A Zone-Based Algorithm for Timed Parity Games

Gilles Geeraerts
Université libre de Bruxelles, Belgium

Frédéric Herbreteau
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, 33400 Talence, France

Jean-François Raskin
Université libre de Bruxelles, Belgium

Alexis Reynouard
Université libre de Bruxelles, Belgium

Abstract

This paper revisits timed games by building upon the semantics introduced in "The Element of Surprise in Timed Games" [9]. We introduce some modifications to this semantics for two primary reasons: firstly, we recognize instances where the original semantics appears counterintuitive in the context of controller synthesis; secondly, we present methods to develop efficient zone-based algorithms. Our algorithm successfully addresses timed parity games, and we have implemented it using UppAal's zone library. This prototype effectively demonstrates the feasibility of a zone-based algorithm for parity objectives and a rich semantics for timed interactions between the players.

2012 ACM Subject Classification Software and its engineering → Formal methods

Keywords and phrases Timed Parity Games, Realtime Controller Synthesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.33

Related Version *Extended Version*: https://arxiv.org/abs/2510.07361

1 Introduction

Timed automata [1] are a well-established formalism for modeling real-time systems. Their semantics captures the continuous evolution of time along with discrete transitions, making them particularly suitable for specifying and verifying time-critical behaviors. A crucial aspect in their analysis is the treatment of infinite executions: runs in which time converges are usually deemed unrealistic, as they imply infinitely many actions occurring within a finite amount of time, a phenomenon called *zenoness*.

Timed games extend the model of timed automata to a two-player setting, where one player represents the system under design and the other represents its environment. Defining the semantics of such games is challenging. A central issue is that one player might attempt to win by taking actions increasingly closer in time, effectively blocking the opponent's ability to respond. However, such strategies rely on blocking the progress of time and are thus physically implausible and should not be permitted.

The paper "The Element of Surprise in Timed Games" [9] proposes an elegant solution to this problem by introducing a timed game model that preserves both the element of surprise (players are allowed to play quickly in order to overtake their adversary), and the need for time divergence. In this model, players independently and simultaneously choose both an action and a delay before executing it. The action with the shortest delay is taken, introducing inherent nondeterminism in the case of equal delays. To ensure physical realism, the semantics prevents a player from winning by stopping time and introduces symmetric winning conditions applicable to all ω-regular objectives. The authors show that the ability to surprise the opponent increases the expressive power of the model and demonstrate that memory strategies may be necessary even for reachability objectives. Although the semantics

is shown to be decidable via region-based abstraction, no implementation currently exists. In particular, this semantics requires solving parity games over timed structures, a task for which no practical implementation is available. Existing tools only support simpler objectives such as safety and reachability.

For simpler games like safety and reachability games, the paper [6] presents a fully symbolic zone-based on-the-fly algorithm for solving reachability and safety timed games. This approach is based on a forward exploration of the state space using zones and supports early termination. The algorithm is efficient in practice and was evaluated experimentally, but it relies on a simpler semantics that does not include the surprise mechanism or handle general ω-regular objectives.

Our objective in this work is to provide an efficient implementation of a semantics closely related to that introduced by de Alfaro et al. [9], which preserves the notion of *surprise* in real-time interactions. This will be the first efficient implementation of synthesis for parity objective in a timed two-player context. To this end, we slightly adapt the original semantics with two goals: to make it amenable to a symbolic zone-based implementation; and to handle the controller synthesis problem in a natural way, while retaining the essential features of the original model. We then develop a symbolic algorithm for solving parity games under this expressive timed semantics. Our method builds on ideas from Zielonka's recursive algorithm for parity games on finite graphs [14]. To the best of our knowledge, this is the *first implementation* of an algorithm for solving *timed parity games*, marking a significant advance in the analysis and synthesis of real-time systems with complex winning conditions.

Related works

Above, we discussed the foundational contributions of de Alfaro et al. [9] and Cassez et al. [6], which address the semantics and algorithmic analysis of timed games respectively. Beyond these, several studies have explored related challenges in timed games.

Cassez et al. [7] examine various formulations of control problems for timed and hybrid systems, highlighting the limitations of dense-time controllers and advocating for more implementable discrete-time approaches. Their solution is obtained through a discretization of the timed semantics which is avoided by [9]. At the initial phase of developing real-time systems, discretization is typically deemed less suitable than continuous time semantics that are more abstract. Brihaye et al. [5] focuses on the minimum-time reachability problem in timed games, presenting methods to compute the least time required for a player to reach a target location against all possible strategies of the opponent. In a case study paper, Cassez et al. [8] demonstrate the application of synthesis tools like UppAal-TiGa for the automatic generation of robust and near-optimal controllers in an industrial application.

Addressing the implementability of timed models, De Wulf et al. [13] introduce the "Almost ASAP" semantics, a relaxation of the traditional ASAP semantics, designed to better reflect the limitations of physical systems, especially when implementing precisely timed strategies on hardware with digital clocks.

Bouyer et al. [3] investigate the value problem in weighted timed games, proving its undecidability in the general case [4], but also providing approximation algorithms for certain subclasses.

Realizability problems for real-time logics have been explored in [11, 10], where it was shown that only specific fragments of MITL are decidable, while the full logic is undecidable.

Finally, Asarin et al. [2] propose symbolic techniques for controller synthesis in both discrete and timed systems, aiming to manage large state spaces without relying on exhaustive enumeration, by leveraging logical formulas. This line of work has been further developed through contributions involving the use of zones [12].

2 Preliminaries

We begin by defining the arenas on which our games are played. These arenas are essentially Alur-Dill timed automata [1], augmented with both discrete and continuous transitions, which are partitioned into transitions controlled by one player and transitions controlled by the other. We then define a family of functions that associate a game to each arena. These functions correspond to the semantics applied to arenas. A game, in this context, is a transition system equipped with a winning condition for one player; the objective of the other player is adversarial and defined as the complement of the first player's objective.

Clocks, Valuations, and Guards

An *interval* is a nonempty convex subset of the set $\mathbb{R}_{\geq 0}$ of non-negative real numbers. Intervals can be left-open or left-closed, right-open or right-closed, and bounded or unbounded. An interval takes one of the following forms: $[a, b]$, $[a, b)$, $[a, \infty)$, $(a, b]$, (a, b), (a, ∞), with $a, b \in \mathbb{N}$ and $a \leq b$. We write $\emptyset$ for any empty interval.

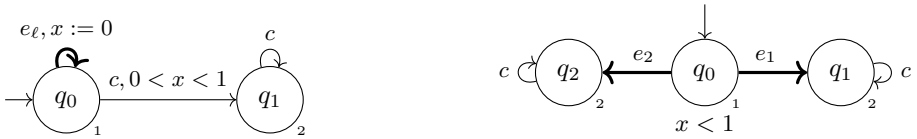
Let X be a finite set $\{x_1, x_2, \ldots, x_n\}$ of variables called clocks. An *atomic clock constraint* is a formula of the form $x \in I$ where I is an interval. A *guard* is a Boolean combination of atomic clock constraints. We denote by $\text{Guards}(X)$ the set of all guards on X. A *valuation* for the clocks in X is a function $v : X \to \mathbb{R}_{\geq 0}$. To simplify the notation, we sometimes write v_x for $v(x)$. We write $v \models g$ whenever the valuation v satisfies the guard g. $\text{Vals}(X)$ is the set of all valuations of X and $\mathbf{0}_X$ is the valuation that assigns 0 to all clocks of X. $\top \in \text{Guards}(X)$ is called *universal guard* and is satisfied by all valuations. $\bot \in \text{Guards}(X)$ is the *empty guard* and is never satisfied.

For $R \subseteq X$ (resp. $x \in X$), we write $v[R := 0]$ (resp. $v[x := 0]$) for the valuation that assigns 0 to all clocks $x \in R$ (resp. to x), and $v(x')$ to all the other clocks. For $t \in \mathbb{R}_{\geq 0}$, we write $v + t$ for the valuation that assigns the value $v(x) + t$ to each clock $x \in X$. Given two valuations v and v', we write $v \bowtie v'$, for $\bowtie \in \{<, \leq, =, \geq, >\}$ when $\forall x \in X : v(x) \bowtie v'(x)$. We can now define the syntax of timed games arenas.

Timed Game Arena

We study games played on arenas. An arena models the system under study. The definition of the game associated with an arena depends on the semantics under consideration. The game rules are determined by the specific semantics used.

