Time Flies When Looking out of the Window: Timed Games with Window Parity Objectives

James C. A. Main
UMONS – Université de Mons, Belgium

Mickael Randour
F.R.S.-FNRS & UMONS – Université de Mons, Belgium

Jeremy Sproston
University of Torino, Italy

Abstract

The *window mechanism* was introduced by Chatterjee et al. to reinforce mean-payoff and total-payoff objectives with time bounds in two-player turn-based games on graphs [17]. It has since proved useful in a variety of settings, including parity objectives in games [14] and both mean-payoff and parity objectives in Markov decision processes [12].

We study *window parity* objectives in timed automata and timed games: given a bound on the window size, a path satisfies such an objective if, in all states along the path, we see a sufficiently small window in which the smallest priority is even. We show that checking that all time-divergent paths of a timed automaton satisfy such a window parity objective can be done in polynomial space, and that the corresponding timed games can be solved in exponential time. This matches the complexity class of timed parity games, while adding the ability to reason about time bounds. We also consider multi-dimensional objectives and show that the complexity class does not increase. To the best of our knowledge, this is the first study of the window mechanism in a real-time setting.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Window objectives, timed automata, timed games, parity games

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.25

Related Version *Full Version*: `https://arxiv.org/abs/2105.06686` [23]

Funding Research supported by F.R.S.-FNRS under Grant n° F.4520.18 (ManySynth).
Mickael Randour: F.R.S.-FNRS Research Associate.

1 Introduction

Timed automata and games. *Timed automata* [2] are extensions of finite automata with real-valued variables called *clocks*. Clocks increase at the same rate and measure the elapse of time between actions. Transitions are constrained by the values of clocks, and clocks can be reset on transitions. Timed automata are used to model real-time systems [4]. Not all paths of timed automata are meaningful; infinite paths that take a finite amount of time, called *time-convergent* paths, are often disregarded when checking properties of timed automata. Timed automata induce uncountable transition systems. However, many properties can be checked using the region abstraction, which is a finite quotient of the transition system.

Timed automaton games [24], or simply *timed games*, are games played on timed automata: one player represents the system and the other its environment. Players play an infinite amount of rounds: for each round, both players simultaneously present a delay and an action, and the play proceeds according to the fastest move (note that we use paths for automata and plays for games to refer to sequences of consecutive states and transitions). When defining winning conditions for players, convergent plays must be taken in account; we must not allow a player to achieve its objective by forcing convergence but cannot either require a player to force divergence (as it also depends on its opponent). Given an objective as a set of plays, following [20], we declare a play winning for a player if either it is *time-divergent and belongs to the objective*, or it is *time-convergent and the player is not responsible for convergence.*

Parity conditions. The class of ω-regular specifications is widely used (e.g., it can express liveness and safety), and parity conditions are a canonical way of representing them. In (timed) parity games, locations are labeled with a non-negative integer priority and the parity objective is to ensure the smallest priority occurring infinitely often along the path/play is even. Timed games with ω-regular objectives given as parity automata are shown to be solvable in [20]. Furthermore, a reduction from timed parity games to classical turn-based parity games on a graph is established in [19].

Real-timed windows. The parity objective can be reformulated: for all odd priorities seen infinitely often, a smaller even priority must be seen infinitely often. One can see the odd priority as a request and the even one as a response. The parity objective does not specify any *timing constraints* between requests and responses. In applications however, this may not be sufficient: for example, a server should respond to requests in a timely manner.

We revisit the window mechanism introduced by Chatterjee et al. for mean-payoff and total-payoff games [17] and later applied to parity games [14] and to parity and mean-payoff objectives in Markov decision processes [12]: we provide *the first* (to the best of our knowledge) *study of window objectives in the real-time setting.* More precisely, we lift the (resp. direct) fixed window parity objective of [14] to its real-time counterpart, the *(resp. direct) timed window parity objective*, and study it in timed automata and games.

Intuitively, given a non-negative integer bound λ on the window size, the *direct* timed window parity objective requires that *at all* times along a path/play, we see a window of size at most λ such that the smallest priority in this window is even. While time was counted as steps in prior works (all in a discrete setting), *we naturally measure window size using delays* between configurations in real-time models. The *(non-direct)* timed window parity objective is simply a prefix-independent version of the direct one, thus more closely matching the spirit of classical parity: it asks that some suffix satisfies the direct objective.

Contributions. We extend window parity objectives to a dense-time setting, and study both *verification* of timed automata and *realizability* in timed games. We consider adaptations of the *fixed window parity objectives* of [14], where the window size is given as a parameter. We establish that (a) verifying that all time-divergent paths of a timed automaton satisfy a timed window parity specification is PSPACE-complete; and that (b) checking the existence of a winning strategy for a window parity objective in timed games is EXPTIME-complete. These results (Thm. 8) hold for both the direct and prefix-independent variants, and they extend to multi-dimensional objectives, i.e., conjunctions of window parity.

All algorithms are based on a reduction to an *expanded timed automaton* (Def. 4). We establish that, similarly to the discrete case, it suffices to keep track of one window at a time (or one per objective in the multi-dimensional case) instead of all currently open windows, thanks to the so-called *inductive property of windows* (Lem. 3). A window can be summarized using its smallest priority and its current size: we encode the priorities in a window by extending locations with priorities and using an additional clock to measure the window's size. The (resp. direct) timed window parity objective translates to a co-Büchi (resp. safety) objective on the expanded automaton. Locations to avoid for the co-Büchi (resp. safety) objective indicate a window exceeding the supplied bound without the smallest priority of the window being even – a *bad window*. To check that all time-divergent paths of the expanded automaton satisfy the safety (resp. co-Büchi) objective, we check for the existence of a time-divergent path visiting (resp. infinitely often) an unsafe location using the PSPACE algorithm of [2]. To solve the similarly-constructed expanded game, we use the EXPTIME algorithm of [20].

Lower bounds are established by encoding safety objectives on timed automata as (resp. direct) timed window parity objectives. Checking safety properties over time-divergent paths in timed automata is PSPACE-complete [2] and solving safety timed games is EXPTIME-complete [21].

Comparison. Window variants constitute *conservative approximations* of classical objectives (e.g., [17, 14, 12]), strengthening them by enforcing timing constraints. Complexity-wise, the situation is varied. In one-dimension turn-based games on graphs, window variants [17, 14] provide *polynomial-time alternatives* to the classical objectives, bypassing long-standing complexity barriers. However, in multi-dimension games, their complexity becomes worse than for the original objectives: in particular, fixed window parity games are EXPTIME-complete for multiple dimensions [14]. We show that timed games with multi-dimensional window parity objectives are in the same complexity class as untimed ones, i.e., dense time comes for free.

For classical parity objectives, timed games can be solved in exponential time [19, 20]. The approach of [19] is as follows: from a timed parity game, one builds a corresponding turn-based parity game on a graph, the construction being polynomial in the number of priorities and the size of the region abstraction. We recall that despite recent progress on quasi-polynomial-time algorithms (starting with [16]), no polynomial-time algorithm is known for parity games; the blow-up comes from the number of priorities. Overall, *the two sources of blow-up* – region abstraction and number of priorities – *combine in a single-exponential solution* for timed parity games. We establish that (multi-dimensional) window parity games can be solved in time polynomial in the size of the region abstraction, the number of priorities and the window size, and exponential in the number of dimensions. Thus *even for conjunctions of objectives, we match the complexity class of single parity objectives of timed games, while avoiding the blow-up related to the number of priorities and enforcing time bounds between odd priorities and smaller even priorities via the window mechanism.*

Outline. Due to space constraints, we only provide an overview of our work. All technical details, intermediary results and proofs can be found in the full version of this paper [23]. This work is organized as follows. Sect. 2 summarizes all prerequisite notions and vocabulary. In Sect. 3, we introduce the timed window parity objective, compare it to the classical parity objective, and establish some useful properties. Our reduction from window parity objectives to safety or co-Büchi objectives is presented in Sect. 4. Finally, Sect. 5 presents complexity results.

