Timed Games with Bounded Window Parity Objectives

James C. A. Main [1] Mickael Randour [1] Jeremy Sproston [2]

[1]UMONS – Université de Mons and F.R.S.-FNRS, Belgium

[2]Università degli Studi di Torino, Italy

FORMATS 2022 – September 14, 2022

Talk overview

- Window objectives reinforce long-term objectives (e.g., mean-payoff objectives, parity objectives) with timing constraints.
- We study window parity objectives in a continuous-time setting.

[1] Main, Randour, and Sproston, "Time Flies When Looking out of the Window: Timed Games with Window Parity Objectives", CONCUR 2021.

Talk overview

- Window objectives reinforce long-term objectives (e.g., mean-payoff objectives, parity objectives) with timing constraints.
- We study window parity objectives in a continuous-time setting.

Intuition of bounded window parity objectives

- A good window for the parity objective is a time frame of size less than λ such that the smallest priority in this time frame is even.

[1] Main, Randour, and Sproston, "Time Flies When Looking out of the Window: Timed Games with Window Parity Objectives", CONCUR 2021.

Talk overview

- Window objectives reinforce long-term objectives (e.g., mean-payoff objectives, parity objectives) with timing constraints.
- We study window parity objectives in a continuous-time setting.

Intuition of bounded window parity objectives

- A good window for the parity objective is a time frame of size less than λ such that the smallest priority in this time frame is even.
- The bounded timed window parity objective requires that there exists a λ s. t., from all configurations of a run, there is a good window for λ.

[1] Main, Randour, and Sproston, "Time Flies When Looking out of the Window: Timed Games with Window Parity Objectives", CONCUR 2021.

Talk overview

- Window objectives reinforce long-term objectives (e.g., mean-payoff objectives, parity objectives) with timing constraints.
- We study window parity objectives in a continuous-time setting.

Intuition of bounded window parity objectives

- A good window for the parity objective is a time frame of size less than λ such that the smallest priority in this time frame is even.
- The bounded timed window parity objective requires that there exists a λ s. t., from all configurations of a run, there is a good window for λ.

- Fixed timed window parity objectives were first studied in [MRS21][1].
- We discuss the complexity of verification and realizability for bounded objectives.

[1] Main, Randour, and Sproston, "Time Flies When Looking out of the Window: Timed Games with Window Parity Objectives", CONCUR 2021.

Table of contents

Table of contents

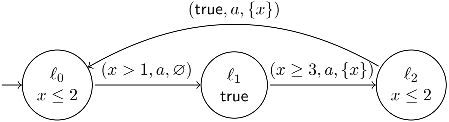
Timed automata

Introduction

- Timed automata [AD94][2] are used to model real-time systems.

A timed automaton consists of:

- a finite set C of clocks progressing at the same rate;
- a finite set of locations L constrained by invariants;
- a finite set of edges E labelled by actions, guards and clock resets.



- Guards and invariants are given by conjunctions of conditions of the form $x \leq c$, $x < c$, $x \geq c$ and $x > c$ where x is a clock and $c \in \mathbb{N}$.

[2] Alur and Dill, "A Theory of Timed Automata", TCS 1994.

Timed automata

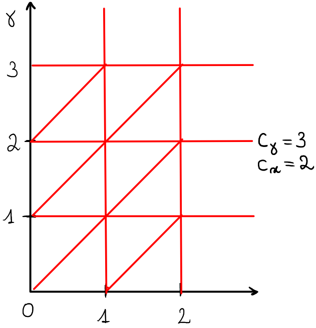
Semantics

A timed automaton gives rise to an uncountable transition system.

- The state space S of this transition system consists of pairs of locations and clock valuations (mappings $C \to \mathbb{R}_{\geq 0}$).
- The initial state is $(\ell_{\mathsf{init}}, \mathbf{0}^C)$.
- Moves are pairs (d, a) where $d \in \mathbb{R}_{\geq 0}$ is a delay and a is an action of the timed automaton or a special standby action $\perp$.
- Transitions are constrained by the invariants and guards of the timed automaton. We distinguish two types of transitions.
  - Delays transitions: for any $\delta \geq 0$, $(\ell, v) \xrightarrow{\delta, \perp} (\ell, v + \delta)$ if $v + \delta \models \mathit{Inv}(\ell)$
  - Edge transitions: for any $\delta \geq 0$ and action a, $(\ell, v) \xrightarrow{\delta, a} (\ell', v')$ if there is an edge $(\ell, g, a, D, \ell') \in E$, $v + \delta \models \mathit{Inv}(\ell) \wedge g$, $v' = \mathsf{reset}_D(v + \delta)$ and $v' \models \mathit{Inv}(\ell')$.

