Automata-theoretic Decision of Timed Games ⋆

Marco Faella [a], Salvatore La Torre [b], Aniello Murano [a]

[a] *Università degli Studi di Napoli "Federico II", 80126 Napoli*
{faella, murano}@na.infn.it

[b] *Università degli Studi di Salerno, 84081 Baronissi (SA)*
slatorre@unisa.it

Abstract

The solution of games is a key decision problem in the context of verification of open systems and program synthesis. Given a game graph and a specification, we wish to determine if there exists a strategy of the protagonist that allows to select only behaviors fulfilling the specification. In this paper, we consider timed games, where the game graph is a timed automaton and the specification is given by formulas of the temporal logics LTL and CTL. We present an automata-theoretic approach to solve the addressed games, extending to the timed framework a successful approach to solve discrete games. The main idea of this approach is to translate the timed automaton A, modeling the game graph, into a tree automaton A^T accepting all trees that correspond to a strategy of the protagonist. Then, given an automaton corresponding to the specification, we intersect it with the tree automaton A^T and check for the nonemptiness of the resulting automaton. Our approach yields a decision algorithm running in exponential time for CTL and in double exponential time for LTL. The obtained algorithms are optimal in the sense that their computational complexity matches the known lower bounds.

1 Introduction

The theory of games was originally introduced as a theoretical model for economic studies (see for example [37]). In the years, this theory has received an increasing interest by many researchers in both computer science and control

⋆ A preliminary version of this paper appears in the Proceedings of the 3rd International Workshop on Verification, Model Checking, and Abstract Interpretation, VMCAI'02. Electronic Notes in LNCS 2294, 2002.

theory. Games have been studied in the context of discrete, timed, and hybrid systems (see [12,26,39,44], for some research). They provide a suitable framework for program synthesis and analysis of *open systems*, that is, systems whose behavior depend on the current state as well as the choices of the environment in which they are embedded.

The notion of open system naturally arises in the compositional modelling and design of *reactive systems*, that is, systems that maintain an ongoing interaction with their environment [10,39]. 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 if φ holds for the computations of M (*model checking*). Model checking is a very successful technology, which has been implemented in many tools [18].

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 the 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 computations satisfy the given specification.

To refer to delays, time needs to be explicitly included in the model of a reactive system. In this paper, we focus on timed games: a game graph given by a nondeterministic timed automaton [3] along with a winning condition. A *timed automaton* is a finite automaton augmented with a finite set of real-valued *clocks*. Its transitions 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 (timed game graph), a move of a player consists of a discrete action along with the time at which it will be issued. 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 when the protagonist stays *idle* and the antagonist moves, we use a special

[1] Notice that this formulation of games, which is asymmetric with respect to the two players, is equivalent to the symmetric one.

symbol ξ. 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 usual way. At the beginning, both players declare their first moves, that is, how long they will wait idling and their next discrete actions. At the time one of the players or both move, both of them declare their next move. This means that, in case only a player moves, the other player is allowed to confirm his previous declared move or to declare a new one. Technically, a play is a run of the automaton modelling the game. A game is giving by a game graph along with a winning condition that establishes which computations are winning for the protagonist. Thus, the decision problem we wish to solve on a given timed game graph is to determine if the protagonist has a strategy to ensure that all the resulting computations satisfy the given winning condition. In this paper, we consider winning conditions expressed as LTL [38] and CTL [17] formulas.

Linear winning conditions are suitable when the design target is to find a strategy that ensures that some property holds regardless of the environment behavior. In some domains, it is useful to consider more general conditions. As instance, consider the problem of designing an ATM controller. There, the controller is the protagonist of the game, while the adversary is the ATM user. Consider the property "No matter what the user does, there is a way for her to have the card back". This property cannot be expressed in a linear time language, while it is expressible in a branching-time temporal logic such as CTL. Indeed, if we call p the proposition "the card is returned", then the previous ATM requirement can be stated as "$\forall\Box\exists\Diamond p$". [2]

A way to solve the decision problem related to game graphs is to reduce it to the emptiness problem for tree automata. This approach has been successfully exploited to decide discrete games (see, for example [41]). In this paper, we extend the automata-theoretic approach to timed games. We propose a general framework that can be used with any class of winning conditions, that can be translated to tree automata with decidable emptiness problem and closure under intersection. Given a timed game (A, W), where A is a timed automaton and W is a winning condition, we construct a tree automaton A^T that accepts all the trees corresponding to a strategy of the protagonist. We construct this automaton exploiting the clock region relation [3]. Then, we construct another tree automaton A_W accepting all the trees satisfying the property expressed by W. Since strategies of the protagonist correspond to trees accepted by A^T, to construct A_W we only need to capture models with branching degree bounded above by the one of A^T (i.e. the maximum branching degree of A^T transitions). 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.

[2] This property is violated by some ATM machines, which trap the card if the user exceeds the maximum number of PIN trials.

For winning conditions given by LTL formulas, our approach yields decision algorithm running in doubly exponential time. Since LTL games are 2EXPTIME-hard [39] even for discrete game graphs our result is complete. For winning conditions given by CTL formulas, our approach yields decision algorithm running in exponential time. Since reachability can be exponential in CTL, we have that our result is complete also for CTL.

In literature, different formulations of games with winning conditions expressed by temporal logics have been considered. In [4], alternating temporal logics are introduced. While linear temporal logics assume universal quantifications on all paths and branching temporal logics allow explicit existential and universal quantifications on all paths, alternating temporal logics offer "selective" quantification over those paths that are possible outcomes of games, such as the game in which the system and the environment alternate moves. Thus, alternating temporal logics are natural specification languages for open systems. Depending on whether an arbitrary nesting of selective path quantifiers and temporal operators is allowed, two alternating temporal logics are defined: ATL and ATL*. It turns out that the model checking problem with respect to ATL and ATL* formulas is solvable, respectively, in quadratic time and exponential time [18]. In [28] the model checking of open systems (*module checking*) is studied. This problem turns out to be EXPTIME-compete for specification given by CTL formulas and 2EXPTIME-complete for specification given by CTL* formulas. Recently, the module checking framework has been fruitfully extended to pushdown systems[13,14], showing that this problem becomes 2EXPTIME-compete for specification given by CTL formulas and 3EXPTIME-complete for specification given by CTL* formulas. Rectangular hybrid games with winning conditions expressed by LTL formulas were solved in [26]. The results from [26] subsume 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. Recently, timed games with specification in dense real–time, have also been introduced. In [35] the module checking in dense real time for specification in TCTL has been introduced. This problem has been proved to be undecidable using a reduction from the problem of deciding whether a nondeterministic two-counter machine has recurring computations (this problem is known to be Σ_1^1-hard [2]). In [25], a real–time game has been introduced as a natural framework to model open systems in dense real time. It has been proved that the problem of finding winning strategies is undecidable for specification in TCTL, while it becomes decidable by relaxing the punctuality of the temporal operators of the logic.

The rest of the paper is organized as follows. In Section 2, we introduce the basic definitions and the notation relatively to games and automata. In Section 3, we introduce our model of timed game. In Section 5, we consider games whose winning condition is given in the linear–time temporal logic LTL. In particular,

we discuss the construction of a tree automaton accepting all strategies of the protagonist in a timed game. Then, in Section 4, we study winning conditions expressed in the branching–time temporal logic CTL. We conclude with few remarks in Section 6.

2 Preliminaries

In this section, we recall the basic definitions concerning words, trees, automata on infinite words and trees, and timed automata [3].

2.1 Words, trees, and automata

Let Σ be an alphabet, we denote by Σ^* the set of all finite words over Σ, including the *empty word* ε. We denote by Σ^i the set of all words of length i over Σ. An *ω-word* over Σ is an infinite sequence of symbols over Σ. We denote by Σ^ω the set of all ω-words over Σ. Let $w = \sigma_0\sigma_1\ldots$ be an ω-word, we denote by $w_{\leq i}$ the *prefix* of w ending at position i, that is, $w_{\leq i} = \sigma_0\ldots\sigma_i$. In the following, we always use Σ to denote a finite alphabet.

A Σ-valued *graph* is a tuple $G = (\Sigma, V, E, v_{ini}, \mu)$, where V is a set of nodes, $E \subseteq V^2$ is the transition relation, $v_{ini} \in V$ is the initial node, and $\mu : V \to \Sigma$ is the labeling function. Whenever $(u, v) \in E$, we say that v is a *successor* of u. If u has a finite number of successors, the *branching degree* $deg(u)$ of u is the number of such successors. Otherwise, we say that the branching degree of u is infinite. The branching degree of a graph is the maximum branching degree of its nodes. For all nodes $u \in V$ we set $succ(G, u)$ to be the set of successors of u in G, that is $succ(G, u) = \{v \in S \mid (u, v) \in E\}$.

For $i \in \{1, 2\}$, let $G_i = (\Sigma, V_i, E_i, z_i, \mu_i)$ be a Σ-valued graph. A relation $R \subseteq V_1 \times V_2$ is a *bisimulation* if and only if, for all $(u_1, u_2) \in R$, the following conditions hold:

(1) $\mu_1(u_1) = \mu_2(u_2)$,
(2) for all $v_1 \in succ(G_1, u_1)$, there exists $v_2 \in succ(G_2, u_2)$ such that $(v_1, v_2) \in R$, and
(3) for all $v_2 \in succ(G_2, u_2)$, there exists $v_1 \in succ(G_1, u_1)$ such that $(v_1, v_2) \in R$.

We say that G_1 and G_2 are *bisimilar* (denoted by $G_1 \sim G_2$), if there exists a bisimulation R such that $(z_1, z_2) \in R$.

A *path* in G is a (possibly finite) sequence of nodes $\pi = v_0v_1\ldots$ such that,

for all $i \geq 0$, $(v_i, v_{i+1}) \in E$. Moreover, π is *rooted* if $v_0 = v_{ini}$. Given a path $\pi = v_0 v_1 \ldots$, we denote by $\mu(\pi)$ the sequence $\mu(v_0)\mu(v_1)\ldots$. A Σ-valued ω-tree (*tree*, for short) is a graph G that satisfies the following:

(1) There are no cicles: for all paths $\pi = v_0 v_1 \ldots$ and for all $i, j \geq 0$, if $i \neq j$ then $v_i \neq v_j$;
(2) Every node has exactly one predecessor: for all $v \in V \setminus \{v_{ini}\}$, there is one (and only one) $u \in V$ such that $(u, v) \in E$;
(3) Every node has (at least) one successor: for all $u \in V$ there is $v \in V$ such that $(u, v) \in E$ (equivalently, E is total).

If G is a tree, v_{ini} is called the *root* of G. A *subgraph* of G is a graph $G' = (\Sigma, V', E', v'_{ini}, \mu')$ such that $V' \subseteq V$, v'_{ini} is the initial node of G', E' and μ' are the restrictions of E and μ to V', respectively. Given a tree T, any subgraph of T which is itself a tree is a *subtree* of T.

A *transition table* A over the alphabet Σ is a tuple $(\Sigma, Q, \Delta, q_{ini})$, where Q is a finite set of *locations*, $\Delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*, and $q_{ini} \in Q$ is the *initial location*. A *run* of A on an ω-word $\sigma_0\sigma_1\ldots$ over Σ is an infinite sequence of locations $q_0 q_1 \ldots$ such that $q_0 = q_{ini}$ and $(q_i, \sigma_i, q_{i+1}) \in \Delta$ for all $i \geq 0$.