Related work. In addition to the aforementioned foundational works, the window mechanism has seen diverse extensions and applications: e.g., [5, 3, 11, 15, 22, 26, 8]. Window parity games are strongly linked to the concept of *finitary ω-regular games*: see, e.g., [18], or [14] for a complete list of references. The window mechanism can be used to ensure a certain form of (local) guarantee over paths: different techniques have been considered in stochastic models, notably variance-based [10] or worst-case-based [13, 7] methods. Finally, let us recall that game models provide a useful framework for controller synthesis [25], and that timed automata have been extended in a number of directions (see, e.g., [9] and references therein): applications of the window mechanism in such richer models could be of interest.

2 Preliminaries

Timed automata. A clock variable, or *clock*, is a real-valued variable. Let C be a set of clocks. A *clock constraint* over C is a conjunction of formulae of the form $x \sim c$ with $x \in C$, $c \in \mathbb{N}$, and $\sim \in \{\leq, \geq, >, <\}$. We write $x = c$ as shorthand for the clock constraint $x \geq c \wedge x \leq c$. Let $\Phi(C)$ denote the set of clock constraints over C.

Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative real numbers. We refer to functions $\nu \in \mathbb{R}^C_{\geq 0}$ as *clock valuations* over C. A clock valuation ν over C satisfies a clock constraint of the form $x \sim c$ if $\nu(x) \sim c$ and a conjunction $g \wedge h$ if it satisfies both g and h. For a clock constraint g and clock valuation ν, we write $\nu \models g$ if ν satisfies g.

For a clock valuation ν and $d \geq 0$, we let $\nu + d$ be the valuation defined by $(\nu + d)(x) = \nu(x) + d$ for all $x \in C$. For any valuation ν and $D \subseteq C$, we define $\mathsf{reset}_D(\nu)$ to be the valuation agreeing with ν for clocks in $C \setminus D$ and that assigns 0 to clocks in D. We denote by $\mathbf{0}^C$ the zero valuation, assigning 0 to all clocks in C.

A *timed automaton* (TA) is a tuple $(L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$ where L is a finite set of *locations*, $\ell_{\mathsf{init}} \in L$ is an initial location, C a finite set of *clocks* containing a special clock γ which keeps track of the total time elapsed, Σ a finite set of actions, $I \colon L \to \Phi(C)$ an *invariant* assignment function and $E \subseteq L \times \Phi(C) \times \Sigma \times 2^{C \setminus \{\gamma\}} \times L$ an edge relation. We only consider deterministic timed automata, i.e., we assume that in any location ℓ, there are no two outgoing edges $(\ell, g_1, a, D_1, \ell_1)$ and $(\ell, g_2, a, D_2, \ell_2)$ sharing the same action such that the conjunction $g_1 \wedge g_2$ is satisfiable. For an edge (ℓ, g, a, D, ℓ'), the clock constraint g is called the *guard* of the edge.

A TA $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$ gives rise to an uncountable transition system $\mathcal{T}(\mathcal{A}) = (S, s_{\mathsf{init}}, M, \to)$ with the state space $S = L \times \mathbb{R}^C_{\geq 0}$, the initial state $s_{\mathsf{init}} = (\ell_{\mathsf{init}}, \mathbf{0}^C)$, set of actions $M = \mathbb{R}_{\geq 0} \times (\Sigma \cup \{\bot\})$ and the transition relation $\to \subseteq S \times M \times S$ defined as follows: for any action $a \in \Sigma$ and delay $d \geq 0$, we have that $((\ell, \nu), (d, a), (\ell', \nu')) \in \to$ if and only if there is some edge $(\ell, g, a, D, \ell') \in E$ such that $\nu + d \models g$, $\nu' = \mathsf{reset}_D(\nu + d)$, $\nu + d \models I(\ell)$ and $\nu' \models I(\ell')$; for any delay $d \geq 0$, $((\ell, \nu), (d, \bot), (\ell, \nu + d)) \in \to$ if and only if $\nu + d \models I(\ell)$. Let us note that the satisfaction set of clock constraints is convex: it is described by a conjunction of inequalities. Whenever $\nu \models I(\ell)$, the above conditions $\nu + d \models I(\ell)$ (the invariant holds after the delay) are equivalent to requiring $\nu + d' \models I(\ell)$ for all $0 \leq d' \leq d$ (the invariant holds at each intermediate time step).

A *move* is any pair in $\mathbb{R}_{\geq 0} \times (\Sigma \cup \{\bot\})$ (i.e., an action in the transition system). For any move $m = (d, a)$ and states s, $s' \in S$, we write $s \xrightarrow{m} s'$ or $s \xrightarrow{d,a} s'$ as shorthand for $(s, m, s') \in \to$. Moves of the form $(d, \bot)$ are called *delay moves*. We say a move m is enabled in a state s if there is some s' such that $s \xrightarrow{m} s'$. There is at most one successor per move in a state, as we do not allow two guards on edges labeled by the same action to be simultaneously satisfied.

A *path* in a TA $\mathcal{A}$ is a finite or infinite sequence $s_0(d_0, a_0)s_1 \ldots \in S(MS)^* \cup (SM)^\omega$ such that for all j, s_j is a state of $\mathcal{T}(\mathcal{A})$ and for all $j > 0$, $s_{j-1} \xrightarrow{d_{j-1}, a_{j-1}} s_j$ is a transition in $\mathcal{T}(\mathcal{A})$. A path is *initial* if $s_0 = s_{\mathsf{init}}$. For clarity, we write $s_0 \xrightarrow{d_0, a_0} s_1 \xrightarrow{d_1, a_1} \cdots$ instead of $s_0(d_0, a_0)s_1(d_1, a_1) \ldots$.

An infinite path $\pi = (\ell_0, \nu_0) \xrightarrow{d_0, a_0} (\ell_1, \nu_1) \ldots$ is *time-divergent* if the sequence $(\nu_j(\gamma))_{j \in \mathbb{N}}$ is not bounded from above. A path which is not time-divergent is called *time-convergent*; time-convergent paths are traditionally ignored in analysis of timed automata [1, 2] as they model unrealistic behavior. This includes ignoring *Zeno paths*, which are time-convergent paths along which infinitely many actions appear. We write $\mathsf{Paths}(\mathcal{A})$ for the set of paths of $\mathcal{A}$.

Priorities. A *priority function* is a function $p\colon L \to \{0, \dots, d-1\}$ with $d \leq |L|$. We use priority functions to express parity objectives. A *k-dimensional priority function* is a function $p\colon L \to \{0, \dots, d-1\}^k$ which assigns vectors of priorities to locations.

Timed games. We consider two player games played on TAs. We refer to the players as player 1 ($\mathcal{P}_1$) for the system and player 2 ($\mathcal{P}_2$) for the environment. We use the notion of timed automaton games of [20].

A *timed* (automaton) *game* (TG) is a tuple $\mathcal{G} = (\mathcal{A}, \Sigma_1, \Sigma_2)$ where $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$ is a TA and (Σ_1, Σ_2) is a partition of Σ. We refer to actions in Σ_i as $\mathcal{P}_i$ actions for $i \in \{1, 2\}$.

