Automata-theoretic Decision of Timed Games[☆] **(revised)**

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

[a] *Università degli Studi di Napoli "Federico II", 80126 Napoli, Italy*
[b] *Università degli Studi di Salerno, 84084 Fisciano (SA), Italy*

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.

Keywords: timed games, tree automata, temporal logic

1. Introduction

The theory of games was originally introduced as a theoretical model for economic studies (see for example [39]). In the years, this theory has received an increasing interest by many researchers in computer science. In particular, it has been deeply studied in the context of discrete, timed, and hybrid systems (see [12, 13, 14, 15, 18, 28, 32, 41, 46, 47]). Games provide a suitable framework for program synthesis and for the analysis of *open systems*, that is, systems whose

[☆] A preliminary version of this paper appeared in the Proceedings of the 3rd International Workshop on Verification, Model Checking, and Abstract Interpretation, VMCAI'02 [26]

Email addresses: m.faella@unina.it (Marco Faella), slatorre@unisa.it (Salvatore La Torre), aniello.murano@unina.it (Aniello Murano)

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, 41]. 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*).

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 [2] 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 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

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

is given 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 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 linear- and branching-time winning conditions expressed as LTL [40] and CTL [19] formulas, respectively.

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. For 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 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 [43]). 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 [2]. Then, we construct another tree automaton A_W accepting all the trees satisfying the property 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 that of A^T (i.e. the maximum branching degree over the 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.

For winning conditions given by LTL formulas, our approach yields a decision algorithm running in doubly exponential time. Since LTL games are 2EXPTIME-hard [41] already for discrete game graphs, our result is complete. For winning conditions given by CTL formulas, our approach yields a decision algorithm running in exponential time. Since reachability games for Timed Automata are already EXPTIME-hard [12, 29], we have that our result is complete also for CTL.

In the literature, different formulations of games with winning conditions expressed by temporal logics have been considered. In [4], the *alternating-time temporal logic* ATL is introduced and the related model checking problem (for

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

discrete systems) is shown to be solvable in quadratic time. Recent research has focused on extending ATL to incorporate more powerful strategic constructs, but often this results in a much more complex model checking problem that becomes even non-elementary in some cases (see for example [4, 20, 21, 36, 37]). In *module checking* [30], a system interacts with its environment, and while the system is forced to consider all possible successors, the environment can select a non-empty subset of the them. The module checking problem for finite-state systems has been deeply investigated and turns out to be EXPTIME-compete for specification given as CTL formulas. This framework has been extended to pushdown systems in [17, 9]. In the setting of pushdown systems, also *modular strategies* with respect to several winning conditions, and in particular LTL specifications, have been considered [6, 7]. Games with winning conditions expressed as parametric LTL [3] have been recently studied [51]. *Rectangular hybrid games* with winning conditions expressed by LTL formulas were solved in [28]. The results from [28] 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. Other research on timed games has concerned: timed games with TCTL [1, 33] winning conditions [27]; optimal time control synthesis for timed automata [11]; optimal cost control synthesis [32] for weighted timed automata [8].

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 4, we introduce winning conditions expressed in the branching–time temporal logic CTL. Then, in Section 5, we solve CTL timed games. In particular, we discuss the construction of a tree automaton accepting all strategies of the protagonist in a timed game. In Section 6, we introduce and study timed games whose winning condition is given in the linear–time temporal logic LTL. Finally, in Section 7, we provide some concluding remarks. In particular, we discuss a comparison with ATL and module checking.

2. Preliminaries

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

Words. 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 fix a finite alphabet Σ.

Trees. An ω-tree $\mathcal{T}$ (a *tree*, for short) is (v_{ini}, V, E) such that V is a set of nodes, $v_{ini} \in V$ is the root, and $E \subseteq V \times V$ is a set of edges such that:

1. v_{ini} has no *predecessors*, i.e, $(u, v_{ini}) \notin E$ for all $u \in V$;

2. every other node has exactly one *predecessor*, i.e., for each $v \in V \setminus \{v_{ini}\}$ there is exactly one $u \in V$ such that $(u, v) \in E$;
3. every node is *reachable* from the root through a finite sequence of edges, i.e., for each $v \in V$ there is a sequence of nodes $v_0 \dots v_m$ such that $v_0 = v_{ini}$, $v_m = v$ and $(v_i, v_{i+1}) \in E$ for $i = 0, \dots, m$ (*a path from* v_{ini} *to* v);
4. every node has at least a *successor*, i.e., for each $u \in V$ there is $v \in V$ such that $(u, v) \in E$.

In the following, we fix a tree $\mathcal{T} = (v_{ini}, V, E)$.

For all nodes $u \in V$ we set $succ(\mathcal{T}, u)$ to be the set of successors of u in $\mathcal{T}$, that is $succ(\mathcal{T}, u) = \{v \in V \mid (u, v) \in E\}$. If a node 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 tree is the maximum branching degree over all its nodes.

A *path* in $\mathcal{T}$ is a (possibly finite) sequence of nodes $\pi = v_0 v_1 \dots$ such that, for all $i \geq 0$, $(v_i, v_{i+1}) \in E$. Moreover, π is *rooted* if $v_0 = v_{ini}$.

For a tree $\mathcal{T}$, the *subtree* of $\mathcal{T}$ rooted at a node $u \in V$ is the tree (u, V', E') where $V' \subseteq V$ is the set of all the nodes that are reachable from u and E' is $E \cap (V' \times V')$.

A Σ-valued tree T is $(\mu, \mathcal{T})$ where $\mu : V \to \Sigma$ is the labeling function. All of the above definitions naturally extend to Σ-valued trees, and thus we omit to state them. In addition, for a path $\pi = v_0 v_1 \dots$, we denote by $\mu(\pi)$ the sequence $\mu(v_0)\mu(v_1) \dots$.

Automata. A *transition table* A over the alphabet Σ is a tuple $(\Sigma, S, \Delta, s_{ini})$, where Q is a finite set of *states*, $\Delta \subseteq S \times \Sigma \times S$ is the *transition relation*, and $s_{ini} \in S$ is the *initial state*. A *run* of A on an ω-word $\sigma_0 \sigma_1 \dots$ over Σ is an infinite sequence of states $s_0 s_1 \dots$ such that $s_0 = s_{ini}$ and $(s_i, \sigma_i, s_{i+1}) \in \Delta$ for all $i \geq 0$.

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

An *automaton* is a (tree) transition table A paired with 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 states, we denote by $Inf(\rho)$ the set of states 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 rooted 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. The set of words (resp., trees) that are accepted by an automaton (resp., tree automaton) A is denoted by $L(A)$. In these regards, we treat a (tree) transition table A as a Büchi (tree) automaton whose acceptance condition F is the set of all states from A, and thus any run is an accepting one.

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 ([25, 31]). *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 denote respectively the number of states and the branching degree of A.*

Proposition 2 ([42, 50]). *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 denotes the number of states in A.*

3. Timed Games

3.1. 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 [2]. 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 $\Xi(C)$ is inductively defined as the minimal set containing:

- $x \leq y + c$, $x \geq y + c$, $x \leq c$, and $x \geq c$, where $x, y \in C$ and c is a natural number;
- $\delta_1 \wedge \delta_2$, where $\delta_1, \delta_2 \in \Xi(C)$.

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 $q \in Q$ and ν is a clock valuation satisfying $inv(q)$. We denote by S the set of all states. 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 a transition table with the infinite set of states S and the infinite alphabet $\mathbb{R}_{\geq 0} \times Acts$. In order to define the transitions, we first introduce the concepts of discrete and time step. 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 $0 \leq d' < d$. 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 $\langle q_i, \nu_i \rangle \xrightarrow{d_i,\sigma_i} \langle q_{i+1}, \nu_{i+1} \rangle$, for all $i = 0, \ldots, n-1$. 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 into a finite number of equivalence classes called *regions*. A region is defined by a location and a *clock region*, which is a set of equivalent clock valuations. We denote with c_x

[3] Notice that we disallow zero-time steps which are not needed in our game formulation.

the largest constant in clock constraints involving the clock variable x, and for $d \in \mathbb{R}_{\geq 0}$, with $frac(d)$ the fractional part of d and with $\lfloor d \rfloor$ its integral part. The equivalence of clock valuations is formally defined as follows. Two clock valuations ν and ν' are *(region) equivalent* if the following conditions are met [2]:

- 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 $frac(\nu'(x)) = 0$.