A *tree transition table* over Σ is a tuple $A = (\Sigma, Q, \Delta, q_{ini})$, where Q and q_{ini} are defined as in transition tables, and $\Delta \subseteq \cup_{i=1}^{k}(Q \times \Sigma \times Q^i)$, where k is a positive integer called the *branching degree* of A and denoted by $deg(A)$. Given a tree $t = (\Sigma, V, E, v_{ini}, \mu)$ with a finite branching degree, a *run* of A over t is a Q-labeled tree $\rho = (Q, V, E, v_{ini}, \nu)$, such that *(i)* $\nu(v_{ini}) = q_{ini}$, and *(ii)* for all nodes $u \in V$, there exists an ordering of $succ(t, u)$ as $(u_1, \ldots, u_n)$ such that $(\nu(u), \mu(u), \nu(u_1), \ldots, \nu(u_n)) \in \Delta$. A *run* of a tree transition table can thus be seen as a rewriting of a tree by locations.

With a (tree) transition table A we can associate an acceptance condition. In the literature, several acceptance conditions have been introduced and fruitfully investigated. Here we recall those defining *Büchi* and *Rabin* (tree) automata.

Let ρ be an infinite sequence of locations, we denote by $Inf(\rho)$ the set of locations occurring infinitely often in ρ. The Büchi condition is defined with respect to a set of final states $F \subseteq Q$.

- A run ρ of a transition table satisfies the *Büchi* condition F iff $Inf(\rho) \cap F \neq \emptyset$;
- A run ρ of a tree transition table satisfies the *Büchi* condition F iff for each path π of ρ, $Inf(\pi) \cap F \neq \emptyset$.

The Rabin condition is defined with respect to a set $F = \{(B_1, F_1), \ldots, (B_k, F_k)\}$

where $B_i, F_i \subseteq Q$ for each $1 \leq i \leq k$.

- A run ρ of a transition table satisfies the *Rabin* condition F iff there is a pair $(B_i, F_i) \in F$ such that $Inf(\rho) \cap B_i = \emptyset$ and $Inf(\rho) \cap F_i \neq \emptyset$;
- A run ρ of a tree transition table satisfies] the *Rabin* condition F iff for each path π of ρ there is a pair $(B_i, F_i) \in F$ such that $Inf(\pi) \cap B_i = \emptyset$ and $Inf(\pi) \cap F_i \neq \emptyset$.

A word w (resp., a tree t) is *accepted* by an automaton (resp., a tree automaton) A w.r.t. an acceptance condition F if there exists a run ρ of A on w (resp., t) that satisfies F.

In the following, we treat a (tree) transition table A also as a Büchi (tree) automaton whose acceptance condition F is the set of all states from A. The set of words (resp., trees) that are accepted by an automaton (resp., tree automaton) A is denoted by $L(A)$. The *emptiness problem* for an automaton A asks whether $L(A)$ is empty. In the following, we give some results concerning the emptiness problem for Rabin and Büchi tree automata. Notice that our definition of tree transition tables differs from the classical one in that our transition tables accept unordered trees. This does not affect the complexity of the emptiness problem, since the emptiness problem according to our definition of $L(A)$ is equivalent to the classical one.

Proposition 1 ([23,32]) *The emptiness problem for a Rabin tree automaton A is NP-complete. Moreover, it can be solved in deterministic time $O(n^{2m+1} \cdot m!)$, where n and m are the number of locations and the branching degree of A, respectively.*

Proposition 2 ([40,47]) *The emptiness problem for a Büchi tree automata A is decidable in polynomial time. In particular, it can be solved in time $O(n^2)$ where n is the number of locations in A.*

2.2 Timed Graphs

In this section, we describe the model we use to represent timed games: *timed graphs*. A timed graph is a model of a real-time system [3]. A central (real-valued) clock is used to scan time, and a finite set of *clock variables* (also simply named *clocks*) along with timing constraints are used to check the satisfaction of timing requirements. Each clock can be seen as a chronograph synchronized with the central clock, that can be read or set to zero (*reset*). After a reset, a clock restarts automatically. In each graph, timing constraints are formally 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;
- $\neg\delta$ and $\delta_1 \wedge \delta_2$, where $\delta, \delta_1, \delta_2 \in \Xi(C)$.

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

A *timed graph* $\mathcal{G}$ is a tuple $(Q, Acts, q_{ini}, C, \Delta, inv)$ where:

- Q is a finite set of locations;
- $Acts$ is a finite alphabet of actions;
- $q_{ini} \in Q$ is the initial location;
- C is a set of clock variables;
- Δ is a finite set of edges $Q \times Acts \times \Xi(C) \times 2^C \times Q$;
- $inv : Q \longrightarrow \Xi(C)$ maps each location q to its invariant $inv(q)$.

Let $|C| = n$, a *state* of the timed graph $\mathcal{G}$ is a pair $\langle q, \nu \rangle$, where $q \in Q$, $\nu \in \mathbb{R}^n_{\geq 0}$ and ν satisfies $inv(q)$. The *initial state* is $\langle q_{ini}, \nu_{ini} \rangle$, where $\nu_{ini}(x) = 0$ for all $x \in C$. The semantics of a timed graph is given by an infinite transition table with an infinite set of states $Q \times \mathbb{R}^n_{\geq 0}$ and an infinite alphabet $\mathbb{R}_{\geq 0} \times Acts$. In order to define transitions, we first introduce discrete steps and time steps. A *discrete step* is $\langle q, \nu \rangle \xrightarrow{\sigma} \langle q', \nu' \rangle$ 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 $\langle q, \nu \rangle \xrightarrow{d} \langle q, \nu' \rangle$ where $d \in \mathbb{R}_{>0}$, [3] $\nu' = \nu + d$ and $\nu + d'$ satisfies $inv(q)$ for all $0 \leq d' \leq d$. Then, a transition (also called *step*) is $\langle q, \nu \rangle \xrightarrow{d,\sigma} \langle q', \nu' \rangle$ where $\langle q, \nu \rangle \xrightarrow{d} \langle q, \nu'' \rangle$ and $\langle q, \nu'' \rangle \xrightarrow{\sigma} \langle q', \nu' \rangle$, for some $\nu'' \in \mathbb{R}^n_{\geq 0}$. A *run* ρ of a timed graph $\mathcal{G}$ from a state $\langle q_0, \nu_0 \rangle$ is a finite sequence of states $\langle q_0, \nu_0 \rangle \langle q_1, \nu_1 \rangle \ \ldots \ \langle q_n, \nu_n \rangle$, where $\langle q_i, \nu_i \rangle \xrightarrow{\mathrm{d}_i, \sigma_i} \langle q_{i+1}, \nu_{i+1} \rangle$, for all $i = 1, \ldots, k-1$. We set $first(r) = \langle q_0, \nu_0 \rangle$ and $last(r) = \langle q_n, \nu_n \rangle$. An infinite run is defined in the obvious way. Given two runs ρ_1, ρ_2, such that $first(\rho_2) = last(\rho_1)$, we denote by $\rho_1 \cdot \rho_2$ the run obtained by concatenating the two runs and removing the duplicate state $last(\rho_1)$. We denote by $runs^{\mathcal{G}}$ the set of all finite runs of $\mathcal{G}$. We omit the $\mathcal{G}$ superscript when the timed graph is clear from the context.

Consider a timed graph $\mathcal{G} = (Q, Acts, q_{ini}, C, \Delta, inv)$. By definition, its state space is infinite. However, it can be partitioned in a finite number of equivalence classes called *regions*. A region is defined by a location and a *clock region*. A clock region is a set of equivalent clock valuations. Let c_x be the largest constant in clock constraints involving the clock variable x. For $d \in \mathbb{R}_{\geq 0}$, let $frac(d)$ denote the fractional part of d, and let $\lfloor d \rfloor$ denote its integral part.

[3] Differently from other presentations, we chose to disallow zero-time steps for technical reasons discussed later in this section.

The equivalence of clock valuations is formally defined as follows. Let C be the set of clocks of $\mathcal{G}$, and ν and ν' be two clock valuations, then ν is *(region) equivalent* to ν' if the following conditions are met [3].

- for all $x \in C$, either $\nu(x) > c_x$ and $\nu'(x) > c_x$, or $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$;
- for all $x, y \in C$ such that $\nu(x) < c_x$ and $\nu(y) < c_y$, we have $frac(\nu(x)) \leq frac(\nu(y))$ if and only if $frac(\nu'(x)) \leq frac(\nu'(y))$;
- for all $x \in C$ such that $\nu(x) < c_x$, we have $frac(\nu(x)) = 0$ if and only if $frac(\nu'(x)) = 0$.

We denote $[\nu]$ the clock region containing the clock valuation ν. The key property of this equivalence relation is that all the clock valuations belonging to a clock region satisfy the same set of clock constraints from the graph $\mathcal{G}$. Consequently, we say that a clock region α satisfies a constraint δ if ν satisfies δ, for all $\nu \in \alpha$. A region is a pair $\langle q, \alpha \rangle$, where q is a location and α is a clock region satisfying $inv(q)$. By abuse of notation, we identify the pair $\langle q, \alpha \rangle$ and the set of states $\{q\} \times \alpha$. We use *Regs* to denote the set of regions of $\mathcal{G}$. We extend the notion of step from states to regions as follows. Given two regions r and r', and an action $\sigma \in Acts^{\xi}$, we write $r \xrightarrow{\sigma} r'$ if there exists $s \in r$ and $s' \in r'$ such that $s \xrightarrow{\sigma} s'$.

A clock region α is *punctual* if either it is a singleton set or it has a clock x with integral value (i.e., $frac(x) = 0$) and all clocks have value different from zero. All other clock regions are *non-punctual*. The definition of punctual clock region applies to regions in the obvious way. Intuitively, in a two-clock system, α is punctual if it is either a point in $\mathbb{N}^2$ or it is a line which is parallel to an axis and does not lie on it. This distinction will be useful in the following, and it is motivated by the following property.

Lemma 1 *Starting from a given state of the timed graph, for each punctual region there is at most one point in that region which is reachable through a step. Conversely, for each non-punctual region, if there is a point in that region which is reachable through a step then there are infinitely many such points.*

Proof : Let $s = \langle q, \nu \rangle$ be a state in the timed graph and $r = \langle q', \alpha \rangle$ a region. Let $s' = \langle q', \nu' \rangle$ be a state in r which is reachable from s with a single step $s \xrightarrow{d,\sigma} s'$.

First, assume that r is punctual. If r is a singleton, the result is trivially true. Otherwise, there is a clock x that has the same integral value i in all points of r. Moreover, since all clocks have non-zero value in r, any step leading to r does not contain clock resets. Hence, $\nu(x) + d = \nu'(x) = i$. Clearly, any other time delay $d' \neq d$ would lead to a different value for x, and therefore not in r.

Now, assume that r is non-punctual. One of the following two cases holds.

- The region r has a clock x with value zero (i.e., in all points of the region it holds $x = 0$). Then, it also has at least one clock with value different from zero (otherwise, r would be a singleton). Since zero-time steps are not allowed, the only way to reach r is by resetting x.
- The region r has no clock with integral value (including zero).