Recall a move is a pair $(d, a) \in \mathbb{R}_{\geq 0} \times (\Sigma \cup \{\bot\})$. Let S denote the set of states of $\mathcal{T}(\mathcal{A})$. In each state $s = (\ell, \nu) \in S$, the moves available to $\mathcal{P}_1$ are the elements of the set $M_1(s)$ where $M_1(s) = \{(d, a) \in \mathbb{R}_{\geq 0} \times (\Sigma_1 \cup \{\bot\}) \mid \exists\, s',\, s \xrightarrow{d,a} s'\}$ which contains moves with $\mathcal{P}_1$ actions and delay moves that are enabled in s. The set $M_2(s)$ is defined analogously with $\mathcal{P}_2$ actions. We write M_1 and M_2 for the set of all moves of $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively.

At each state s along a play, both players simultaneously select a move $(d^{(1)}, a^{(1)}) \in M_1(s)$ and $(d^{(2)}, a^{(2)}) \in M_2(s)$. Intuitively, the fastest player gets to act and in case of a tie, the move is chosen non-deterministically. This is formalized by the *joint destination function* $\delta : S \times M_1 \times M_2 \to 2^S$, defined by

$$\delta(s, (d^{(1)}, a^{(1)}), (d^{(2)}, a^{(2)})) = \begin{cases} \{s' \in S \mid s \xrightarrow{d^{(1)}, a^{(1)}} s'\} & \text{if } d^{(1)} < d^{(2)} \\ \{s' \in S \mid s \xrightarrow{d^{(2)}, a^{(2)}} s'\} & \text{if } d^{(1)} > d^{(2)} \\ \{s' \in S \mid s \xrightarrow{d^{(i)}, a^{(i)}} s', i = 1, 2\} & \text{if } d^{(1)} = d^{(2)}. \end{cases}$$

For $m^{(1)} = (d^{(1)}, a^{(1)}) \in M_1$ and $m^{(2)} = (d^{(2)}, a^{(2)}) \in M_2$, we write $\mathsf{delay}(m^{(1)}, m^{(2)}) = \min\{d^{(1)}, d^{(2)}\}$ to denote the delay occurring when $\mathcal{P}_1$ and $\mathcal{P}_2$ play $m^{(1)}$ and $m^{(2)}$ respectively.

A play is defined similarly to a path: it is a finite or infinite sequence of the form $s_0(m_0^{(1)}, m_0^{(2)})s_1(m_1^{(1)}, m_1^{(2)}) \ldots \in S((M_1 \times M_2)S)^* \cup (S(M_1 \times M_2))^\omega$ where for all indices j, $m_j^{(i)} \in M_i(s_j)$ for $i \in \{1, 2\}$, and for $j > 0$, $s_j \in \delta(s_{j-1}, m_{j-1}^{(1)}, m_{j-1}^{(2)})$. A play is *initial* if $s_0 = s_{\mathsf{init}}$. For a finite play $\pi = s_0 \ldots s_n$, we set $\mathsf{last}(\pi) = s_n$. For an infinite play $\pi = s_0 \ldots$, we write $\pi_{|n} = s_0(m_0^{(0)}, m_0^{(1)}) \ldots s_n$. A play follows a path in the TA, but there need not be a unique path compatible with a play: if along a play, at the nth step, the moves of both players share the same delay and target state, either move can label the nth transition in a matching path.

Similarly to paths, an infinite play $\pi = (\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \cdots$ is *time-divergent* if and only if $(\nu_j(\gamma))_{j \in \mathbb{N}}$ is not bounded from above. Otherwise, we say a play is *time-convergent*. We define the following sets: $\mathsf{Plays}(\mathcal{G})$ for the set of plays of $\mathcal{G}$; $\mathsf{Plays}_{\mathit{fin}}(\mathcal{G})$ for the set of finite plays of $\mathcal{G}$; $\mathsf{Plays}_\infty(\mathcal{G})$ for the set of time-divergent plays of $\mathcal{G}$. We also write $\mathsf{Plays}(\mathcal{G}, s)$ to denote plays starting in a state s of $\mathcal{T}(\mathcal{A})$.

Note that our games are built on *deterministic* TAs. From a modeling standpoint, this is not restrictive, as we can simulate a non-deterministic TA through the actions of $\mathcal{P}_2$.

Strategies. A strategy for $\mathcal{P}_i$ is a function describing which move a player should use based on a play history. Formally, a strategy for $\mathcal{P}_i$ is a function $\sigma_i\colon \mathsf{Plays}_{\mathit{fin}}(\mathcal{G}) \to M_i$ such that for all $\pi \in \mathsf{Plays}_{\mathit{fin}}(\mathcal{G})$, $\sigma_i(\pi) \in M_i(\mathsf{last}(\pi))$. This last condition requires that each move given by a strategy be enabled in the last state of a play.

A play $s_0(m_0^{(1)}, m_0^{(2)})s_1 \ldots$ is said to be consistent with a $\mathcal{P}_i$-strategy σ_i if for all indices j, $m_j^{(i)} = \sigma_i(\pi_{|j})$. Given a $\mathcal{P}_i$-strategy σ_i, we define $\mathsf{Outcome}_i(\sigma_i)$ (resp. $\mathsf{Outcome}_i(\sigma_i, s)$) to be the set of plays (resp. set of plays starting in state s) consistent with σ_i.

Objectives. An objective represents the property we desire on paths of a TA or a goal of a player in a TG. Formally, we define an *objective* as a set $\Psi \subseteq \mathsf{Paths}(\mathcal{A})$ of infinite paths (when studying TAs) or a set $\Psi \subseteq \mathsf{Plays}(\mathcal{G})$ of infinite plays (when studying TGs). An objective is *state-based* (resp. *location-based*) if it depends solely on the sequence of states (resp. of locations) in a path or play. Any location-based objective is state-based.

▶ Remark 1. In the sequel, we present objectives exclusively as sets of plays. Definitions for paths are analogous as all the objectives defined hereafter are state-based.

We use the following classical location-based objectives. The *safety* objective for a set F of locations is the set of plays that never visit a location in F. The *co-Büchi* objective for a set F of locations consists of plays traversing locations in F finitely often. The *parity* objective for a priority function p over the set of locations requires that the smallest priority seen infinitely often is even.

Fix F a set of locations and p a priority function. The aforementioned objectives are formally defined as follows:

- $\mathsf{Safe}(F) = \{(\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \ldots \in \mathsf{Plays}(\mathcal{G}) \mid \forall n,\, \ell_n \notin F\}$;
- $\mathsf{coBüchi}(F) = \{(\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \ldots \in \mathsf{Plays}(\mathcal{G}) \mid \exists j,\, \forall n \geq j,\, \ell_n \notin F\}$;
- $\mathsf{Parity}(p) = \{(\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \ldots \in \mathsf{Plays}(\mathcal{G}) \mid (\liminf_{n\to\infty} p(\ell_n)) \bmod 2 = 0\}$.

Winning conditions. In games, we distinguish objectives and *winning conditions*. We adopt the definition of [20]. Let Ψ be an objective. It is desirable to have victory be achieved in a physically meaningful way: for example, it is unrealistic to have a safety objective be achieved by stopping time. This motivates a restriction to time-divergent plays. However, this requires $\mathcal{P}_1$ to force the divergence of plays, which is not reasonable, as $\mathcal{P}_2$ can stall using delays with zero time units. Thus we also declare winning time-convergent plays where $\mathcal{P}_1$ is *blameless*. Let $\mathsf{Blameless}_1$ denote the set of $\mathcal{P}_1$-blameless plays, which we define in the following way.

Let $\pi = s_0(m_0^{(1)}, m_0^{(2)})s_1 \ldots$ be a (possibly finite) play. We say $\mathcal{P}_1$ is *not responsible* (or not to be blamed) for the transition at step n in π if either $d_n^{(2)} < d_n^{(1)}$ ($\mathcal{P}_2$ is faster) or $d_n^{(1)} = d_n^{(2)}$ and $s_n \xrightarrow{d_n^{(1)}, a_n^{(1)}} s_{n+1}$ does not hold in $\mathcal{T}(\mathcal{A})$ ($\mathcal{P}_2$'s move was selected and did not have the same target state as $\mathcal{P}_1$'s) where $m_n^{(i)} = (d_n^{(i)}, a_n^{(i)})$ for $i \in \{1, 2\}$. The set $\mathsf{Blameless}_1$ is formally defined as the set of infinite plays π such that there is some j such that for all $n \geq j$, $\mathcal{P}_1$ is not responsible for the transition at step n in π.

