Automata-theoretic Decision of Timed Games*

Marco Faella[1], Salvatore La Torre[1,2], and Aniello Murano[1,3]

[1]Università degli Studi di Salerno
[2]University of Pennsylvania
[3]Rice University
mfaella,sallat,murano@dia.unisa.it

Abstract. The solution of games is a key decision problem in the context of verification of open systems and program synthesis. We present an automata-theoretic approach to solve timed games. Our solution gives a general framework to solve many classes of timed games via a translation to tree automata, extending to timed games a successful approach to solve discrete games. Our approach relies on translating a timed automaton into a tree automaton that accepts all the trees corresponding to a given strategy of the protagonist. This construction exploits the region automaton introduced by Alur and Dill. We use our framework to solve timed Büchi games in exponential time, timed Rabin games in exponential time, CTL games in exponential time and LTL games in doubly exponential time. All these results are tight in the sense that they match the known lower bounds on these decision problems.

1 Introduction

The theory of games was originally introduced as a theoretical model for economic studies (see for example [20]). In the years, this theory has received an increasing interest by many researchers in both computer science and control theory. Games have been studied in the context of discrete [11, 22, 27, 6], timed [9, 8], and hybrid systems [14]. They provide a suitable framework for the program synthesis and the verification of *open systems*, that is, systems whose behavior depends on the current state as well as the behavior of the environment in which they are embedded.

The notion of open system naturally arises in the compositional modeling and design of *reactive systems*, that is systems that maintain an on-going interaction with their environment [7, 22]. A reactive system can be seen as divided into many components interacting with each other, and each component can be modeled as an open system. In automated verification, systems are often modeled as *closed systems*, where a system behavior is completely determined by the current state. The verification problem can thus be phrased as: given an abstract model (transition system) M and a specification φ, we wish to determine

* This research was partially supported by the NSF awards CCR99-70925 and CCR99-88322, SRC award 99-TJ-688, DARPA ITO Mobies award F33615-00-C-1707, NSF ITR award, and by the MURST project "MEFISTO".

if φ holds for the computations of M (*model-checking*). Model-checking is a very successful technology which has been implemented in many tools.

The decision problem we consider in this paper is analogous to model-checking. We are given a specification and a game graph (alternating transition system), where the transitions are determined by the moves of two players. We wish to determine if a player has a strategy to ensure that, independently from the choices of the other player, the resulting computation satisfies the specification. We recall that for simple specifications such as "always p" and relatively to discrete-time systems, such model-checking games have already been implemented in the software Mocha [5], and shown to be useful in construction of the most-general environments for automating assume-guarantee reasoning [1].

To refer to delays, time needs to be explicitly included in the model for a reactive system. In this paper we focus on timed games and model them as nondeterministic *timed automata* [3]: a finite automaton augmented with a finite set of real-valued *clocks*. The transitions of a timed automaton are enabled according to the current state, that is, the current *location* and the current clock values. In a transition, clocks can be instantaneously reset. The value of a clock is exactly the time elapsed since the last time it was reset. A clock constraint (*guard*) is associated to each transition with the meaning that a transition can be taken only if the associated guard is enabled. Moreover, a clock constraint (*invariant*) is also associated to each location with the meaning that the automaton can stay in a location as long as the corresponding invariant remains true.

When interpreting a nondeterministic timed automaton as a game graph, we capture the choices of the *protagonist* by the symbols associated with the transitions and nondeterminism is used to model the possible choices of the *antagonist*[1]. To model the case that the protagonist stays *idle* and the antagonist is moving, we use a special symbol denoted by ε. The case that both players stay idle is captured by letting time elapse in a location. A *play* of a timed game is thus constructed in the following way. At each time, a player declares how long it will wait idling and its next choice. At the time one of the players or both move, both players are allowed to redeclare their next move and the time they will issue it. That is, if a player moves before the other, this latter is allowed to change its last declared decision. Technically, a play is a run of the automaton modeling the game. The winning condition for the protagonist is expressed by a predicate over system behaviors (runs). Thus, the decision problem we wish to solve is to establish if the protagonist has a strategy to ensure that the resulting computations satisfy the winning condition.

A way to solve games is to reduce them to the emptiness problem for tree automata. This approach has been successfully exploited to solving discrete games [24, 28]. In this paper we extend the automata-theoretic approach to solving timed games. We propose a general framework that can be used with any class of winning predicates over the untimed runs of a given timed automaton, which

[1] We recall that this formulation of games, which is asymmetric with respect to the two players, is substantially equivalent to the symmetric one, and thus our results can be stated also for the general case.

admit direct translation to a class of tree automata with decidable emptiness problem and closure under intersection. Given a timed automaton A and a winning predicate W we construct the tree automata A^T and A_W such that A^T accepts all the trees corresponding to a strategy of the protagonist in the game (A, W), and A_W accepts all the trees whose paths satisfy the predicate W and having arity (branching degree) upper bounded by the maximum arity among all the trees accepted by A^T. Thus, there exists a winning strategy of the protagonist in the game (A, W) if and only if the intersection between the languages accepted by A^T and A_W is not empty. To construct A^T we exploit the region automaton due to Alur and Dill [3].

We analyze in more detail the case of winning predicates expressed by temporal logic formulas. Temporal logic is a widely accepted formalism to specify and verify reactive systems introduced by Pnueli in 1977 [21]. Here we consider formulas of the logics CTL [12] and LTL [21]. Using our approach, we solve timed Büchi games in exponential time, timed Rabin games in exponential time, CTL games in exponential time and LTL games in doubly exponential time. Since timed reachability games are known to be EXPTIME-hard even if the antagonist is allowed to move only when the protagonist does [18], and LTL games are 2EXPTIME-hard [22], our results are complete. Combining our construction with the results on LTL generators from [6], we can prove an upper bound smaller than 2EXPTIME for meaningful subclasses of LTL timed games. Rectangular hybrid games with winning conditions expressed by LTL formulas were solved in [14], where the authors also prove EXPTIME-hardness of timed reachability games. The results from [14] subsumes our results on LTL timed games, but the approach we follow here is different, and mainly, we are giving a systematic way of solving timed games for different classes of winning predicates. Different formulations of games with winning conditions expressed by temporal logic formulas have been also considered in [4, 15, 16].