In both cases, it is easy to see that there exists a neighbourhood of d such that for all values d' in the neighbourhood, $s \xrightarrow{\mathrm{d}',\sigma} s''$, where $s'' \in r$ and $s'' \neq s'$. $\square$

Note that allowing zero-time steps would falsify the second part of the above lemma. Indeed, in a two-clock timed graph, consider the non-punctual region $r = \langle q, (0 < x < 1, y = 0)\rangle$ and any state $s \in r$. The only way to reach r from s is by taking a zero-time step, contradicting the above lemma.

Two runs are said to be *(region) equivalent* if they have the same length and they are state-wise equivalent. Formally, ρ is equivalent to ρ' if $\rho = \langle q_0, \nu_0\rangle\langle q_1, \nu_1\rangle \ \ldots \ \langle q_n, \nu_n\rangle$, $\rho' = \langle q_0, \nu'_0\rangle\langle q_1, \nu'_1\rangle \ldots \langle q_n, \nu'_n\rangle$, and, for all $i = 0, \ldots, n$, we have $[\nu_i] = [\nu'_i]$. As before, we denote by $[\rho]$ the equivalence class containing the run ρ. A clock region α' is said to be a *successor* of a clock region α if and only if for all $\nu \in \alpha$ there is a $d \in \mathbb{R}_{\geq 0}$ such that $\nu + d \in \alpha'$. A region $\langle q', \alpha'\rangle$ is said to be a successor of a region $\langle q, \alpha\rangle$ (denoted $\langle q, \alpha\rangle \preceq \langle q', \alpha'\rangle$) if $q' = q$, α' is a successor of α and all clock regions between α and α' satisfy $inv(q)$. Moreover, we say that $\langle q, \alpha\rangle \prec \langle q', \alpha'\rangle$ if $\langle q, \alpha\rangle \preceq \langle q', \alpha'\rangle$ and $\alpha \neq \alpha'$.

3 Timed Games

A *timed game* is a pair $(\mathcal{G}, \varphi)$, where $\mathcal{G}$ is a timed graph and φ is the winning condition. Different classes of winning condition define different classes of timed games. The alphabet symbols (actions) of the timed graph represents the choices of the *protagonist*. The nondeterminism on the actions issued by the protagonist is used to model the possible choices of the *antagonist*, including the case in which the protagonist moves and the antagonist stays *idle*. To model the case that the protagonist stays idle and the antagonist chooses to move, we use the special action ξ. In the following, we denote by *Acts* a finite set of symbols not containing the antagonist action ξ. We also set $Acts^{\xi} = Acts \cup \{\xi\}$, and if not differently specified, σ denotes an element of *Acts*. A *play* of a timed game is a run of the corresponding timed graph, and it is constructed in the following way. At the beginning, both players declare their first move, that is, how long they will wait idling and their next discrete moves. At the time one of the players or both move, both players are allowed to declare their next discrete moves and the time these will be issued. That is,

if a player moves before the other, this latter is allowed to change its former decision.

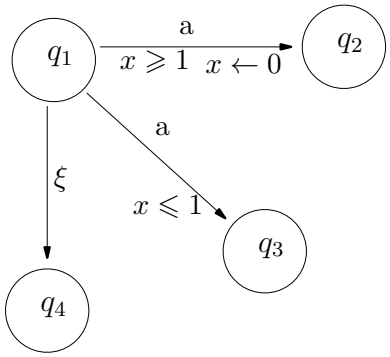

Fig. 1. A fragment of a timed game.

As instance, take the fragment of a 1-clock timed game shown in Figure 1. For the sake of simplicity, the invariants on the locations are not shown and they are supposed to hold always *true*. Suppose the game is in location q_1, with clock x equal to zero and the current strategy of the protagonist is to take an a-move, after a delay of one time units. Since at time $x = 1$ there are two a-moves enabled, the antagonist can choose which one is to be taken ad accordingly the game will proceed either to location q_2 or to location q_3. Moreover, note that form q_1 the antagonist can also move at his own, by taking a ξ-move before the antagonist move. Then, accordingly to the above strategy, the game will then proceed to location q_4, with $x \leq 1$.

In the following, if not differently specified, we always refer to a fixed timed graph $\mathcal{G} = (Q, Acts^{\xi}, q_{ini}, C, \Delta, inv)$. A *strategy* is a function $\mathcal{F} : runs \longrightarrow \mathbb{R}_{>0} \times Acts$, whose choices are allowed by the game. Formally, let $\rho = s_0 s_1 \dots s_n$ be a run of $\mathcal{G}$ and $\mathcal{F}(r) = (d, \sigma)$. Then, there is a step $s_n \xrightarrow{d,\sigma} s'$ in $\mathcal{G}$. The only exception to this rule occurs when there are no allowed moves for the protagonist from the current location. In this case, the value of the strategy can be arbitrary.

Given a strategy $\mathcal{F}$, we can collect all runs that are consistent with $\mathcal{F}$ in an infinite tree $T_{\mathcal{F}}$ called *strategy tree*. Moreover, we define an auxiliary labeling $act_{\mathcal{F}}$ which labels each node of the strategy tree with the action which leads to it. Formally, let $S = Q \times \mathbb{R}_{\geq 0}^{|C|}$, the strategy tree $T_{\mathcal{F}} = (S, V, E, v_{ini}, \mu)$ is inductively defined as follows. First, $v_{ini} \in V$, $\mu(v_{ini}) = \langle q_{ini}, \nu_{ini} \rangle$, and $act_{\mathcal{F}}(v_{ini})$ is arbitrarily set to any element of $Acts^{\xi}$. Then, let $\pi = u_1 u_2 \dots u_n$ be a finite rooted path in $T_{\mathcal{F}}$ and let $\mu(u_n) = s$; assume $\mathcal{F}(\mu(\pi)) = (d, \sigma)$ and consider the sets $S^{\text{Prot}} = \{s' \mid s \xrightarrow{d,\sigma} s'\}$ and $S^{\text{Ant}} = \{s' \mid s \xrightarrow{d',\xi} s',$ where $0 < d' < d\} \setminus S^{\text{Prot}}$. In words, S^{Prot} is the set of states that the game can reach if the protagonist is allowed to take the desired move σ after the desired delay d, while S^{Ant} is the set of states that the game can reach if the antagonist moves before the protagonist. Notice that there might be states that are reachable

both via a protagonist move and via an antagonist move. For instance, consider again the game in Figure 1 and, in addition, suppose that (by means of resets) state $\langle q_1, x = 0\rangle$ can be reached from $\langle q_{ini}, x = 0\rangle$ both via action a taken with some delay $0 < d < 1$ or via an action ξ taken with some delay $0 < d' < d$. However, actions are not part of the runs, so they are not visible to the players. Thus, in either case we obtain the same run $\rho = \langle q_{ini}, x = 0\rangle\langle q_1, x = 0\rangle$. The definition of the strategy tree mirrors this situation by removing duplicate states among the destinations. Ambiguous states, such as $\langle q_1, x = 0\rangle$ in the example, are ascribed to the protagonist and therefore assigned to S^{Prot}.

For each state which is either in S^{Prot} or in S^{Ant}, we add a corresponding child to u_n. Formally, for $x \in \{\text{Prot}, \text{Ant}\}$ and for each $s' \in S^x$, u_n has a child $v \in V$ whose labeling is $\mu(v) = s'$ and whose action is $act_{\mathcal{F}}(v) = \sigma$ if $x = \text{Prot}$ and $act_{\mathcal{F}}(v) = \xi$ otherwise. We denote $succ^x(T_{\mathcal{F}}, u_n)$ the set of all such children. We remark that the states associated to the children of u_n are all distinct.

By $untime(T_{\mathcal{F}})$ we denote the projection of $T_{\mathcal{F}}$ on Q. Formally, $untime(T_{\mathcal{F}}) = (2^Q, V, E, v_{ini}, \mu')$, where $\mu'(u) = \{q\}$ if and only if $\mu(u) = \langle q, \nu\rangle$ for some clock valuation ν. We set $plays^{\omega}_{\mathcal{F}} = \{\mu(\pi) \mid \pi \text{ is a rooted infinite path in } T_{\mathcal{F}}\}$ as the set of *infinite plays* of $\mathcal{F}$. Similarly, we define $plays_{\mathcal{F}}$ as the set of finite prefixes of runs in $plays^{\omega}_{\mathcal{F}}$. By construction, there exists a bijection between the elements of $plays^{\omega}_{\mathcal{F}}$ and the infinite paths in $T_{\mathcal{F}}$.

Although the definition allows strategies to choose among an infinite number of moves, it is natural to partition these moves in a finite number of classes, according to their intended destinations. We aim at declaring equivalent the moves that lead to the same set of possible next regions. Given a state $s = \langle q, \nu\rangle$ and a move $m = \langle d, \sigma\rangle$, we set $\theta(s, m) = \langle [\nu + d], \sigma\rangle$. Intuitively, given two states $s_1 = \langle q, \nu_1\rangle$ and $s_2 = \langle q, \nu_2\rangle$ that share the same location and two moves m_1 and m_2, if $\theta(s_1, m_1) = \theta(s_2, m_2)$, then playing m_1 from s_1 has the same effect as playing m_2 from s_2, in terms of possible next regions.

Given a strategy $\mathcal{F}$, let $[plays_{\mathcal{F}}]$ and $[\mathcal{F}] : [plays_{\mathcal{F}}] \to 2^{Regs \times Acts}$ be, respectively, the quotient of $plays_{\mathcal{F}}$ and of $\mathcal{F}$ with respect to the region equivalence relation, where we recall that *Regs* is the set of regions of $\mathcal{G}$. Formally, $[plays_{\mathcal{F}}] = \{[\rho] \mid \rho \in plays_{\mathcal{F}}\}$, and $[\mathcal{F}]([\rho]) = \bigcup_{\rho' \in [\rho]} \theta(last(\rho'), \mathcal{F}(\rho'))$. For an integer $k \geq 0$, we say that a strategy is *k-splitting* if it assigns up to k different destinations to equivalent histories. Formally, $\mathcal{F}$ is k-splitting iff, for all runs ρ, $|[\mathcal{F}]([\rho])| \leq k$. We further say that a strategy is *region stable* if it is 1-splitting. Region stable strategies choose equivalent moves for equivalent histories.

4 Branching Games

In this section, we deal with timed games whose winning condition is a formula from a branching-time temporal logic. In particular, we focus on the well-known logic CTL and we say that $(\mathcal{G}, \varphi)$ is an CTL game whenever φ is an CTL formula using as atomic propositions the locations of $\mathcal{G}$. Intuitively, in such a game we say that a strategy is winning if the dense tree $T_{\mathcal{F}}$ satisfies φ.