Given an objective Ψ, we set the winning condition $\mathsf{WC}_1(\Psi)$ for $\mathcal{P}_1$ to be the set of plays

$$\mathsf{WC}_1(\Psi) = (\Psi \cap \mathsf{Plays}_\infty(\mathcal{G})) \cup (\mathsf{Blameless}_1 \setminus \mathsf{Plays}_\infty(\mathcal{G})).$$

Winning conditions for $\mathcal{P}_2$ are defined by exchanging the roles of the players in the former definition. We consider that the two players are adversaries and have opposite objectives, Ψ and $\neg\Psi$ (shorthand for $\mathsf{Plays}(\mathcal{G}) \setminus \Psi$). There may be plays π such that $\pi \notin \mathsf{WC}_1(\Psi)$ and $\pi \notin \mathsf{WC}_2(\neg\Psi)$, e.g., any time-convergent play in which neither player is blameless.

A *winning strategy* for $\mathcal{P}_i$ for an objective Ψ from a state s_0 is a strategy σ_i such that $\mathsf{Outcome}_i(\sigma_i, s_0) \subseteq \mathsf{WC}_i(\Psi)$.

Decision problems. We consider two different problems for an objective Ψ. The first is the *verification problem* for Ψ, which asks given a timed automaton whether all *time-divergent initial* paths satisfy the objective. Second is the *realizability problem*, which asks whether in a timed automaton game with objective Ψ, $\mathcal{P}_1$ has a winning strategy from the initial state.

3 Window objectives

We consider the *fixed window parity* and *direct fixed window parity* problems from [14] and adapt the discrete-time requirements from their initial formulation to dense-time requirements for TAs and TGs. Intuitively, a direct fixed window parity objective for some bound λ requires that at all points along a play or a path, we see a window of size less than λ in which the smallest priority is even. The (non-direct) window parity objective requires that the direct objective holds for some suffix. In the sequel, we drop "fixed" from the name of these objectives.

In this section, we formalize the timed window parity objective in TGs as sets of plays. The definition for paths of TAs is analogous (see Rmk. 1). First, we define the *timed good window objective*, which formalizes the notion of good windows. Then we introduce the timed window parity objective and its direct variant. We compare these objectives to the parity objective and argue that satisfying a window objective implies satisfying a parity objective, and that window objectives do not coincide with parity objectives in general, via an example. We conclude this section by presenting some useful properties of this objective.

For this entire section, we fix a TG $\mathcal{G} = (\mathcal{A}, \Sigma_1, \Sigma_2)$ where $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma_1 \cup \Sigma_2, I, E)$, a priority function $p\colon L \to \{0, \dots, d-1\}$ and a bound $\lambda \in \mathbb{N} \setminus \{0\}$ on the size of windows.

3.1 Definitions

Good windows. A window objective is based on a notion of good windows. Intuitively, a good window for the parity objective is a fragment of a play in which less than λ time units pass and the smallest priority of the locations appearing in this fragment is even.

The timed good window objective encompasses plays in which there is a good window at the start of the play. We formally define the *timed good window (parity) objective* as the set

$$\begin{aligned}\mathsf{TGW}(p, \lambda) = \big\{(\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \ldots \in \mathsf{Plays}(\mathcal{G}) \mid \exists j \in \mathbb{N},\ \min_{0 \leq k \leq j} p(\ell_k) \bmod 2 = 0 \\ \wedge\, \nu_j(\gamma) - \nu_0(\gamma) < \lambda\big\}.\end{aligned}$$

The timed good window objective is a state-based objective.

We introduce some terminology related to windows. Let $\pi = (\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, \nu_1) \ldots$ be an infinite play. We say that the window opened at step n *closes* at step j if $\min_{n \leq k \leq j} p(\ell_k)$ is even and for all $n \leq j' < j$, $\min_{n \leq k \leq j'} p(\ell_k)$ is odd. Note that, in this case, we must have $\min_{n \leq k \leq j} p(\ell_k) = p(\ell_j)$. In other words, a window closes when an even priority smaller than all other priorities in the window is encountered. The window opened at step n is said to *close immediately* if $p(\ell_n)$ is even.

If a window does not close within λ time units, we refer to it as a *bad* window: the window opened at step n is a bad window if there is some $j^\star \geq n$ such that $\nu_{j^\star}(\gamma) - \nu_n(\gamma) \geq \lambda$ and for all $j \geq n$, if $\nu_j(\gamma) - \nu_n(\gamma) < \lambda$, then $\min_{n \leq k \leq j} p(\ell_k)$ is odd.

Direct timed window objective. The direct window parity objective in graph games requires that every suffix of the play belongs to the good window objective. To adapt this objective to a dense-time setting, we must express that *at all times, we have a good window.* We require that this property holds not only at states which appear explicitly along plays, but also in the continuum between them (during the delay within a location). To this end, let us introduce a notation for suffixes of play.

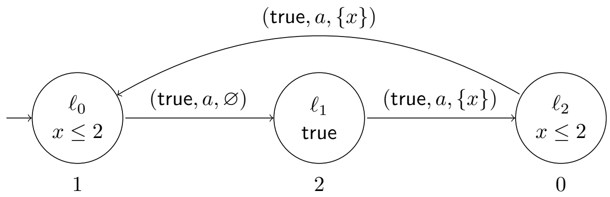

Figure 1 Timed automaton $\mathcal{A}$. Edges are labeled with triples guard-action-resets. Priorities are beneath locations. The initial state is denoted by an incoming arrow with no origin.

Let $\pi = (\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, \nu_1) \ldots \in \mathsf{Plays}(\mathcal{G})$ be a play. For all $i \in \{1, 2\}$ and all $n \in \mathbb{N}$, write $m_n^{(i)} = (d_n^{(i)}, a_n^{(i)})$ and $d_n = \mathsf{delay}(m_n^{(1)}, m_n^{(2)}) = \nu_{n+1}(\gamma) - \nu_n(\gamma)$. For any $n \in \mathbb{N}$ and $d \in [0, d_n]$, let $\pi_{n\rightarrow}^{+d}$ be the *delayed suffix* of π starting in position n delayed by d time units, defined as $\pi_{n\rightarrow}^{+d} = (\ell_n, \nu_n + d)((d_n^{(1)} - d, a_n^{(1)}), (d_n^{(2)} - d, a_n^{(2)}))(\ell_{n+1}, \nu_{n+1})(m_{n+1}^{(1)}, m_{n+1}^{(2)}) \ldots$ If $d = 0$, we write $\pi_{n\rightarrow}$ rather than $\pi_{n\rightarrow}^{+0}$.

Using the notations above, we define the *direct timed window (parity) objective* as the set

$$\mathsf{DTW}(p, \lambda) = \{\pi \in \mathsf{Plays}(\mathcal{G}) \mid \forall n \in \mathbb{N}, \forall d \in [0, d_n], \pi_{n\rightarrow}^{+d} \in \mathsf{TGW}(p, \lambda)\}.$$

The direct timed window objective is state-based: the timed good window objective is state-based and the delays d_n are encoded in states (by clock γ), thus all conditions in the definition of the direct timed window objective depend only the sequence of states of a play.