The rest of the paper is organized as follows. In Section 2, we introduce the definition and the notation relatively to timed games. We also briefly discuss the automata-theoretic approach to solve discrete games. In Section 3, we discuss the construction of a tree automaton accepting all the strategies of the protagonist in a timed game and give the complexity results on timed Büchi and Rabin games. The general algorithm to solve timed games and its application to timed games with winning conditions expressed by CTL and LTL formulas are discussed in Section 4. In Section 5 we give a few conclusions.

2 Timed games

In this section we introduce the concept of timed game. We start defining discrete games and discuss an automata-theoretic approach to decide them. Then, we introduce timed games.

We model a game as a nondeterministic automaton along with a winning condition. The alphabet symbols (actions) of the automaton represent the choices of the *protagonist* and the nondeterminism is used to model the possible choices of the *antagonist*. To model the case that the protagonist stays *idle* and the antagonist is moving, we use a special symbol denoted by ε. Therefore, all transitions are to be considered joint moves of the two players, except transitions labeled with ε. In the following, we will denote by Σ^ε the set of symbols including the idle action ε, and, if not differently specified, by σ a member of Σ.

An automaton A is a tuple $(Q, \Sigma^\varepsilon, q_0, \Delta)$ where Q is a finite set of locations (or *vertices*), $q_0 \in Q$ is the initial location, and $\Delta \subseteq Q \times \Sigma^\varepsilon \times Q$ is the transition relation. A *run* of A is a sequence of locations $q_0 q_1 \dots q_k$ such that $(q_i, \sigma, q_{i+1}) \in \Delta$ for $i = 0, \dots, k-1$. We will also be interested in infinite runs, that is runs on ω-words over Σ^ε. An ω-*word* over a given alphabet Γ is a mapping from $\mathbb{N}$ into Γ, that is, an infinite sequence of symbols over Γ. Let $w = w_0 w_1 w_2 \dots$ be an ω-word, with w^i we denote the subsequence of w starting at position i, that is, the mapping defined by $w^i_n = w_{i+n}$. A game is a pair (A, W) where A is an automaton (*game graph*) and W is the winning condition, that is a predicate over ω-words of vertices. According to the kind of winning condition, we obtain different kinds of games. A *play* of a given game corresponds to a run of the automaton. A *strategy* is a function that with any run associates an action among those that are enabled. A strategy f has associated a tree t_f (*strategy tree*) in the following way:

- the root of t_f is $\langle q_0 \rangle$, and
- if $\langle q_0 \dots q_k \rangle$ is a node of t and $f(q_0 \dots q_k) = \sigma \in \Sigma^\varepsilon$, then $\langle q_0 \dots q_k q_{k+1} \rangle$ is a child of $\langle q_0 \dots q_k \rangle$ for all q_{k+1} such that $(q_k, \sigma, q_{k+1}) \in \Delta$.

A node $\langle q_0 \dots q_k \rangle$ of t_f is labeled by q_k. Thus, t_f has nodes corresponding to plays of the game constructed according to a strategy f. We observe that paths of t_f are in general not finite (i.e., t_f is an ω-tree), and thus are ω-words over Q. Given a game (A, W), a strategy f is *winning* if W holds on all the ω-words corresponding to paths of t_f. We consider the decision problem: "Is there a strategy satisfying the winning condition W?"

A way to solve games is to reduce them to the emptiness problem for a suitable class of ω-tree automata [24, 28]. An ω-tree automaton is defined by a tuple (Q, Σ, q_0, Δ) where Q is a finite set of locations, $q_0 \in Q$ is the initial location, and Σ is an alphabet as in the definition of an automaton on ω-words. The only difference concerns the transition relation Δ, which here is a subset of $\cup_{i=1}^{K} (Q \times \Sigma \times Q^i)$, where K is a positive integer (called the *arity* of the automaton). A run of a tree automaton can thus be seen as a rewriting of a tree by locations, and generalizes the definition of run we have given for automata on ω-words in the obvious way. With a tree automaton we associate an acceptance condition which is usually a predicate over ω-words of locations (we will return on acceptance conditions for automata on ω-objects in Section 3.1). A tree is accepted by an automaton A if and only if there exists a run r of A such that all the paths of r satisfy the acceptance condition. See [26] for a survey on automata on ω-words and ω-trees.

Given a game (A, W) we construct a corresponding tree automaton A' accepting t_f for all the winning strategies f of (A, W) in the following way. The set of locations, the initial location, and the alphabet of A' are respectively the set of locations, the initial location, and the alphabet of A. The acceptance condition of A' is W, and the transition relation of A' is the set of the tuples $(q, \sigma, q_1, \ldots, q_k)$ such that $(q, \sigma, q_1), \ldots, (q, \sigma, q_k)$ are all the transitions on σ of A. It is easy to verify that A' accepts a tree t if and only if there exists a winning strategy f of (A, W) such that $t = t_f$.