Clock-equivalence and regions

- We assume that there is a clock γ that cannot be reset.
- We use clock-equivalence and region-equivalence [AD94][3].



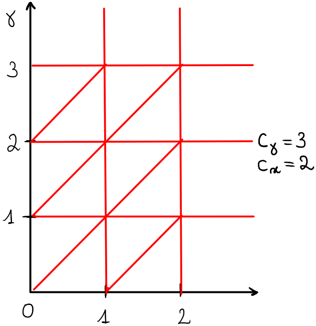
We have $v \equiv v'$ if

- for all $x \in C$, $v(x) > c_x$ iff $v'(x) > c_x$;
- for all $x \in \{z \in C \mid v(z) \leq c_x\}$, $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$;
- for all $x, y \in \{z \in C \mid v(z) \leq c_x\} \cup \{\gamma\}$, $v(x) \in \mathbb{N}$ iff $v'(x) \in \mathbb{N}$, and $\mathsf{frac}((v(x)) \leq \mathsf{frac}((v(y))$ iff $\mathsf{frac}((v'(x)) \leq \mathsf{frac}((v'(y))$.

[3] Alur and Dill, "A Theory of Timed Automata", TCS 1994.

Clock-equivalence and regions

- We assume that there is a clock γ that cannot be reset.
- We use clock-equivalence and region-equivalence [AD94][3].



We have $v \equiv v'$ if

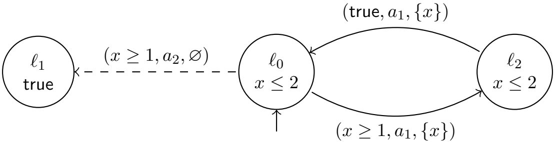
- for all $x \in C$, $v(x) > c_x$ iff $v'(x) > c_x$;
- for all $x \in \{z \in C \mid v(z) \leq c_x\}$, $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$;
- for all $x, y \in \{z \in C \mid v(z) \leq c_x\} \cup \{\gamma\}$, $v(x) \in \mathbb{N}$ iff $v'(x) \in \mathbb{N}$, and $\mathsf{frac}((v(x)) \leq \mathsf{frac}((v(y))$ iff $\mathsf{frac}((v'(x)) \leq \mathsf{frac}((v'(y))$.

- There are exponentially many clock regions.
- We let Reg denote the set of clock regions.

[3] Alur and Dill, "A Theory of Timed Automata", TCS 1994.

Timed games

- We consider two-player games played on timed automata [Alf+03][4].
- A timed game is given by a timed automaton and a partition (Σ_1, Σ_2) of the actions of the timed automaton in $\mathcal{P}_1$ actions and $\mathcal{P}_2$ actions.
- These games are concurrent: at each round, both players present a move and the play proceeds following a fastest move.

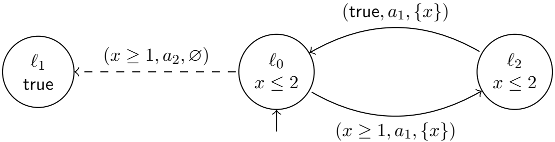


- Examples of the first round of a timed game:
  - $(\ell_0, 0)\,((1, a_1), (0.5, a_2))$

[4] Alfaro et al., "The Element of Surprise in Timed Games", CONCUR 2003.

Timed games

- We consider two-player games played on timed automata [Alf+03][4].
- A timed game is given by a timed automaton and a partition (Σ_1, Σ_2) of the actions of the timed automaton in $\mathcal{P}_1$ actions and $\mathcal{P}_2$ actions.
- These games are concurrent: at each round, both players present a move and the play proceeds following a fastest move.

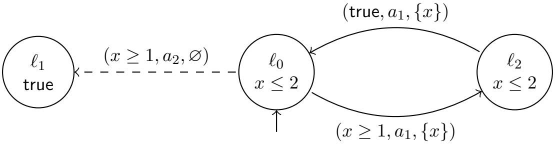


- Examples of the first round of a timed game:
  - $(\ell_0, 0)\,((1, a_1), (0.5, a_2))$

[4] Alfaro et al., "The Element of Surprise in Timed Games", CONCUR 2003.