A *timed game arena*, TGA for short, over actions $A = A_0 \uplus A_1$ and clocks X is a tuple $G = (Q, E, C, U_0, U_1, \alpha)$ where Q is a finite set of *locations*, $E \subseteq Q \times A \times \text{Guards}(X) \times 2^X \times Q$ is a finite set of labeled *edges*, $C : Q \to \text{Guards}(X)$ is an *invariant function*, $U_p : Q \to \text{Guards}(X)$ are the *no-wait functions* of the players $p \in \{0, 1\}$, and $\alpha : Q \to \{0, 1, \ldots, d\}$ is a *coloring function of the locations*. $\alpha(q)$ is the *color* of q (sometimes called its *priority*). The partition of A into A_0 and A_1 represents the possible actions of two players: Player 0, which is called the *Controller* and Player 1, the *Environment*. Given a TGA G, $\text{Guards}(G)$ is the set of guards that appear in G (in the image of C, U_0, or U_1, or as an edge guard in E). Figure 1a shows a TGA with one clock x and two locations. The number next to a location



(a) A TGA from [9]. Under the surprise semantics, the Controller can reach q_1 from q_0 by playing fast enough and retrying as many time as needed.

(b) A TGA where the Controller should win (by reaching either q_1 or q_2).

Figure 1 Two TGAs. Thick arrows are Environment's actions. The Controller aims to visit locations with even colors, indicated at their bottom right, infinitely often.

q is its color $\alpha(q)$. The Controller's actions set is $A_0 = \{c\}$ and the Environment's actions set (thick arrows) is $A_1 = \{e_\ell\}$. The guard below a location is its invariant. Intuitively, the invariant of a location q is a condition on the clocks that must be satisfied for the game to be in q. On this example, all the no-wait functions always return $\bot$ for both players and we do not display them. Intuitively, the no-wait functions of the players force them to play without a delay when the clocks have certain value. For example, if $U_0(q) = x < 1$, then Player 0 cannot let time elapse in location q whenever the clock x is smaller than 1. Those no-wait functions usually do not appear in the classical definitions of timed games, but they will turn out to be useful in the sequel.

To play games on such TGA, we need further assumptions. We extend all TGA $G = (Q, E, C, U_0, U_1, \alpha)$ over actions $A = A_0 \uplus A_1$ and clocks X with two extra actions $\bot_0$, $\bot_1$ and an extra clock z. The actions $\bot_p$ will be used in the game setting to mean that Player p will choose to stay in the current location: we assume that each location q bears two self-loops labeled respectively by $\bot_0$ and $\bot_1$. Formally, we let: $A_{\bot,p} := A_p \cup \{\bot_p\}$ for $p \in \{0,1\}$ and $E_\bot := E \cup \big\{(q, \bot_p, \top, \emptyset, q) \mid \forall q \in Q, p \in \{0,1\}\big\}$. When necessary, we call A_p the set of *regular actions* of p. The clock z will be used to measure elapsed time and is never reset or tested. We let $X_z = X \cup \{z\}$. From now on, we assume that all TGA are equipped with z, $\bot_0$ and $\bot_1$. Note that we do not formally fix an initial location in the definition of a TGA. We may give one on the figures to help intuition.

Alur-Dill semantics

All the semantics we will discuss in Section 3 are based on the following definitions. A *configuration* is a pair $(q, v) \in Q \times \mathrm{Vals}(X_z)$. The TGA can only be in such a configuration if v satisfies the invariant of q; and time can elapse, increasing the valuation, as long as the invariant stays satisfied. From a configuration (q, v), an edge (q, a, g, R, q') can be fired iff the current valuation v satisfies g. This leads to configuration (q', v'), where v' is obtained from v by resetting the clocks in R. Formally, given (q, v), we say that an action a is *enabled* whenever there is $(q, a, g, R, q') \in E$ s.t. $v \models g \wedge v[R := 0] \models C(q')$. We write $(q, v) \xrightarrow{\Delta, a} (q', v')$ with $\Delta \in \mathbb{R}_{\geq 0}$ to say that a is enabled in $v + \Delta$, leading from $(q, v + \Delta)$ to (q', v'), and that one can wait from (q, v) to $(q, v + \Delta)$ without violating the invariant $C(q)$ nor the no-wait function $U_p(q)$, with p s.t. $a \in A_{\bot,p}$. Formally, $(q, v) \xrightarrow{\Delta, a} (q', v')$ iff:

$$\begin{aligned}
&\forall \delta \in [0, \Delta] : v + \delta \models C(q) \wedge \exists (q, a, g, R, q') \in E_\bot : \\
&\big(v + \Delta \models g \wedge v' = v + \Delta[R := 0] \wedge v' \models C(q') \\
&\qquad\qquad \wedge a \in A_{\bot,p} \implies \forall \delta' \in (0, \Delta) : v + \delta' \not\models U_p(q)\big).
\end{aligned}$$

A *deadlock* occurs whenever there is a location-valuation pair (q, v) s.t. C limits the time one can stay in q starting from v and there is no action enabled in that time. Formally, there is a deadlock in (q, v) whenever $\nexists \Delta, a, q', v' : (q, v) \xrightarrow{\Delta, a} (q', v')$. From now on, we assume that our arenas are deadlock-free.

Clock Regions

We rely on the usual region construction [1] that we adapt slightly to account for the special clock z. It is based on a finite equivalence relation over clock valuations of X_z. A clock region is an equivalence class of this relation. A *region* is a pair (q, r) with r a clock region. Since there are finitely many regions, and regions form a time-abstract bisimulation, regions form a finite abstraction of the state space that is suitable for our purpose.

Given a TGA G, for each clock $x \in X$, let c_x be the largest constant in Guards(G) that involves x, where $c_x = 0$ if x does not occur in any guard, no-wait or invariant of G. Let v_1 and v_2 be two clock valuations in Vals(X_z). We let $X' = \{y \in X \mid v_1(y) < c_y\} \cup \{z\}$ denote the set of clocks y which have a value smaller than c_y in v_1 plus the special clock z. Then, v_1 and v_2 are *clock equivalent*, denoted $v_1 \equiv_G v_2$, iff:

1. for all $x \in X$: either $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$ or $\lfloor v_1(x) \rfloor > c_x$ and $\lfloor v_2(x) \rfloor > c_x$;
2. the ordering of the fractional parts of all clocks in X' is the same in v_1 and v_2;
3. for all clocks $x \in X'$: $v_1(x)$ is an integer if and only if $v_2(x)$ is an integer.

Note that the special case we apply to z can be interpreted as if z was automatically reset to 0 every time it reaches 1. Indeed, the first condition about the integer parts of the clocks does not constrain z, and the two other conditions constrain only the fractional part of z's value. Then, a *clock region* is an equivalence class of $\equiv_G$, and we write $[\![v]\!]_G$ for the clock region of the clock valuation v. It is well-known that $\equiv_G$ is a finite time-abstract bisimulation relation, hence that the number of clock regions is finite for all TGA [1].

Next, to obtain games, we will associate a *transition system* (whose states and transitions will define the possible game states and moves) and a *winning condition* to each TGA. We will consider *concurrent games*, where both players choose a move simultaneously and the game state is updated according to the choice of both players. In Section 3, we will discuss different possible ways to build a transition system for a given TGA.

Game

A *(labeled) transition system* over a potentially infinite set M of labels is a tuple $\Phi = (S, M, \rightarrow, \beta)$ where S is a potentially infinite set of states, $\rightarrow \subseteq S \times M \times S$ is a labeled transition relation, and $\beta : S \mapsto \mathbb{N}$ is a coloring function of the states whose image must be finite. As usual, we write $s \xrightarrow{m} s'$ instead of $(s, m, s') \in \rightarrow$.

To model concurrent games, the set of labels are of the form $M = M_0 \times M_1$. The elements of M_p are called the *moves* of Player p. Intuitively, the game is played *round by round* and each round is characterized by the current game state (assuming a fixed initial state $s_{in} \in S$). When the game is in state s, both players concurrently choose respective moves $m_0 \in M_0$ and $m_1 \in M_1$, and the game moves to any state s' s.t. $s \xrightarrow{m_0, m_1} s'$. Then, a new round starts and the game continues *ad infinitum*, forming an infinite *play*.

Formally, a *play from* s_{in} is an infinite sequence of the form $s^0 \left(m_0^i m_1^i s^i\right)_{i \in 1,2,3\ldots}$ s.t.: $s^0 = s_{in}$ and $s^{i-1} \xrightarrow{m_0^i, m_1^i} s^i$ for all $i \geq 1$. A *history* is a sequence as above that is finite. Let $K(\Phi)$ denote the set of all possible plays on a transition system Φ and $H(\Phi)$ be the set of its possible histories. A *run* is a history or a play.

We use *parity conditions* to define winning plays. Let $\rho = \left(s^0\left(m_0^i m_1^i s^i\right)_{i\in 1,2,3\dots}\right)$ be a play and β a coloring function of the states. We lift the coloring function β from states to runs: $\beta(\rho) = \left(\beta(s^i)\right)_{i\in 0,1,2\dots}$. We denote $\inf(\beta(\rho))$ the set of colors that occur infinitely often in ρ. Then, we write $\rho \models \text{Parity}_0(\beta)$ iff $\max(\inf(\beta(\rho)))$ is even and $\rho \models \text{Parity}_1(\beta)$ iff $\max(\inf(\beta(\rho)))$ is odd.