Timed window objective. We define the *timed window (parity) objective* as a prefix-independent variant of the direct timed window objective. Formally, we let

$$\mathsf{TW}(p, \lambda) = \{\pi \in \mathsf{Plays}(\mathcal{G}) \mid \exists n \in \mathbb{N}, \pi_{n\rightarrow} \in \mathsf{DTW}(p, \lambda)\}.$$

The timed window objective requires the direct timed window objective to hold from some point on. This implies that the timed window objective is state-based.

3.2 Comparison with parity objectives

Both the direct and non-direct timed window objectives reinforce the parity objective with time bounds. It can easily be shown that satisfying the direct timed window objective implies satisfying a parity objective. Any odd priority seen along a play in $\mathsf{DTW}(p, \lambda)$ is answered within λ time units by a smaller even priority. Therefore, should any odd priority appear infinitely often, it is followed by a smaller even priority. As the set of priorities is finite, there must be some smaller even priority appearing infinitely often. This in turn implies that the parity objective is fulfilled. It follows from prefix-independence of the parity objective, that satisfying the non-direct timed window objective implies satisfying the parity objective.

However, in some cases, the timed window objectives may not hold even though the parity objective holds. For simplicity, we provide an example on a TA, rather than a TG. Consider the timed automaton $\mathcal{A}$ depicted in Fig. 1.

All time-divergent paths of $\mathcal{A}$ satisfy the parity objective. We can classify time-divergent paths in two families: either ℓ_2 is visited infinitely often, or from some point on only delay transitions are taken in ℓ_1. In the former case, the smallest priority seen infinitely often is 0 and in the latter case, it is 2.

However, there is a path π such that for all window sizes $\lambda \in \mathbb{N} \setminus \{0\}$, π violates the direct and non-direct timed window objectives. Initialize n to 1. This path can be described by the following loop: play action a in ℓ_0 with delay 0, followed by action a with delay n in ℓ_1 and action a in ℓ_2 with delay 0, increase n by 1 and repeat. The window opened in ℓ_0 only closes when location ℓ_2 is entered. At the n-th step of the loop, this window closes after n time units. As we let n increase to infinity, there is no window size λ such that this path satisfies the direct and non-direct timed window objectives for λ.

This example demonstrates the interest of reinforcing parity objectives with time bounds; we can enforce that there is a bounded delay between an odd priority and a smaller even priority in a path.

3.3 Properties of window objectives

We present several properties of the timed window objective. First, we show that we need only check good windows for non-delayed suffixes $\pi_{n\rightarrow}$. Once this property is introduced, we move on to the inductive property of windows, which is the crux of the reduction in the next section. This inductive property states that when we close a window in less than λ time units all other windows opened in the meantime also close in less than λ time units.

The definition of the direct timed window objective requires checking uncountably many windows. This can be reduced to a countable number of windows: those opened when entering states appearing along a play. Let us explain why no information is lost through such a restriction. We rely on a timeline-like visual representation given in Fig. 2. Consider a window that does not close immediately and is opened in some state of the play delayed by d time units, of the form $(\ell_n, \nu_n + d)$ (depicted by the circle at the start of the bottom line of Fig. 2). This implies that the priority of ℓ_n is odd, otherwise this window would close immediately. Assume the window opened at step n closes at step j (illustrated by the middle line of the figure) in less than λ time units. As the priority of ℓ_n is odd, we must have $j \geq n+1$ (i.e., the window opened at step n is still open as long as ℓ_n is not left). These lines cover the same locations, i.e., the set of locations appearing along the time-frame given by both the dotted and dashed lines coincide. Thus, the window opened d time units after step n closes in at most $\lambda - d$ time units, at the same time as the window opened at step n.

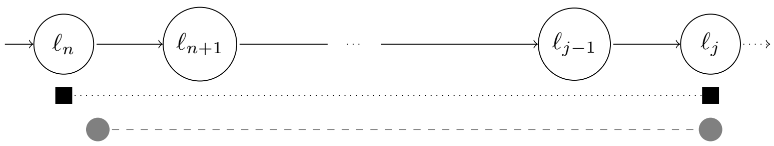

Figure 2 A timeline representation of a play. Circles with labels indicate entry in a location. The dotted line underneath represents a window opened at step n and closed at step j and the dashed line underneath the window opened d time units after step n.

▶ **Lemma 2.** *Let $\pi = (\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \ldots \in \mathsf{Plays}(\mathcal{G})$ and $n \in \mathbb{N}$. Let d_n denote $\mathsf{delay}(m_n^{(1)}, m_n^{(2)})$. Then $\pi_{n\rightarrow} \in \mathsf{TGW}(p, \lambda)$ if and only if for all $d \in [0, d_n]$, $\pi_{n\rightarrow}^{+d} \in \mathsf{TGW}(p, \lambda)$. Furthermore, $\pi \in \mathsf{DTW}(p, \lambda)$ if and only if for all $n \in \mathbb{N}$, $\pi_{n\rightarrow} \in \mathsf{TGW}(p, \lambda)$.*

In turn-based games on graphs, window objectives exhibit an inductive property: when a window closes, all subsequently opened windows close (or were closed earlier) [14]. This is also the case for the timed variant. A window closes when an even priority smaller than all priorities seen in the window is encountered. This priority is also smaller than priorities in all windows opened in the meantime, therefore they must close at this point (if they are not yet closed). We state this property only for windows opened at steps along the run and neglect the continuum in between due to Lem. 2.

▶ **Lemma 3** (Inductive property). *Let* $\pi = (\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, \nu_1) \ldots \in \mathsf{Plays}(\mathcal{G})$. *Let* $n \in \mathbb{N}$. *Assume the window opened at step* n *closes at step* j. *Then, for all* $n \leq i \leq j$, $\pi_{i\to} \in \mathsf{TGW}(p, \lambda)$.

It follows from this inductive property that it suffices to keep track of one window at a time when checking whether a play satisfies the (direct) timed window objective.

4 Reduction

We establish that the realizability (resp. verification) problem for the direct/non-direct timed window parity objective can be reduced to the realizability (resp. verification) problem for safety/co-Büchi objectives on an expanded TG (resp. TA). Our reduction uses the same construction of an expanded TA for both the verification and realizability problems. A state of the expanded TA describes the status of a window, allowing the detection of bad windows. Due to space constraints, we only describe the reduction here and sketch the main technical hurdles along the way. The full approach, including intermediate results, is presented in the full version of our paper [23].

The sketch is as follows. First, we describe how a TA can be expanded with window-related information. Second, we show that time-divergent plays in a TG and its expansion can be related, by constructing two (non-bijective) mappings, in a manner such that a time-divergent play in the base TG satisfies the direct/non-direct timed window parity objective if and only if its related play in the expanded TG satisfies the safety/co-Büchi objective. These results developed for plays are (indirectly) applied to paths in order to show the correctness of the reduction for the verification problem. Third, we establish that the mappings developed in the second part can be leveraged to translate strategies in TGs, and prove that the presented translations preserve winning strategies, proving correctness of the reduction for TGs.

For this section, we fix a TG $\mathcal{G} = (\mathcal{A}, \Sigma_1, \Sigma_2)$ with TA $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$, a priority function p and a bound λ on the size of windows.

Encoding the objective in an automaton. The inductive property (Lem. 3) implies that it suffices to keep track of one window at a time when checking a window objective. Following this, we encode the status of a window in the TA.

A window can be summarized by two characteristics: the lowest priority within it and for how long it has been open. To keep track of the first trait, we encode the lowest priority seen in the current window in locations of the TA. An expanded location is a pair (ℓ, q) where $q \in \{0, \ldots, d-1\}$; the number q represents the smallest priority in the window currently under consideration. We say a pair (ℓ, q) is an even (resp. odd) location if q is even (resp. odd). To measure how long a window is opened, we use an additional clock $z \notin C$ that does not appear in $\mathcal{A}$. This clock is reset whenever a new window opens or a bad window is detected.