A *clock region* is an equivalence class of the above equivalence relation. We denote by $[\nu]$ 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 $\{q\} \times \alpha$. A region $r' = \langle q, \alpha' \rangle$ is a *time-successor* of a region $r = \langle q, \alpha \rangle$, denoted $r \preceq r'$, if for each $\nu \in \alpha$ there is a $d \in \mathbb{R}_{\geq 0}$ such that $\nu + d \in \alpha'$. We use *Regs* to denote the set of regions of $\mathcal{G}$ and extend the notion of step from states to regions as follows. Given two regions r and r', and an action $\sigma \in Acts$, we write $r \xrightarrow{\sigma} r'$ if there exist $s \in r$, $s' \in r'$ and $d > 0$ such that $s \xrightarrow{d,\sigma} s'$. Also, for a run $\rho = \langle q_0, \nu_0 \rangle \langle q_1, \nu_1 \rangle \ldots$, with $[\rho]$ we denote the sequence $\langle q_0, [\nu_0] \rangle \langle q_1, [\nu_1] \rangle \ldots$.

3.2. 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.

Intuitively, a *play* of a timed game is constructed as follows. 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, the latter is allowed to change the previously declared move.

We use the alphabet symbols (actions) of the timed graph to represent 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 a special action ξ. In the following, we denote with *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 modeled as a run of the corresponding timed graph.

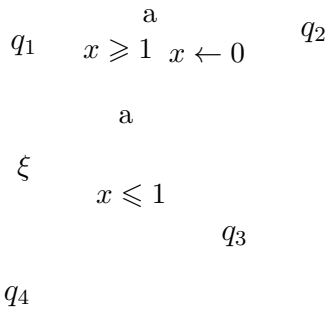

Figure 1: A fragment of a timed game.

As instance, consider 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 that 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 unit. Since at time $x = 1$ there are two a-moves which are enabled, the antagonist can choose which one is to be taken and the game will proceed either to location q_2 or to location q_3. Moreover, from q_1 the antagonist can also move on his own, by taking a ξ-move before the move of the protagonist. In that case, according to the above strategy, the game will proceed to location q_4, with $x \leq 1$.

In the following, if not differently specified, we always refer to a fixed timed graph $\mathcal{G} = (Q, Acts^{\xi}, q_{ini}, C, \Delta, inv)$. Without loss of generality, we assume that a move of the protagonist is always defined in each state s of $\mathcal{G}$, i.e., for each s there is always at least a step of the form $s \xrightarrow{d,\sigma} s'$ with $\sigma \in Acts$.

A *strategy* is a function $\mathcal{F} : runs \longrightarrow \mathbb{R}_{>0} \times Acts$ that assigns to each run a move of the protagonist which is allowed by the underlying timed graph, that is for each run $\rho = s_0 s_1 \dots s_n$ of $\mathcal{G}$, if $\mathcal{F}(\rho) = (d, \sigma)$ then there is a step $s_n \xrightarrow{d,\sigma} s$ in $\mathcal{G}$.

A play $\rho = s_0 s_1 \dots s_n \dots$ is *consistent with* a strategy $\mathcal{F}$ if for each $n \geq 0$, denoting $\mathcal{F}(\rho_{\leq n}) = (d_n, \sigma_n)$, either $s_n \xrightarrow{d_n,\sigma_n} s_{n+1}$ or $s_n \xrightarrow{d',\xi} s_{n+1}$ with $d' < d_n$. The set of all infinite plays which are consistent with $\mathcal{F}$ is denoted with $plays^{\omega}_{\mathcal{F}}$, and the set of their finite prefixes with $plays_{\mathcal{F}}$.

For a given a strategy $\mathcal{F}$, we can collect all runs that are consistent with it in an infinite tree $T_{\mathcal{F}}$ called the *strategy tree* of $\mathcal{F}$. Moreover, we define an auxiliary labeling $act_{\mathcal{F}}$ which labels each node of $T_{\mathcal{F}}$ with the action of the step taken to reach the corresponding state from the state labeling the parent node (in other words, this exposes on each path the actions taken in the corresponding play).

Formally, the strategy tree $T_{\mathcal{F}}$ is an S-labeled tree $(\mu, \mathcal{T}_{\mathcal{F}})$, where $\mathcal{T}_{\mathcal{F}} = (v_{ini}, V, E)$, inductively defined as follows. For the root v_{ini}, we define $\mu(v_{ini}) =$

$\langle q_{ini}, \nu_{ini}\rangle$ and $act_{\mathcal{F}}(v_{ini})$ is arbitrarily set to any element of $Acts^{\xi}$.

Now, let $\pi = u_0 u_1 \dots u_n$ be a finite rooted path in $T_{\mathcal{F}}$, $\mu(u_n) = s$ and $\mathcal{F}(\mu(\pi)) = (d, \sigma)$. 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 $0 < d' < d\} \setminus S^{\text{Prot}}$. For each state which is either in S^{Prot} or in S^{Ant}, we add a corresponding child of u_n in $T_{\mathcal{F}}$, that is, for $x \in \{\text{Prot}, \text{Ant}\}$ and $s' \in S^x$, we add a child $v \in V$ of u_n with labeling $\mu(v) = s'$, and action $act_{\mathcal{F}}(v) = \sigma$, if $x = \text{Prot}$, and $act_{\mathcal{F}}(v) = \xi$, otherwise. We denote $succ^x(T_{\mathcal{F}}, u_n)$ the set of all children corresponding to S^x states.

We remark that from the definition of S^{Prot} and S^{Ant}, the states associated to the children of u_n are all distinct. Also, S^{Prot} is the set of states that can be reached 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 can be reached if the antagonist moves before the protagonist and that are not in S^{Prot}. Observe that there can be states that are reached 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}.

By $untime(T_{\mathcal{F}})$ we denote the tree obtained from $T_{\mathcal{F}}$ by projecting out the time valuation from the states labeling it. Formally, $untime(T_{\mathcal{F}}) = (\mu', \mathcal{T}_{\mathcal{F}})$, where $\mu'(u) = \{q\}$ if and only if $\mu(u) = \langle q, \nu\rangle$ for some clock valuation ν. By construction, $plays^{\omega}_{\mathcal{F}} = \{\mu(\pi) \mid \pi$ is a rooted infinite path in $T_{\mathcal{F}}\}$ and 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 after equivalent histories.

3.3. Counting the number of region-equivalent successors in a strategy tree

In a strategy tree, the number of successors of a node that are labeled with states from a given region r, if any, depends only on the features of r and the transitions of the timed graph. We define the function *split* that assigns a value to any two regions r, r' and a symbol σ. We will prove that indeed this function exactly gives the number of states of r' that are reachable from any state in r in a single step over σ. We start giving some additional notation.

A clock constraint is *punctual* if it is of the form $x = c$, for a natural number $c > 0$. A guard is *punctual* if it contains a punctual clock constraint (among its conjuncts). Given a timed graph, an edge is *punctual* if it has a punctual guard.

A clock region α is a *point* if it contains just one clock valuation assigning to all clocks an integral value (i.e., for all clocks x, $frac(x) = 0$). Intuitively, in a two-clock system, α is a point if it belongs to $\mathbb{N}_0 \times \mathbb{N}_0$. A clock region α is *punctual* if it is a point or it has a punctual clock constraint. Intuitively, in a two-clock system, α is punctual if either it belongs to $\mathbb{N}_0 \times \mathbb{N}_0$ or it is a line that is parallel to an axis and does not lie on it. A clock region is a *boundary* region if it has a clock constraint of the form $x = c$ for $c \geq 0$ (i.e. it is either punctual or has a clock constraint of the form $x = 0$). An *open* clock region is any clock region that is not boundary. The definitions of point and punctual clock regions apply to regions in the obvious way.

