th International Workshop Hybrid Systems, Computation and Control (HSCC'06)*, p. 153-168, 2006.

[DOY 07] DOYEN L., "Robust Parametric Reachability for Timed Automata", *Information Processing Letters*, vol. 102, num. 5, p. 208-213, Elsevier Science, 2007.

[DSO 02] D'SOUZA D., MADHUSUDAN P., "Timed Control Synthesis for External Specifications", *Proc. 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, vol. 2285 of *Lecture Notes in Computer Science*, p. 571-582, Springer, 2002.

[FAE 02a] FAËLLA M., LA TORRE S., MURANO A., "Automata-Theoretic Decision of Timed Games", *Proc. 3rd International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI'02)*, vol. 2294 of *Lecture Notes in Computer Science*, p. 240-254, Venise, Italie, Springer, January 2002.

[FAE 02b] FAËLLA M., LA TORRE S., MURANO A., "Dense Real-Time Games", *Proc. 17th IEEE Symposium on Logic in Computer Science (LICS'02)*, p. 167-176, IEEE Computer Society Press, 2002.

[FRE 05] FREHSE G., "PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech", *HSCC'05*, vol. 3414 of *LNCS*, p. 258-273, Springer, 2005.

[HEN 96] HENZINGER T.A., "The Theory of Hybrid Automata", *Proc. 11th IEEE Symposium on Logic in Computer Science (LICS'96)*, p. 278-292, 1996.

[HEN 97] HENZINGER T.A., HO P.H., WONG-TOI H., "HYTECH: A Model-Checker for Hybrid Systems", *Journal on Software Tools for Technology Transfer*, vol. 1, num. 1-2, p. 110-122, Springer, 1997.

[HEN 98] HENZINGER T.A., KOPKE P.W., PURI A., VARAIYA P., "What's decidable about hybrid automata?", *Journal of Computer and System Sciences*, vol. 57, p. 94-124, Elsevier Science, 1998.

[HEN 99a] HENZINGER T.A., HOROWITZ B., MAJUMDAR R., "Rectangular Hybrid Games", *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, vol. 1664 of *Lecture Notes in Computer Science*, p. 320-335, Springer, 1999.

[HEN 99b] HENZINGER T.A., KOPKE P.W., "Discrete-time control for rectangular hybrid automata", *Theoretical Computer Science*, vol. 221, p. 369-392, Elsevier Science, 1999.

[HEN 06a] HENZINGER T.A., PRABHU V.S., "Timed Alternating-Time Temporal Logic", *Proc. 4th International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'06)*, vol. 4202 of *Lecture Notes in Computer Science*, p. 1-17, Springer, September 2006.

[HEN 06b] HENZINGER T.A., PRABHU V.S., "Timed Alternating-Time Temporal Logic", *Proc. 4th International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'06)*, vol. 4202 of *Lecture Notes in Computer Science*, p. 1-17, Springer, September 2006.

[HOF 92] HOFFMANN G., WONG-TOI H., "The input-output control of real-time discrete-event systems", *Proc. 13th Annual Real-time Systems Symposium*, p. 256-265, IEEE Computer Society Press, 1992.

[HOL 07] HOLTZMANN G., "SPIN", 2007, http://spinroot.com.

[KOY 90] KOYMANS R., "Specifying Real-Time Properties with Metric Temporal Logic", *Real-Time Systems*, vol. 2, num. 4, p. 255-299, 1990.

[LAF 00] LAFFERRIERE G., PAPPAS G., SASTRY S., "O-minimal hybrid systems", *Mathematics of Control Signals and Systems*, vol. 13, num. 1, p. 1-21, 2000.

[LAR 98] LAROUSSINIE F., LARSEN K.G., "CMC: A Tool for Compositional Model-Checking of Real-Time Systems", *Proc. IFIP Joint International Conference on Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, p. 439-456, Kluwer Academic, 1998.

[LAR 05] LAROUSSINIE F., "CMC", 2005, http://www.lsv.ens-cachan.fr/~fl/cmcweb.html.

[LAR 06] LAROUSSINIE F., MARKEY N., OREIBY G., “Model Checking Timed ATL for Durational Concurrent Game Structures”, *Proc. 4th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS’06)*, vol. 4202 of *Lecture Notes in Computer Science*, p. 245-259, Paris, France, Springer, September 2006.

[LAT 02] LA TORRE S., MUKHOPADHYAY S., MURANO A., “Optimal-Reachability and Control for Acyclic Weighted Timed Automata”, *Proc. 2nd IFIP International Conference on Theoretical Computer Science (TCS’02)*, vol. 223 of *IFIP Conference Proceedings*, p. 485-497, Kluwer, 2002.

[LIU 98] LIU X., SMOLKA S.A., “Simple Linear-Time Algorithm for Minimal Fixed Points”, *Proc. 26th International Conference on Automata, Languages and Programming (ICALP’98)*, vol. 1443 of *Lecture Notes in Computer Science*, p. 53-66, Aalborg, Danemark, Springer, 1998.

[MAL 95] MALER O., PNUELI A., SIFAKIS J., “On the Synthesis of Discrete Controllers for Timed Systems”, *Proc. 12th Symposium on Theoretical Aspects of Computer Science (STACS’95)*, vol. 900 of *Lecture Notes in Computer Science*, p. 229-242, Springer, 1995.

[MAR 75] MARTIN D.A., “Borel determinacy”, *Annals of Mathematics*, vol. 102, num. 2, p. 363-371, 1975.

[MCN 93] MCNAUGHTON R., “Infinite Games Played on Finite Graphs”, *Annals of Pure and Applied Logic*, vol. 65, num. 2, p. 149-184, Elsevier Science, 1993.

[MER 74] MERLIN P., A study of the recoverability of computing systems, PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA, 1974.

[RAM 87] RAMADGE P.J., WONHAM W.M., “Supervisory Control of a Class of Discrete Event Processes”, *SIAM Journal of Control and Optimization*, vol. 25, num. 1, p. 1202-1218, 1987.

[RAM 89] RAMADGE P.J., WONHAM W.M., “The Control of Discrete Event Systems”, *Proceedings of the IEEE*, vol. 77, num. 1, p. 81-98, 1989.

[RIE 03] RIEDWEG S., PINCHINAT S., “Quantified Mu-Calculus for Control Synthesis”, *Proc. 28th International Symposium on Mathematical Foundations of Computer Science (MFCS’03)*, vol. 2747 of *Lecture Notes in Computer Science*, p. 642-651, Springer, 2003.

[SCH 99] SCHNOEBELEN P., BÉRARD B., BIDOIT M., LAROUSSINIE F., PETIT A., *Vérification de logiciels : Techniques et outils de model-checking*, Vuibert, Paris, 1999.

[THO 95] THOMAS W., “On the Synthesis of Strategies in Infinite Games”, *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS’95)*, vol. 900 of *Lecture Notes in Computer Science*, p. 1-13, Springer, 1995.

[YOV 97] YOVINE S., “KRONOS: A Verification Tool for Real-Time Systems”, *Journal of Software Tools for Technology Transfer*, vol. 1, num. 1-2, p. 123-133, Springer, 1997.