A *game* is obtained by associating a winning condition to a transition system. A game is thus a pair $\mathcal{G} = (\Phi, WC)$ where Φ is a transition system over a set $M_0 \times M_1$ and $WC \subseteq K(\Phi)$ a set of Player 0's *winning* plays. In games, players can play according to a strategy. A Player p's *strategy* is a function $\sigma_p : H(\Phi) \to M_p$ that associates a Player p's move to all histories. We denote by $Z_p(\Phi)$ the set of all strategies of Player p in Φ. A run $\rho = s^0(m_0^i m_1^i s^i)_{i\in 1,2,\dots}$ is played according strategies σ_0 and σ_1 if for every $i \geq 1$, $m_0^i = \sigma_0(h_i)$ and $m_1^i = \sigma_1(h_i)$ where h_i is the prefix of ρ up to s^{i-1}. Then, given two strategies σ_0 and σ_1, the *outcome* $\text{Out}^{s_{in}}(\sigma_0, \sigma_1)$ is the set of all plays from s_{in} played according to σ_0 and σ_1. A strategy $\sigma^\star$ of the Controller (Player 0) is *winning from* s_{in} iff $\text{Out}^{s_{in}}(\sigma^\star, \sigma_1) \subseteq WC$ **for all** Environment's strategy σ_1.

Finally, we assume that all games $\mathcal{G} = (\Phi, WC)$ are *well-defined*, i.e. (1) Φ is deadlock-free and (2) in all states, the moves available to a player do not depend on the move played by the other player. Formally, for any state $s \in S$, let $\text{Moves}_0(s) = \{m_0 \in M_0 \mid \exists m_1 \in M_1 : \exists s' \in S \text{ s.t. } s \xrightarrow{m_0,m_1} s'\}$. We define $\text{Moves}_1(s)$ symmetrically for Player 1. Then, for all $s \in S$, there are m_0, m_1, s' s.t. $s \xrightarrow{m_0,m_1} s'$, *and* for all $s \in S$, for all $m_0 \in \text{Moves}_0(s)$, for all m_1 in $\text{Moves}_1(s)$: there is s' s.t. $s \xrightarrow{m_0,m_1} s'$.

Deciding how to build a game from an arena is not a trivial problem, and we devote Section 3 to discuss several possibilities.

3 Three Semantics for Timed Games

In the present section, we discuss three possible ways to associate a game $\mathcal{G} = (\Phi, WC)$ to a TGA G, thereby providing three possible definitions of game semantics in a unified framework. This framework constitutes the first contribution of the paper. We first recall the "classical" semantics of timed games, and the "surprise" semantics of [9]. Then, as a second contribution we introduce a third, novel, semantics that refines and improves the "surprise" semantics. This refinement results in a semantics which is more natural, and that allows for efficient winning strategy computation. In Section 4, we will introduce a zone-based symbolic algorithm to compute winning strategies in this semantics.

Let us fix a TGA $G = (Q, E, C, U_0, U_1, \alpha)$ over actions $A = A_0 \uplus A_1$ and clocks X. We show how to build a game (Φ, WC). Since we consider timed games, the states of the transition system Φ are configurations (q, v) of the TGA G, and the moves of Player p are pairs $(\Delta, a) \in (\mathbb{R}_{\geq 0} \times A_{\perp,p})$. Intuitively, each round of the game starts in a state (q, v). Both players propose concurrently a delay Δ and an action a. The player with the shortest delay wins the round and his proposal (Δ, a) is played in G: we let Δ time units (t.u.) elapse, reaching $(q, v + \Delta)$; then follow an a-labeled edge. The special actions $\perp_0$ and $\perp_1$ mean that the player only wants to wait without following any edge.

Plays are now of the form $\rho = (q^0, v^0)(\Delta_0^1, a_0^1)(\Delta_1^1, a_1^1)(q^1, v^1)\dots$. We say that such a play is *time-divergent* iff the special clock z cannot be bounded along $v^0, v^1, v^2, \dots$, i.e. $\forall t \in \mathbb{R}_{\geq 0} : \exists i \in \mathbb{N} : v_z^i > t$. We write $\rho \models td$ if ρ is time-divergent.

We formally define a semantics $\mathcal{Q}$ as a function that associates a game $\mathcal{G} = (\Phi, WC)$ to a TGA G. We now present the three semantics mentioned above. Given a TGA G, $\mathcal{T}(G)$ is the timed game classically associated to G [1], $\mathcal{S}(G)$ is the game from [9] which accounts for the surprise effect, and $\mathcal{R}(G)$ is a new refinement of $\mathcal{S}(G)$. These three semantics are *sound*, i.e., given a deadlock-free arena, they give a game with a well-defined transition system.

Classical Timed Semantics $\mathcal{T}$

The "classical" timed semantics associated to G is the game $\mathcal{T}(G)$. In this settings, players can only play moves of the form $(0, a)$, meaning "play a immediately"; or $(\Delta, \bot)$, meaning "wait for Δ t.u.". For example, consider the TGA in Figure 1a. The objective of the Controller is to reach q_1 from q_0. To do so, he needs to wait in q_0 until x becomes non-null, because of the guard from q_0 to q_1. However, the Environment can always overtake him by playing $(0, e_\ell)$ that resets the clock. The Environment can keep playing that way, so, with the classical semantics, there is no winning strategy for the Controller in this game.

Formally, $\mathcal{T}(G) = (\Phi = (S,\ M,\ \to,\ \beta),\ WC)$ is defined as follows. The state set is $S = Q \times \mathrm{Vals}(X_z)$. The possible moves of Player p are $M_p = (\mathbb{R}_{\geq 0} \times \{\bot_p\}) \cup (0 \times A_p)$. The definition of the transition relation $\to$ follows the above intuition: $(q, v) \xrightarrow{(\Delta_0,a_0),(\Delta_1,a_1)} (q', v')$ iff there is $j \in 0, 1$ s.t. $\Delta_j \leq \Delta_{1-j}$ and $q' = q_j$, $v' = v_j$ with[1] $(q, v) \xrightarrow{\Delta_0,a_0} (q_0, v_0)$ and $(q, v) \xrightarrow{\Delta_1,a_1} (q_1, v_1)$. For the winning condition, we lift the coloring function α on the locations of G to the states of Φ, letting $\beta(q, v) = \alpha(q)$ for all states (q, v). Then, WC is defined as a parity condition: $WC = \{\rho \in K(\Phi) \mid \rho \models \mathrm{Parity}_0(\beta)\}$.

TEOS Surprise Semantics $\mathcal{S}$

The winning strategy of the Environment described above (always playing $(0, e_\ell)$ in q_0) is problematic since any play in its outcome is *not* time-divergent. This is regarded as a flaw in the model, since playing an infinite number of actions in a finite time is not realistic. This observation has prompted the introduction of the "surprise" semantics by de Alfaro et al. [9]. We called this semantics the "TEOS semantics", from the title of the paper, and denote it $\mathcal{S}$. It differs from $\mathcal{T}$ by the definition of the player moves and of the set of winning plays. In the TEOS semantics, we let $M_p = (\mathbb{R}_{\geq 0} \times A_{\bot,p})$. Thus, players can propose to wait *and, immediately after*, play an action. This is where the "surprise" stems from: it gives players the ability to *act quickly* and to take their opponent by surprise by playing an action after an arbitrarily short delay. However, plays where the time does not diverge should be avoided, so the game $\mathcal{S}(G)$ introduces a *blame function* bl to identify the player blocking the time. If the Controller is to blame in a play where time does not diverge, he looses the game even if the play satisfies his parity objective.

Formally, we let: $\mathrm{bl} : S \times M_0 \times M_1 \times S \to \{0, 1\}$ be s.t. $\mathrm{bl}(s, m_0, m_1, s') = 0$ iff $s \xrightarrow{m_0} s'$. Then, bl is lifted to runs by recording who is playing at each round: $\mathrm{bl}\left(s^0 \left(m_0^i m_1^i s^i\right)_{i\in 1,2,3\ldots}\right) = \left(\mathrm{bl}(s^{i-1}, m_0^i, m_1^i, s^i)\right)_{i\in 1,2,3\ldots}$. Player p will be to blame in play ρ iff p occurs infinitely often in $\mathrm{bl}(\rho)$. The Controller then wins iff it satisfies his parity objective on time-divergent plays *and* is not to blame if the time converges. Formally, for $p \in \{0, 1\}$, we write $\rho \models \mathrm{bl}_p$ iff $p \in \inf(\mathrm{bl}(\rho))$, and let: $WC = \{\rho \mid \left(\rho \models td \wedge \rho \models \mathrm{Parity}_0(\beta)\right) \vee \left(\rho \not\models td \wedge \rho \not\models \mathrm{bl}_0\right)\}$. The other components of $\mathcal{S}(G)$ are the same as for $\mathcal{T}(G)$.