Definition 1. *Let $\mathcal{G}$ be a timed graph, r, r' be two regions of $\mathcal{G}$ and $\sigma \in Acts^{\xi}$. If r' is not reachable from r through a transition over σ, then $split(r, \sigma, r') = 0$. Otherwise, one of the following cases holds:*

1. *if r' is punctual, then $split(r, \sigma, r') = 1$;*
2. *if r' open, then $split(r, \sigma, r') = \infty$;*
3. *otherwise (i.e., r' is a non-punctual boundary region) we distinguish between the following two sub-cases:*
   (a) *if there exists an edge in $\mathcal{G}$ that is not punctual and can move from r to r' in one transition over σ, then $split(r, \sigma, r') = \infty$;*
   (b) *otherwise (i.e., all edges in $\mathcal{G}$ moving from r to r' over σ are punctual), let $e_1, \ldots, e_l$ be all the edges in $\mathcal{G}$ that can be taken from r to r' over σ. For each $1 \leq i \leq l$, let g_i be the guard of e_i, $x_i = c_i$ be one of the punctual clock constraints in g_i and X be the set of all such clocks x_i. Then, $split(r, \sigma, r')$ equals to the number of clocks in X having different fractional parts in the clock valuations of r (i.e., $|\{frac(\nu(x)) \mid x \in X\}|$ for any state $\langle q, \nu \rangle$ of r).*

The last case is clearly the most complex one and deserves some further explanation. First, from each guard g_i we pick an arbitrary punctual clock constraint $x_i = c_i$. Notice that if g_i contains any other punctual clock constraint, say $y = d$, it holds that x_i and y have the same fractional part in r, otherwise e_i could not be enabled in a time-successor of r. Therefore, it is enough to insert either x_i or y into X.

Second, we pick an arbitrary state $\langle q, \nu \rangle \in r$. Recall that a clock region fixes an ordering between the fractional parts of all clocks x whose value is less than c_x. In particular, the equivalence relation distinguishes between the case $frac(x) < frac(y)$ and $frac(x) = frac(y)$, for all clocks x, y. Hence, the number of clocks having different fractional parts is the same for all $\langle q, \nu \rangle \in r$.

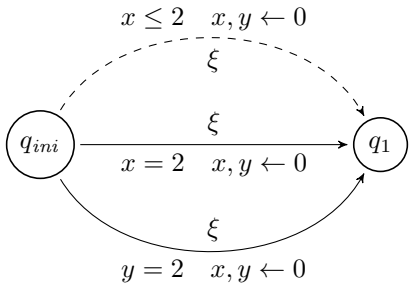

Figure 2: A game showing different cases in the evaluation of the function *split* (see Example 1).

Let us consider some examples where one can retrieve the bounds we have introduced in the definition above.

Example 1. *Consider the 3-clock timed graph in Figure 2, including the dashed edge. Let* $r = \langle q_{ini}, \alpha \rangle$ *where* α *is such that* $x, y, z \in (0, 1)$ *and* $frac(x) < frac(y) < frac(z)$*, and* $r' = \langle q_1, \alpha' \rangle$ *where* α' *is such that* $x = y = 0$ *and* $z \in (1, 2)$*. Notice that* r' *is not punctual but it does have integral-valued clocks, ruling out cases* 1 *and* 2 *of the definition of split. The dashed edge, with guard* $x \leq 2$*, is not punctual, thus we are in case* 3(a) *of the definition and thus* $split(r, \xi, r') = \infty$*. Indeed, from any state in* r *there are infinitely many delays for which the dashed edge can be taken, all leading to different states in* r'*, each with a different value of the clock* z*.*

Next, assume that the dashed edge is removed from the graph. Now, all the remaining edges from r *to* r' *are punctual, so we end up in case* 3(b) *of the definition of split. The set of clocks that are subject to a punctual constraint is* $X = \{x, y\}$*. Since* x *and* y *have different fractional part in* r*, we obtain* $split(r, \xi, r') = 2$*. Indeed, from any state in* r *there exist a delay which enables the edge with guard* $x = 2$ *and another delay which enables the edge with guard* $y = 2$*. These delays lead to two states in* r' *which differ in the value of the clock* z*.*

Finally, consider the region $r'' = \langle q_{ini}, \alpha'' \rangle$*, where* α'' *is the clock region such that* $x, y, z \in (0, 1)$ *and* $frac(x) = frac(y) < frac(z)$*. When evaluating* $split(r'', \xi, r')$*, we fall in case* 3(b) *again, and as before we have* $X = \{x, y\}$*, but this time* x *and* y *have the same fractional part in* r''*. Hence, we obtain* $split(r'', \xi, r') = 1$*. Indeed, from any state in* r *there exists a unique delay which enables both edges, leading to a unique state in* r'*.*

The following result states the main property of the *split* function.

Lemma 1. *Starting from a given state of a region r of a timed graph $\mathcal{G}$, for each region r' of $\mathcal{G}$, it holds that the maximal number of different states of r' which are reachable through a single step over a symbol $\sigma \in Acts^{\xi}$ is equal to $split(r, \sigma, r')$.*

Proof : Let $s = \langle q, \nu \rangle$, $r = \langle q, \alpha \rangle$ and $r' = \langle q', \alpha' \rangle$. We recall that a state s' in r' is reachable from s in one step over σ if there exists a step $s \xrightarrow{d,\sigma} s'$, for a given delay d. Note that the number of different states s' of r' that can be reached from s in one step over σ does not depend on the particular state s we pick in r, therefore let denote this number with $\sharp_{r,\sigma,r'}$. We show that $\sharp_{r,\sigma,r'} = split(r, \sigma, r')$ by case inspection.

First observe that in $\mathcal{G}$ if the region r' is not reachable from r over σ, clearly $\sharp_{r,\sigma,r'} = 0$, which matches the definition of $split(r, \sigma, r')$ in this case. In all the remaining cases, there is always at least a state of r' which is reachable in one step over σ, and denote with $\nu' \in \alpha'$ a clock valuation such that $s \xrightarrow{d,\sigma} \langle q', \nu' \rangle$ for some $d > 0$. We distinguish among the following cases.

- r' is a point. Thus, it contains only the state $\langle q', \nu' \rangle$ and for all clocks x, $frac(\nu'(x)) = 0$ must hold (i.e., ν' assigns an integer to x). Therefore, $\sharp_{r,\sigma,r'} = 1$ and thus $\sharp_{r,\sigma,r'} = split(r, \sigma, r')$ holds.

- r' is punctual but not a point. In this case, ν' evaluates x to an integer $c > 0$ and hence x cannot be reset in any transition from r to r'. Thus, starting from a state s in ν there is only one possible delay d for the step from s to a state in r', namely $d = c - \nu(x)$. Indeed, any other delay $d' \neq d$ would lead to a point with $x \neq c$ and thus outside r'. So, there is only one reachable state in r' from any state of r. Therefore, also in this case we have $\sharp_{r,\sigma,r'} = 1$ and thus $\sharp_{r,\sigma,r'} = split(r, \sigma, r')$ holds.

- r' is an open region. Let x be the clock with the greatest fractional part according to ν', i.e., $frac(\nu'(x)) \geq frac(\nu'(y))$, for all clocks y in $\mathcal{G}$. Clearly, $0 < frac(\nu'(x)) < 1$. Now, consider the set of states $H = \{s'' \mid \langle q', \nu' + d \rangle$, with $d \in \mathbb{R}$ and $0 < d < 1 - frac(\nu(x))\}$. The set H is infinite and contained in r', and all the states it contains are reachable from s with a different delay. Therefore, $\sharp_{r,\sigma,r'} = \infty$ which equals $split(r, \sigma, r')$.

- r' is a non-punctual boundary region (i.e., it is not a point and for all the clock constraints of the form $x = c$, c is 0) and there exists a non punctual edge of $\mathcal{G}$ that can be taken to get from s to a state in r' in one step. This case is very similar to the previous one, except that s has a clock that has been reset while moving from q to q' (recall that the delay d must be greater than 0). This means that by taking the set H from the previous case and setting to 0 all clocks y such that $\nu'(y) = 0$, we obtain a set of states H' that is still infinite (because at least one clock is not zero in r') and is contained in r'. Thus, also in this case $\sharp_{r,\sigma,r'} = \infty$ which equals $split(r, \sigma, r')$.

- r' is a non-punctual boundary region and all the edges of $\mathcal{G}$ that can be taken to get from r to r' (in one step over σ) are punctual. As r' is not punctual, for all clocks x either $\nu'(x) = 0$ or $\mathit{frac}(\nu'(x)) \neq 0$ depending on whether x is reset or not on the transition. In particular, ν' evaluates at least one clock x to zero and at least one clock z to a positive value. Moreover, since the guards are punctual, all the clocks x such that $x = c$ is a conjunct of the guard must be reset on the transition.