Timed games

- We consider two-player games played on timed automata [Alf+03][4].
- A timed game is given by a timed automaton and a partition (Σ_1, Σ_2) of the actions of the timed automaton in $\mathcal{P}_1$ actions and $\mathcal{P}_2$ actions.
- These games are concurrent: at each round, both players present a move and the play proceeds following a fastest move.

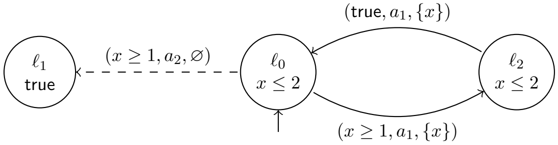


- Examples of the first round of a timed game:
  - $(\ell_0, 0)\,((1, a_1), (0.5, a_2))\,(\ell_1, 0.5)$

[4] Alfaro et al., "The Element of Surprise in Timed Games", CONCUR 2003.

Timed games

- We consider two-player games played on timed automata [Alf+03][4].
- A timed game is given by a timed automaton and a partition (Σ_1, Σ_2) of the actions of the timed automaton in $\mathcal{P}_1$ actions and $\mathcal{P}_2$ actions.
- These games are concurrent: at each round, both players present a move and the play proceeds following a fastest move.

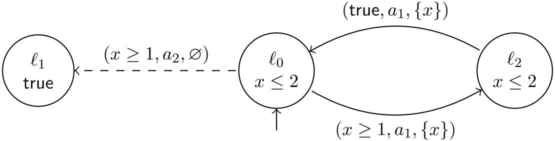


- Examples of the first round of a timed game:
  - $(\ell_0, 0)\,((1, a_1), (0.5, a_2))\,(\ell_1, 0.5)$
  - $(\ell_0, 0)\,((1, a_1), (1, a_2))$

[4] Alfaro et al., "The Element of Surprise in Timed Games", CONCUR 2003.

Timed games

- We consider two-player games played on timed automata [Alf+03][4].
- A timed game is given by a timed automaton and a partition (Σ_1, Σ_2) of the actions of the timed automaton in $\mathcal{P}_1$ actions and $\mathcal{P}_2$ actions.
- These games are concurrent: at each round, both players present a move and the play proceeds following a fastest move.

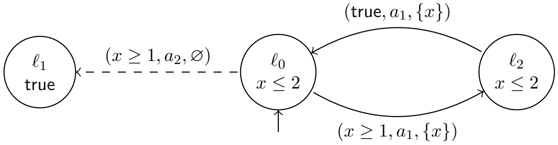


- Examples of the first round of a timed game:
  - $(\ell_0, 0)\,((1, a_1), (0.5, a_2))\,(\ell_1, 0.5)$
  - $(\ell_0, 0)\,((1, a_1), (1, a_2))\,(\ell_2, 0)$

[4] Alfaro et al., "The Element of Surprise in Timed Games", CONCUR 2003.

Timed games

- We consider two-player games played on timed automata [Alf+03][4].
- A timed game is given by a timed automaton and a partition (Σ_1, Σ_2) of the actions of the timed automaton in $\mathcal{P}_1$ actions and $\mathcal{P}_2$ actions.
- These games are concurrent: at each round, both players present a move and the play proceeds following a fastest move.

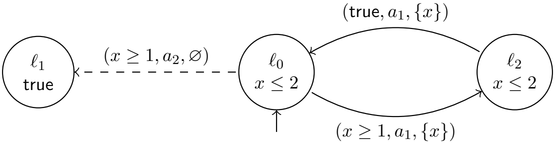


- Examples of the first round of a timed game:
  - $(\ell_0, 0)\,((1, a_1), (0.5, a_2))\,(\ell_1, 0.5)$
  - $(\ell_0, 0)\,((1, a_1), (1, a_2))\,(\ell_2, 0)$
  - $(\ell_0, 0)\,((1, a_1), (1, a_2))$

[4] Alfaro et al., "The Element of Surprise in Timed Games", CONCUR 2003.

Timed games

- We consider two-player games played on timed automata [Alf+03][4].
- A timed game is given by a timed automaton and a partition (Σ_1, Σ_2) of the actions of the timed automaton in $\mathcal{P}_1$ actions and $\mathcal{P}_2$ actions.
- These games are concurrent: at each round, both players present a move and the play proceeds following a fastest move.