Consider again Figure 1a, and the Environment's strategy $\sigma_1^\star$ that consists in always playing $(0, e_\ell)$ in q_0. In the TEOS semantics, the Controller has a strategy $\sigma_0^\star$ to win against $\sigma_1^\star$ that consists in always playing (ε, c), for $0 < \varepsilon < 1$. In the outcome of those strategies, the time converges, but the Controller is not to blame, so he wins those plays although his parity objective is not fulfilled. However, such a $\sigma_0^\star$ is *not* winning for the Controller against *all* strategies of the Environment, who can always play $(\frac{\varepsilon}{2}, e_\ell)$. Yet, the Controller now has a winning strategy $\sigma_{2^{-i}}$ that consists in playing $(2^{-i}, c)$ at the i-th round until q_1 is reached,

[1] Recall that $\rightharpoonup$ is defined above under the paragraph about the Alur-Dill semantics.

and then playing $(1, c)$ for ever. Against $\sigma_{2^{-i}}$, the only way for the Environment to avoid reaching q_1 is to always play smaller delays to win each round, such as $(2^{-i-1}, c)$ at the i-th round. Then, if q_1 is not reached, time converges and the Environment is to blame, so the Controller always wins, even if q_1 is never reached. This illustrates the *ability to act quickly* of the surprise semantics.

Refined Surprise Semantics $\mathcal{R}$

Our last semantics is a novel refinement of $\mathcal{S}$. The main idea with this new semantics is that we allow the Controller *not to act* in the states of the system where he has no (discrete) action at his disposal. Consider an example where the Controller needs to observe the system and act after the environment has chosen between two possible actions. Such a situation is modeled in Figure 1b, where the Controller observes the system in state q_0 (where he has no action) and waits for the Environment to play either e_1 or e_2, that will arrive within one time unit. In the $\mathcal{S}$ semantics, when in q_0, the Controller can only propose to play a sequence of the form $(\Delta_0, \bot), (\Delta_1, \bot), \ldots, (\Delta_i, \bot), \ldots$ where $\sum_{i\geq 0} \Delta_i < 1$, in order to satisfy the invariant. Then, the Environment can always play (t, e_1) with $t \geq t_i$ to let the Controller play at all rounds. In such plays, the time converges with the blame on the Controller who loses the game. Thus, surprisingly, the Environment wins by doing nothing, forcing the Controller to block time. Indeed, the TEOS semantics makes no distinction between a player who waits because he has no other choice and a player who voluntarily blocks the time. This prompts the first refinement of our semantics that consists in introducing a new move, denoted "$-$" and meaning that the player does not play anything.

Formally, we extend the set of possible moves of Player p to: $M_p = (\mathbb{R}_{\geq 0} \times A_{\bot,p}) \cup \{-\}$. We define an ancillary predicate $\mathrm{P}_p(q, v)$ which is true iff Player p can propose a regular action from (q, v), i.e. iff there are $\Delta \in \mathbb{R}_{\geq 0}$, $a \in A_p$, $q' \in Q$ and $v' \in \mathrm{Vals}(X_z)$ s.t. $(q, v) \xrightarrow{\Delta, a} (q', v')$. Observe that $\neg\, \mathrm{P}_p(q, v)$ implies $\mathrm{P}_{1-p}(q, v)$ because we have assumed that G is deadlock-free. Then, we adapt the transition relation as follows.

- $(q, v) \xrightarrow{(\Delta_0, a_0), (\Delta_1, a_1)} (q', v')$ iff both players can play: $\mathrm{P}_0(q, v) \wedge \mathrm{P}_1(q, v)$ and $\exists j \in 0, 1$ s.t. $\Delta_j \leq \Delta_{1-j}$ and $q' = q_j$, $v' = v_j$ with $(q, v) \xrightarrow{\Delta_0, a_0} (q_0, v_0)$ and $(q, v) \xrightarrow{\Delta_1, a_1} (q_1, v_1)$.
- $(q, v) \xrightarrow{(\Delta_0, a_0), -} (q', v')$ iff only Player 0 can play: $\neg\, \mathrm{P}_1(q, v)$ and $(q, v) \xrightarrow{\Delta_0, a_0} (q', v')$.
- $(q, v) \xrightarrow{-, (\Delta_1, a_1)} (q', v')$ iff only Player 1 can play: $\neg\, \mathrm{P}_0(q, v)$ and $(q, v) \xrightarrow{\Delta_1, a_1} (q', v')$.

With this change, the Controller no longer needs to propose delays from q_0 in Figure 1b. Instead, in q_0, the Controller does not play, and the Environment must play e_1 or e_2 to model an event; otherwise, the Environment loses by blocking time.

In the game $\mathcal{R}(G)$, we also adapt the *blame* predicate. In the $\mathcal{S}$ semantics, the blame only records which Player played last, no matter which delay he has played. We introduce a *lazy blame* policy, which differs from the classic blame in that a player who has played *a wait* $(\Delta, \bot_p)$ *long enough to leave the current clock region* is not blamed. While this change might sound inconsequential, it is important to obtain an efficient zone-based algorithm later in the paper. Formally, we let $\mathrm{lbl} : S \times M_0 \times M_1 \times S \to \{-, 0, 1\}$ be s.t. $\mathrm{lbl}((q, v), m_0, m_1, (q', v')) = -$ if $q' = q$ and $v \not\equiv_G v'$; *otherwise* $\mathrm{lbl}((q, v), m_0, m_1, (q', v')) = \mathrm{bl}((q, v), m_0, m_1, (q', v'))$. We lift lbl to runs like we did for bl and write $\rho \models \mathrm{lbl}_p$ iff $p \in \inf(\mathrm{lbl}(\rho))$. One can show that it is feasible to replace the blame $(-)$, assigned by lbl s.t. no one is blamed, with a repetition of the previous blame. This allows reducing the state space. In the next section, we focus on this $\mathcal{R}$ semantics and show how to solve the synthesis problem in this setting.

4 Solving the Synthesis Problem On $\mathcal{R}$

Given a well-defined game $\mathcal{G}$ and an initial state s_{in}, the *synthesis problem* asks to compute, if possible, a winning strategy from s_{in} for the Controller. Solving the synthesis problem on $\mathcal{R}$ is to solve the synthesis problem for all game $\mathcal{G} = \mathcal{R}(G)$ for all TGA G, and all initial configuration $(q, \mathbf{0}_X)$ for all location q of G.

In this section, we present an algorithm to solve the synthesis problem on $\mathcal{R}$. The main technicality to achieve this is to define an *augmented*[2] TGA G^+, and associate to it an appropriate semantics[3] to obtain a game $\mathcal{R}^+(G^+)$ on which the synthesis problem is easier to solve. This transformation moves just enough complexity from the $\mathcal{R}$ semantics to the TGA G^+: for instance, in G^+, the locations are augmented with information about the past. This allow us to obtain a game with a *pure parity objective*, i.e., a game whose objective is of the form $\{\rho \mid \rho \models \text{Parity}_0(\alpha)\}$ (while in $\mathcal{R}(G)$ the objective must also account for the blame).

Equiped with $\mathcal{R}^+(G^+)$, we could imagine a simple algorithm to solve the synthesis problem. It would work as follows. First, remove the concurrency of the game by letting the Controller make his choice first, then letting the Environment play against this move. This reduction is valid since we are trying to compute a winning strategy of the Controller against *any* strategy of the environment. One can also let the environment lift the non-determinism at this stage by letting him choose from whom the action will be played when the two have chosen the same delay. This is valid since we want a Controller's strategy s.t. *all the plays* in all the outcome sets are winning. Second, use the classical region construction to obtain a finite game. The resulting game is a finite, turn-based game played on graph with a parity objective, i.e. a *parity game*. Such games are well-studied [14] and enjoy positional strategies that can be computed by the classical Zielonka algorithm.

While this naive algorithm would be correct, we argue it would be inefficient and would not be benefit from the zone data-structure classically used to effectively handle set of regions. We argue in Section 4.4 that the fixed point computation would end up splitting many zones into individual regions, which would obliterate the advantage of using zones instead of regions. Instead, we propose a different algorithm (see Algorithm 1), which is inspired from the Zielonka algorithm, but works directly on the TGA G^+ (and not on the underlying TS) to avoid these pitfalls. In particular, we prove that, in G^+ (under semantics $\mathcal{R}^+$), one can solve synthesis by considering a restricted class of strategies only (Sections 4.2 and 4.4). We exploit these properties to obtain an efficient zone-based algorithm. Finally, from the winning strategies computed in $\mathcal{R}^+(G^+)$, we can extract winning strategies on $\mathcal{R}(G)$.