We end this section by introducing timed games. We define timed games analogously to games via a nondeterministic *timed automaton* [3]. A timed automaton is a model of a real-time system. We assume that there is a central (real-valued) clock scanning time, and the model can use a finite set of *clock variables* (also simply named *clocks*) along with timing constraints to check the satisfaction of timing requirements. Each clock can be seen as a chronograph synchronized with the central clock. It can be read or set to zero (reset); after a reset, it restarts automatically. In each automaton, timing constraints are expressed by clock constraints. Let C be a set of clocks, the set of clock constraints $\Xi(C)$ contains:

- $x \leq y + c$, $x \geq y + c$, $x \leq c$ and $x \geq c$, where $x, y \in C$ and c is a natural number; we call such constraints *atomic* clock constraints;
- $\neg\delta$ and $\delta_1 \wedge \delta_2$ where $\delta, \delta_1, \delta_2 \in \Xi(C)$.

Furthermore, let $\mathbb{R}_+$ be the set of nonnegative real numbers, a *clock interpretation* is a mapping $\nu : C \longrightarrow \mathbb{R}_+$. If ν is a clock interpretation, λ is a set of clocks and d is a real number, we denote with $[\lambda \leftarrow 0](\nu + d)$ the clock interpretation that for each clock $x \in \lambda$ gives 0 and for each clock $x \notin \lambda$ gives the value $\nu(x) + d$.

A *timed automaton* A is a tuple $(Q, \Sigma^\varepsilon, q_0, C, \Delta, inv)$ where:

- Q is a finite set of locations;
- $q_0 \in Q$ is the initial location;
- C is a finite set of n clock variables;
- Δ is a finite subset of $Q \times \Sigma \times \Xi(C) \times 2^C \times Q$ (edges);
- $inv : Q \longrightarrow \Xi(C)$ maps each location q to its invariant $inv(q)$.

A *state* of a timed automaton A is a pair (q, ν) where $q \in Q$ and $\nu \in \mathbb{R}^n_+$. The *initial state* is the pair (q_0, ν_0) where $\nu_0(x) = 0$ for all $x \in C$. The semantics of a timed automaton is given by a transition system over the set of states. The transitions of this system are divided into *discrete steps* and *time steps*. A discrete step is $(q, \nu) \xrightarrow{\sigma} (q', \nu')$ where $(q, \sigma, \delta, \lambda, q') \in \Delta$, ν satisfies δ, $\nu' = [\lambda \leftarrow 0]\nu$, and ν' satisfies $inv(q')$. A time step is $(q, \nu) \xrightarrow{d} (q, \nu')$ where $d \in \mathbb{R}_+$, $\nu' = \nu + d$ and $\nu + d'$ satisfies $inv(q)$ for all $0 \leq d' \leq d$. A *step* is $(q, \nu) \xrightarrow{d,\sigma} (q', \nu')$ where $(q, \nu) \xrightarrow{d} (q, \nu'')$ and $(q, \nu'') \xrightarrow{\sigma} (q', \nu')$, for some $\nu'' \in \mathbb{R}^n$. A *timed word* $(\bar{\sigma}, \bar{\tau})$ over the alphabet Σ is such that $\bar{\sigma} \in \Sigma^*$, $\bar{\tau} \in \mathbb{R}^*_+$, and $|\bar{\sigma}| = |\bar{\tau}|$. In a timed word, each symbol σ_i at input is associated with a positive real number τ_i, which expresses (except for the first symbol σ_1) the time which has elapsed since the symbol σ_{i-1} was at input. Time τ_1 represents instead the time at which the symbol σ_1 appears at input assuming that the computation starts at time 0. A run r of a timed automaton A on a timed word $(\bar{\sigma}, \bar{\tau})$, where $\bar{\sigma} = \sigma_1 \ldots \sigma_k$

and $\bar{\tau} = \tau_1 \ldots \tau_k$, is a finite sequence $(q_0, \nu_0) \xrightarrow{\tau_1, \sigma_1} (q_1, \nu_1) \xrightarrow{\tau_2, \sigma_2} \ldots \xrightarrow{\tau_k, \sigma_k} (q_k, \nu_k)$. We say that r starts at q_0 and ends at q_k. Also, for all $i \leqslant k$, we denote by r_i the run $(q_0, \nu_0) \xrightarrow{\tau_1, \sigma_1} (q_1, \nu_1) \xrightarrow{\tau_2, \sigma_2} \ldots \xrightarrow{\tau_i, \sigma_i} (q_i, \nu_i)$, that is the prefix of r up to the i-th step. Finally, $\mathrm{Run}(A)$ is the set of all runs of A.

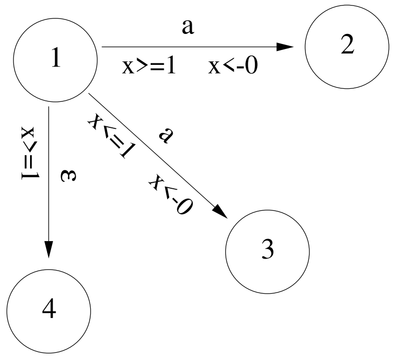

Fig. 1. A fragment of a timed game A.

In a timed game, both players can decide to stay idle for a while. We capture this case by time steps. A *play* of a timed game is constructed in the following way. At each time, a player declares how long it will wait idling until its next choice. At the time one of the players or both move, both players are allowed to redeclare their next move and the time they will issue it. That is, if a player moves before the other, this latter is allowed to change its former decision. A play is represented by a run of the automaton.

As instance, take the fragment of a one clock timed game depicted in figure 1, where the symbol above each edge is the action taken, while under the edge there is a timing constraint and possibly a clock reset. For sake of simplicity, the invariants on the locations are not shown and they are supposed to be *True*. Suppose that the game is in location 1, with clock x equal to zero and the current strategy of the protagonist is to take an a-move, after a delay of 1. Since at time $x = 1$ there are two a-moves enabled, the antagonist can choose which one is to be taken, and the game will proceed either to location 2 or to location 3. Notice that, since there is no ε-move enabled before time 1, this strategy does not allow the antagonist to move on his own.