  Let $e_1, \ldots, e_l$ be the punctual edges in $\mathcal{G}$ that allow moving from s to any state in r' over σ. For each $1 \leq i \leq l$, let g_i the guard in e_i and x_i be such that $x_i = c_i$ is a clock constraint in g_i. By taking the one such edge e_j, for the punctual clock constraint $x_j = c_j$ in g_j, there is only one possible delay, namely $d_j = c_j - \nu(x_j)$, to trigger the transition and thus only one state in r' can be reached through e_j. If we use any other edge e_t, with $t \neq j$, and $\mathit{frac}(\nu(x_j)) \neq \mathit{frac}(\nu(x_t))$, a simple calculation shows that $d_t \neq d_j$. In general, let X be the set of the punctual clocks x_i in g_i and $k = |\{\mathit{frac}(\nu(x)) \mid x \in X\}|$. We can take k different delays to move from s to a state in r' taking a transition over σ, which lead to k different states (since the clock z is not reset by any edge $e_1, \ldots, e_l$). Therefore, $\sharp_{r,\sigma,r'}$ is exactly k, and also in this case $\sharp_{r,\sigma,r'} = \mathit{split}(r, \sigma, r')$, which concludes the proof.

□

4. Branching-time Timed Games

4.1. Computation Tree Logic

CTL was introduced by Emerson and Clarke [19] 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 *tree* $T = (\mu, (v_{ini}, V, E))$. Given a CTL formula φ and a node $v \in V$, the satisfaction relation $(T, v) \models \varphi$, meaning that φ is true in G at v, is defined inductively as follows ($p \in AP$):

- $(T, v) \models p$ if and only if $p \in \mu(v)$;
- $(T, v) \models \neg p$ if and only if $(T, v) \not\models p$;
- $(T, v) \models \varphi_1 \wedge \varphi_2$ if and only if $(T, v) \models \varphi_1$ and $(T, v) \models \varphi_2$;

- $(T, v) \models \varphi_1 \vee \varphi_2$ if and only if $(T, v) \models \varphi_1$ or $(T, v) \models \varphi_2$;
- $(T, v) \models \exists X\varphi$ if and only if there exists a $v' \in V$ such that $(v, v') \in E$ and $(T, v') \models \varphi$;
- $(T, v) \models \forall X\varphi$ if and only if $(T, v') \models \varphi$ for all $v' \in V$ such that $(v, v') \in E$;
- $(T, v) \models \exists(\varphi_1 \,\mathcal{U}\, \varphi_2)$ if and only if there exists an infinite path $v_0v_1\ldots$, with $v = v_0$, such that for some $i \geq 0$ $(T, v_i) \models \varphi_2$, and $(T, v_j) \models \varphi_1$ for all $0 \leq j < i$;
- $(T, v) \models \forall(\varphi_1 \,\mathcal{U}\, \varphi_2)$ if and only if for all infinite paths $v_0v_1\ldots$, with $v = v_0$, there exists $i \geq 0$ such that $(T, v_i) \models \varphi_2$, and $(T, v_j) \models \varphi_1$ for all $0 \leq j < i$;
- $(T, v) \models \exists(\varphi_1 \;\mathcal{R}\; \varphi_2)$ if and only if there exists an infinite path $v_0v_1\ldots$, with $v = v_0$, such that for all $i \geq 0$, if $(T, v_i) \not\models \varphi_2$ then there exists $0 \leq j < i$ such that $(T, v_j) \models \varphi_1$;
- $(T, v) \models \forall(\varphi_1 \;\mathcal{R}\; \varphi_2)$ if and only if for all infinite paths $v_0v_1\ldots$, with $v = v_0$, and for all $i \geq 0$, if $(T, v_i) \not\models \varphi_2$ then there exists $0 \leq j < i$ such that $(T, v_j) \models \varphi_1$.

The usual abbreviation $\Diamond\varphi$ stands for $\mathit{true}\,\mathcal{U}\varphi$. To denote that T is a model of φ, i.e. $(T, v_{ini}) \models \varphi$, we also write $T \models \varphi$. It is known that, given a branching degree k, it is possible to characterize all the trees of degree k that satisfy a CTL formula by a Büchi tree automaton, as reported in the following proposition.

Proposition 3 ([49]). *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.*

CTL Timed Games. A CTL timed game (CTL game, for short) is a timed game $(\mathcal{G}, \varphi)$ where the winning condition is expressed by a CTL formula φ which uses as atomic propositions the locations of $\mathcal{G}$. We say that a strategy $\mathcal{F}$ is *winning* in a CTL game $(\mathcal{G}, \varphi)$ if $\mathit{untime}(T_\mathcal{F}) \models \varphi$.

Decision problem: Given a CTL game $(\mathcal{G}, \varphi)$, we wish to determine whether there exists a winning strategy.

We will address this decision problem in Section 5. In the rest of this section we will show some properties of the winning strategies for CTL games.

4.2. Splitting and Memory Requirements in Branching-time Timed Games

The ability of strategies to be splitting is a key argument in branching games. Indeed, we show in the following lemma that splitting strategies are needed for winning CTL games.

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

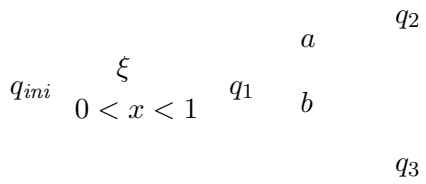

Figure 3: A game where region stable strategies are not sufficient to win.

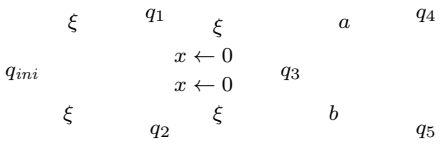

Figure 4: A game where memory is needed.

Proof : Consider the game in Figure 3, 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. □

Another crucial property is that in CTL games it is not sufficient for a player to use *memoryless strategies* in order to win. A strategy is memoryless if it only depends on the last state of each play.

Lemma 3. *Memory is needed to win* CTL *games.*

Proof : Figure 4 gives an example of a timed game where all winning strategies need memory. The timed graph has only the clock x that is reset on the transitions that lead to q_3. Consider the winning condition $\exists\Diamond q_4 \wedge \exists\Diamond q_5$. Once the game reaches q_3, the value of x is always 0 and thus any memoryless strategy can only choose either move a or move b, independently of the play. Clearly, either moves will not suffice to satisfy the winning condition. In fact, along all plays which are consistent with the resulting memoryless strategy will lead either all to q_4 or all to q_5. On the other hand, a winning strategy exists. For example, one can issue a if q_3 is reached through q_2, and b otherwise. □

5. Solving CTL Timed Games

In this section, we present our solution to CTL timed games, i.e., an algorithm to determine whether there exists a winning strategy of the protagonist, given a CTL timed game.

We follow the automata-theoretic approach. In particular, we build an automaton that accepts a tree if and only if there is a winning strategy of the protagonist in the game. From Proposition 3, we can build an automaton accepting all the trees (with a given branching degree) that satisfy a given CTL formula, thus we just need to capture with trees of bounded branching degree all the possible strategies.

The tree counterpart of strategies in timed games, i.e., the strategy trees, have a continuous branching degree. The crucial step in our approach is to show that indeed for each strategy $\mathcal{F}$ of a CTL timed game, we define a corresponding discrete tree $t_{\mathcal{F}}$ of bounded branching-degree, 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 the strategy tree $T_{\mathcal{F}}$ which "samples" paths which are meaningful for the fulfillment of the winning condition. In particular, such discrete tree preserves all the moves taken by the protagonist, as well as all the 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 bounded number 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). Moreover, as we 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 in the formula. Accordingly, we define an automaton $A_k^{\mathcal{G}}$ accepting all trees $t_{\mathcal{F}}$ corresponding to k-splitting strategies.

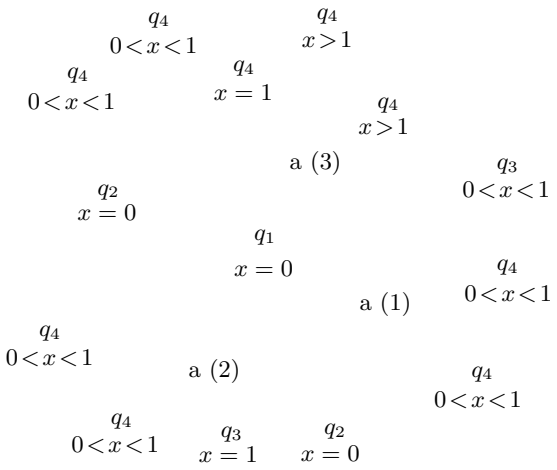

Figure 5: Transitions of $A_2^{\mathcal{G}}$ from state $\langle 1, x=0\rangle$, for the timed graph $\mathcal{G}$ of Figure 1. Edges that are linked by an arc belong to the same tree transition.