Throughout this section, we fix a TGA $G = (Q, E, C, U_0, U_1, \alpha)$ over actions $A = A_0 \uplus A_1$, with set of clocks X_z, and maximum color d. We also fix $\mathcal{R}(G) = \mathcal{G} = (\Phi, WC)$. We extend the notion of region to states (q, v). We say that two states $s = (q, v)$ and $s' = (q', v')$ are *region equivalent*, or *in the same region*, written $s \equiv s'$, whenever $q = q'$ and $v \equiv v'$. Then, a *region* is an equivalence class of states wrt $\equiv$. We write $[\![s]\!]_G$ for the region containing s. Those notions extend naturally to plays and histories of the same length.

Zones and zone federations

To manipulate convex sets of clock valuations, we rely on the classical zone data-structure. An *atom* is an inequality of the form $x \bowtie c$ or $x - y \bowtie c$, where $x, y \in X_z$ are clocks of G; $\bowtie \in \{<, \leq, =, \geq, >\}$; and c is an integer. Then, a *zone* is a finite conjunction of atoms. For

[2] As we will see later, "augmented" means that each location contains two extra discrete pieces of information: a color and the identifier of a Player.

[3] $\mathcal{R}^+$ is the same as $\mathcal{R}$ but adapted to the new syntax of G^+.

example, $(x \leq 5) \wedge (y - x \geq 3)$. Given a zone Z, its semantics $[\![Z]\!] \subseteq \text{Vals}(X_z)$ is the set of all valuations that satisfy all the atoms in Z, using the usual semantics for the comparison operators. To manipulate non-convex sets of valuations, we rely on *zone federations* [6], which are finite sets of zones. The semantics of a federation $F = \{Z_1, Z_2, \ldots, Z_n\}$ is the union $[\![F]\!] = [\![Z_1]\!] \cup [\![Z_2]\!] \cup \cdots \cup [\![Z_n]\!]$. Note that guards can be encoded as federations.

Given a federation F, one can compute efficiently [6] a federation $F_{\not\downarrow}$ representing all the valuations that can reach F by letting time elapse, i.e. $[\![F_{\not\downarrow}]\!] = \{v \mid \exists \Delta \in \mathbb{R}_{\geq 0} : v + \Delta \in [\![F]\!]\}$. Similarly, one can compute $Z[X := 0]$ s.t. $[\![Z[X := 0]]\!] = [\![Z]\!][X := 0]$ for a set of clocks X to reset. Given federations F_1 and F_2, one can also compute $F_1 \vee F_2$, $F_1 \wedge F_2$ and $F_1 - F_2$ s.t. $[\![F_1 \vee F_2]\!] = [\![F_1]\!] \cup [\![F_1]\!]$, $[\![F_1 \wedge F_2]\!] = [\![F_1]\!] \cap [\![F_1]\!]$ and $[\![F_1 - F_2]\!] = [\![F_1]\!] \setminus [\![F_2]\!]$. Federations also allow for inclusion and intersection checks. Thus, computing all the valuations of F that satisfy some guard or invariant g encoded into a federation F_g amounts to computing $F \wedge F_g$. We write $\top$ for the universal federation: $[\![\top]\!] = \text{Vals}(X_z)$; and $\bot$ for the empty: $[\![\bot]\!] = \emptyset$. For convenience, we identify a zone to the singleton federation containing it. Given a set of clocks X, $\mathbb{F}(X)$ is the set of all the zone federations one can create with the clocks of X. Federations naturally allow for the encoding of infinite set of states as finite set of pairs (location, federation): $[\![\{(q_1, F_1), (q_2, F_2), \ldots\}]\!] = \{(q, v) \in Q \times \text{Vals} \mid \exists i : q = q_i \wedge v \in [\![F_i]\!]\}$.

4.1 Reduction to a Parity Game

We apply ideas from [9] to reduce the game $\mathcal{R}(G) = (\Phi, WC)$ to a parity game $\mathcal{R}^+(G^+)$. First, we prove that for all strategies σ in $\mathcal{R}(G)$, there exists a strategy σ' that stops at each integer bound of z (called a *tick*) and is winning if σ is winning. A *z-stepped strategy* is a strategy that always plays a delay sufficiently small to prevent z from going beyond its next integer bound. Formally, a strategy σ is a z-stepped Player p's strategy when, for all histories h ending in a state (q, v), we have $\sigma(h) = (\Delta, a)$ with $v_z + \Delta \leq \lfloor v_z \rfloor + 1$. Given a TS Φ, $Z_p^z(\Phi)$ is the set of all z-stepped Player p strategies in Φ. Then:

▶ **Lemma 1** (z-stepped Strategies Are Sufficient). *Let G be a TGA. If a player has a winning strategy σ in $\mathcal{R}(G)$, then one can construct from σ a winning z-stepped strategy σ'.*

Intuitively, σ' is built to mimic σ closely. For example, from a state where $z = 0.5$, σ plays $(1, a)$, then there is a winning strategy σ' that first plays $(0.5, \bot_p)$ to reach $z = 1$, then, if the proposed wait actually happened, plays $(0.5, a)$ to play a in the valuation chosen by σ.

Lemma 1 allows us to incorporate in the arena a modulo 1 behavior of z. Intuitively, in $G^+ = (Q^+, E^+, C^+, U_0^+, U_1^+, \alpha^+)$, z evolves in $[0, 1)$, and is always reset by the only possible action $\circlearrowleft$ whenever it reaches 1. On all locations q, the invariant becomes $C(q) \wedge z \leq 1$ and we add self-loops with action $\circlearrowleft$ that reset z when $z = 1$. We also check that $z < 1$ on all other edges.

Next, we need to encode the blaming mechanism in the colors of the parity game. Whenever there is a tick of z, we see color $2 + c$, where c is the largest color seen since the last tick (in the original game). On all the states where no tick has occurred, we see color $1 - b$, where b is the player to be blamed. So, if the Controller is to be blamed in a time-convergent play, color 1 will be seen infinitely often and the Controller will lose.

More formally, we have $Q^+ = \{(q, c, b) \in Q \times \{0, \ldots, d\} \times \{0, 1, -\} \mid c \geq \alpha(q)\}$. Each location G^+, is thus of the form (q, c, b), where c is the largest color seen since the last tick, and b is the blamed player. The set of actions is $A \cup \{\circlearrowleft\}$. No-wait and coloring functions are lifted to G's locations: for $f \in \{U_0, U_1, \alpha\}$, $f^+((q, c, b)) = f(q)$. Finally, the set of edges is $E^+ = \{((q,\ c,\ b),\ \circlearrowleft,\ z = 1,\ \{z\},\ (q,\ \alpha(q),\ b)) \mid (q,\ c,\ b) \in Q^+\} \cup \{((q, c, b), a, g \wedge z < 1, R, (q', \max(c, \alpha(q')), b')) \mid (q, a, g, R, q') \in E, b' \text{ s.t. } a \in A_{\bot, b'}\}$.

Now that we have defined the arena G^+, we need to associate to it a semantics $\mathcal{R}^+(G^+)$. To this end, we define $\mathcal{R}^+$ as $\mathcal{R}$ except that: (1) players cannot play $(\Delta, \circlearrowleft)$ with $\Delta > 0$ and (2) the coloring function is β^+ defined as $\beta^+((q,c,b),v) = 1-b$ if $v_z < 1$, and $\beta^+((q,c,b),v) = c+2$ if $v_z = 1$. Hence, $\mathcal{R}^+(G^+)$ is a game with a pure parity condition, where the colors have been adapted to let any player that blocks the time lose the game. Further, players cannot play $(\Delta, \circlearrowleft)$ with $\Delta > 0$, which preserves the winner of the game by Lemma 1. Since $\mathcal{R}^+(G^+)$ is built from $\mathcal{R}$, it also incorporates the *refined surprise rule* (that removes the need for a player without an enabled regular action to play), and the *lazy blame policy* (that does not assign blame to a player who has waited long enough to exit the current clock region).

4.2 Further restricting the strategies

Let us now show that the *lazy blame policy* and the *refined surprise rule* allow us to further restrict the family of strategies we can consider to solve synthesis on $\mathcal{R}$.