The focus of the reduction is over time-divergent plays. Some time-convergent plays may violate a timed good window objective without ever seeing a bad window, e.g., when time does not progress up to the supplied window size. Along time-divergent plays however, the lack of a good window at any point equates to the presence of a bad window. We encode the (resp. direct) timed window objective as a co-Büchi (resp. safety) objective. Locations to avoid in both cases indicate bad windows and are additional expanded locations (ℓ, bad), referred to as *bad locations*. We introduce two new actions β_1 and β_2, one per player, for entering and exiting bad locations. While only the action β_1 is sufficient for the reduction to be correct, introducing two actions allows for a simpler correctness proof in the case of TGs; we can exploit the fact that $\mathcal{P}_2$ can enter and exit bad locations.

It remains to discuss how the initial location, edges and invariants of an expanded TA are defined. We discuss edges and invariants for each type of expanded location, starting with even locations, then odd locations and finally bad locations. Each rule we introduce hereafter is followed by an application on an example. We depict the TA of Fig. 1 and the reachable fragment of its expansion in Fig. 3 and use these TAs for our example. For this explanation, we use the terminology of TAs (paths) rather than that of TGs (plays).

The initial location of an expanded TA encodes the window opened at the start of an initial path of the original TA. This window contains only a single priority, that is the priority of the initial location of the original TA. Thus, the initial location of the expanded TA is the expanded location $(\ell_{\mathsf{init}}, p(\ell_{\mathsf{init}}))$. In our example, the initial location of the expanded TA is $(\ell_0, 1)$.

Even expanded locations encode windows that are closed and do not need to be monitored anymore. Therefore, the invariant of an even expanded location is unchanged from the invariant of the original location in the original TA. Similarly, we do not add any additional constraints on the edges leaving even expanded locations. Leaving an even expanded location means opening a new window: any edge leaving an even expanded location has an expanded location of the form $(\ell, p(\ell))$ as its target ($p(\ell)$ is the only priority in the new window) and resets z to start measuring the size of the new window. For example, the edge from $(\ell_2, 0)$ to $(\ell_0, 1)$ of the expanded TA of Fig. 3 is derived from the edge from ℓ_2 to ℓ_0 of the original TA.

Odd expanded locations represent windows that are still open. The clock z measures how long a window has been opened. If z reaches λ in an odd expanded location, that equates to a bad window in the original TA. In this case, we force time-divergent paths of the expanded TA to visit a bad location. This is done in three steps. We strengthen the invariant of odd expanded locations to prevent z from exceeding λ. We also disable the edges that leave odd expanded locations and do not go to a bad location whenever $z = \lambda$ holds, by reinforcing the guards of such edges by $z < \lambda$. Finally, we include two edges to a bad location (one per additional action β_1 and β_2), which can only be used whenever there is a bad window, i.e., when $z = \lambda$. In the case of our example, if z reaches λ in $(\ell_0, 1)$, we redirect the path to location (ℓ_0, bad), indicating a window has not closed in time in ℓ_0. When z reaches λ in $(\ell_0, 1)$, no more non-zero delays are possible, the edge from $(\ell_0, 1)$ to $(\ell_1, 1)$ is disabled and only the edges to (ℓ_0, bad) are enabled.

When leaving an odd expanded location using an edge, assuming we do not go to a bad location, the smallest priority of the window has to be updated. The new smallest priority is the minimum between the smallest priority of the window prior to traversing the edge and the priority of the target location. In our example for instance, the edge from $(\ell_1, 1)$ to $(\ell_2, 0)$ is derived from the edge from ℓ_1 to ℓ_2 in the original TA. As the priority of ℓ_2 is 0 and is smaller than the current smallest priority of the window encoded by location $(\ell_1, 1)$, the smallest priority of the window is updated to $0 = \min\{1, p(\ell_2)\}$ when traversing the edge. Note that we do not reset z despite the encoded window closing upon entering $(\ell_2, 0)$: the value of z does not matter while in even locations, thus there is no need for a reset when closing the window.

A bad location (ℓ, bad) is entered whenever a bad window is detected while in location ℓ. Bad locations are equipped with the invariant $z = 0$ preventing the passage of time. In this way, for time-divergent paths, a new window is opened immediately after a bad window is detected. For each additional action β_1 and β_2, we add an edge exiting the bad location. Edges leaving a bad location (ℓ, bad) have as their target the expanded location $(\ell, p(\ell))$; we reopen a window in the location in which a bad window was detected. The clock z is not reset by these edges, as it was reset prior to entering the bad location and the invariant

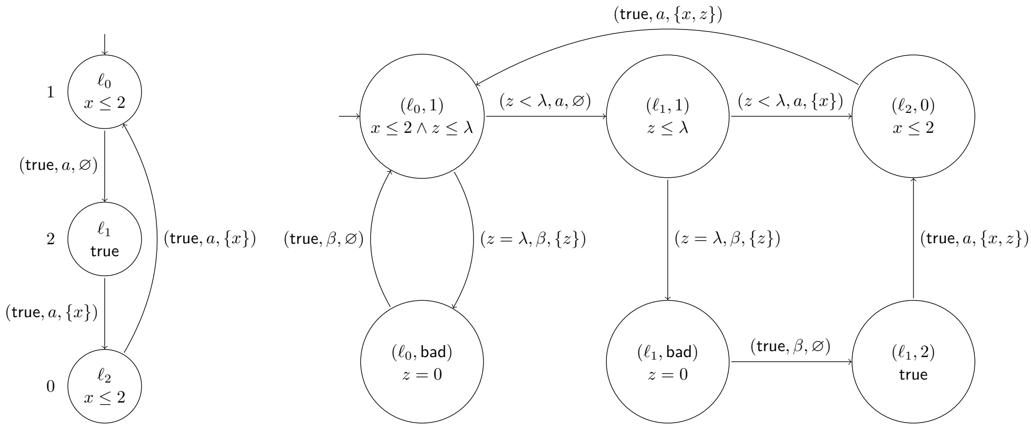

Figure 3 The TA of Fig. 1 (left) and the reachable fragment of its expansion (right). We write β for actions β_1 and β_2.

$z = 0$ prevents any non-zero delay in the bad location. For instance, the edges from (ℓ_1, bad) to $(\ell_1, 2)$ in our example represent that when reopening while in location ℓ_1, the smallest priority of this window is $p(\ell_1) = 2$.

The expansion depends on the priority function p and the bound on the size of windows λ. Therefore, we write $\mathcal{A}(p, \lambda)$ for the expansion. The formal definition of $\mathcal{A}(p, \lambda)$ follows.

▶ **Definition 4.** *Given a TA $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$, the TA $\mathcal{A}(p, \lambda)$ is defined to be the TA $(L', \ell'_{\mathsf{init}}, C', \Sigma', I', E')$ such that*

- $L' = L \times (\{0, \ldots, d-1\} \cup \{\mathsf{bad}\})$*;*
- $\ell'_{\mathsf{init}} = (\ell_{\mathsf{init}}, p(\ell_{\mathsf{init}}))$*;*
- $C' = C \cup \{z\}$ *where* $z \notin C$ *is a new clock;*
- $\Sigma' = \Sigma \cup \{\beta_1, \beta_2\}$ *is an expanded set of actions with special actions* β_1*,* $\beta_2 \notin \Sigma$ *for bad locations;*
- $I'(\ell, q) = I(\ell)$ *for all* $\ell \in L$ *and even* $q \in \{0, \ldots, d-1\}$*,* $I'(\ell, q) = (I(\ell) \wedge z \leq \lambda)$ *for all* $\ell \in L$ *and odd* $q \in \{0, \ldots, d-1\}$*, and* $I'(\ell, \mathsf{bad}) = (z = 0)$ *for all* $\ell \in L$*;*
- *the set of edges* E' *of* $\mathcal{A}(p, \lambda)$ *is the smallest set satisfying the following rules:*
  - *if* q *is even and* $(\ell, g, a, D, \ell') \in E$*, then* $((\ell, q), g, a, D \cup \{z\}, (\ell', p(\ell'))) \in E'$*;*
  - *if* q *is odd and* $(\ell, g, a, D, \ell') \in E$*, then* $((\ell, q), (g \wedge z < \lambda), a, D, (\ell', \min\{q, p(\ell')\})) \in E'$*;*
  - *for all* $\ell \in L$*, odd* q *and* $\beta \in \{\beta_1, \beta_2\}$*,* $((\ell, q), (z = \lambda), \beta, \{z\}, (\ell, \mathsf{bad})) \in E'$ *and* $((\ell, \mathsf{bad}), \mathsf{true}, \beta, \varnothing, (\ell, p(\ell)) \in E'$*.*