4.1 Computation Tree Logic

CTL was introduced by Emerson and Clarke [17] 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 *disjunction* ($\vee$), and the temporal operators *Next* (X), *Until* ($\mathcal{U}$), and *Release* ($\mathcal{R}$), coupled with path quantifiers *for all paths* ($\forall$) and *for some path* ($\exists$). CTL formulas are built up in the usual way from the above operators and connectives, according to the following grammar:

$$\varphi := p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists X\varphi \mid \forall X\varphi \mid \exists(\varphi \mathcal{U} \varphi) \mid \forall(\varphi \mathcal{U} \varphi) \mid \exists(\varphi \mathcal{R} \varphi) \mid \forall(\varphi \mathcal{R} \varphi)$$

where $p \in AP$. The semantics of CTL is defined with respect to a 2^{AP}-valued *graph* $G = (2^{AP}, V, E, v_{ini}, \mu)$. Given a CTL formula φ and a node $v \in V$, the satisfaction relation $(G, v) \models \varphi$, meaning that φ is true in G at v, is defined inductively as follows:

- if $\varphi \in AP$, then $(G, v) \models \varphi$ if and only if $\varphi \in \mu(v)$;
- $(G, v) \models \varphi_1 \wedge \varphi_2$ if and only if $(G, v) \models \varphi_1$ and $(G, v) \models \varphi_2$;
- $(G, v) \models \varphi_1 \vee \varphi_2$ if and only if $(G, v) \models \varphi_1$ or $(G, v) \models \varphi_2$;
- $(G, v) \models \forall X\varphi$ if and only if $(G, v') \models \varphi$ for all $v' \in V$ such that $(v, v') \in R$;
- $(G, v) \models \exists(\varphi_1 \mathcal{U} \varphi_2)$ if and only if there exists a path $v_0 v_1 \ldots$, with $v = v_0$, such that for some $i \geq 0$ $(G, v_i) \models \varphi_2$, and $(G, v_j) \models \varphi_1$ for all $0 \leq j < i$;
- $(G, v) \models \forall(\varphi_1 \mathcal{U} \varphi_2)$ if and only if for all paths $v_0 v_1 \ldots$, with $v = v_0$, there exists $i \geq 0$ such that $(G, v_i) \models \varphi_2$, and $(G, v_j) \models \varphi_1$ for all $0 \leq j < i$;
- $(G, v) \models \exists(\varphi_1 \mathcal{R} \varphi_2)$ if and only if there exists a path $v_0 v_1 \ldots$, with $v = v_0$, such that for all $i \geq 0$, if $(G, v_i) \not\models \varphi_2$ then there exists $0 \leq j < i$ such that $(G, v_j) \models \varphi_1$;
- $(G, v) \models \forall(\varphi_1 \mathcal{R} \varphi_2)$ if and only if for all paths $v_0 v_1 \ldots$, with $v = v_0$, and for all $i \geq 0$, if $(G, v_i) \not\models \varphi_2$ then there exists $0 \leq j < i$ such that $(G, v_j) \models \varphi_1$.

Notice that, for technical convenience, we presented CTL in *positive normal form*, as negation can only be applied to atomic propositions. This formulation is equivalent to more general ones where negation can be applied to arbitrary formulas. Given a CTL formula φ, it is possible to characterize all trees that

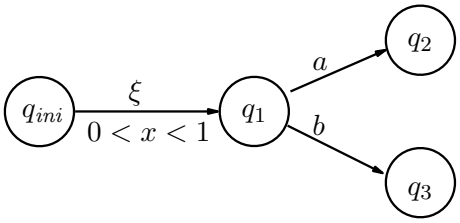

Fig. 2. A game where splitting strategies are needed.

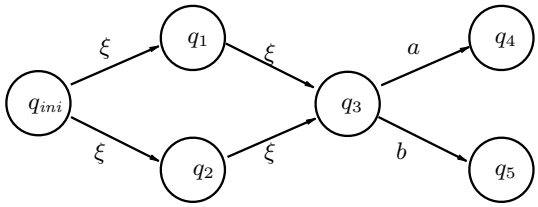

Fig. 3. A game where memory is needed.

satisfy φ via a Büchi tree automaton, as reported in the following proposition.

Proposition 3 ([46]) *Given a* Ctl *formula φ and an integer k, there exists a Büchi tree automaton A_k^φ accepting all ω-trees t with a branching degree k and such that $t \models \varphi$. Moreover, A_k^φ has a number of states that is exponential in the size of φ.*

Given a Ctl game $(\mathcal{G}, \varphi)$, we say that a strategy $\mathcal{F}$ is *winning* if $(untime(T_\mathcal{F}), v_{ini}) \models \varphi$. In the remaining of this section we show that deciding whether there exists a winning strategy $\mathcal{F}$ for a Ctl game is Exptime-complete.

4.2 Splitting and Memory are Needed

By means of an example, the following can be shown.

Lemma 2 *Region stable strategies are not sufficient for winning* Ctl *games.*

Proof : Consider the game in Figure 2, whose winning condition is the Ctl formula $\exists\Diamond q_2 \wedge \exists\Diamond q_3$. At the beginning, the antagonist must take a ξ-move in the clock region $0 < x < 1$. If the protagonist uses a region stable strategy, then, once the game is in location q_1, he can choose either action a or action b. Clearly, he cannot win the game in this way.

On the other hand, if the protagonist is allowed to use a general strategy, he could choose a if the antagonist played before $x = 0.5$ and b otherwise, thus winning the game. □

Notice that we are considering strategies with memory, that is strategies whose choices depend on the history of the game. Figure 3 gives an example of a game

where all winning strategies need memory. Indeed, let the winning condition of the game be $(\Diamond q_1 \vee \Diamond q_4) \wedge (\Diamond q_3 \vee \Diamond q_5)$, where $\Diamond\varphi$ is the usual LTL abbreviation for $true\,\mathcal{U}\varphi$. Once the game reaches q_3, any memoryless strategy can only choose either move a or move b. Such a strategy does not satisfy the winning condition. On the other hand, a memoryful strategy can issue a if q_3 is reached through q_2, and b otherwise. Such strategy is winning.

4.3 Solving CTL *Games*

Given a CTL game $(\mathcal{G}, \varphi)$, we wish to determine whether there exists a winning strategy. The automata-theoretic approach suggests to build an automaton accepting all possible strategies in the game. However, we show that it is sufficient to consider simpler structures, namely discrete infinite trees. Therefore, for each strategy $\mathcal{F}$ we define a corresponding discrete tree $t_{\mathcal{F}}$, in such a way that $\mathcal{F}$ is winning if and only if $t_{\mathcal{F}}$ satisfies φ. Intuitively, $t_{\mathcal{F}}$ is the untimed version of a suitable subtree of $T_{\mathcal{F}}$. Such subtree preserves all moves taken by the protagonist, as well as all moves taken by the antagonist from a punctual region. Furthermore, for each non-punctual region from which the antagonist can move, the subtree contains only a finite selection of the antagonist moves. In order for this subtree to faithfully represent the strategy it is derived from, the size of this selection must be at least equal to the splitting degree of the strategy (i.e., at least k if the strategy is k-splitting).

As we will see in the following, in order to win CTL games we may need strategies that have a splitting degree at least equal to the number of existential quantifiers present in the winning condition. Accordingly, we define an automaton $A_k^{\mathcal{G}}$ accepting all trees $t_{\mathcal{F}}$ corresponding to k-splitting strategies.

Then, by Proposition 3 we can build an automaton A_k^{φ} accepting all trees which satisfy φ and have branching degree k. By taking the intersection of $A_k^{\mathcal{G}}$ and A_k^{φ}, we obtain an automaton whose language is nonempty if and only if there exists a winning strategy in the game.

As an example, consider the timed graph in Figure 1, and suppose we want to accept all trees corresponding to 2-splitting strategies. The corresponding fragment of the tree automaton $A_2^{\mathcal{G}}$ is depicted in Figure 4. The automaton uses as locations the regions of the timed graph.

To see how this automaton works, suppose that the game is in the state $\langle q_1, x = 0\rangle$ and the protagonist chooses to issue the action a after a delay of 0.5 time units. One can see from Figure 1 that there is only one transition labeled with a and enabled at time $x = 0.5$, leading to the region $\langle q_3, 0 < x < 1\rangle$. Also, the antagonist can take a ξ-move for all delays in the interval $(0, 0.5)$, always leading to the region $\langle q_4, 0 < x < 1\rangle$. Since we are considering 2-

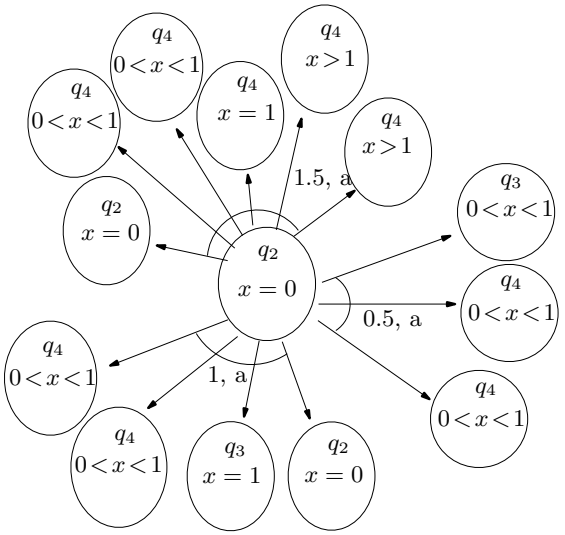

Fig. 4. Transitions of $A_2^{\mathcal{G}}$ from state $\langle 1, x = 0\rangle$, for the timed graph $\mathcal{G}$ of Figure 1. Edges that are linked by an arc belong to the same tree transition.

splitting strategies, the automaton keeps two copies for each non-punctual region of the antagonist. Therefore, the automaton contains a transition with three destinations: one for the region chosen by the protagonist and two copies of the (unique) region possibly chosen by the antagonist.

Similar arguments hold when the a-move is issued by the protagonist at time $x = 1$ or $x > 1$.

4.3.1 Strategy Samples

Given a strategy $\mathcal{F}$, let its strategy tree be $T_{\mathcal{F}} = (S, V, E, v_{ini}, \mu)$. For an integer $k > 0$, we define a class of discrete trees (called *k-samples of $\mathcal{F}$*) associated with $\mathcal{F}$. As intermediate step, we define a *pre-k-sample* to be a suitable sub-tree t of $T_{\mathcal{F}}$. Then, a k-sample is the projection of a pre-k-sample along the Q component, that is, if u is a node of a pre-k-sample and $\mu(u) = \langle q, \nu\rangle$, then the label of u in the corresponding k-sample is $\{q\}$. We denote $Sample_k(\mathcal{F})$ the set of all k-samples of $\mathcal{F}$. To conclude the construction, we provide the definition of a pre-sample.

For each $u \in V$, let $(succ^{\text{Prot}}(T_{\mathcal{F}}, u), succ^{\text{Ant}}(T_{\mathcal{F}}, u))$ be the partition of $succ(T_{\mathcal{F}}, u)$ as given in the definition of $T_{\mathcal{F}}$. A pre-sample of $\mathcal{F}$ is a sub-tree $t = (S, V', E', v_{ini}, \mu')$ of $T_{\mathcal{F}}$, where V' is defined as follows:

- For each $v \in succ^{\text{Prot}}(T_{\mathcal{F}}, u)$, we have $v \in V'$;
- for each $v \in succ^{\text{Ant}}(T_{\mathcal{F}}, u)$ such that $[\mu(v)]$ is punctual, we have $v \in V'$;
- for each non-punctual region α such that there exists $v \in succ^{\text{Ant}}(T_{\mathcal{F}}, u)$ with $\mu(v) \in \alpha$, there exist k distinct nodes $v_1, \ldots, v_k \in succ^{\text{Ant}}(T_{\mathcal{F}}, u)$ such that, for all $i = 1 \ldots k$, $\mu(v_i) \in \alpha$ and $v_i \in V'$.