First, we fix some vocabulary. We say that, in a given state, a delay δ is *short* if we stay in the same region after waiting δ t.u.. Otherwise, a delay Δ is long (we use δ to emphasize that a delay is short). A short wait (long wait, short action, long action) is a move of the form $(\delta, \bot_p)$ (resp. $(\Delta, \bot_p)$, (δ, a), (Δ, a)). Finally, a wait or action is *immediate* when $\delta = 0$, and *delayed* when $\Delta > 0$. Next, tick is a function that associates to all runs $s^0 m_0^1 m_1^1 s^1 \dots$ a sequence of Booleans $(t^i)_{i>0}$ s.t. $t^i = \top$ iff s^i is a tick. Formally, $\mathrm{tick}((q,v),(q',v')) = \top$ iff $(\mathrm{frac}(v'_z) = 0) \wedge (v_z \neq v'_z)$; $\mathrm{tick}(s^0) = \top$; and $\mathrm{tick}(\dots s^{i-1}(m_0^i, m_1^i)s^i) = \mathrm{tick}(s^{i-1}, s^i)$ for all $i \geq 1$. Then, a strategy σ is a *region strategy* whenever, for all pairs of history (ρ_1, ρ_2) s.t. $\rho_1 \equiv \rho_2$ and $\mathrm{tick}(\rho_1) = \mathrm{tick}(\rho_2)$, we have $\mathrm{last}(\rho_1) \xrightarrow{\sigma(\rho_1)} s_1$ and $\mathrm{last}(\rho_2) \xrightarrow{\sigma(\rho_2)} s_2$ with $s_1 \equiv s_2$. We denote the set of region strategies for Player p as $Z_p^{\mathrm{r}}(G)$. Then:

▶ **Lemma 2** (Region Strategies Are Sufficient). *Let G be a TGA. If a player has a winning strategy σ in $\mathcal{R}(G)$, then one can construct from σ a winning region z-stepped strategy σ'.*

The previous lemma implies that if a strategy is winning, then any strategy that differs only in the chosen delays, but not enough to change the regions or ticks, is also winning. Indeed, by Lemma 2, any short delayed move can be replaced by an immediate one:

▶ **Corollary 3.** *If there is a Player p's winning strategy σ, then there is a Player p's winning strategy σ' that never plays short delayed moves.*

Next, we show that *split strategies* are sufficient. Intuitively, split strategies always propose either an immediate action or a long wait. Formally, a Player p's strategy σ is a *split strategy* if for all history h we have $\sigma(h) = (\Delta, a)$ with $(a \neq \bot_p \implies \Delta = 0) \wedge (\Delta > 0 \implies a = \bot_p)$.

We claim that, under the $\mathcal{R}$ semantics, for every winning strategy, there exists a split winning strategy. This might sound surprising since the ability to play the combination of a delay and a move seems to be at the heart of the notion of "surprise" in timed games. Let us explain why, *under our new $\mathcal{R}$ semantics*, it is not the case. First observe that, thanks to Corollary 3, players do not need to play *short actions* with a non-null delay. Next, we need to explain why playing *long actions* are not necessary either thanks to our new definition of the blame. To illustrate this, we consider the TGA on Figure 1a. Recall that the controller has a winning strategy $\sigma_{2^{-i}}$ that consists in playing $(2^{-i}, c)$ at the i-th round until q_1 is reached. That is, the Environment may prevent the Controller to play c, but the Controller is able to retry as long as needed. Consider now the split strategy $\sigma'_{2^{-i}}$ that plays $(2^{-i}, \bot_0)$ while on (q_0, v) with $v_x = 0$, and $(0, c)$ whenever in (q_0, v) with $v_x > 0$. We argue that this strategy is losing with $\mathcal{S}$'s blame policy, but winning in $\mathcal{R}$. Under $\mathcal{S}$, the

Environment wins by playing $(1, \bot_1)$ in $(q_0, v_x = 0)$, and $(0, e_\ell)$ in $(q_0, v_x > 0)$. Indeed, consider the following time-converging play in the outcome of $\sigma_{2^{-i}}$ and $\sigma'_{2^{-i}}$: $(q_0, v_x = 0) \xrightarrow{(0.5,\bot_0),(1,\bot_1)} (q_0, v_x = 0.5) \xrightarrow{(0,c),(0,e_\ell)} (q_0, v_x = 0) \xrightarrow{(0.125,\bot_0),(1,\bot_1)} (q_0, v_x = 0.125) \cdots$, where the non-determinism is always resolved for the Environment. In this play, every other move played by the Controller is a long wait (because we leave the region where the clock x is equal to 0). Under $\mathcal{S}$'s blame policy, the Controller is blamed for the time-convergence and looses the game. Now, consider $\mathcal{R}$'s lazy blame policy which does not blame players that *play a long wait*. Under this semantics, $\sigma'_{2^{-i}}$ is now a winning *split-strategy*. This example shows why, in the $\mathcal{R}$ semantics, we can restrict ourselves to such strategies:

▶ **Lemma 4** (Split Strategies Are Sufficient). *Let G be a TGA. If a player has a winning strategy σ in $\mathcal{R}(G)$, then one can construct from σ a winning split region z-stepped strategy σ'.*

4.3 Synthesis Algorithm

We propose Algorithm 1 to solve the synthesis problem on all TGA, under semantics $\mathcal{R}$. Inspired by Zielonka algorithm, our algorithm is recursive, it computes the winning partition $W = (W_0, W_1)$ of the state space of $\mathcal{R}^+(G^+)$, and thanks to a trivial modification we obtain the winning Controller's strategy from each state of W_0. By construction, from all winning strategies in $\mathcal{R}^+(G^+)$ we can construct a winning strategy in $\mathcal{R}(G)$. We first describe the objects that the algorithm operates on. We then explain the algorithm step by step, providing intuition for the proof of correctness.

Algorithm 1 Synthesis Algorithm for $\mathcal{R}$ semantics, with S the set of states of $\mathcal{R}^+(G^+)$, $[\![k]\!]_{\beta^+} \subseteq S$ the set of states with β^+-color k, and $\textsc{Attractor}(G, j, [\![k]\!]_{\beta^+})$ the attractor for Player j to $[\![k]\!]_{\beta^+}$.

```
1:  function Solve(G)
2:      k ← max(β⁺(S))
3:      j ← k mod 2 ; i ← 1 − j
4:      T_j ← Attractor(G, j, [[k]]_β⁺)
5:      if T_j = S then
6:          W_i ← ∅, W_j ← S
7:          return W
8:      W' ← Solve(G \_j T_j)
9:      if W'_i = ∅ then
10:         W_i ← ∅, W_j ← S
11:         return W
12:     T_i ← Attractor(G, i, W'_i)
13:     W'' ← Solve(G \_i T_i)
14:     W_i ← S \ W''_j, W_j ← W''_j
15:     return W
```

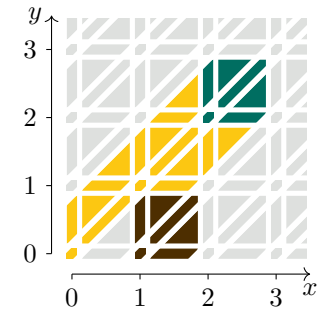


Figure 2 TPred Valuations.

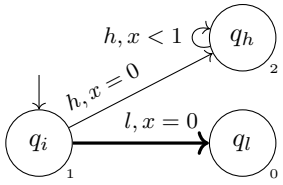


Figure 3 A TGA.

Overview

Algorithm 1 operates on the arena $G^+ = (Q^+, E^+, C^+, U_0^+, U_1^+, \alpha^+)$ and relies on the coloring function β^+, as defined in Section 4.1 (w.l.o.g., we assume that there is at least one state of each color from 0 to the maximum color d). It manipulates infinite sets of states of $\mathcal{R}^+(G^+)$ in the S, T_i and W_i variables. These set of states are symbolically represented as finite sets of pairs $(q, f) \in Q^+ \times \mathbb{F}(X)$, i.e. q is a location and f is a federation.

Like the Zielonka algorithm, Algorithm 1 solves a game by recursively solving subgames. These subgames are constructed so that the winning strategy for one of the player in the subgame remains a valid partial winning strategy in the current game.

The core components of Algorithm 1 are the ATTRACTOR function (lines 4 and 12), and a difference operator $\setminus_j$ (parameterized by a player number j) that reduces the size of the arena before recursive calls (lines 8 and 13). Intuitively, ATTRACTOR(G, j, T) returns all the states from which Player j can force game G to reach the target set of states T.

During each call, j is the player with the highest color in the game and i is his adversary. Hence, initially $j = 1$ iff d is odd and $i = 1 - j$. The attractor T_j is computed for j on the set of all states of maximal color (line 4). If all states of the arena are winning (line 5), the algorithm terminates with $T_j = S = W_j$. Otherwise, a subarena is computed by removing T_j from the initial arena G, and a recursive call is performed (line 8) to compute the winning states *for the other player*. If no states are discovered winning for him (line 9) the algorithm terminates with $T_j = S = W_j$. Otherwise, the attractor to these states T_i is removed and a recursive call is performed (line 13). Since Player i's attractor T_i was removed, we know that states winning for j in the subgame are exactly the states winning for j in the current game (line 14). To initialize this process, the algorithm first computes k, the largest β^+-color in G; j, the player for which k is a good color; and i, the other player. The target of the first attractor is $[\![k]\!]_{\beta^+}$, defined as the set of all states of maximal β^+ color.