As an example, consider the timed graph in Figure 1, and suppose we want to accept all trees corresponding to 2-splitting strategies. A fragment of the corresponding tree automaton $A_2^{\mathcal{G}}$ is depicted in Figure 5. 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-splitting strategies, the automaton keeps two copies for each non-punctual region of the antagonist. Therefore, the automaton contains a transition with three destinations (case (1) in Figure 5): 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 $x = 1$ (case (2) in Figure 5) or $x > 1$ (case (3) in Figure 5).

5.1. Strategy Samples

Consider a strategy $\mathcal{F}$ and let $T_{\mathcal{F}} = (\mu, \mathcal{T}_{\mathcal{F}})$, where $\mathcal{T}_{\mathcal{F}} = (v_{ini}, V, E)$, be the corresponding strategy tree. For an integer $k > 0$, we define a class of discrete trees (called *k-samples of* $\mathcal{F}$) associated with $\mathcal{F}$. As an intermediate step, we define a *pre-k-sample* to be a suitable subtree of $T_{\mathcal{F}}$. A k-sample is then 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 $\mathcal{F}$. We now provide the definition of a pre-k-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 $T_{\mathcal{F}}$. For $k > 0$, a pre-k-sample of $\mathcal{F}$ is a tree $T = (\mu', \mathcal{T})$ with $\mathcal{T} = (v_{ini}, V', E')$ where μ' is the restriction of μ to V', $E' = E \cap (V')^2$ and V' is a minimal subset of V such that:

- $v_{ini} \in V'$;
- For each $u \in V'$:
  - for each $v \in succ^{\text{Prot}}(T_{\mathcal{F}}, u)$, we have $v \in V'$;
  - for each region r such that there exists $v \in succ^{\text{Ant}}(T_{\mathcal{F}}, u)$ with $\mu(v) \in r$, let $k' = \min\{k, split([\mu(u)], \xi, r)\}$; there exist k' distinct nodes $v_1, \ldots, v_{k'} \in succ^{\text{Ant}}(T_{\mathcal{F}}, u)$ such that, for all $i = 1 \ldots k'$, $\mu(v_i) \in r$ and $v_i \in V'$.

Notice that the existence of such k' distinct nodes in $succ^{\text{Ant}}(T_{\mathcal{F}}, u)$ is ensured by Lemma 1.

For instance, consider again the game in Figure 3, 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 6 shows two 2-samples of $\mathcal{F}$.

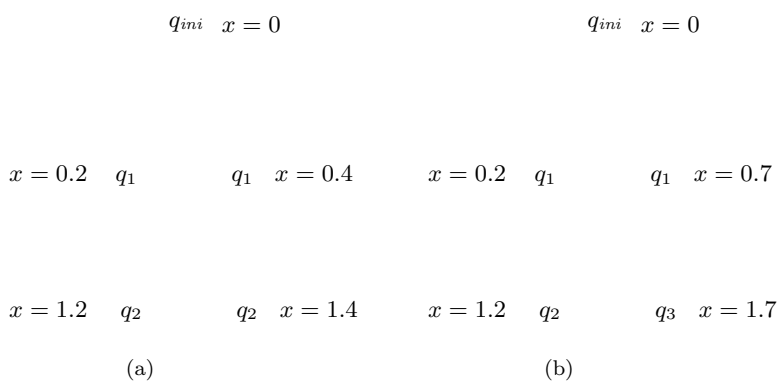

Figure 6: Two samples obtained from the game of Figure 3 and the strategy that, from q_1, picks move $(1, a)$ when $x < 0.5$ and move $(1, b)$ otherwise.

5.2. The Automaton $A_k^{\mathcal{G}}$

For an integer $k > 0$, we now define the tree automaton $A_k^{\mathcal{G}}$, whose task is to accept all and only the k-samples corresponding to some strategy in $\mathcal{G}$. The locations of $A_k^{\mathcal{G}}$ are the regions of the timed graph $\mathcal{G}$, while its labels are the 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 the set of all the tuples defined as follows.

Consider a $A_k^{\mathcal{G}}$ location $r = \langle q, \alpha \rangle \in \mathit{Regs}$. There exists a region z which is a time-successor of r and such that there is at least a σ-transition enabled from z, with $\sigma \in \mathit{Acts}$ (recall our assumption that in any state at least a move of the protagonist is enabled). For any such $\sigma \in \mathit{Acts}$, let $\mathit{Regs}_{z,\sigma} = \{\, r' \mid z \xrightarrow{\sigma} r' \,\}$ and $\mathit{Regs}_{r,z,\xi} = \{\, r' \mid z' \xrightarrow{\xi} r' \text{ and } r \preceq z' \preceq z \,\}$. We number the elements of these sets such that $\mathit{Regs}_{z,\sigma} = \{r'_1, \ldots, r'_m\}$ and $\mathit{Regs}_{r,z,\xi} = \{r'_{m+1}, \ldots, r'_{m+h}\}$, and for all $i = m+1, \ldots, m+h$, let $k_i = \min\{k, \mathit{split}(r, \xi, r'_i)\}$. Then, the following transition belongs to Δ:

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

In the following, we refer to $r'_1, \ldots, r'_m$ as the $\mathit{Regs}_{z,\sigma}$ part of the above transition. Analogously, what follows on the right is referred to as the $\mathit{Regs}_{r,z,\xi}$ part.

Note that the branching degree of $A_k^{\mathcal{G}}$ is linear in the product of the maximum branching degree of $\mathcal{G}$, $\max\{\, c_x \mid x \in C \,\}$, and k. From [2], 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 4. *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 6(a). It satisfies the formula $\forall X \forall X q_2$, while the strategy from which the sample is derived does not. The problem is that the strategy has extra behaviors that are not present in the sample in Figure 6(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 sample, but first we recall the notions of bisimilarity and isomorphism on trees.

For $i \in \{1, 2\}$, let $T_i = (\mu_i, (root_i, V_i, E_i))$ be a Σ-valued tree. T_1 and T_2 are *isomorphic* if they are equal modulo a renaming of their vertices. 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 $v_1 \in succ(T_1, u_1)$, there exists $v_2 \in succ(T_2, u_2)$ such that $(v_1, v_2) \in R$, and
3. for all $v_2 \in succ(T_2, u_2)$, there exists $v_1 \in succ(T_1, u_1)$ such that $(v_1, v_2) \in R$.

We say that T_1 and T_2 are *bisimilar* if there exists a bisimulation R such that $(root_1, root_2) \in R$.

Given a tree T, we say that a strategy $\mathcal{F}$ is a *canonical strategy for* T if *(i)* T is isomorphic 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 5. *There exists a canonical strategy for each* $T \in L(A_k^{\mathcal{G}})$.

Proof : Let $A_k^{\mathcal{G}} = (2^Q, Regs, \Delta, r_{ini})$ and $T \in L(A_k^{\mathcal{G}})$. We first introduce some notation, then we inductively construct a strategy $\mathcal{F}$ using a notion of consistency. This notion of consistency is also the key to show that indeed such $\mathcal{F}$ is canonical, which concludes the proof.

Preliminary notation. Fix $T = (\mu_Q, \mathcal{T})$ with $\mathcal{T} = (v'_{ini}, V', E')$ and an accepting run $(\mu_R, \mathcal{T})$ of $A_k^{\mathcal{G}}$ on T. From the definition of run and the construction of $A_k^{\mathcal{G}}$, we derive the following notation.

Recall that for each $u \in V'$, $(\mu_R(u), \{q\}, \mu_R(v_1), \ldots, \mu_R(v_c)) \in \Delta$ where $\mu_R(u) = \langle q, \alpha \rangle$ and $succ(T, u) = \{v_1, \ldots, v_c\}$. Now, let $\sigma^u, z^u, Regs_\sigma^u$ and $Regs_\xi^u$ be respectively the action $\sigma \in Acts$, the region z, and the sets of regions $Regs_{z,\sigma}$ and $Regs_{r,z,\xi}$ used in the definition of the above $A_k^{\mathcal{G}}$ transition.

For each $u \in V'$, we set $succ(T, u, \sigma) = \{v \in succ(T, u) \mid v$ corresponds to a position in the $Regs_\sigma^u$ part of the transition from $u\}$, and for $r \in Regs_\xi^u$, $succ(T, u, \xi, r) = \{v \in succ(T, u) \mid v$ corresponds to a position in the $Regs_\xi^u$ part of the transition from u which assigns $r\}$. Thus, the sets $succ(T, u, \sigma)$ and $succ(T, u, \xi, r)$ for $r \in Regs_\xi^u$ form a partition of $succ(T, u)$.