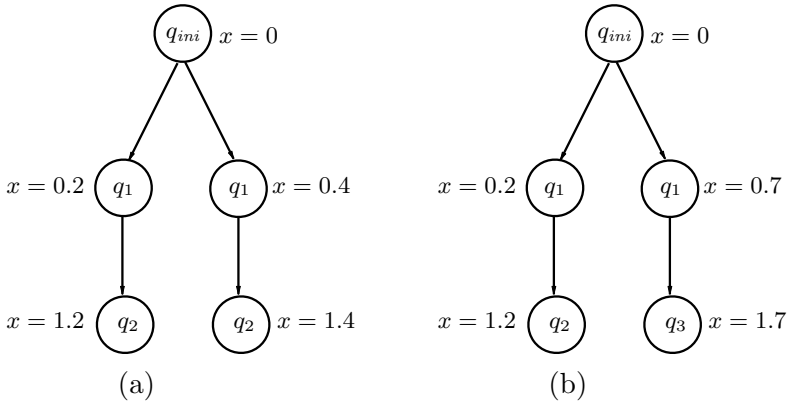

Fig. 5. Two samples obtained from the game of Figure 2 and the strategy that, from q_1, picks move $(1, a)$ when $x < 0.5$ and move $(1, b)$ otherwise.

Notice that the existence of k distinct points in the last case is ensured by Lemma 1. Components E' and μ' are simply the restrictions of E and μ to V'.

For instance, consider again the game in Figure 2, and consider the strategy $\mathcal{F}$ that, from state $\langle q_1, \nu \rangle$ chooses move $(1, a)$ if $\nu(x) < 0.5$ and move $(1, b)$ otherwise. Figure 5 shows two 2-samples of $\mathcal{F}$.

4.3.2 The Automaton $A_k^{\mathcal{G}}$

For an integer $k > 0$, we now define a tree automaton $A_k^{\mathcal{G}}$, whose task is to accept all and only the k-samples corresponding to some strategy. The locations of $A_k^{\mathcal{G}}$ are the regions of the timed automaton $\mathcal{G}$, while its labels are sets of locations of $\mathcal{G}$. We set $A_k^{\mathcal{G}} = (2^Q, \mathit{Regs}, \Delta, r_{ini})$, where $r_{ini} = \langle q_{ini}, [\nu_{ini}] \rangle$ and $\Delta \subseteq \cup_{i \geq 1} \mathit{Regs} \times 2^Q \times \mathit{Regs}^i$ is defined as follows.

Consider $r = \langle q, \alpha \rangle \in \mathit{Regs}$ as a location of $A_k^{\mathcal{G}}$. We distinguish the following two cases:

(1) Suppose that there exists a region z which is a successor of r and such that there is at least a σ-transition enabled from z, with $\sigma \in \mathit{Acts}$. Then, let $Z_\sigma = \{\, r' \mid z \xrightarrow{\sigma} r' \,\}$ and $Z_\xi = \{\, r' \mid z' \xrightarrow{\xi} r', r \preceq z' \prec z$ if z is boundary, and $r \preceq z' \preceq z$ otherwise $\}$, where a region is boundary if at least one clock has integral value. Let Z_ξ^{np} be the subset of Z_ξ containing only non-punctual regions. Let Z_ξ^p be the subset of Z_ξ containing all punctual regions except those already contained in Z_σ. We remove duplicate punctual regions from Z_ξ^p in accordance with the definition of strategy tree. Let us number the elements of these sets as follows: $Z_\sigma = \{r'_1, \ldots, r'_m\}$, $Z_\xi^p = \{r'_{m+1}, \ldots, r'_{m+h}\}$, and $Z_\xi^{np} = \{r'_{m+h+1}, \ldots, r'_{m+h+l}\}$.

The following transition belongs to Δ.

$$(r, \{q\}, r'_1, \ldots, r'_m, r'_{m+1}, \ldots, r'_{m+h}, \underbrace{r'_{m+h+1}, \ldots, r'_{m+h+1}}_{k \text{ times}}, \ldots, \underbrace{r'_{m+h+l}, \ldots, r'_{m+h+l}}_{k \text{ times}}).$$

(2) On the other hand, suppose that for all successors z of r no σ-transition is enabled in z. Let $Z_\xi = \{\, r' \mid z' \xrightarrow{\xi} r', r \preceq z' \,\}$, and, let Z_ξ be partitioned into the punctual regions $\{r'_1, \ldots, r'_m\}$, and non-punctual regions $\{r'_{m+1}, \ldots, r'_{m+h}\}$. Then, the following transition belongs to Δ.

$$(r, \{q\}, r'_1, \ldots, r'_m, \underbrace{r'_{m+1}, \ldots, r'_{m+1}}_{k \text{ times}}, \ldots, \underbrace{r'_{m+h}, \ldots, r'_{m+h}}_{k \text{ times}}).$$

In the second case, the game is in a state where the protagonist is not allowed to play anymore. This may result for two reasons: either there is no σ-move from the current location q, or all σ-moves have *expired*, meaning that they are disabled in α and will not be enabled in the future.

The branching degree of $A_k^{\mathcal{G}}$ is bounded by the product of the maximum branching degree of $\mathcal{G}$, $\max\{\, c_x \mid x \in C \,\}$, and k. From [3], we have that the number of regions of $\mathcal{G}$ is $O(|C|!\, 2^{|C|}\, \Pi_{x \in C}(2c_x + 2))$. Thus, the following result holds.

Lemma 3 *The size of $A_k^{\mathcal{G}}$ is exponential in the size of the clock constraints and linear in the number of locations of $\mathcal{G}$.*

Suppose we have a sample t that satisfies a CTL formula φ. We could be tempted to conclude that the strategy from which t is derived satisfies φ as well. However, this is not the case, since samples are rather arbitrary selections of paths from the original strategy tree. For instance, consider the 2-sample in Figure 5(a). It satisfies the formula $\forall X \forall X q_2$, while the strategy from which the sample is derived does not satisfy that formula. The problem is that the strategy has extra behaviors that are not present in the sample in Figure 5(a). However, we could modify the strategy in such a way as to exhibit all and only the behaviors that are present in a given sample. To this aim, we introduce the concept of *canonical* strategy for a tree.

Given a tree t, we say that a strategy $\mathcal{F}$ is a *canonical strategy for t* if *(i)* t is isomorphic [4] to a tree in $Sample_k(\mathcal{F})$ and *(ii)* t is bisimilar to $untime(T_{\mathcal{F}})$. We now show that, given a tree t accepted by $A_k^{\mathcal{G}}$, there exists a canonical strategy $\mathcal{F}$ for t.

Lemma 4 *Given a tree $t \in L(A_k^{\mathcal{G}})$, there exists a canonical strategy for t.*

[4] Two trees are isomorphic if there is a bijection between their sets of nodes which preserves the structure and the labeling of the trees.

Proof : Let $A_k^{\mathcal{G}} = (2^Q, \mathit{Regs}, \Delta, r_{ini})$. We split the proof in two parts: first, we inductively define a strategy $\mathcal{F}$ and we prove that $t \in \mathit{Sample}_k(\mathcal{F})$. In the second part, we show that t is bisimilar to $\mathit{untime}(T_{\mathcal{F}})$.

Building the strategy. Let $t = (2^Q, V', E', v'_{ini}, \hat{\mu}')$ and let $(\mathit{Regs}, V', E', v'_{ini}, \bar{\mu})$ be an accepting run of $A_k^{\mathcal{G}}$ over t. Together with $\mathcal{F}$, we inductively define the following functions:

- $\mathit{dir} : V' \to \{1, \ldots, k\}$; $\mathit{dir}(v)$ keeps track of the splitting history along the path from v'_{ini} to v;
- $\mathit{act} : V' \to \mathit{Acts}^{\xi}$; $\mathit{act}(v)$ stores the action which leads to v;
- $\mathit{trans} : V' \to \Delta$; $\mathit{trans}(v)$ keeps track of the transition of the automaton used to consume the children of v;
- $\mathit{rest} : V' \to \mathit{Regs}$; $\mathit{rest}(v)$ stores the successor region of $\bar{\mu}(v)$ where the protagonist takes his action (discrete step).

Suppose $\pi = u_0, \ldots, u_n$ is a path in t. We proceed by induction on n. For the root, we have $\mathit{dir}(v'_{ini}) = 1$, while $\mathit{act}(v_{ini})$ is arbitrarily defined. Suppose $\bar{\mu}(u_n) = \langle q, \alpha \rangle$ and let $v_1, \ldots, v_c$ be the children of u_n in t. Then, let $\mathit{trans}(u_n) \in \Delta$ be a transition of the form $(\bar{\mu}(u_n), \{q\}, \bar{\mu}(v_1), \ldots, \bar{\mu}(v_c))$. We assume w.l.o.g. that we are in case *(1)* of the definition of Δ. Accordingly, let $\sigma \in \mathit{Acts}$ and $z \in \mathit{Regs}$ be the action and the region which lead to the addition of $\mathit{trans}(u_n)$ to Δ. We set $\mathit{rest}(u_n) = z$. Also, there exist the following sets of regions: $Z_\sigma = \{r'_1, \ldots, r'_m\}$, such that nodes $v_1, \ldots, v_m$ correspond to the protagonist destinations $r'_1, \ldots, r'_m$; $Z_\xi^p = \{r'_{m+1}, \ldots, r'_{m+h}\}$, such that nodes $v_{m+1}, \ldots, v_{m+h}$ correspond to the punctual antagonist destinations $r'_{m+1}, \ldots, r'_{m+h}$; $Z_\xi^{np} = \{r'_{m+h+1}, \ldots, r'_{m+h+l}\}$, such that, for all $i = 1, \ldots, l$ and $j = 1, \ldots, k$, $v_{m+h+(i-1)\cdot k+j}$ corresponds to the j-th copy of the i-th non-punctual destination r'_{m+h+i} of the antagonist.

For technical convenience, let $w_i = v_{m+h+i}$ for all $i = 1, \ldots, k \cdot l$. Thus, $\mathit{trans}(u_n)$ can be written as:

$$(\bar{\mu}(u_n), \{q\}, \bar{\mu}(v_1), \ldots, \bar{\mu}(v_m), \bar{\mu}(v_{m+1}), \ldots, \bar{\mu}(v_{m+h}), \bar{\mu}(w_1), \ldots, \bar{\mu}(w_k), \ldots, \bar{\mu}(w_{k\cdot(l-1)+1}), \ldots, \bar{\mu}(w_{k\cdot l})).$$

For all $i = 1, \ldots, m$, we set $\mathit{dir}(v_i) = 1$ and $\mathit{act}(v_i) = \sigma$. For all $i = m + 1, \ldots, m + h$, we set $\mathit{dir}(v_i) = 1$ and $\mathit{act}(v_i) = \xi$. For all $i = 1, \ldots, l$, and $j = 1, \ldots, k$, we set $\mathit{dir}(w_{k\cdot(i-1)+j}) = j$ and $\mathit{act}(w_{k\cdot(i-1)+j}) = \xi$.