- Examples of the first round of a timed game:
  - $(\ell_0, 0)\,((1, a_1), (0.5, a_2))\,(\ell_1, 0.5)$
  - $(\ell_0, 0)\,((1, a_1), (1, a_2))\,(\ell_2, 0)$
  - $(\ell_0, 0)\,((1, a_1), (1, a_2))\,(\ell_1, 0)$

[4] Alfaro et al., "The Element of Surprise in Timed Games", CONCUR 2003.

Timed games

- A play is an infinite sequence in $(S((\mathbb{R}_{\geq 0} \times \Sigma_1) \times (\mathbb{R}_{\geq 0} \times \Sigma_2)))^\omega$ constructed according to the rules of the game.
- A history is a prefix of a play ending in a state.
- A strategy for $\mathcal{P}_i$ is a function mapping histories to moves of $\mathcal{P}_i$.

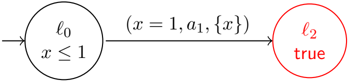
Finite-memory region strategies

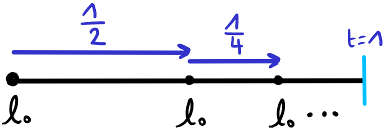
A strategy of $\mathcal{P}_i$ is a finite-memory region strategy if it can be encoded by a Mealy machine $\mathcal{M} = (\mathfrak{M}, \mathfrak{m}_{\text{init}}, \alpha_{\text{up}}, \alpha_{\text{mov}})$ where

- $\mathfrak{M}$ is a finite set of memory states, $\mathfrak{m}_{\text{init}} \in \mathfrak{M}$;
- $\alpha_{\text{up}} \colon \mathfrak{M} \times L \times \text{Reg} \to \mathfrak{M}$ is a memory update function;
- $\alpha_{\text{mov}} \colon \mathfrak{M} \times S \to \mathbb{R}_{\geq 0} \times \Sigma_i$ is a next-move function such that for all $\mathfrak{m} \in \mathfrak{M}$ and two region-equivalent states $s, s' \in S$, the delays of the moves $\alpha_{\text{mov}}(\mathfrak{m}, s)$ and $\alpha_{\text{mov}}(\mathfrak{m}, s')$ move to the same regions.

Passage of time

- It is possible to have a play in which a finite amount of time passes.
  - Example: $(\ell_0, 0)((\frac{1}{2}, \perp), (\frac{1}{2}, \perp))(\ell_0, \frac{1}{2})((\frac{1}{4}, \perp), (\frac{1}{4}, \perp))(\ell_0, \frac{3}{4}) \ldots$





- Plays in which the sum of delays converges are called time-convergent.
- Otherwise, a play is referred to as time-divergent.
- We use winning conditions that prevent a player from winning by making time converge, following [Alf+03][5].

[5] Alfaro et al., "The Element of Surprise in Timed Games", CONCUR 2003.

Winning conditions

- An objective is a set of plays that represents the specification to be enforced.
- Given an objective, we say a play belongs to its associated winning condition for $\mathcal{P}_1$ if one of the two following conditions is fulfilled:
  - the play is time-divergent and satisfies the objective;
  - the play is time-convergent and from some point on, transitions in the play cannot be achieved by $\mathcal{P}_1$'s moves.
- We say a strategy of $\mathcal{P}_1$ is winning from some initial state if all plays starting in this state consistent with the strategy satisfy the winning condition of $\mathcal{P}_1$.

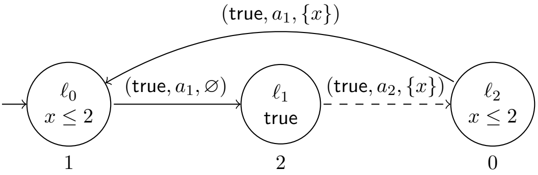
Realizability problem

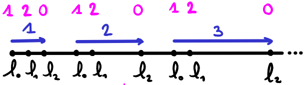
Given an objective, check whether $\mathcal{P}_1$ has a winning strategy from the initial state.

Table of contents

A motivation for windows

- Parity objective: given a priority function $p \colon L \to \mathbb{N}$, the objective consists of plays along which the smallest priority appearing infinitely often is even.
- The parity objective imposes no timing constraints between odd priorities and smaller even priorities.