Strategy $\mathcal{F}$ and consistency. We start giving a labeling μ_S of T that assigns to each node a state of $\mathcal{G}$ such that the sequence of labels of each path of T is a run of $\mathcal{G}$. Then, we introduce a partition of the regions r among each nonempty set of nodes $succ(T, u, \xi, r)$. All this is used to define inductively a notion of consistency for the plays of $\mathcal{G}$ with respect to a path in T and the strategy $\mathcal{F}$.

From the definition of run and the construction of $A_k^{\mathcal{G}}$, we can define the labeling function μ_S of $\mathcal{T}$ such that for each $u \in V'$:

- $\mu_S(u) \in \mu_R(u)$;
- there exists a delay $d > 0$ such that for each $v \in succ(T, u, \sigma^u)$, $\mu_S(u) \xrightarrow{d,\sigma^u} \mu_S(v)$; we denote such d as d^u in the following;
- for each $v \in succ(T, u) \setminus succ(T, u, \sigma^u)$, there exists a delay d' with $0 < d' < d^u$ such that $\mu_S(u) \xrightarrow{d',\xi} \mu_S(v)$.

From the definition of region, it is simple to verify that for each path $\pi = u_1 \dots u_n \dots$ of T, the sequence $\mu_S(\pi) = \mu_S(u_1) \dots \mu_S(u_n) \dots$ is a play of $\mathcal{G}$. Indeed, the tree $(\mu_S, \mathcal{T})$ is a pre-k-sample of the strategy tree for the strategy $\mathcal{F}$ we are going to construct.

Now, we define a partition of the regions r among each nonempty set of nodes $succ(T, u, \xi, r)$. In particular, for each $u \in V'$, we split a region $r \in Regs_\sigma^u$ among $slice(v_1, r), \dots, slice(v_h, r)$ such that $\{v_1, \dots, v_h\} = succ(T, u, \xi, r)$ and $\mu_S(v_i) \in slice(v_i, r)$ for $i = 1, \dots, h$.

The consistency notion for the plays of $\mathcal{G}$ and the strategy $\mathcal{F}$ are given together inductively on the length of the plays. Denote with $\rho = s_0 \dots s_n$ a play of $\mathcal{G}$ and with $\pi = u_0 \dots u_n$ a path of T.

If $n = 0$, ρ is *consistent* with π if $s_0 \in \mu_R(u_0)$. Moreover, we define $\mathcal{F}(\rho) = (d^{u_0}, \sigma^{u_0})$ (by definition there is only one initial state in $\mathcal{G}$ and the sole initial state of $A_k^{\mathcal{G}}$ is the region containing just that state).

Let $n > 0$, and suppose that $\rho' = s_0 \dots s_{n-1}$ is consistent with $u_0 \dots u_{n-1}$ and let $\mathcal{F}(\rho') = (d, \sigma^{u_{n-1}})$.

- If $s_{n-1} \xrightarrow{d,\sigma^{u_{n-1}}} s_n$, then ρ is *consistent* with π if $u_n \in succ(T, u_{n-1}, \sigma^{u_{n-1}})$ and $s_n \in \mu_R(u_n)$. Otherwise (i.e., $s_{n-1} \xrightarrow{d',\xi} s_n$ with $d' < d$), ρ is *consistent* with π if $u_n \in succ(T, u_{n-1}, \xi, \mu_R(u_n))$ and $s_n \in slice(u_n, \mu_R(u_n))$.
- if ρ is consistent with π, we let $\mathcal{F}(\rho) = (d, \sigma^{u_n})$ where $s_n = \langle q, \nu \rangle$ and $\langle q, \nu + d \rangle \in z^{u_n}$ (recall that z^{u_n} is the region from which the σ steps are taken, relatively to the $A_k^{\mathcal{G}}$ transition from u_n in the run $(\mu_R, \mathcal{T})$).

We complete the definition of $\mathcal{F}$ by arbitrarily assigning a move for all the plays of $\mathcal{G}$ that are not consistent with any path of T.

Observe that the strategy is well-defined. In fact, since the definition is by disjoint cases, for each play ρ of $\mathcal{G}$, we assign exactly one move. Moreover, the following claim holds:

Claim 1. *Each play of $\mathcal{G}$ which is consistent with $\mathcal{F}$ is also consistent with a path of T.*

Proof : By contradiction let $\rho = s_0 \dots s_n$ be the shortest play of $\mathcal{G}$ that is consistent with $\mathcal{F}$ but is not consistent with any path of T. By hypothesis, since $\rho' = s_0 \dots s_{n-1}$ is consistent with $\mathcal{F}$, there is a path $u_0 \dots u_{n-1}$ of T that is consistent with it. From the definition of $\mathcal{F}$, $\mathcal{F}(\rho') = (d, \sigma)$ for some delay $d > 0$ and $\sigma = \sigma^{u_{n-1}}$. Thus, since ρ is consistent with $\mathcal{F}$, either $s_{n-1} \xrightarrow{d,\sigma} s_n$ or $s_{n-1} \xrightarrow{d',\xi} s_n$ with $d' < d$ must hold. In the first case, there is a $u_n \in succ(T, u_{n-1}, \sigma)$ such that $s_n \in \mu_R(u_n)$, and thus ρ is consistent with $u_0 \dots u_{n-1}u_n$ which contradicts the hypothesis. In the second case, there is a $u_n \in succ(T, u_{n-1}, \xi, r)$ such that $\mu_R(u_n) = r$ and $s_n \in slice(u_n, r)$. Thus also in this case we get a contradiction, and the claim is shown. □

We extend the notion of consistency by saying that a path ρ of the strategy tree $T_{\mathcal{F}}$ is *consistent* with a path π of T if the play defined by ρ (i.e., defined by the sequence of the labels on ρ) is consistent with π. From the above claim, we get that each path ρ of $T_{\mathcal{F}}$ is consistent with a path π of T and from the definition of consistency this path π is unique.

$\mathcal{F}$ is canonical. From the definition of $\mathcal{F}$, we get that $(\mu_S, \mathcal{T})$ is isomorphic to a tree T' contained in $T_{\mathcal{F}}$. Also, T' is a pre-k-sample of $\mathcal{F}$. Thus, from the definition of k-sample, we can obtain from T' a k-sample of $\mathcal{F}$ that is isomorphic to T, which proves the first condition in the definition of canonical strategy.

Let $T_{\mathcal{F}} = (\mu, \mathcal{T}_{\mathcal{F}})$ where $\mathcal{T}_{\mathcal{F}} = (v_{ini}, V, E)$ and $untime(T_{\mathcal{F}}) = (\hat{\mu}, \mathcal{T}_{\mathcal{F}})$. Define a relation R as the set of all pairs $(v, v') \in V \times V'$ such that the unique path from the root to v is consistent with the unique path of T from the root to v'. Since $(v_{ini}, v'_{ini}) \in R$, the following claim shows that $\mathcal{F}$ is canonical.

Claim 2. *R is a bisimulation.*

Proof : For all $(v, v') \in R$, let $\rho(v)$ be the path in $T_{\mathcal{F}}$ from v_{ini} to v and $\pi(v')$ be the path in T from v'_{ini} to v'. From the definition of R, $\rho(v)$ is consistent with $\pi(v')$, and thus from the definition of $untime(T_{\mathcal{F}})$ we get that $\mu_Q(v') = \hat{\mu}(v)$, which shows part 1 of the definition of bisimulation. Also, since $(\mu_S, \mathcal{T})$ is isomorphic to a tree contained in $T_{\mathcal{F}}$, we get that the pairs (u, u') of the nodes that are matched by the isomorphism are also in R, and thus for each $u' \in succ(T, v')$ we can pick exactly the $u \in succ(untime(T_{\mathcal{F}}), v)$ matched by the isomorphism to show part 3 of the definition of bisimulation. Now, consider a node $u \in succ(untime(T_{\mathcal{F}}), v)$. By the notion of consistency, we have that the path of T which is consistent to $\rho(u)$ is given by $\pi(v')$ followed by a node $u' \in succ(T, v')$ and therefore also $(u, u') \in R$, which concludes the proof of the claim. □

This concludes the proof of Lemma 5. □

We can now prove the following result.