Formally, a *timed game* is a tuple (A, W) where A is a timed automaton and W is a winning condition. A *strategy* is a function $\mathcal{F} : \mathit{Plays}(\mathcal{F}) \longrightarrow \mathbb{R}_+ \times \Sigma$, where $\mathit{Plays}(\mathcal{F}) \subseteq \mathrm{Run}(A)$, $(q_0, \nu_0) = (q_0, \bar{0}) \in \mathit{Plays}(\mathcal{F})$ and for all $r = (q_0, \nu_0) \xrightarrow{\tau_1, \sigma_1} (q_1, \nu_1) \xrightarrow{\tau_2, \sigma_2} \ldots \xrightarrow{\tau_k, \sigma_k} (q_k, \nu_k)$ belonging to $\mathit{Plays}(\mathcal{F})$, it holds that for $i = 0, \ldots, k-1$, either $\mathcal{F}(r_i) = (\tau_{i+1}, \sigma_{i+1})$ or $\mathcal{F}(r_i) = (d, \sigma)$, $\tau_{i+1} < d$ and $\sigma_{i+1} = \varepsilon$. In other words, a strategy gives the moves of the protagonist on each play which is "consistent" with the strategy itself and the case $\sigma_{i+1} = \varepsilon$ corresponds to a move of the antagonist taken before the next declared move of the protagonist. The set of ω-runs r such that any prefix of r is a play consistent with $\mathcal{F}$, is called the set of maximal plays of $\mathcal{F}$. Each strategy $\mathcal{F}$ has a *dense tree*[2] $T_{\mathcal{F}} = (S, \mu, b)$ associated with it, where:

- $S = Q \times \mathbb{R}^{|C|}$ the state space of A;
- $\mu(q, \nu) = q$;
- $b(q, \nu)$ is the set of suffixes starting from (q, ν) of maximal plays of $\mathcal{F}$.

A path in $T_{\mathcal{F}}$ is thus a maximal play of $\mathcal{F}$. Given a timed ω-word $w = (\bar{\sigma}, \bar{\tau})$, we define $Untime(w) = \bar{\sigma}$, that is the ω-word obtained by discarding the time occurrence of each symbol. A strategy $\mathcal{F}$ is *winning* if $Untime(r)$ satisfies the winning condition W for any maximal play r of $\mathcal{F}$.

3 From timed games to tree automata

In this section, we discuss how to extend to timed games the automata-theoretic approach to solve discrete games. This approach relies on the construction of a tree automaton accepting a non-empty language if and only if there exists a winning strategy of the protagonist in the given game. Our approach is based on the region automaton construction introduced by Alur and Dill in [3].

Consider a timed automaton $A = (Q, \Sigma^{\varepsilon}, q_0, C, \Delta, inv)$. By definition, its set of states is infinite. However, it can be partitioned in a finite number of equivalence classes, called *regions*, which are defined by a location and a *clock region*. Denoted by c_x the largest constant in clock constraints involving the clock variable x, a clock region is described by:

- a constraint of the type $c - 1 < x < c$, $x > c_x$, or $x = c$ for each clock variable x and a natural number $c \leq c_x$;
- the ordering of the fractional parts of the clock variables x such that $x < c_x$.

Thus, a clock region denotes a set of clock valuations. Given a clock valuation ν, $[\nu]$ denotes the clock region containing ν. A state (q, ν) belongs to a region $\langle q', \alpha \rangle$ if $q = q'$ and $\nu \in \alpha$. A clock region α is said to be *open* if for any clock variable x and $c \leq c_x$, $x = c$ does not hold in α. Otherwise α is said to be a *boundary* clock region. These definitions apply to regions in an obvious way. The key property of this equivalence relation is that all the valuations belonging to a region satisfy the same set of clock constraints from the given timed automaton. Consistently we say that a clock region α satisfies a constraint δ if ν satisfies δ for any $\nu \in \alpha$. A clock region α' is said to be a *time-successor* of a clock region α if and only if for all $\nu \in \alpha$ there is a $d \in \mathbb{R}_+$ such that $\nu + d \in \alpha'$. A *next time-successor* α' of α is a time-successor of α such that $\alpha' \neq \alpha$ and for all time-successors α'' of α such that $\alpha'' \notin \{\alpha, \alpha'\}$, α'' is also a time-successor of α'.

Denoted by ξ a symbol not in Σ^{ε}, the *region automaton* of A is a transition system defined by:

[2] For a general definition of dense tree see [2].

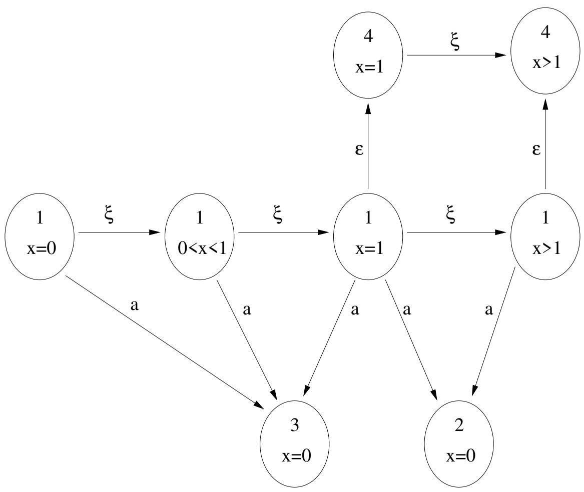

Fig. 2. The region automaton $R(A)$.