For a TG $\mathcal{G} = (\mathcal{A}, \Sigma_1, \Sigma_2)$*, we set* $\mathcal{G}(p, \lambda) = (\mathcal{A}(p, \lambda), \Sigma_1 \cup \{\beta_1\}, \Sigma_2 \cup \{\beta_2\})$*.*

We write $(\ell, q, \bar{\nu})$ for states of $\mathcal{T}(\mathcal{A}(p, \lambda))$ instead of $((\ell, q), \bar{\nu})$, for conciseness. The bar over the valuation is a visual indicator of the different domain. We write $\mathsf{Bad} = L \times \{\mathsf{bad}\}$ for the set of bad locations.

Expanding and projecting plays. A bijection between the set of plays of $\mathcal{G}$ and the set of plays of $\mathcal{G}(p, \lambda)$ cannot be achieved naturally due to the additional information encoded in the expanded TA, notably the presence of bad locations. We illustrate this by showing there are some plays of $\mathcal{G}(p, \lambda)$ that are intuitively indistinguishable if seen as plays of $\mathcal{G}$.

Consider the initial location ℓ_{init} of $\mathcal{G}$, and assume that its priority is odd and its invariant is true. Consider the initial play $\bar{\pi}_1$ of $\mathcal{G}(p, \lambda)$ where the actions β_i are used by both players with a delay of λ at the start of the play and then only delay moves are taken in the reached bad location, i.e., $\bar{\pi}_1 = (\ell, p(\ell), \mathbf{0}^{C\cup\{z\}})((\lambda,\beta_1),(\lambda,\beta_2))\big((\ell, \mathsf{bad}, \bar{\nu})((0,\bot),(0,\bot))\big)^\omega$, where $\bar{\nu}(x) = \lambda$ for all $x \in C$ and $\bar{\nu}(z) = 0$. As the actions β_i and z do not exist in $\mathcal{G}$, $\bar{\pi}_1$ cannot be discerned from the similar play $\bar{\pi}_2$ of $\mathcal{G}(p,\lambda)$ where instead of using the actions β_i, delay moves were used instead, i.e., $\bar{\pi}_2 = (\ell, p(\ell), \mathbf{0}^{C\cup\{z\}})((\lambda,\bot),(\lambda,\bot))((\ell, p(\ell), \bar{\nu}')((0,\bot),(0,\bot)))^\omega$ with $\bar{\nu}'(x) = \lambda$ for all $x \in C \cup \{z\}$.

This motivates using two mappings instead of a bijection. Using the expansion mapping $\mathsf{Ex}\colon \mathsf{Plays}(\mathcal{G}) \to \mathsf{Plays}(\mathcal{G}(p,\lambda))$ and projection mapping $\mathsf{Pr}\colon \mathsf{Plays}(\mathcal{G}(p,\lambda)) \to \mathsf{Plays}(\mathcal{G})$ defined in the full paper [23], we obtain the following equivalence.

▶ **Theorem 5.** *Let $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$ be a TA, p a priority function and $\lambda \in \mathbb{N} \setminus \{0\}$. All time-divergent paths of $\mathcal{A}$ satisfy the (resp. direct) timed window objective if and only if all time-divergent paths of $\mathcal{A}(p,\lambda)$ satisfy the co-Büchi (resp. safety) objective over bad locations.*

Translating strategies. We translate $\mathcal{P}_1$-strategies of the expanded TG to $\mathcal{P}_1$-strategies of the original TG by evaluating the strategy on the expanded TG on expansions of plays provided by the expansion mapping and by replacing any occurrences of β_1 by $\bot$. The fact that translating a winning strategy of the expanded TG this way yields a winning strategy of the original TG is not straightforward. When we translate a winning strategy $\bar{\sigma}$ of $\mathcal{G}(p,\lambda)$ to a strategy σ of $\mathcal{G}$, the expansion of an outcome π of σ may not be consistent with $\bar{\sigma}$, e.g., moves of the form (d, β_1) may appear along $\mathsf{Ex}(\pi)$ in places where $\bar{\sigma}$ suggests otherwise, therefore, it cannot be inferred from the fact that $\bar{\sigma}$ is winning that $\mathsf{Ex}(\pi)$ satisfies the winning condition. We circumvent this issue by constructing another play $\bar{\pi}$ in parallel that is consistent with $\bar{\sigma}$ and shares the same sequence of states as $\mathsf{Ex}(\pi)$. This ensures, by the properties of the expansion mapping, that all time-divergent outcomes of σ are also winning. Furthermore, if the constructed play $\bar{\pi}$ is $\mathcal{P}_1$-blameless and time-convergent, then π also is $\mathcal{P}_1$-blameless, ensuring that all time-convergent outcomes of σ are winning.

In the other direction, we translate $\mathcal{P}_1$-strategies of the original TG to strategies of the expanded TG using the projection mapping and by replacing moves that are illegal in the expansion with suitable moves of the form (d, β_1). The technical difficulties for this translation are similar to those encountered in the first one and are handled similarly. From these two translations, we obtain the following equivalence.

▶ **Theorem 6.** *Let s_{init} be the initial state of $\mathcal{G}$ and $\bar{s}_{\mathsf{init}}$ be the initial state of $\mathcal{G}(p,\lambda)$. There is a winning strategy σ for $\mathcal{P}_1$ for the objective* $\mathsf{TW}(p,\lambda)$ *(resp.* $\mathsf{DTW}(p,\lambda)$*) from s_{init} in $\mathcal{G}$ if and only if there is a winning strategy $\bar{\sigma}$ for $\mathcal{P}_1$ for the objective* $\mathsf{coBüchi}(\mathsf{Bad})$ *(resp.* $\mathsf{Safe}(\mathsf{Bad})$*) from $\bar{s}_{\mathsf{init}}$ in $\mathcal{G}(p,\lambda)$.*

Multi-dimensional objectives. The construction of the expanded TA $\mathcal{A}(p,\lambda)$ can be extended to handle *generalized (resp. direct) timed window objectives*, defined as conjunctions of several (resp. direct) timed window objectives. Generalized objectives are given by k-dimensional priority functions and vectors $\lambda \in (\mathbb{N} \setminus \{0\})^k$ of bounds on the size of windows for each dimension. To keep track of the windows for each dimension, locations are expanded using k-dimensional vectors and one new clock per objective is added to the expanded automaton. The expanded TA in this case is of size exponential in k. The objective on the expansion is a co-Büchi or safety objective, even for multiple objectives. Due to space constraints, we omit the full generalization, which can be found in the full paper [23].

5 Algorithms and complexity

This section outlines algorithms for solving the verification and realizability problems for generalized (resp. direct) timed window parity objectives. We consider the general multi-dimensional setting and we denote by k the number of timed window parity objectives under consideration. Technical details and a comparison with timed parity games can be found in the full paper [23].