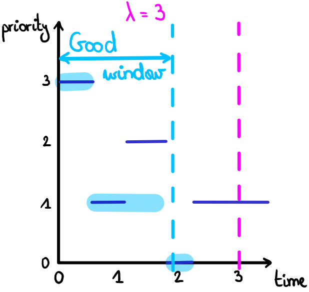


No time bound between priorities 1 and 0

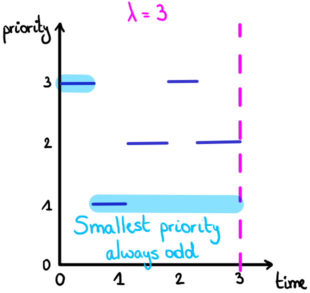
- Through the window mechanism, one can specify such timing constraints.

Good windows

- The window objectives are based on the notion of good windows.
- Fix a bound λ on the length of windows. A good window for the parity objective is a window in which:
  - strictly less than λ time units elapse and
  - the smallest priority appearing in the window is even.



Good window



Bad window

Window objectives

Let $\pi = (\ell_0, v_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, v_1)\ldots$ be a play.

- π satisfies the timed good window parity objective $\mathsf{TGW}(\lambda)$ if the window at the start of π is good. Formally, $\pi \in \mathsf{TGW}(\lambda)$ if and only if

$$\exists\, n,\ \left(\min_{0\leq k\leq n} p(\ell_k)\right) \bmod 2 = 0 \wedge \sum_{k=0}^{n-1} \mathsf{delay}(m_k^{(1)}, m_k^{(2)}) < \lambda.$$

Window objectives

Let $\pi = (\ell_0, v_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, v_1)\dots$ be a play.

- π satisfies the timed good window parity objective $\mathsf{TGW}(\lambda)$ if the window at the start of π is good. Formally, $\pi \in \mathsf{TGW}(\lambda)$ if and only if

$$\exists\, n,\ \left(\min_{0\le k\le n} p(\ell_k)\right) \bmod 2 = 0 \wedge \sum_{k=0}^{n-1} \mathsf{delay}(m_k^{(1)}, m_k^{(2)}) < \lambda.$$

- π satisfies the direct fixed timed window parity objective $\mathsf{DFTW}(\lambda)$ if all suffixes of π satisfy $\mathsf{TGW}(\lambda)$.

Window objectives

Let $\pi = (\ell_0, v_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, v_1)\dots$ be a play.

- π satisfies the timed good window parity objective $\mathsf{TGW}(\lambda)$ if the window at the start of π is good. Formally, $\pi \in \mathsf{TGW}(\lambda)$ if and only if

$$\exists\, n,\ \left(\min_{0 \le k \le n} p(\ell_k)\right) \bmod 2 = 0 \wedge \sum_{k=0}^{n-1} \mathsf{delay}(m_k^{(1)}, m_k^{(2)}) < \lambda.$$

- π satisfies the direct fixed timed window parity objective $\mathsf{DFTW}(\lambda)$ if all suffixes of π satisfy $\mathsf{TGW}(\lambda)$.
- π satisfies the direct bounded timed window parity objective DBTW if there exists λ such that $\pi \in \mathsf{DFTW}(\lambda)$.

Window objectives

Let $\pi = (\ell_0, v_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, v_1)\ldots$ be a play.

- π satisfies the timed good window parity objective $\mathsf{TGW}(\lambda)$ if the window at the start of π is good. Formally, $\pi \in \mathsf{TGW}(\lambda)$ if and only if

$$\exists\, n,\ \left(\min_{0\leq k\leq n} p(\ell_k)\right) \bmod 2 = 0 \wedge \sum_{k=0}^{n-1} \mathsf{delay}(m_k^{(1)}, m_k^{(2)}) < \lambda.$$

- π satisfies the direct fixed timed window parity objective $\mathsf{DFTW}(\lambda)$ if all suffixes of π satisfy $\mathsf{TGW}(\lambda)$.
- π satisfies the direct bounded timed window parity objective DBTW if there exists λ such that $\pi \in \mathsf{DFTW}(\lambda)$.
- π satisfies the fixed timed window parity objective $\mathsf{FTW}(\lambda)$ if some suffix of π satisfies $\pi \in \mathsf{DFTW}(\lambda)$.
- π satisfies the bounded timed window parity objective BTW if some suffix of π satisfies $\pi \in \mathsf{DBTW}$.

Table of contents

Direct bounded window objective

Request-response objectives

- We can reduce the realizability problem for DBTW to the realizability problem for request-response objectives.