Lemma 6. *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 $T' = (\mu, \mathcal{T})$, with $\mathcal{T} = (v_{ini}, V, E)$, be the pre-sample corresponding to T,

and let $[T']$ be the tree $(\mu', \mathcal{T})$, 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 5, there exists a canonical strategy $\mathcal{F}$ for T, and thus $T \in \mathit{Sample}_k(\mathcal{F})$. $\square$

5.3. Bounding the Splitting Degree

Lemma 7. *Given a* CTL *formula φ, let $\mathcal{F}$ be a winning strategy w.r.t. φ. There exists $T \in \mathit{Sample}_k(\mathcal{F})$ such that $T \models \varphi$, where k is the number of existential quantifiers in φ.*

Proof : We essentially prove that if a tree $T_{\mathcal{F}}$ satisfies a CTL formula φ, there exists a subtree of $T_{\mathcal{F}}$ that also satisfies φ and whose branching degree is bounded by the number k of existential quantifiers in φ. Moreover, the subtree can be defined in such a way that it is a k-sample of $\mathcal{F}$.

Let $T_{\mathcal{F}} = (\mu, (v_{ini}, V, E))$. To define $T = (\mu', (v_{ini}, V', E'))$, it is sufficient to define its set of nodes V', as E' and μ' are simply the restrictions of E and μ to V', respectively.

For all $v \in V$, let $\mathit{Sub}^{\exists}(v)$ be the set of existential subformulas of φ that are true in $T_{\mathcal{F}}$ at v. Formally, $\mathit{Sub}^{\exists}(v) = \{\varphi' \in \mathit{Sub}(\varphi) \mid \varphi' = \exists\theta \text{ and } (T_{\mathcal{F}}, v) \models \varphi'\}$. For each node $v \in V$, one can identify the existential subformulas of φ that are true at v and select a witness path for each such formula. Witness paths may be finite, if associated to formulas of the type $\exists\theta_1 \mathcal{U} \theta_2$, or infinite, due to formulas of the type $\exists\theta_1 \mathcal{R} \theta_2$. Since we need at most k witnesses for all existential subformulas of φ, the subtree needs to contain at most k children of v in order to preserve the witnesses. However, this observation is not sufficient to bound the branching degree of the whole subtree, because v may lie on one or more witness paths that have been chosen by its ancestors, thus requiring more than k children. To address this concern, we notice that a witness $u_0 u_1 \ldots$ for an existential formula ψ at a node u_0 is also a witness for the same formula at all nodes u_i belonging to the witness.

The extraction of the subtree argument is formalized by the recursive function **AddLayer**, provided in the following. Precisely, the wanted set V' is obtained as the least fixpoint of the **AddLayer** function (starting from $(\{v_{ini}\}, \emptyset)$).

Function **AddLayer**, together with the supporting functions **AddChildren** and **Update**, maintains a labeling which assigns to each node $u \in V'$ a set $W \subseteq \mathit{Sub}(\varphi) \times V^{\infty}$ (for *Witnesses*), where $V^{\infty} = V^* \cup V^{\omega}$. If a pair $(\psi, u_0 u_1 \ldots)$ belongs to W, it means that the path $u_0 u_1 \ldots$ was chosen as a witness for ψ at u_0. The extra steps taken by the function **AddChildren** guarantee that the resulting subtree is a k-sample of $\mathcal{F}$. Given $U \subseteq V$ and $u \in U$, we say that u is a *leaf w.r.t. U* if for all $v \in U$ we have $(u, v) \notin E$. $\square$

function AddLayer$(V' \subseteq V; f : V' \to 2^{\mathit{Sub}(\varphi) \times V^{\infty}})$
vars:
U : set of nodes
$g : V \to 2^{\mathit{Sub}(\varphi) \times V^{\infty}}$
main:

for each leaf u w.r.t. V' **do**
$(U, g) := \mathbf{AddChildren}(u, f(u))$
$V' := V' \cup U$
$f := f \cup g$
done
return (V', f)

function $\mathbf{AddChildren}(u \in V; W \subseteq \mathit{Sub}(\varphi) \times V^\infty)$
vars:
U : set of nodes
$g : V \to 2^{\mathit{Sub}(\varphi) \times V^\infty}$
RelNodes (relevant nodes) : set of nodes
NewW (new witnesses) : $\mathit{succ}(T_\mathcal{F}, u) \to 2^{\mathit{Sub}(\varphi) \times V^\infty}$
init:
$U := \emptyset$
RelNodes $:= \emptyset$
for each $v \in \mathit{succ}(T_\mathcal{F}, u)$ **do** $\mathit{NewW}(v) := \emptyset$
main:
/ collect nodes preserving existential formulas that are true in u */*
for each $\exists\psi \in \mathit{Sub}^\exists(u)$ **do**
if $\psi = X\theta$ **then**
pick a node $v \in \mathit{succ}(T_\mathcal{F}, u)$ 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 $(\psi, uv_1v_2\ldots) \in W$ **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* $\cup \{v_1\}$
$\mathit{NewW}(v_1) := \mathit{NewW}(v_1) \cup \{(\psi, v_1v_2\ldots)\}$
endif
endif
done
/ add relevant nodes */*
for each $v \in$ *RelNodes* **do**
$U := U \cup \{v\}$
$g(v) := \mathit{NewW}(v) \cup \mathbf{Update}(v, W)$
done
/ protagonist moves */*
for each $v \in \mathit{succ}^{\mathrm{Prot}}(T_\mathcal{F}, u) \setminus$ *RelNodes* **do**
$U := U \cup \{v\}$
$g(v) := \mathbf{Update}(v, W)$
done
/ antagonist moves */*
for each region r such that there exists $v \in \mathit{succ}^{\mathrm{Ant}}(T_\mathcal{F}, u)$ with $\mu(v) \in r$ **do**
$n := \min\{k, \mathit{split}([\mu(u)], \xi, r)\} - |\{v \in U \mid \mu(v) \in r\}|$
while $n > 0$ **do**
pick v in $\mathit{succ}^{\mathrm{Ant}}(T_\mathcal{F}, u) \setminus U$ with $\mu(v) \in r$
$U := U \cup \{v\}$
$g(v) := \mathbf{Update}(v, W)$

```
            n := n − 1
        done
    done
    return (U, g)
function Update(v ∈ V, W ⊆ Sub(φ) × V^∞)
vars:
    B : subset of Sub(φ) × V^∞
init:
    B := ∅
main:
    for each (ψ, v0 v1 ...) ∈ W do
        if v = v1 then B := B ∪ {(ψ, v1 v2 ...)}
    done
    return B
```

5.4. 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. Since the transitions of $A_k^{\mathcal{G}}$ and A_k^{φ} may differ on their branching degree, we augment them in a standard way such that they are uniformed on their maximum branching degree.

Lemma 8. *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 6, 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 5, 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 7, there exists a k-sample T of $\mathcal{F}$ that satisfies φ. By Lemma 6, 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 8 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.*

6. Linear-time Timed 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.

6.1. Linear Temporal Logic

Linear Temporal Logic (LTL) was introduced by Pnueli to specify and verify properties of reactive systems [40]. 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:

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

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 [49]. 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 [35, 50]; 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 [44]. Thus, we obtain a deterministic Rabin generator for φ with a doubly exponential number of states and exponentially many accepting pairs.

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 $\rho = \langle q_{ini}, \nu_{ini}\rangle\langle q_1, \nu_1\rangle \dots \langle q_n, \nu_n\rangle$ in $\mathcal{G}$, we define $Untime(\rho) = \{q_{ini}\}\{q_1\}\dots\{q_n\}$, 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 $\rho \in plays^{\omega}_{\mathcal{F}}$, $Untime(\rho) \models \varphi$.

6.2. Solving LTL *Games*

The following result greatly simplifies the treatment of linear games compared to the CTL games of Section 5.

Proposition 5 ([22]). *Region stable strategies are sufficient for winning* LTL *games.*

By Proposition 4, let A^{φ}_k the Rabin tree automaton corresponding accepting all trees satisfying φ whose branching degree is at most $deg(A^T_1)$. Since Rabin tree automata are closed under intersection [45], we can build a Rabin tree automaton $A^{\cap}$ accepting the intersection of the languages accepted by A^T_1 and A^{φ}_k. In particular, since by [45] the size of $A^{\cap}$ is polynomial in the size of A^T_1 and A^{φ}_k, and the size of A^{φ}_k is doubly exponential in φ, the size of $A^{\cap}$ is doubly exponential in the size of φ. By Lemma 4, 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 9. *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 $T \in L(A^T_1)$, by Lemma 5 there exists a canonical strategy $\mathcal{F}$ for T. By Proposition 4, $t \models \varphi$. Therefore, $untime(T_{\mathcal{F}}) \models \varphi$, and so $\mathcal{F}$ is a winning strategy.