In the end, given an initial location q_{in}, the Controller has a winning strategy $\mathcal{R}(G)$ from q_{in} whenever there is $((q_{in}, \alpha(q_{in}), -), F) \in W_0$ s.t. $\mathbf{0}_{X_z} \in [\![F]\!]$.

Attractor Computation

▶ **Definition 5** (Attractor). *The* attractor *for a Player p to a set of target states T is the set of all states from which p has a strategy that guarantees that the game evolves to a state of T (including T). Formally, given a set of states T, a Controller's attractor to T is the set of all states (q, v) s.t. $\exists \sigma_0 : \forall \sigma_1 : \forall \rho \in \mathrm{Out}^{(q,v)}(\sigma_0, \sigma_1)$: ρ visits T.*

An Environment's attractor to T is defined accordingly. It is the set of all states (q, v) s.t. $\forall \sigma_0 : \exists \sigma_1 : \exists \rho \in \mathrm{Out}^{(q,v)}(\sigma_0, \sigma_1)$: ρ visits T.

Given an arena G, a Player p and a set of target states T, the call to ATTRACTOR(G, p, T) returns the attractor for p to T. The efficient use of zones for the attractor computation relies on the properties of semantics $\mathcal{R}$ emphasized in 4.4. In order to compute attractors, we rely on the classical *one-step controllable predecessors*. The *one-step controllable predecessors* for a Player p to a set of target states T is the set of all states from which p will surely get into T in one step, whatever the move of the opponent, and is noted $\mathrm{CPre}_p(T)$. Assuming split strategies, the computation of $\mathrm{CPre}_p(T)$ itself can be broken down into several steps that we describe now.

TPred: Given a game $\mathcal{G}$ with states S, the *temporal predecessor* of a set of target states T avoiding a set of bad states B is, for $p \in \{0, 1\}$: $\mathrm{TPred}_p(T, B) = (T \setminus B) \cup \{s \in S \mid \exists \Delta \in \mathbb{R}_{\geq 0}$ s.t. $s \xrightarrow{(\Delta, \perp_p)} s', s' \in T \wedge (\nexists \Delta' \in \mathbb{R}_{\geq 0}$ s.t. $\Delta' \leq \Delta \wedge s \xrightarrow{(\Delta', \perp_{1-p})} s'' \wedge s'' \in B)\}$. For example, suppose we want to reach $\{(q, v) \mid v \models 2 \leq x < 3 \wedge 2 \leq y < 3\} = T$ (in green on Figure 2), but q has an Environment's outgoing edge with guard $(1 \leq x < 2 \wedge y < 1)$ (in dark) going into a location we want to avoid. So, B is $\{(q, v') \mid v' \models 1 \leq x < 2 \wedge y < 1\}$. Then $\mathrm{TPred}_0(T, B)$ is $\{(s, v'') \mid v'' \in f\}$ with f the set of valuations in yellow and green on the figure.

$\mathrm{TPred}_p(T, B)$ can be computed by manipulating federations of zones. Given a target set T, let $T_{\downarrow} = \{(q, f_{\downarrow}) \mid (q, f) \in T\}$. As shown in [6], $\mathrm{TPred}_p(T, B) = (T \setminus B) \cup (T_{\downarrow} \setminus B_{\downarrow})$ where $T \setminus B = \{(q, f) \in T \mid \nexists (q, f') \in B\} \cup \{(q, f \setminus f') \in T \in T \mid \exists (q, f') \in B\}$.

$\mathbf{DCPre_0}$: The Controller's discrete controllable predecessors of a set T is the set of states s s.t.: $\mathrm{DCPre}_0(T) = \{s \in S \mid (\exists a_0 \in A_0 : s \xrightarrow{0,a_0} s', s' \in T \wedge \nexists a_1 \in A_1 : s \xrightarrow{0,a_1} s'', s'' \notin T) \vee (\neg P_0(s) \wedge \exists a_1 \in A_1 : s \xrightarrow{0,a_1} s'', s'' \in T \wedge \nexists a_1' \in A_1 : s \xrightarrow{0,a_1'} s'', s'' \notin T)\}$. The second part of the disjunction captures the case where the Controller does not play in the $\mathcal{R}$ semantics.

$\mathbf{DCPre_1}$: The Environment's discrete controllable predecessor of a set of target states T is the set of states s s.t.: $\mathrm{DCPre}_1(T) = \{s \in S \mid \exists a_1 \in A_1 : s \xrightarrow{0,a_1} s', s' \in T \vee \forall a_0 \in A_0 : s \xrightarrow{0,a_0} s'', s'' \in T\}$.

$\mathbf{CPre_p}$: Finally, Player p's controllable predecessor of a set of target states T can be computed as: $\mathrm{CPre}_p(T) = \mathrm{TPred}_p(\mathrm{DCPre}_p(T), \mathrm{DCPre}_{1-p}(S \setminus T))$.

Now that we have the computation of $\mathrm{CPre}_p(T)$, we can use a standard fixed point algorithm to obtain the attractor:

▶ **Lemma 6.** *The attractor for a Player p to a set of states T is the closure* $(\mathrm{CPre}_p)^*(T)$.

Subgame Creation

Let us finally explain how we compute subgames with the $\setminus_i$ operator. Let us first explain why this requires careful consideration through the example in Figure 3. Here, the Controller wins by reaching either q_h or q_l. The Controller wins this game by playing $(0, h)$ in q_i: either this move gets played and q_h is reached, or the Environment plays $(0, l)$ and reaches q_l.

The algorithm starts by computing the attractor of the locations with maximal colors, i.e. $\textsc{Attractor}\big(G, 0, (q_h, x = 1)\big)$. This attractor is $(q_h, 0 \leq x \leq 1)$ and does not contain $(q_i, x = 0)$ since the Environment can force the transition from q_i to q_l in this state and thus avoid q_h. Next, the algorithm needs to solve a subgame from which we have removed at least the attractor $(q_h, 0 \leq x \leq 1)$. However, this example shows we need to remove more than simply location q_h and the transition from q_i to q_h. Indeed, in the same game without q_h, the Environment wins by waiting a non-null delay in q_i, which prevents reaching q_l (although this strategy does not make him win in the full game). This shows the difficulty: we want to remove transitions and states from the transition system while the algorithm works on the arena. Our difference operator is defined to take this into account.

Concretely, the $\setminus_i$ operator does not only remove locations and transitions, but also modifies the no-wait functions U_p to disable continuous transitions. Thus, let us first refine the TPred operator to account for the U functions: Every time we write $\mathrm{TPred}_p(T, B)$, we implicitly mean $\mathrm{TPred}_p(T, B \cup \overline{U})$ with $\overline{U} = \{(q, U_p(q)) \mid q \in Q)\} \setminus T$.

Given an attractor T_0 of Player 0, let T' denote the set of states s not in T_0 where there exists an enabled discrete transition of the Controller from s to some state in T_0. (So we know that there is an enabled Environment's discrete transition from s to a state out of T_0.) Then we define: $G \setminus_0 T_0 := (Q, E, C', U_0, U_1, \alpha)$ with $C'(q) = C(q) \setminus f$ if $\exists (q, f) \in T_0$ else $C(q)$, $U_1'(q) = U_1(q) \setminus f$ if $\exists (q, f) \in T'$ else $U_1(q)$. Removing a Player 1's attractor is straightforward. Given an arena $G = (Q, E, C, U_0, U_1, \alpha)$ and a set of states T_1 that is an attractor for Player 1, we define: $G \setminus_1 T_1 := (Q, E, C', U_0, U_1, \alpha)$ with $C'(q) = C(q) \setminus f$ if $\exists (q, f) \in T_1$ else $C(q)$.

We close this section by sketching the main ideas behind the proof of correctness of the algorithm.

▶ **Theorem 7.** *Algorithm 1 terminates and returns the winning sets of both players.*

Termination is established by showing that the depth of recursive calls is bounded, using the classical region construction. The set of regions $Q \times \text{CRegs}(G)$ of a game $\mathcal{R}(G)$ is finite. By definition, the attractor $\textsc{Attractor}(G, j, T)$ never returns an empty set, since it contains T. Thus, each recursive call is performed on a game with at least one region less than the game in the calling procedure, which establishes termination.

The correctness is established inductively. The **induction hypothesis** is that $\textsc{Solve}(G)$ returns a partition (W_0, W_1) of the states of $\mathcal{R}^+(G^+)$ s.t. p has a winning strategy from all states of W_p.

To prove the inductive step, we consider all the return statements. The partition returned on line 7 satisfies the induction hypothesis by properties of the attractor. On line 11, we have a Player j attractor T_j and a j-winning strategy from every state in the subgame $G \setminus_j T_j$. For the final return statement, note that T_i is an i-attractor to a set W_i' constructed s.t. i has a winning strategy from W_i' in G. Thus, T_i is winning for i. Finally, from the states of $G \setminus_i T_i$, if a player has a winning strategy in this subgame, then they also have a winning strategy in G. To prove that the return on line 15 is correct, we rely on the properties of our dedicated difference operator $\setminus_i$.