- the set of states $R(Q) = \{\langle q, \alpha \rangle \mid q \in Q$ and α is a clock region satisfying $inv(\mathrm{q})\}$;
- the transition rules $R(\Delta)$ such that:
  - $(\langle q, \alpha \rangle, \sigma, \langle q', \alpha' \rangle) \in R(\Delta)$ if and only if $(q, \sigma, \lambda, \delta, q') \in \Delta$, α satisfies δ, and $\alpha' = [\lambda \leftarrow 0]\alpha$ (σ-transition);
  - $(\langle q, \alpha \rangle, \xi, \langle q, \alpha' \rangle) \in R(\Delta)$ if and only if α' is the next time-successor of α (ξ-transition).

We denote by $R(A)$ the region automaton corresponding to A.

We observe that the action ξ captures the case that both players stay idle and let the time pass, while the other symbols in Σ^ε have the same meaning as in A. Given an A step $r = (q, \nu) \xrightarrow{d,\sigma} (q', \nu')$, define $[r]$ as $\langle q, [\nu] \rangle \xrightarrow{\xi} \langle q_0, \alpha_1 \rangle \ldots \langle q_0, \alpha_{k-1} \rangle \xrightarrow{\xi} \langle q_0, \alpha_k \rangle \xrightarrow{\sigma} \langle q', [\nu'] \rangle$ where $\alpha_0 = [\nu]$, α_{i+1} is the next successor of α_i, and $\nu + d \in \alpha_k$. From the definition of $R(A)$, we have that $[r]$ is a run of $R(A)$ from $\langle q, [\nu] \rangle$ to $\langle q', [\nu'] \rangle$. We can extend to any arbitrary run of A the previous definition. Clearly, it holds that if r is a run of A from (q, ν) to (q', ν') then $[r]$ is a run of A from $\langle q, [\nu] \rangle$ to $\langle q', [\nu'] \rangle$. Given a strategy $\mathcal{F}$ in the game graph A we can analogously define a strategy $[\mathcal{F}]$ in $R(A)$.

The above observations would suggest to use $R(A)$ as a game graph in a discrete game corresponding to a game on A, however it is not obvious how to translate the winning conditions. To see this, suppose that the winning condition of the considered game defines a language which is not closed under stuttering, that is a language where repetitions of an assignment of atomic propositions matter. Since the ξ-transitions do not change the atomic propositions (the location remains the same), the winning condition may hold false on a run of $R(A)$ while it holds true on the corresponding run of A.

We solve this problem by constructing a tree automaton that hides the ξ-transitions and keeps the information relative to the ε-moves that can be taken by the antagonist while the protagonist chooses to stay idle. Namely, consider a state $\langle q, \alpha_1 \rangle$ of $R(A)$. Suppose that there exist a sequence of ξ-transitions from $\langle q, \alpha_i \rangle$ to $\langle q, \alpha_{i+1} \rangle$, for $i = 1, \ldots, k-1$ and $k \geq 1$, such that there is at least a σ-transition from $\langle q, \alpha_k \rangle$ for $\sigma \in \Sigma$. In the new transition system, we define a σ-transition from $\langle q, \alpha_1 \rangle$ to all the states $\langle q', \alpha' \rangle$ such that there exists a σ-transition from $\langle q, \alpha_k \rangle$ to $\langle q', \alpha' \rangle$.
Also, we add an ε-transition from $\langle q, \alpha_1 \rangle$ to all the states $\langle q', \alpha' \rangle$ such that:

- there exists an ε-transition from $\langle q, \alpha_i \rangle$ to $\langle q', \alpha' \rangle$ for some $i = 1, \ldots, k-1$, or
- there exists an ε-transition from $\langle q, \alpha_k \rangle$ to $\langle q', \alpha' \rangle$, if α_k is an open region and $k \neq 1$.

Given a state $\langle q, \alpha_1 \rangle$ of $R(A)$, if for all the time-successors α of α_1 there exist no σ-transition from $\langle q, \alpha \rangle$, then we define an ε-transition from $\langle q, \alpha_1 \rangle$ to all the states $\langle q', \alpha' \rangle$ such that there is an ε-transition from $\langle q, \alpha \rangle$ to $\langle q', \alpha' \rangle$ and α is a time-successor of α_1.

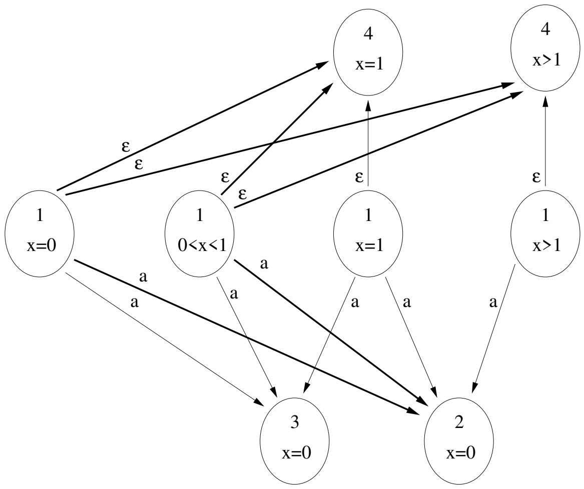

Fig. 3. Transitions of A^T. The transitions that do not belong to the region automaton are in boldface.

Let us take into consideration the A^T automaton depicted in figure 3. By comparing this automaton with the corresponding region automaton in figure 2, it can be seen that the transitions corresponding to the mere passage of time (ξ-transitions) have been removed, while others have been added. As instance, starting from region $\langle 1, x = 0 \rangle$, there is a new a-move, leading to $\langle 2, x = 0 \rangle$, which corresponds to the path $\langle 1, x = 0 \rangle$, $\langle 1, 0 < x < 1 \rangle$, $\langle 1, x = 1 \rangle$,$\langle 2, x = 0 \rangle$ in the region automaton.