Algorithms. Our solution to the verification and realizability problem for generalized (resp. direct) timed window objectives consists of a reduction to a co-Büchi (resp. safety) objective on an expanded automaton. The following lemma establishes the time necessary for the reduction.

▶ **Lemma 7.** *Let $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$ be a TA. Let $p\colon L \to \{0, \ldots, d-1\}^k$ be a k-dimensional priority function and $\lambda \in (\mathbb{N} \setminus \{0\})^k$ be a vector of bounds on window sizes. The expanded TA $\mathcal{A}(p, \lambda) = (L', \ell'_{\mathsf{init}}, C', \Sigma', I', E')$ can be computed in time exponential in k and polynomial in the size of L, the size of E, the size of C, d, the length of the encoding of the clock constraints of $\mathcal{A}$ and the encoding of λ.*

Let $\mathcal{G}$ be a TG. To solve the realizability problem on the expanded TG $\mathcal{G}(p, \lambda)$, we use the algorithm of [20], which checks the existence of a winning strategy for $\mathcal{P}_1$ for location-based objectives specified by a deterministic parity automaton. Using this algorithm, we can check the existence of a winning strategy for $\mathcal{P}_1$ for the co-Büchi (resp. safety) objective on $\mathcal{G}(p, \lambda)$ in time $\mathcal{O}\big(\big(|L|^2 \cdot (d^k+1)^2 \cdot (|C|+k)! \cdot 2^{|C|+k} \prod_{x \in C}(2c_x+1) \cdot \prod_{1 \leq i \leq k}(2\lambda_i+1)\big)^4\big)$, where L is the set of locations of $\mathcal{G}$, C the set of clocks of $\mathcal{G}$ and c_x is the largest constant to which $x \in C$ is compared to in $\mathcal{G}$ (recall that λ_i refers to the bound on window sizes for the ith dimension). This establishes membership of the realizability problem in EXPTIME as the construction of the expanded game can be done in exponential time by Lem. 7.

Let us move on to the verification problem. For one dimension, the verification problem for the (resp. direct) timed window objective is reducible in polynomial time to the verification problem for co-Büchi (resp. safety) objectives by Lem. 7. This establishes membership in PSPACE for the case of a single dimension. For multiple dimensions, the single-dimensional case can be used as an oracle to check that, for each individual objective, all time-divergent paths belong to the objective. This shows membership of the verification problem for generalized objectives in $\mathsf{P}^{\mathsf{PSPACE}} = \mathsf{PSPACE}$ [6].

Lower bounds. Lower bounds are established by reducing the verification/realizability problem for safety objectives to the verification/realizability problem for (resp. direct) timed window parity objectives. The verification problem for safety objectives is PSPACE-complete [2] and the realizability problem for safety objectives is EXPTIME-complete [21].

The reduction is as follows. Given a TA $\mathcal{A}$ and a set F of locations to avoid, we construct a TA $\mathcal{A}'$ obtained by extending locations of $\mathcal{A}$ with a Boolean indicating whether F has been visited. We assign an odd priority to expanded locations that indicate F has been visited, such that windows can no longer close if F has been visited.

Complexity wrap-up. We summarize our complexity results in the following theorem.

▶ **Theorem 8.** *The verification problem for the (direct) generalized timed window parity objective is PSPACE-complete and the realizability problem for the (direct) generalized timed window parity objective is EXPTIME-complete.*

References

1 Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993. doi:10.1006/inco.1993.1024.

2 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.

3 Christel Baier. Reasoning about cost-utility constraints in probabilistic models. In Mikolaj Bojanczyk, Slawomir Lasota, and Igor Potapov, editors, *Reachability Problems – 9th International Workshop, RP 2015, Warsaw, Poland, September 21-23, 2015, Proceedings*, volume 9328 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2015. doi:10.1007/978-3-319-24537-9_1.

4 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

5 Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Weight monitoring with linear temporal logic: complexity and decidability. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14–18, 2014*, pages 11:1–11:10. ACM, 2014. doi:10.1145/2603088.2603162.

6 Theodore P. Baker, John Gill, and Robert Solovay. Relativizations of the P =? NP question. *SIAM J. Comput.*, 4(4):431–442, 1975. doi:10.1137/0204037.

7 Raphaël Berthon, Mickael Randour, and Jean-François Raskin. Threshold constraints with guarantees for parity objectives in Markov decision processes. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.121.

8 Benjamin Bordais, Shibashis Guha, and Jean-François Raskin. Expected window mean-payoff. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 32:1–32:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.32.

9 Patricia Bouyer, Thomas Brihaye, Mickael Randour, Cédric Rivière, and Pierre Vandenhove. Decisiveness of stochastic systems and its application to hybrid models. In Jean-François Raskin and Davide Bresolin, editors, *Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2020, Brussels, Belgium, September 21-22, 2020*, volume 326 of *EPTCS*, pages 149–165, 2020. doi:10.4204/EPTCS.326.10.

10 Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Trading performance for stability in Markov decision processes. *J. Comput. Syst. Sci.*, 84:144–170, 2017. doi:10.1016/j.jcss.2016.09.009.

11 Tomás Brázdil, Vojtech Forejt, Antonín Kucera, and Petr Novotný. Stability in graphs and games. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 10:1–10:14. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.10.

12 Thomas Brihaye, Florent Delgrange, Youssouf Oualhadj, and Mickael Randour. Life is random, time is not: Markov decision processes with window objectives. *Log. Methods Comput. Sci.*, 16(4), 2020. URL: https://lmcs.episciences.org/6975.

13 Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Inf. Comput.*, 254:259–295, 2017. doi:10.1016/j.ic.2016.10.011.

14 Véronique Bruyère, Quentin Hautem, and Mickael Randour. Window parity games: an alternative approach toward parity games with time bounds. In Domenico Cantone and Giorgio Delzanno, editors, *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016*, volume 226 of *EPTCS*, pages 135–148, 2016. doi:10.4204/EPTCS.226.10.

15 Véronique Bruyère, Quentin Hautem, and Jean-François Raskin. On the complexity of heterogeneous multidimensional games. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.11.

16 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263. ACM, 2017. doi:10.1145/3055399.3055409.

17 Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at mean-payoff and total-payoff through windows. *Inf. Comput.*, 242:25–52, 2015. doi:10.1016/j.ic.2015.03.010.

18 Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. Finitary winning in omega-regular games. *ACM Trans. Comput. Log.*, 11(1):1:1–1:27, 2009. doi:10.1145/1614431.1614432.

19 Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. *Log. Methods Comput. Sci.*, 7(4), 2011. doi:10.2168/LMCS-7(4:8)2011.

20 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.

21 Thomas A. Henzinger and Peter W. Kopke. Discrete-time control for rectangular hybrid automata. *Theor. Comput. Sci.*, 221(1-2):369–392, 1999. doi:10.1016/S0304-3975(99)00038-9.

22 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Looking at mean payoff through foggy windows. *Acta Inf.*, 55(8):627–647, 2018. doi:10.1007/s00236-017-0304-7.

23 James C. A. Main, Mickael Randour, and Jeremy Sproston. Time flies when looking out of the window: Timed games with window parity objectives. *CoRR*, abs/2105.06686, 2021. arXiv:2105.06686.

24 Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In Ernst W. Mayr and Claude Puech, editors, *STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 2-4, 1995, Proceedings*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 1995. doi:10.1007/3-540-59042-0_76.

25 Mickael Randour. Automated synthesis of reliable and efficient systems through game theory: A case study. In *Proc. of ECCS 2012*, Springer Proceedings in Complexity XVII, pages 731–738. Springer, 2013. doi:10.1007/978-3-319-00395-5_90.

26 Stéphane Le Roux, Arno Pauly, and Mickael Randour. Extending finite-memory determinacy by Boolean combination of winning conditions. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 38:1–38:20. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.38.