We define the following auxiliary functions. For all $\rho \in \mathit{plays}_{\mathcal{F}}$ and non-punctual region r, let $\mathit{slice}(\rho, r)$ be a partition $(\beta_1, \ldots, \beta_k)$ of the set of states in r that are reachable from $s = \mathit{last}(\rho)$ taking an antagonist move before

the intended protagonist move. Formally, let $\mathcal{F}(\rho) = (\bar{d}, \bar{\sigma})$, $(\beta_1, \ldots, \beta_k)$ is a partition of the set $\{s' \in r \mid s \xrightarrow{d,\xi} s', \text{ for some } 0 < d < \bar{d}\}$. Moreover, for all $i = 1, \ldots, k$, we set $\mathit{slice}(\rho, r, i) = \beta_i$. For a punctual region r, or an empty run $\rho = \varepsilon$, we set $\mathit{slice}(\rho, r) = \mathit{slice}(\rho, r, 1) = r$.

Let $\rho = s_0 \ldots s_n$ be a run in $\mathit{plays}_{\mathcal{F}}$. Since there is a bijection between $\mathit{plays}_{\mathcal{F}}$ and paths in $T_{\mathcal{F}}$, there is a path $\bar{u}_0 \ldots \bar{u}_n$ in $T_{\mathcal{F}}$ corresponding to ρ. We say that ρ is *consistent* with π if for all $i = 0, \ldots, n$, it holds *(i)* if $\mathit{act}_{\mathcal{F}}(\bar{u}_i) = \sigma$, then $\mathit{act}(u_i) = \sigma$ and $s_i \in \bar{\mu}(u_i)$, and *(ii)* otherwise (i.e., if $\mathit{act}_{\mathcal{F}}(\bar{u}) = \xi$), then $\mathit{act}(u_i) = \xi$ and $s_i \in \mathit{slice}(\rho_{\leq i-1}, \bar{\mu}(u_i), \mathit{dir}(u_i))$. We denote by $\mathit{Cons}(\pi)$ the set of all runs that are consistent with π. The following *uniqueness property* holds and will be proved in the following: each finite run of length n is consistent with at most one path of length n in t. Using this property, for all $\rho \in \mathit{Cons}(\pi)$, we fix $\mathcal{F}(\rho) = (d, \sigma)$, where d is a time delay such that $\mathit{last}(\rho) + d \in \mathit{rest}(u_n)$.

To conclude the definition of $\mathcal{F}$, we prove the above uniqueness property. Let $\rho = s_0 \ldots s_n$ be a run in $\mathit{plays}_{\mathcal{F}}$. By contradiction, assume that ρ is consistent with two distinct paths $\pi' = u'_0 \ldots u'_n$ and $\pi'' = u''_0 \ldots u''_n$ in t. Let b be an index such that ${\pi'}_{\leq b-1} = {\pi''}_{\leq b-1}$ and $u'_b \neq u''_b$. Let $\mathit{trans}(u'_{b-1})$ be:

$$(z, \{q\}, z'_1, \ldots, z'_m, z'_{m+1}, \ldots, z'_{m+h}, \underbrace{z'_{m+h+1}, \ldots, z'_{m+h+1}}_{k \text{ times}}, \ldots, \underbrace{z'_{m+h+l}, \ldots, z'_{m+h+l}}_{k \text{ times}}),$$

where $\bar{\mu}(u'_{b-1}) = z$ and there exist $i, j \in \{1, \ldots, m+h+l\}$ such that $\bar{\mu}(u'_b) = z'_i$ and $\bar{\mu}(u''_b) = z'_j$. Recall from the definition of $A^{\mathcal{G}}_k$ that nodes $z'_1, \ldots, z'_m$ correspond to protagonist moves, nodes $z'_{m+1}, \ldots, z'_{m+h}$ correspond to antagonist moves leading to punctual regions, and the remaining nodes to antagonist moves leading to non-punctual regions.

Since $\rho \in \mathit{Cons}(\pi') \cap \mathit{Cons}(\pi'')$, we have in particular $\mathit{act}_{\mathcal{F}}(s_b) = \mathit{act}(u'_b) = \mathit{act}(u''_b)$, $s_b \in z'_i \cap z'_j$, and therefore $z'_i = z'_j$. If $\mathit{act}_{\mathcal{F}}(s_b) \in \mathit{Acts}$, we obtain an immediate contradiction with the definition of $A^{\mathcal{G}}_k$, since there are no duplicates among the regions corresponding to destinations of the protagonist. Otherwise, assume $\mathit{act}_{\mathcal{F}}(s_b) = \xi$. Then, we also have $s_b \in \mathit{slice}(\rho_{\leq b-1}, \bar{\mu}(u'_b), \mathit{dir}(u'_b)) \cap \mathit{slice}(\rho_{\leq b-1}, \bar{\mu}(u''_b), \mathit{dir}(u''_b))$. If both regions $\bar{\mu}(u'_b)$ and $\bar{\mu}(u''_b)$ are punctual, an argument similar to the one used in the previous case applies. Otherwise, both regions are non-punctual. There exists a unique $g \in \{1, \ldots, l\}$ such that $\bar{\mu}(u'_b) = \bar{\mu}(u''_b) = z'_{m+h+g}$. Moreover, since $u'_b \neq u''_b$, it must be $\mathit{dir}(u'_b) \neq \mathit{dir}(u''_b)$. So, we found two different slices of the same region having the state s_b in common, which is a contradiction.

This concludes the proof of the property and the definition of $\mathcal{F}$. From the definition of k-sample, it holds that t is isomorphic to a k-sample of $\mathcal{F}$, which proves the first condition in the definition of canonical strategy.

Bisimilarity. In this part, we prove that t is bisimilar to $\mathit{untime}(T_{\mathcal{F}})$, where $\mathcal{F}$

is the strategy defined in the previous part of the proof. Let $T_{\mathcal{F}} = (S, V, E, v_{ini}, \mu)$ and $untime(T_{\mathcal{F}}) = (2^Q, V, E, v_{ini}, \hat{\mu})$. In the following, we assume w.l.o.g. that t is a k-sample of $\mathcal{F}$. Let $t' = (S, V', E', v'_{ini}, \mu')$ be the pre-sample corresponding to t. Let dir be the auxiliary function used to define the canonical strategy $\mathcal{F}$.

We inductively define the candidate bisimulation $R \subseteq V' \times V$ as follows. First, we have $(v'_{ini}, v_{ini}) \in R$. Then, let $(v', v) \in R$, π be the path in t from v'_{ini} to v', and ρ be the path in $T_{\mathcal{F}}$ from v_{ini} to v. While defining R, we simultaneously prove that $\rho \in Cons(\pi)$. We proceed by induction on $|\rho|$. The thesis is obviously true for $|\rho| = 1$ (i.e., $v_{ini} \in Cons(v'_{ini})$).

Let w be a successor of v in $T_{\mathcal{F}}$ due to a protagonist move (i.e., $act_{\mathcal{F}}(w) \in Acts$). There is a successor w' of v' in t that is due to the same action of the protagonist, leading to the same region (i.e., $[\mu'(w')] = [\mu(w)]$). We set $(w', w) \in R$. It follows from our choice of w' that $\rho w \in Cons(\pi w')$. Similarly for the successors of v which are due to the antagonist and represent punctual regions.

Finally, let w be a successor of v in $T_{\mathcal{F}}$ due to an antagonist move leading to a non-punctual region r (i.e., $act_{\mathcal{F}}(w) = \xi$ and $\mu(w) \in r$). In the construction of $\mathcal{F}$, region r is split into k slices according to function $slice(\rho, r)$, and v' has a child for each such slice. Nodes are linked to slices by function dir. In particular, there is a child w' of v' whose slice contains $\mu(w)$. This is the node which we associate to w in R. Formally, w' is the child of v' such that $\mu(w) \in slice(\rho, r, dir(w'))$, and we set $(w', w) \in R$. It follows from our choice of w' that $\rho w \in Cons(\pi w')$.

Next, we prove that R is a bisimulation between t and $untime(T_{\mathcal{F}})$. For all $(v', v) \in R$, let π be the path in t from v'_{ini} to v', and let ρ be the path in $T_{\mathcal{F}}$ from v_{ini} to v.

Condition *(1)* of the definition of bisimulation (i.e., $\hat{\mu}'(v') = \hat{\mu}(v)$) is an immediate consequence of the fact that $\rho \in Cons(\pi)$. Condition *(3)* of the definition of bisimulation follows directly from the construction of R, since for every successor w of v we have explicitly exhibited a successor w' of v' such that $(w', w) \in R$.

Regarding condition *(2)* of the definition of bisimulation, let $w' \in succ(t, v')$. We show that there exists a node $w \in succ(T_{\mathcal{F}}, v)$ such that $\rho w \in Cons(\pi w')$. By condition *(3)* of bisimulation, there exists $w'' \in V'$ such that $(w'', w) \in R$. Let π'' be the path in t from the root to the parent of w'', we have $\rho w \in Cons(\pi'' w'')$. By the uniqueness property, $\pi w' = \pi'' w''$ and so $(w', w) \in R$.

To conclude the proof, we show the existence of such w. Consider the following cases:

(1) w' is due to the protagonist, i.e. $act(w') \in Acts$. Then, recall that $rest(v')$ is a successor of the region $\bar{\mu}(v')$ such that $rest(v') \xrightarrow{act(w')} \bar{\mu}(w')$. Since $\rho \in Cons(\pi)$, $\mathcal{F}(\rho) = (d, \sigma)$, where $\sigma = act(w')$ and $\mu(v) + d \in rest(v')$. In conclusion, there must be $w \in succ(T_{\mathcal{F}}, v)$ such that $\mu(v) \xrightarrow{d,\sigma} \mu(w)$ and $\mu(w) \in \bar{\mu}(w')$. Thus, $\rho w \in Cons(\pi w')$ by condition *(i)* of consistency.

(2) w' is due to the antagonist and it corresponds to a punctual region, i.e. $act(w') = \xi$ and $\bar{\mu}(w')$ is punctual. We proceed similarly to the previous case.

(3) w' is due to the antagonist and it corresponds to a non-punctual region, i.e. $act(w') = \xi$ and $\bar{\mu}(w')$ is non-punctual. Consider the set $slice(\rho, \bar{\mu}(w'), dir(w'))$.

As recalled in case 1, the delay $\bar{d}$ chosen by the strategy $\mathcal{F}$ after ρ leads to the region $rest(v')$, which is the region from which the protagonist takes the discrete step according to t. By the definition of $A_k^{\mathcal{G}}$, the region $rest(v')$ is a successor of $\bar{\mu}(w')$. Finally, recall that $slice(\rho, \bar{\mu}(w'))$ is a partition of the set

$$\{s' \in \bar{\mu}(w') \mid \mu(v) \xrightarrow{d,\xi} s', \text{ for some } 0 < d < \bar{d}\}.$$

Since $v = last(\rho)$, the above set is not empty.

Let $s \in slice(\rho, \bar{\mu}(w'), dir(w'))$, there is a delay $0 < d < \bar{d}$ such that $\mu(v) \xrightarrow{d,\xi} s$. By definition of $T_{\mathcal{F}}$, there is then a node $w \in succ(T_{\mathcal{F}}, v)$ such that $\mu(w) = s$ and $act_{\mathcal{F}}(w) = \xi$. By definition of consistency, we conclude that $\rho w \in Cons(\pi w')$.