Since the winning conditions are predicates over the runs of the starting timed game, to complete the construction we need to define the transitions of the above tree automaton on locations of A. To do this, we simply replace each transition $(\langle q, \alpha\rangle, \sigma, \langle q_1, \alpha_1\rangle, \dots, \langle q_k, \alpha_k\rangle)$ by $(\langle q, \alpha\rangle, q, \langle q_1, \alpha_1\rangle, \dots, \langle q_k, \alpha_k\rangle)$.

We denote by A^T the tree automaton which is obtained by the above construction. (Notice that the action corresponding to a given transition can be easily determined by translating back a run of A^T into a run of $R(A)$.) From [3], we have that the number of regions of A is $O(|C|!\, 2^{|C|}\, \Pi_{x \in C}(2c_x + 2))$. Thus, the following result holds.

Lemma 1. *The size of A^T is exponential in the size of the clock constraints and linear in the number of locations.*

By the construction of A^T, we have that the following theorem holds.

Theorem 1. *Given a timed game (A, W), each strategy $\mathcal{F}$ of the protagonist can be mapped into a tree t accepted by A^T, and vice-versa. Moreover, if $\mathcal{F}$ is winning then W holds on all the paths of t.*

3.1 Timed Büchi and Rabin games

Here we briefly discuss some results that can be obtained directly from the construction presented in the first part of this section. In the next section we extend this approach to solving timed games to temporal logic games.

We recall that a *Büchi condition* asserts that at least a final location repeats infinitely often in an infinite run. That is, denoted by $Inf(r)$ the locations that repeat infinitely often in an infinite run r and by $W \subseteq Q$ a set of acceptance locations, the Büchi condition is expressed as $Inf(r) \cap W \neq \emptyset$. A *Rabin condition* is given by a set of pairs $\{(U_i, L_i) \mid i = 1, \dots, k\}$ such that $U_i, L_i \subseteq Q$, and for some $i = 1, \dots, k$, a vertex in U_i must repeat infinitely often and none of the vertices in L_i repeats infinitely often. Formally, the Rabin condition is: "there exists a pair (U_i, L_i) such that $Inf(r) \cap U_i \neq \emptyset$ and $Inf(r) \cap L_i = \emptyset$". Combining these two conditions with the definition of timed games, we define respectively *timed Büchi games* and *timed Rabin games*.

Given a timed Büchi game (A, W), define W' as the set $\{\langle q, \alpha\rangle \mid q \in W$ and α is a clock region of $A\}$. We have that A^T coupled with W' is a Büchi tree automaton which, by Theorem 1, accepts a tree t if and only if there exists a winning strategy of the protagonist in (A, W). We recall that the non-emptiness problem for Büchi tree automata is decidable in polynomial time [10]. Moreover, timed reachability games are EXPTIME-hard [14]. Thus by Lemma 1, the following result holds.

Theorem 2. *Timed Büchi games are* EXPTIME-*complete.*

We prove the same result on timed Rabin games. Given a timed Rabin game (A, W), for $(U_i, L_i) \in W$ define $U'_i = \{\langle q, \alpha\rangle \mid q \in U_i$ and α is a clock region of $A\}$ and $L'_i = \{\langle q, \alpha\rangle \mid q \in L_i$ and α is a clock region of $A\}$. Denote by W' the set

$\{(U_i', L_i' \mid (U_i, L_i) \in W\}$, we have that A^T coupled with W' is a Rabin tree automaton which, by Theorem 1, accepts a tree t if and only if there exists a winning strategy of the protagonist in (A, W). By Lemma 1 and the fact that the emptiness problem for Rabin tree automata can be solved in deterministic time $O((nm)^{cm})$ [22], where n is the number of locations, m is the number of pairs in the accepting set and c is a constant, we have the following result.

Theorem 3. *Timed Rabin games are* EXPTIME-*complete.*

4 A solution to timed games

In this section we give a general algorithm to solve timed games with winning conditions that can be translated into a tree language accepted by a Rabin tree automaton, and using known results in the field of temporal logic, we apply it to solving CTL and LTL timed games.

Consider a timed game (A, W). By Theorem 1, it is possible to construct a tree automaton A^T such that for any strategy of the protagonist in (A, W) there exists a corresponding tree accepted by A^T and vice-versa. Let $K > 0$ be the maximum arity of a tree accepted by A^T. Suppose that it is possible to construct a tree automaton A_W accepting all the trees t with arity at most K such that W holds on all paths of t. Thus, by Theorem 1 we have that the language accepted by $A^T \cap A_W$ is non-empty is and only if there exists a winning strategy of the protagonist in the game (A, W). Since the automaton A^T is a Büchi tree automaton, it is sufficient that A_W is an automaton from a class of tree automata that are closed under intersection with respect Büchi automata and have a decidable emptiness problem. For example, A_W could be a Rabin, Büchi, Streett, or Muller tree automaton (see [26] for a survey on these classes of automata).

In the rest of the section we apply the above approach to solve CTL and LTL timed games.

4.1 Computation Tree Logic

Computation Tree logic (CTL) was introduced by Emerson and Clarke [12] as a powerful tool for specifying and verifying concurrent programs. Given a set of atomic propositions AP, a CTL formula is composed of atomic propositions, the boolean connectives *conjunction* ($\wedge$) and *negation* ($\neg$), and the linear-temporal operators *Next* ($\bigcirc$) and *Until* ($\mathcal{U}$) coupled with path quantifiers *for all paths* ($\forall$) and *for some path* ($\exists$). Formulas are built up in the usual way from the above operators and connectives, according to the following grammar

$$\varphi := p \mid \neg\,\varphi \mid \varphi \wedge \varphi \mid \forall\bigcirc\varphi \mid \exists\bigcirc\varphi \mid \forall(\varphi\,\mathcal{U}\,\varphi) \mid \exists(\varphi\,\mathcal{U}\,\varphi)$$

where p is an atomic proposition. The semantics of CTL is defined with respect to a *Kripke structure* $M = (S, \mu, R)$, where S is a countable set of states, $\mu : S \to 2^{AP}$ maps each state to a set of atomic proposition true in that state, and R is a binary relation over S. A *path* in M is an infinite sequence of states $s_0, s_1, \ldots$ such that $(s_i, s_{i+1}) \in R$. Given a Kripke structure M, a CTL formula φ and a state $s \in S$, the satisfaction relation $(M, s) \models \varphi$, meaning that φ is true in M at s, is defined inductively as follows:

- if φ is an atomic proposition, then $(M, s) \models \varphi$ iff the truth assignment $\mu(s)$ satisfies φ;
- $(M, s) \models \neg\varphi$ iff $(M, s) \models \varphi$ does not hold;
- $(M, s) \models \varphi_1 \wedge \varphi_2$ iff $(M, s) \models \varphi_1$ and $(M, s) \models \varphi_2$;
- $(M, s) \models \forall\bigcirc\varphi$ iff $(M, s') \models \varphi$ for all $s' \in S$ such that $(s, s') \in R$;
- $(M, s) \models \exists\bigcirc\varphi$ iff $(M, s') \models \varphi$ for some $s' \in S$ such that $(s, s') \in R$;
- $(M, s) \models \forall(\varphi_1\ \mathcal{U}\ \varphi_2)$ iff for every path $s_0, s_1, \ldots s_n$ with $s = s_0$, there exists $i \geq 0$ such that $(M, s_i) \models \varphi_2$, and $(M, s_j) \models \varphi_1$, for all j such that $0 \leq j < i$;
- $(M, s) \models \exists(\varphi_1\ \mathcal{U}\ \varphi_2)$ iff there exist a path $s_0, s_1, \ldots s_n$ with $s = s_0$, and $i \geq 0$ such that $(M, s_i) \models \varphi_2$, and $(M, s_j) \models \varphi_1$, for all j such that $0 \leq j < i$.

Given a CTL formula φ and a positive integer K, it is possible to construct a Büchi tree automaton, of size exponential in the size of φ, which accepts all the ω-trees t with arity at most K such that: t is the unwinding of a Kripke structure M from a state s and $(M, s) \models \varphi$ [29]. Thus by Theorem 1, Lemma 1, and the fact that the emptiness problem for Büchi tree automata is decidable in polynomial time [10], the following theorem holds.

Theorem 4. *Given a timed game (A, φ) where φ is a* CTL *formula, the problem of deciding the existence of a winning strategy of the protagonist in (A, φ) is* EXPTIME*-complete.*

4.2 Linear Temporal Logic

Linear Temporal Logic (LTL) was introduced by Pnueli to specifying and verifying reactive systems [21]. Given a set of atomic propositions AP, an LTL formula is composed of atomic propositions, the boolean connectives *conjunction* ($\wedge$) and *negation* ($\neg$), the temporal operators *Next* ($\bigcirc$) and *Until* ($\mathcal{U}$). Formulas are built up in the usual way from the above operators and connectives, according to the following grammar

$$\varphi := p \,|\, \neg\varphi \,|\, \varphi \wedge \varphi \,|\, \bigcirc\varphi \,|\, \varphi\, \mathcal{U}\, \varphi$$

where p is an atomic proposition. Thus, the syntax of LTL formulas can be obtained by the syntax of CTL formulas by deleting the universal and existential quantifiers. The semantics of LTL formulas is given with respect to on an ω-word $w = w_0 w_1 \ldots w_n \ldots$ over the alphabet $\Sigma = 2^{AP}$. The satisfaction relation $w \models \varphi$ is defined in the standard way:

- if φ is an atomic proposition, then $w \models \varphi$ if and only if the assignment of atomic propositions specified by w_0 assigns φ true;
- $w \models \neg\varphi$ if and only if $w \models \varphi$ does not hold;

– $w \models \varphi_1 \wedge \varphi_2$ if and only if $w \models \varphi_1$ and $w \models \varphi_2$;
– $w \models \bigcirc\varphi$ if and only if $w^1 \models \varphi$;
– $w \models \varphi_1 \,\mathcal{U}\, \varphi_2$ if and only if there exists $i \geq 0$ such that $M, w^i \models \varphi_1$ and $w^j \models \varphi_2$ for all j such that $0 \leq j < i$.

For every LTL formula φ, it is possible to construct a nondeterministic Büchi automaton on ω-words accepting all models of φ. We will refer to such an automaton as a *generator* of models of φ. Since we need to construct a tree automaton, it is necessary to have a deterministic generator. In fact, given a positive integer K and a deterministic automaton on words A, a tree automaton accepting all the trees t with arity bounded above by K and such that any path of t is a word accepted by A, can be easily obtained by adding for any transition rule (q, σ, q') of A all the transition rules $(q, \sigma, q_1, \ldots, q_k)$ such that: $q_i = q'$ for $k = 1, \ldots, k$ and $k \leq K$. Clearly, such a construction does not work for nondeterministic automata.

A deterministic Rabin generator A_φ for an LTL formula φ of doubly exponential size can be obtained in the following way: from the formula φ, by the tableau construction, it is possible to construct a nondeterministic Büchi generator of size $2^{O(|\varphi|)}$ [19, 30]; we recall that a Büchi automaton of size n can be determinized and the resulting deterministic Rabin automaton has $2^{O(n \log n)}$ states and n pairs [25]; thus, we determinize the Büchi generator for φ so obtaining a deterministic Rabin generator of doubly exponential size with exponentially many pairs. Notice that, in general, for a given formula φ, a deterministic Büchi generator may not exist but, when it exists, it has a doubly exponential size in the length of the formula (see [17]), and thus, the above construction is asymptotically optimal.