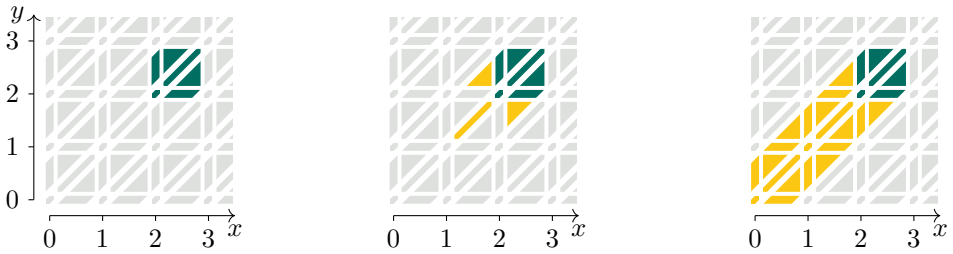
4.4 Efficient zone-based computation thanks to the $\mathcal{R}$ semantics

In this section, we explain why the definition of the $\mathcal{R}$ semantics is essential to obtain efficient zone-based algorithms. As mentioned before, the main takeaway message is that the $\mathcal{R}$ semantics (with the lazy blame policy) allows avoiding splitting zones into too many regions when computing the predecessor operator.

This problem is illustrated in Figure 4. First, remember that, in G^+, the states contain an extra bit indicating which player is to blame. This *discrete* information present in the states will be the source of the difficulties we are about to discuss. Assume we have game with a single location q, and assume we have obtained the symbolic state $(q, 1, T)$, where 1 indicates that the environment has the blame and T is the zone depicted in Figure 4a. First, observe that $\text{CPre}_0(\{(q, 1, T)\}) = \{(q, 1, T)\}$. Indeed, the Controller cannot force, in one step, to reach $(q, 1, T)$ from any other state, because the Controller has never the guarantee to play: either the Environment plays faster, or the non-determinism is resolved in favour of the Environment, who takes the blame (while letting the Controller play would have changed the blame bit to 0). Next, let us consider the computation of the Environment's Controllable Predecessors $\text{CPre}_1(\{(q, 1, T)\})$. In Figure 4b, we can observe that, if we had applied the "classical" blame policy to the $\mathcal{R}$ semantics, $\text{CPre}_1(\{(q, 1, T)\})$ would have a non-convex form, which means that it needs to be represented by a federation containing at least three zones. For example, region $x = 2 \wedge 1 < y < 2$ is not contained in $\text{CPre}_1(\{(q, 1, T)\})$ because the Controller could play a shorter delay than the one proposed by the Environment and take the blame, flipping the blame bit from 1 to 0. Using our lazy blame rule *that does not blame long waits* however preserves the convexity of $\text{CPre}_1(\{(q, 1, T)\})$, as seen in Figure 4c. The lazy blame avoid flipping the blame bit in the states and allows one to compute larger convex sets at once with the CPre operator.

5 Empirical Results

We have a proof of concept implementation which uses UppAal's federation library. We have tested it with artificial examples of varying sizes. We focus on time, as memory usage has remained relatively low in our experiments. These results aim to provide a general idea of the algorithm's behavior rather than measure its effectiveness precisely.



(a) A zone T. With the classic blame, we have $\{(q,1,T)\} = \mathrm{CPre}_0(\{(q,1,T)\})$.

(b) A federation $T'' \supset T$. With the classic blame, $\mathrm{CPre}_1(\{(q,1,T)\}) = \{(q,1,T'')\}$.

(c) A zone $T' \supset T$. With the lazy blame, $\mathrm{CPre}_0(\{(q,1,T)\}) = \mathrm{CPre}_1(\{(q,1,T)\}) = \{(q,1,T')\}$.

Figure 4 The lazy blame simplifies the controllable predecessor computation.

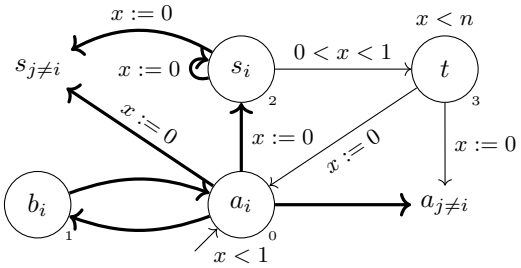


Figure 5 A TGA. Controller owns thin arrows.

Here are the main takeaways. As expected, the algorithm exhibits exponential time growth with respect to the number of clocks, polynomial time growth with respect to the number of distinct guards. We can observe that the algorithm effectively exploits the zone data-structure to capture the structure of the game. The problematic behavior explained in Figure 4b has not been observed. For the sake of brieveity, we now focus on a simple example.

Figure 5 shows a family of arenas. The arena is parametrized by an integer $n \geq 1$. It has $3n+1$ locations and $3n(n+1)$ edges, one clock x with $c_x = n$ and the largest color is 3. The locations are t and $\{a_i, b_i, s_i\}_{i\in\{1,\dots,n\}}$. In this game, the number of locations is growing linearly with n and the number of edges quadratically.

The calculation time is linear in the number of edges, showing that the zone data-structure effectively capture the structure of the game. We tested two variations: one where we change the invariant on t from $x < n$ to $x < 1$, and another where we replace the invariant $x < 1$ on a_i with $x < n$. The first variation (t with $x < n$) yields faster computation than the depicted case. The second variation yields slower computation. Both variations maintain linearity in the number of edges.

Finally, we observed that the tick interval impacts algorithm efficiency. This is easily understood: a tick interval that is too short results in smaller timed controllable predecessors. Experiments seem to indicate that using a tick interval equal to the largest constant found in guards and invariants yields good results. However, the question of how to choose an optimal tick interval, if possible, remains an open issue.

References

1 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

2 Eugene Asarin, Oded Maler, and Amir Pnueli. Symbolic controller synthesis for discrete and timed systems. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II, Proceedings of the Third International Workshop on Hybrid Systems, Ithaca, NY, USA, October 1994*, volume 999 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 1994. doi:10.1007/3-540-60472-3_1.

3 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the value problem in weighted timed games. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 311–324. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.CONCUR.2015.311.

4 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In Paul Pettersson and Wang Yi, editors, *Formal Modeling and Analysis of Timed Systems, Third International Conference, FORMATS 2005, Uppsala, Sweden, September 26-28, 2005, Proceedings*, volume 3829 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2005. doi:10.1007/11603009_5.

5 Thomas Brihaye, Thomas A. Henzinger, Vinayak S. Prabhu, and Jean-François Raskin. Minimum-time reachability in timed games. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 825–837. Springer, 2007. doi:10.1007/978-3-540-73420-8_71.

6 Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005. doi:10.1007/11539452_9.

7 Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In Claire J. Tomlin and Mark R. Greenstreet, editors, *Hybrid Systems: Computation and Control, 5th International Workshop, HSCC 2002, Stanford, CA, USA, March 25-27, 2002, Proceedings*, volume 2289 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2002. doi:10.1007/3-540-45873-5_13.

8 Franck Cassez, Jan Jakob Jessen, Kim Guldstrand Larsen, Jean-François Raskin, and Pierre-Alain Reynier. Automatic synthesis of robust and optimal controllers - an industrial case study. In Rupak Majumdar and Paulo Tabuada, editors, *Hybrid Systems: Computation and Control, 12th International Conference, HSCC 2009, San Francisco, CA, USA, April 13-15, 2009. Proceedings*, volume 5469 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2009. doi:10.1007/978-3-642-00602-9_7.

9 Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR 2003 - Concurrency Theory, 14th International Conference, Marseille, France, September 3-5, 2003, Proceedings*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003. doi:10.1007/978-3-540-45187-7_9.

10 Barbara Di Giampaolo, Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. Safraless procedures for timed specifications. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, volume 6246 of *Lecture Notes in Computer Science*, pages 2–22. Springer, 2010. doi:10.1007/978-3-642-15297-9_2.

11 Laurent Doyen, Gilles Geeraerts, Jean-François Raskin, and Julien Reichert. Realizability of real-time logics. In Joël Ouaknine and Frits W. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings*, volume 5813 of *Lecture Notes in Computer Science*, pages 133–148. Springer, 2009. doi:10.1007/978-3-642-04368-0_12.

12 Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: compact data structure and state-space reduction. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97), December 3-5, 1997, San Francisco, CA, USA*, pages 14–24. IEEE Computer Society, 1997. doi:10.1109/REAL.1997.641265.

13 Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: from timed models to timed implementations. *Formal Aspects Comput.*, 17(3):319–341, 2005. doi:10.1007/S00165-005-0067-8.

14 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.