□

We can now prove the following result.

Lemma 5 *The automaton $A_k^{\mathcal{G}}$ accepts all and only the k-samples of strategies.*

Proof : Let $\mathcal{F}$ be a strategy and $t \in Sample_k(\mathcal{F})$. We prove that $t \in L(A_k^{\mathcal{G}})$. Let $t' = (S, V, E, v_{ini}, \mu)$ be the pre-sample corresponding to t, and let $[t']$ be the tree $(Regs, V, E, v_{ini}, \mu')$, where for all $v \in V$, $\mu'(v) = \langle q, [\nu] \rangle$ if and only if $\mu(v) = \langle q, \nu \rangle$. It is easy to show by induction that $[t']$ is an accepting run of $A_k^{\mathcal{G}}$ over t.

Conversely, let $t \in L(A_k^{\mathcal{G}})$. By Lemma 4, there exists a canonical strategy $\mathcal{F}$ for t, and thus $t \in Sample_k(\mathcal{F})$. □

4.3.3 Bounding the Splitting Degree

Lemma 6 *Given a* CTL *formula φ, let $\mathcal{F}$ be a winning strategy w.r.t. φ. There exists $t \in Sample_k(\mathcal{F})$ such that $t \models \varphi$, where k is the number of existential quantifiers in φ.*

Proof : Let $T_{\mathcal{F}} = (S, V, E, v_{ini}, \mu)$. To define $t = (S, V', E', v_{ini}, \mu')$, it is

sufficient to define V', as E' and μ' are simply the restrictions of E and μ to V', respectively.

For all $v \in V$, let $Sub^{\exists}(v)$ be the set of existential subformulas of φ that are true in $T_{\mathcal{F}}$ at v. Formally, $Sub^{\exists}(v) = \{\varphi' \in Sub(\varphi) \mid \varphi' = \exists\theta \text{ and } (T_{\mathcal{F}}, v) \models \varphi'\}$. The following recursive definition uses a set $P \subseteq Sub(\varphi) \times V^*$ to remember specific paths that are needed to satisfy certain existential subformulas of φ.

Given $W \subseteq V$ and $u \in W$, we say that u is a *leaf w.r.t.* W if for all $v \in W$ we have $(u, v) \notin E$. For a set Σ, we let $\Sigma^{\infty} = \Sigma^* \cup \Sigma^{\omega}$.

function AddLayer($V' \subseteq V, f : V' \to 2^{Sub(\varphi) \times V^{\infty}}$)
vars:
  W : set of nodes
  $g : V \to 2^{Sub(\varphi) \times V^{\infty}}$
main:
  for each leaf u w.r.t. V' **do**
    $(W, g) :=$ **AddChildren**$(u, f(u))$
    $V' := V' \cup W$
    $f := f \cup g$
  done
  return (V', f)

function AddChildren($u \in V, P \subseteq Sub(\varphi) \times V^{\infty}$)
vars:
  W : set of nodes
  $g : V \to 2^{Sub(\varphi) \times V^{\infty}}$
  RelNodes (relevant nodes) : set of nodes
  NewP (new promises) : $succ(T_{\mathcal{F}}, u) \to 2^{Sub(\varphi) \times V^{\infty}}$
init:
  $W := \emptyset$
  RelNodes $:= \emptyset$
  for each $v \in succ(T_{\mathcal{F}}, u)$ **do** $NewP(v) := \emptyset$
main:
  / collect nodes preserving existential formulas that are true in u */*
  for each $\exists\psi \in Sub^{\exists}(u)$ **do**
    if $\psi = X\theta$ **then**
      pick a node $v \in succ(T_{\mathcal{F}}, u)$ such that $(T_{\mathcal{F}}, v) \models \theta$
      RelNodes $:=$ *RelNodes* $\cup \{v\}$
    else if $\big(\psi = \theta_1 \,\mathcal{U}\, \theta_2$ and $(T_{\mathcal{F}}, u) \not\models \theta_2\big)$ or $\big(\psi = \theta_1 \,\mathcal{R}\, \theta_2$ and $(T_{\mathcal{F}}, u) \not\models \theta_1\big)$ **then**
      if $(\psi, uv_1v_2\ldots) \in P$ **then**
        RelNodes $:=$ *RelNodes* $\cup \{v_1\}$
      else
        pick a path $\pi = uv_1v_2\ldots$ such that $(T_{\mathcal{F}}, \pi) \models \psi$
        RelNodes $:=$ *RelNodes* $\cup \{v_1\}$
        $NewP(v_1) := NewP(v_1) \cup \{(\psi, v_1v_2\ldots)\}$

```
            endif
        endif
    done
    /* add relevant nodes */
    for each v ∈ RelNodes do
        W := W ∪ {v}
        g(v) := NewP(v) ∪ Update(v, P)
    done
    /* add mandatory samples */
    for each v ∈ succ^Prot(T_F, u) \ RelNodes do
        W := W ∪ {v}
        g(v) := Update(v, P)
    done
    for each v ∈ succ^Ant(T_F, u) \ RelNodes such that [μ(v)] is punctual do
        W := W ∪ {v}
        g(v) := Update(v, P)
    done
    /* add mandatory samples for open regions */
    for each non-punctual region α such that there exists v ∈ succ^Ant(T_F, u) with μ(v) ∈ α do
        n := k − |{v ∈ RelNodes | μ(v) ∈ α}|
        while n > 0 do
            pick v in succ^Ant(T_F, u) \ W with μ(v) ∈ α
            W := W ∪ {v}
            g(v) := Update(v, P)
            n := n − 1
        done
    done
    return (W, g)
```

```
function Update(v ∈ V, P ⊆ Sub(φ) × V^∞)
vars:
    B : subset of Sub(φ) × V^∞
init:
    B := ∅
main:
    for each (ψ, v_0 v_1 ...) ∈ P do
        if v = v_1 then B := B ∪ {(ψ, v_1 v_2 ...)}
    done
    return B
```

Finally, the wanted set V' is obtained by infinitely many iterative applications of AddLayer on $(\{v_{ini}\}, \emptyset)$. $\square$

4.3.4 The Automata-theoretic Solution

To solve branching games, we build an automaton $A^{\cap}$ accepting the intersection of $L(A_k^{\varphi})$ and $L(A_k^{\mathcal{G}})$, and then check for its emptiness. However, $A_k^{\mathcal{G}}$ may have a different branching degree than A_k^{φ}. Let h be the maximum branching degree among A_k^{φ} and $A_k^{\mathcal{G}}$. Before taking the intersection, we augment each transition of each automaton to have degree exactly h, by repeating existing destinations, in all possible permutations.

Lemma 7 *Given a* CTL *game* $(\mathcal{G}, \varphi)$*, there exists a winning strategy if and only if* $L(A^{\cap})$ *is nonempty.*

Proof : Assume $t \in L(A^{\cap}) = L(A_k^{\mathcal{G}}) \cap L(A_k^{\varphi})$. By Lemma 5, there is a strategy $\mathcal{F}$ such that t is a k-sample of $\mathcal{F}$, and by Proposition 3, $t \models \varphi$. Thus, by Lemma 4, there is a strategy $\mathcal{F}'$ such that $\mathcal{T}_{\mathcal{F}'} \models \varphi$ and thus $\mathcal{F}'$ is a winning strategy.

Conversely, suppose that $\mathcal{F}$ is a winning strategy and $T_{\mathcal{F}}$ is the associated strategy tree. Assuming that φ contains k existential quantifiers, by Lemma 6, there exists a k-sample t of $\mathcal{F}$ that satisfies φ. By Lemma 5, t is accepted by $A_k^{\mathcal{G}}$, and by Proposition 3, it is also accepted by A_k^{φ}. Therefore, $L(A^{\cap})$ is nonempty. □

By taking the intersection of A_k^{φ} and $A_k^{\mathcal{G}}$, we obtain a Büchi tree automaton $A^{\cap}$ whose number of states is the product of the number of states of the original automata. By Lemma 7 and the fact that the emptiness problem for Büchi automata is decidable in polynomial time (Proposition 2), the following holds.

Theorem 1 *Given a* CTL *game* $(\mathcal{G}, \varphi)$*, the problem of deciding the existence of a winning strategy in* $(\mathcal{G}, \varphi)$ *is solvable in exponential time.*

We conclude this section by briefly discussing about lifting the results we have achieved here for CTL directly to game logics, such as ATL, as well as to different game frameworks, such as the module checking approach.

4.4 A comparison with ATL

Alternating-time Temporal Logic [5] (ATL) is a well-known formalism for specifying properties of multi-agent systems. When interpreted on one-agent systems, ATL closely resembles CTL: its team quantifiers correspond to the path quantifiers of CTL. Since ATL model-checking is in PTIME [5], one may wonder whether the timed games with CTL objectives studied in this paper may be more efficiently solved by converting the objective into ATL and then executing an ATL model checking algorithm on the region graph corresponding

to the game. Two obstacles prevent this course of actions: first, due to Theorem **??**, the region graph may not be sufficient to find winning strategies; second, and more importantly, our semantics for CTL does not match the standard semantics for ATL. Consider the CTL goal $\exists\varphi_1 \wedge \exists\varphi_2$, where φ_1 and φ_2 are path formulas, and compare it with the apparently similar ATL formula $\ll 1,2 \gg \varphi_1 \wedge \ll 1,2 \gg \varphi_2$, where $\ll 1,2 \gg$ is the team quantifier which puts the protagonist and the environment in the same team. According to our semantics, the CTL goal is satisfied by the game if the protagonist has a strategy whose tree contains a path satisfying φ_1 and another path satisfying φ_2. On the other hand, the above ATL formula is satisfied if the team has one strategy that satisfies φ_1 and another strategy that satisfies φ_2. In other words, in order to satisfy the ATL formula, the protagonist may use two different strategies for φ_1 and φ_2, respectively.

4.5 A comparison with module checking

In [28,29], Kupferman, Vardi, and Wolper studied the model checking problem for open finite-state systems. In their framework, the open finite-state system is described by a labeled state-transition graph called a *module*, whose set of states is partitioned into a set of *system states* (where the system makes a transition) and a set of *environment states* (where the environment makes a transition). Given a module M describing the system to be verified, and a temporal logic formula φ specifying the desired behavior of the system, the problem of model checking a module, called *module checking*, asks whether for all possible environments M satisfies φ. In particular, it might be that the environment does not enable all the external nondeterministic choices. Module checking thus involves not only checking that the full computation tree $\langle T_M, V_M \rangle$ obtained by unwinding M (which corresponds to the interaction of M with a maximal environment) satisfies the specification φ, but also that every tree obtained from it by pruning children of environment nodes (this corresponds to the different choices of different environments) satisfy φ. In other words, module checking can be seen as a two-player turn-based game where one of the two players (the system) has a deterministic (full) strategy. It is shown in [28,29,36,20] that for formulas in branching time temporal logics, module checking open finite-state systems is exponentially harder than model checking closed finite-state systems. Recently, the module checking problem has been extended to infinite state-systems and, in particular, to pushdown systems[14,9,8], showing that this problem for CTL is 2EXPTIME-complete in the perfect information setting and undecidable in the imperfect one.