Consider now a game (A, φ) and denote by K the maximum arity of the trees accepted by A^T. Denote by A' the Rabin tree automaton corresponding to A_φ and constructed to accept trees of arity at most K. It is easy to construct a Rabin tree automaton $A^\cap$ accepting the intersection of the languages accepted by A^T and A'. From the above arguments, the size of $A^\cap$ is doubly exponential in the size of φ and, by Lemma 1, it is singly exponential in the size of A. Moreover, the number of pairs in the accepting condition of $A^\cap$ is exponential in the size of φ. We recall that, checking for the emptiness of a language accepted by a Rabin tree automaton with n locations and m pairs can be done in deterministic time $O((nm)^{cm})$ [22]. Thus, by Theorem 1 and the fact that LTL games are 2EXPTIME-hard [22], the following theorem holds.

Theorem 5. *Given a timed game (A, φ) where φ is an* LTL *formula, the problem of deciding the existence of a winning strategy of the protagonist in (A, φ) is 2*EXPTIME*-complete.*

We recall that the result stated in the above theorem is subsumed by the result proved in [14] for LTL rectangular hybrid games.

5 Conclusions

We presented an automata-theoretic approach to solve timed games. Our solution relies on the construction of a tree automaton accepting all the ω-trees corresponding to a strategy of the protagonist in the timed game. This approach can be used with any class of winning conditions admitting a direct translation to a class of tree automata with decidable emptiness problem and closure under intersection. We have analyzed in more detail the cases of winning conditions expressed by temporal logic formulas. We can solve timed Büchi games, timed Rabin games and CTL games in exponential time. Since timed reachability games are known to be EXPTIME-hard even if the antagonist is allowed to move only when the protagonist does [18], this results are also complete. We have also applied our approach to solving LTL games. The obtained procedure takes doubly exponential time, and since LTL games are 2EXPTIME-hard [22], our result is tight.

References

1. R. Alur, L. de Alfaro, T. Henzinger, and F. Mang. Automating modular verification. In *CONCUR'99: Concurrency Theory, Tenth Int. Conference*, LNCS 1664, pages 82–97, 1999.
2. R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2 – 34, 1993.
3. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183 – 235, 1994.
4. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. of the 38th IEEE Symposium on Foundations of Computer Science*, pages 100 – 109, 1997.
5. R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proc. of the Tenth Int. Conference on Computer Aided Verification*, LNCS 1427, pages 521 – 525. Springer-Verlag, 1998.
6. R. Alur and S. La Torre. Deterministic generators and games for ltl fragments. In *Proc. of the 16th IEEE Symposium on Logic in Computer Science, LICS'01*, pages 291–300, 2001.
7. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *Proc. of the 16th Intern. Colloquium on Automata, Languages and Programming, ICALP'89*, LNCS 372, pages 1–17, 1989.
8. E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In *Proc. of the 2nd International Workshop on Hybrid Systems: Computation and Control*, LNCS 1569, pages 19 – 30. Springer-Verlag, 1999.
9. E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469 – 474. Elsevier, 1998.
10. J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state stategies. *Trans. Amer. Math. Soc.*, 138:295 – 311, 1969.

11. A. Church. Logic, arithmetic, and automata. In *Proc. of the International Congress of Mathematics*, pages 23–35, 1962.
12. E.A. Emerson and E.M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241 – 266, 1982.
13. E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs. In *Proc. of the 29th IEEE-CS Symposium on Foundations of Computer Science*, pages 328 – 337, 1988.
14. T.A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *Proc. of the 10th International Conference on Concurrency Theory, CONCUR'99*, LNCS 1664, pages 320 – 335, 1999.
15. O. Kupferman and M.Y. Vardi. Module checking. In *Computer Aided Verification, Proc. Eighth Int. Workshop*, LNCS 1102, pages 75 – 86. Springer-Verlag, 1996.
16. O. Kupferman and M.Y. Vardi. Module checking revisited. In *Proc. of the 9th Intern. Conference on Computer Aided Verification, CAV'97*, LNCS 1254, pages 36 –47, June 1997.
17. O. Kupferman and M.Y. Vardi. Freedom, weakness, and determinism: From linear-time to branching-time. In *Proc. of the 13th IEEE Symposium on Logic in Computer Science*, pages 81 – 92, June 1998.
18. S. La Torre and M. Napoli. *Finite Automata on Timed ω-Trees.* To appear in *Theoretical Computer Science.*
19. O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. In *Proc. of the 12th ACM Symposium on Principles of Programming Languages*, pages 97 – 107, 1985.
20. J. Von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944.
21. A. Pnueli. The temporal logic of programs. In *Proc. of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46 – 77, 1977.
22. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of the 16th ACM Symposium on Principles of Programming Languages*, pages 179 – 190, 1989.
23. M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1 – 35, 1969.
24. M.O. Rabin. Automata on infinite objects and Church's problem. *Trans. Amer. Math. Soc.*, 1972.
25. S. Safra. On the complexity of ω-automata. In *Proc. of the 29th IEEE Symposium on Foundations of Computer Science*, pages 319 – 327, 1988.
26. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133 – 191. Elsevier Science Publishers, 1990.
27. W. Thomas. On the synthesis of strategies in infinite games. In Ernst W. Mayr and Claude Puech, editors, *12th Annual Symposium on Theoretical Aspects of Computer Science, STACS'95*, LNCS 900, pages 1 – 13. Springer-Verlag, 1995.
28. M.Y. Vardi. Verification of concurrent programs: the automata-theoretic framework. In *Proc. of the Second IEEE Symposium on Logic in Computer Science*, pages 167 – 176, 1987.
29. M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:182 – 211, 1986.

30. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1 – 37, 1994.