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

By the previous lemma, Proposition 1, and the fact that LTL games are 2EXPTIME-hard [41], 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.*

7. Discussion

7.1. A Comparison with ATL

Alternating-time Temporal Logic [4] (ATL) is a well-known formalism for specifying properties of multi-agent systems. Since ATL model-checking is in PTIME [4], one may wonder whether the timed games with CTL objectives studied in this paper may be solved by simply 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 Lemma 3, the region graph may not be sufficient to find winning strategies; second, and more importantly, not all CTL objectives, which in this paper are applied to the tree of a strategy, can be expressed by an ATL formula on the region graph. 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 antagonist in the same team. According to our semantics, the game is won by the protagonist if she 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.

7.2. A Comparison with Module Checking

In [30], 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 [30, 38, 24] 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 [17, 9], showing that this problem for CTL is 2EXPTIME-complete in the perfect information setting and undecidable in the imperfect one.

7.3. Conclusions and future work

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 an effective 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 [34], these 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 [41], our result is tight. Combining our construction with the results on LTL generators from [5], we can prove an upper bound smaller than 2EXPTIME for meaningful subclasses of LTL timed games.

Recent work has extended to timed automata the results on model-checking against quantitative specifications with parametric constants [23, 16]. The use of parametric constants has been advocated by many authors as a useful tool that allow designers to start the analysis of the system without specifying the actual constants, and thus delay the delicate task of determining the value of such constants to a later time when more will be known about the system. In the discrete time setting the games with parametric winning conditions have been considered [51]. As a future direction, we seek the extension of the automata-theoretic approach also to timed games with such parametric winning conditions.

Bibliography

[1] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1), pages 2 – 34, 1993.

[2] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, vol. 126, pages 183–235, 1994.

[3] Alur, R., Etessami, K., La Torre, S., Peled, D.: Parametric temporal logic for "model measuring". In *ACM Trans. Comput. Log.* vol. 2(3), pages 388–407, 2001.

[4] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. In *Journal of the ACM* vol. 49(5), pages 672–713, 2002.

[5] R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. In *ACM Trans. Comput. Log.*, vol. 5(1), pages 1–25, 2004.

[6] R. Alur, S. La Torre, and P. Madhusudan. Modular Strategies for Infinite Games on Recursive Graphs. In *Proc. of the 15th International Conference on Computer Aided Verification, CAV'03*, LNCS 2725, pages 67-79. Springer, 2003.

[7] R. Alur, S. La Torre, and P. Madhusudan. Modular strategies for recursive game graphs. In *Theor. Comput. Sci.*, vol. 354(2), pages 230–249, 2006.

[8] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. *Theor. Comput. Sci.*, vol 318(3), pages 297–322, 2004.

[9] B. Aminof, A. Legay, A. Murano, O. Serre, and M. Y. Vardi. Pushdown module checking with imperfect information. In *Information and Computation*, vol.223, pages 1-17, 2013.

[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, 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, 1998.

[13] M. Benerecetti, M. Faella, and S. Minopoli. Automatic Synthesis of Switching Controllers for Linear Hybrid Systems: Safety Control. In *Theoretical Computer Science*, to appear.

[14] P. Bouyer, T. Brihaye, F. Chevalier. O-Minimal Hybrid Reachability Games. In *Logical Methods in Computer Science*, vol.6(1), 2010.

[15] P. Bouyer,N. Markey, and O. Sankur. Robust Reachability in Timed Automata: A Game-Based Approach. In *Proc. of 39th International Colloquium on Automata, Languages, and Programming (ICALP 2012)*, LNCS 7392, pages 128–140, 2012.

[16] L. Bozzelli, and S. La Torre. Decision problems for lower/upper bound parametric timed automata. In *Formal Methods in System Design*, vol. 35(2), pages 121–151, 2009.

[17] L. Bozzelli, A. Murano, and A. Peron. Pushdown module checking. *Formal Methods in System Design (FMSD 2010)*, 36(1), pages 65–95, 2010.

[18] L. de Alfaro, M. Faella, and B. Adler. Average reward timed games. In *Proc. of 3rd Int. Conf. on Formal Modelling and Analysis of Timed Systems (FORMATS05)*, LNCS 3829, 2005.

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

[20] K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy Logic. In *Information and Computation*, 208(6), pages 677–693, 2010.

[21] A. Da Costa, F. Laroussinie, and N. Markey. ATL with Strategy Contexts: Expressiveness and Model Checking. In *Proceedings of FSTTCS'10*, LIPIcs 8, pages 120–132, 2010.

[22] L. de Alfaro, M. Faella, T. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *Proc. of 14th International Conference on Concurrency Theory (CONCUR'03)*, LNCS 2761, pages 142–156, 2003.

[23] B. Di Giampaolo, S. La Torre, and M. Napoli. Parametric Metric Interval Temporal Logic. In *Proc. of 4th Int. Conf. on Language and Automata Theory and Applications (LATA 2010)*, LNCS 6031, pages 249–260, 2010.

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

[25] E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs. In *Proc. of 29th IEEE-CS Symposium on Foundations of Computer Science (FOCS'88)*, pages 328 – 337, 1988.

[26] M. Faella, S. La Torre, and A. Murano. Automata-theoretic decision of timed games. In *Proc. of 3rd Int. Work. Verification, Model Checking, and Abstract Interpretation (VMCAI'02)*, LNCS 2294, pages 94–108, 2002.

[27] M. Faella, S. La Torre, and A. Murano. Dense Real-Time Games. *Proc. of 17th Annual IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 167–176, 2002.

[28] T.A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *Proc. of 10th Int. Conf. on Concurrency Theory (CONCUR'99)*, LNCS 1664, pages 320 – 335, 1999.

[29] M. Jurdziński and A. Trivedi. Reachability-time games on timed automata. In *Proc. of 34th Int. Colloquium on Automata, Languages and Programming (ICALP'07)*, LNCS 4596, pages 838–849, 2007.

[30] O. Kupferman, M.Y. Vardi, and P. Wolper. Module checking. In *Information and Computation*, vol. 164(2), pages 322–344, 2001.

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

[32] S. La Torre, S. Mukhopadhyay, and A. Murano. Optimal-Reachability and Control for Acyclic Weighted Timed Automata. In *Proc. of IFIP Int. Conf. on Theoretical Computer Science (IFIP-TCS 2002)*, pages 485–497, 2002.

[33] S. La Torre and M. Napoli. A Decidable Dense Branching-Time Temporal Logic. In *Proc. of 20th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2000)*, pages 139–150, 2000. FSTTCS 2000: 139-150

[34] S. La Torre and M. Napoli. Finite Automata on Timed ω-Trees. *Theoretical Computer Science* 293(3), pages 479–505, 2003.

[35] O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. In *Proc. of 12th ACM Symposium on Principles of Programming Languages*, pages 97–107, 1985.

[36] M. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. What Makes ATL* Decidable? A Decidable Fragment of Strategy Logic. In *Proc. of 23rd International Conference on Concurrency Theory (CONCUR'12*, LNCS 7454, pages 193–208, 2012.

[37] M. Mogavero, A. Murano, and M. Y. Vardi. Reasoning About Strategies. In *Proceedings of FSTTCS'10*, LIPIcs 8, pages 133–144, 2010.

[38] A. Murano, M. Napoli, and M. 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.

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

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

[41] 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.

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

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

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

[45] 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.

[46] W. Thomas. On the synthesis of strategies in infinite games. In *Proc. of 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, LNCS 900, pages 1 – 13. 1995.

[47] C. J. Tomlin, J. Lygeros, and S. S. Sastry. A game theoretic approach to controller design for hybrid systems. In *Proc. of the IEEE*, vol.88(7), pages.949–970, 2000.

[48] M.Y. Vardi. Verification of concurrent programs: the automata-theoretic framework. In *Proc. of the Second IEEE Symposium on Logic in Computer Science (LICS'87)*, pages 167–176, 1987.

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

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

[51] M. Zimmermann. Optimal Bounds in Parametric LTL Games. In *Proc. of Second Int. Symp. on Games, Automata, Logics and Formal Verification (GandALF 2011)*, EPTCS 54, pages 146–161, 2011.