A natural question that arises is whether the techniques introduced above can be lifted to the real-time module checking framework w.r.t. to CTL goals. Clearly, this can be done easily once the given problem is reduced to one on

discrete trees. To be convinced that this can be indeed the case, first observe that in real-time module checking the model is a timed module [35], i.e, an open real-time labeled state-transition graph. Also, observe that the unwinding of this module is a dense tree. Now, as we have done above, consider the unwinding of the region graph of such a model, which is a discrete tree. It is not hard to prove that the former and the latter are equivalent with respect to CTL goals. At this point, we can continue as in the discrete case and get the solution to of our problem.

5 Linear Games

In this section, we deal with timed games whose winning condition is an LTL formula. We say that $(\mathcal{G}, \varphi)$ is an LTL game whenever φ is an LTL formula using as atomic propositions the locations of $\mathcal{G}$. Intuitively, in such a game we say that a strategy for the protagonist is winning if every run which is consistent with the strategy satisfies the formula.

5.1 Linear Temporal Logic

Linear Temporal Logic (LTL) was introduced by Pnueli to specify and verify properties of reactive systems [38]. Given a set of atomic propositions AP, an LTL formula is composed of atomic propositions, the boolean connectives *conjunction* ($\wedge$) and *negation* ($\neg$), and the temporal operators *Next* (X) and *Until* ($\mathcal{U}$). LTL formulas are built up in the usual way from the above operators and connectives, according to the following grammar:

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi \mid X\varphi \mid \varphi \mathcal{U} \varphi$$

where p is an atomic proposition. We denote by $|\varphi|$ the length of formula φ. The semantics of LTL formulas is given with respect to an infinite word $w = \sigma_0\sigma_1 \ldots \sigma_n \ldots$ 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 $\varphi \in \sigma_0$;
- $w \models \neg\varphi$ 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 $w \models \varphi_2$;
- $w \models X\varphi$ if and only if $w_{\geq 1} \models \varphi$;
- $w \models \varphi_1 \mathcal{U} \varphi_2$ if and only if there exists $i \geq 0$ such that $w_{\geq i} \models \varphi_2$ and $w_{\geq j} \models \varphi_1$ for all j such that $0 \leq j < i$.

Given a tree t, we also say that $t \models \varphi$ if, for all paths π of t, we have that $\pi \models \varphi$. For every LTL formula φ, it is possible to construct a nondeterministic

Büchi automaton on ω-words accepting all ω-words models of φ [46]. 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 Büchi automaton on ω-words $A = (\Sigma, Q, \Delta, q_{ini}, F)$, we can easily obtain a tree automaton A' accepting all trees t with branching degree bounded above by k and such that every path of t is a word accepted by A. The tree automaton has the form $A' = (\Sigma, Q, \Delta', q_{ini}, F)$, where Δ' contains all and only the transition $(q, \sigma, q_1, \ldots, q_l)$, where $(q, \sigma, q_i) \in \Delta$, for all $i = 1, \ldots, l$ and $l \leq k$. Clearly, such a construction does not work for nondeterministic automata.

Given an LTL formula φ, we can build a deterministic generator A_k^φ for φ in the following way: we start with a nondeterministic Büchi generator with $2^{O(|\varphi|)}$ states [34,47]; a Büchi automaton with n states can be converted into a deterministic Rabin automaton with $2^{O(n \log n)}$ states and n accepting pairs [42]. Thus, we obtain a deterministic Rabin generator for φ with a doubly exponential number of states and exponentially many accepting pairs. Notice that, in general, for a given formula φ, a deterministic Büchi generator may not exist but, when it exists, it may require a doubly exponential number of states in the length of the formula (see [31]), and thus the above construction is asymptotically optimal.

Proposition 4 *Given an* LTL *formula φ and an integer k, there exists a deterministic Rabin tree automaton A_k^φ, accepting all ω-trees t with branching degree k and such that $t \models \varphi$. Moreover, A_k^φ has a doubly exponential number of states and exponentially many accepting pairs.*

We now define when a strategy is winning with respect to an LTL winning condition. Given a timed graph $\mathcal{G}$ and a (possibly infinite) run $\rho = \langle q_{ini}, \nu_{ini} \rangle \langle q_1, \nu_1 \rangle \ldots \langle q_n, \nu_n \rangle$ in $\mathcal{G}$, we define $Untime(\rho) = \{q_{ini}\}\{q_1\} \ldots \{q_n\}$, that is, the sequence of locations traversed by the run, with each location wrapped in a singleton set. Given an LTL game $(\mathcal{G}, \varphi)$ and a strategy $\mathcal{F}$, we say that $\mathcal{F}$ is *winning* if, for all $\rho \in plays_{\mathcal{F}}^{\omega}$, $Untime(\rho) \models \varphi$.

5.2 Solving LTL *Games*

The following result greatly simplifies the treatment of linear games compared to the CTL games of Section 4.

Proposition 5 ([19]) *Region stable strategies are sufficient for winning* LTL *games.*

By Proposition 4, let A_k^φ the Rabin tree automaton corresponding accepting all trees satisfying φ whose branching degree is at most $deg(A_1^T)$. Since Rabin

tree automata are closed under intersection [43], we can build a Rabin tree automaton $A^{\cap}$ accepting the intersection of the languages accepted by A_1^T and A_k^{φ}. In particular, since by [43] the size of $A^{\cap}$ is polynomial in the size of A_1^T and A_k^{φ}, and the size of A_k^{φ} is doubly exponential in φ, the size of $A^{\cap}$ is doubly exponential in the size of φ. By Lemma 3, the size of $A^{\cap}$ is singly exponential in the size of $\mathcal{G}$. Moreover, the number of pairs in the accepting condition of $A^{\cap}$ is exponential in the size of φ.

Lemma 8 *Given an* LTL *game* $(\mathcal{G}, \varphi)$*, there exists a winning strategy for the protagonist if and only if* $L(A^{\cap})$ *is nonempty.*

Proof : Let $t \in L(A^{\cap})$. Since $t \in L(A_1^T)$, by Lemma 4 there exists a canonical strategy $\mathcal{F}$ for t. By Proposition 4, $t \models \varphi$. Therefore, $\mathit{untime}(T_{\mathcal{F}}) \models \varphi$, and so $\mathcal{F}$ is a winning strategy.

Conversely, let $\mathcal{F}$ be a winning strategy (i.e., $\mathit{untime}(T_{\mathcal{F}}) \models \varphi$). Since the winning condition is an LTL formula, all samples of $\mathcal{F}$ also satisfy φ. Therefore, $L(A^{\cap})$ is not empty. $\square$

By the previous lemma, Proposition 1, and the fact that LTL games are 2EXPTIME-hard [39], the following theorem holds.

Theorem 2 *Given an* LTL *game* $(\mathcal{G}, \varphi)$*, the problem of deciding the existence of a winning strategy for the protagonist is 2*EXPTIME*-complete.*

6 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 [33], 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 [39], our result is tight. Combining our construction with the results on LTL generators from [7], we can prove an upper bound smaller than 2EXPTIME for meaningful subclasses of LTL timed games.

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.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. In *Journal of the ACM* Vol. 49(5), pages 672–713.

[6] 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.

[7] 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.

[8] B. Aminof, A. Legay, A. Murano, O. Serre, and Moshe Y. Vardi. Pushdown module checking with imperfect information. Under review to International Journal http://people.na.infn.it/ murano/pubblicazioni/Module-extended.pdf

[9] Benjamin Aminof, A. Murano, and Moshe Y. Vardi. Pushdown module checking with imperfect information. In *18th International Conference on Concurrency Theory, (CONCUR'07)*, LNCS 4703, pages 461–476, 2007.

[10] 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.

[11] 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.

[12] 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.

[13] Laura Bozzelli, A. Murano, and Adriano Peron. Pushdown module checking. In *12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, pages 504–518, 2005

[14] Laura Bozzelli, A. Murano, and Adriano Peron. Pushdown module checking. *Formal Methods in System Design (FMSD 2010)*, 36(1):65–95, 2010.

[15] J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state stategies. *Trans. Amer. Math. Soc.*, 138:295 – 311, 1969.

[16] A. Church. Logic, arithmetic, and automata. In *Proc. of the International Congress of Mathematics*, pages 23–35, 1962.

[17] E.A. Emerson and E.M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241 – 266, 1982.

[18] E. M. Clarke and R. P. Kurshan. Computer-aided verification. *IEEE Spectrum*, v. 33, 6: 61–67, 1996

[19] L. de Alfaro, M. Faella, T. Henzinger, R. Majumdar and M. Stoelinga. The element of surprise in timed games. In *Proc. of the 14th International Conference on Concurrency Theory*, LNCS, 2003.

[20] Alessandro Ferrante, A. Murano, and Mimmo Parente. Enriched μ-calculi module checking. *Logical Methods in Computer Science (LMCS 2008)*, 4(3:1):1–21, 2008.

[21] E.A. Emerson, Temporal and Modal Logic. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Elsevier Science, pages 995–1072, 1990.

[22] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic. *JACM*, 33(1): 151–178, 1986.

[23] 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.

[24] E. Allen Emerson and A.P. Sistla. Deciding full Branching Time Logic. *Information and Control*, 61(3): 175–201, 1984.

[25] M. Faella, S. La Torre and A. Murano. Dense Real-Time Games. *Proc. of the Seventeenth Annual IEEE Symposium on Logic in Computer Science (LICS02)*, pages 167–176, 2002.

[26] 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.

[27] T. Henzinger and P. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221(1–2):369–392, 1999

[28] O. Kupferman and M.Y. Vardi. Module checking. In *Computer Aided Verification, Proc. Eighth Int. Workshop*, LNCS 1102, pages 75 – 86. Springer-Verlag, 1996.

[29] O. Kupferman and M.Y. Vardi. Module checking. *Information and Computation 164,* 2, 322–344.

[30] 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.

[31] 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.

[32] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th ACM Symposium on Theory of Computing, Dallas*, pages 224–233, 1998.

[33] S. La Torre and M. Napoli. *Finite Automata on Timed ω-Trees*. To appear in *Theoretical Computer Science*.

[34] 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.

[35] Aniello Murano. *Tecision Problems on Tree Automata and Synthesis of Open Timed Systems*. PhD thesis, Universitá degli Studi di Salerno, February 2003. Supervisori: M. Napoli, M.Y.Vardi and S. La Torre.

[36] A. Murano, Margherita Napoli, and Mimmo Parente. Program complexity in hierarchical module checking. In *15th International Conference on Logic for Programming Artificial Intelligence and Reasoning, (LPAR'08)*, LNCS 5330, pages 318–332, 2008.

[37] J. Von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[38] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46 – 77, 1977.

[39] 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.

[40] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1 – 35, 1969.

[41] M.O. Rabin. Automata on infinite objects and Church's problem. *Trans. Amer. Math. Soc.*, 1972.

[42] S. Safra. On the complexity of ω-automata. In *Proc. of the 29th IEEE Symposium on Foundations of Computer Science*, pages 319 – 327, 1988.

[43] 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.

[44] 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.

[45] M.Y. Vardi. Verification of concurrent programs: the automata-theoretic framework. In *Proc. of the Second IEEE Symposium on Logic in Computer Science (LICS87)*, pages 167 – 176, 1987.

[46] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:182 – 211, 1986.

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