Request-response objective

Let $\mathcal{R} = ((\mathsf{Rq}_i, \mathsf{Rp}_i))_{i=1}^{r}$ where for all i, $\mathsf{Rq}_i, \mathsf{Rp}_i \subseteq L \times \mathsf{Reg}$. A play $\pi = (\ell_0, v_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, v_1)\ldots$ satisfies the request-response objective $\mathsf{RR}(\mathcal{R})$ if for all $i \in \{1, \ldots, r\}$ and for all $j \in \mathbb{N}$, there exists $k \geq j$ such that

$$(\ell_j, [v_j]) \in \mathsf{Rq}_i \implies (\ell_k, [v_k]) \in \mathsf{Rp}_i.$$

- Whenever $\mathcal{P}_1$ has a winning strategy for a request-response objective, he also has a region finite-memory winning strategy using delays of at most one.

Direct bounded window objective

Reduction

- From a priority function p, we define a set of pairs of requests and responses $\mathcal{R}(p)$.
  - For each odd priority q, we have the request $p^{-1}(q) \times \mathrm{Reg}$.
  - The matching response set is $\bigcup_{q' \leq q, q \text{ even}} p^{-1}(q') \times \mathrm{Reg}$.

Direct bounded window objective

Reduction

- From a priority function p, we define a set of pairs of requests and responses $\mathcal{R}(p)$.
  - For each odd priority q, we have the request $p^{-1}(q) \times \mathsf{Reg}$.
  - The matching response set is $\bigcup_{q' \leq q, q \text{ even}} p^{-1}(q') \times \mathsf{Reg}$.

Theorem

Let $\mathsf{D} = \max_{\ell \in L}(p(\ell)) + 1$ *and* $\lambda = 8 \cdot |L| \cdot |\mathsf{Reg}| \cdot (\lfloor \frac{\mathsf{D}}{2} \rfloor + 1) + 3$.

- *The sets of winning states for the objectives* $\mathsf{RR}(\mathcal{R}(p))$, $\mathsf{DFTW}(\lambda)$ *and* DBTW *coincide.*
- *There exists a finite-memory region strategy that is simultaneously winning for these three objectives from any winning state.*

Direct bounded window objective

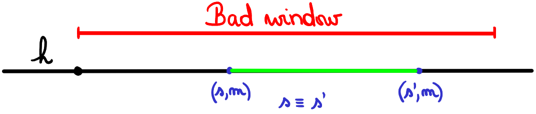
Correctness of the reduction

- The chain of inclusions $\mathsf{DFTW}(\lambda) \subseteq \mathsf{DBTW} \subseteq \mathsf{RR}(\mathcal{R}(p))$ implies the similar chain for the sets of winning states.

Direct bounded window objective

Correctness of the reduction

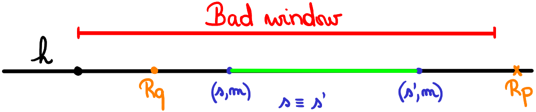
- The chain of inclusions $\mathsf{DFTW}(\lambda) \subseteq \mathsf{DBTW} \subseteq \mathsf{RR}(\mathcal{R}(p))$ implies the similar chain for the sets of winning states.
- To show that if a state is winning for $\mathsf{RR}(\mathcal{R}(p))$, then it must be for $\mathsf{DFTW}(\lambda)$, we proceed by contradiction.
- We fix a finite-memory region winning strategy for $\mathsf{RR}(\mathcal{R}(p))$ with $4 \cdot (\lfloor \frac{D}{2} \rfloor + 1)$ states and assume by contradiction that it is not winning for $\mathsf{DFTW}(\lambda)$.



Direct bounded window objective

Correctness of the reduction

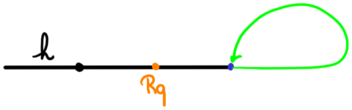
- The chain of inclusions $\mathsf{DFTW}(\lambda) \subseteq \mathsf{DBTW} \subseteq \mathsf{RR}(\mathcal{R}(p))$ implies the similar chain for the sets of winning states.
- To show that if a state is winning for $\mathsf{RR}(\mathcal{R}(p))$, then it must be for $\mathsf{DFTW}(\lambda)$, we proceed by contradiction.
- We fix a finite-memory region winning strategy for $\mathsf{RR}(\mathcal{R}(p))$ with $4 \cdot (\lfloor \frac{\mathsf{D}}{2} \rfloor + 1)$ states and assume by contradiction that it is not winning for $\mathsf{DFTW}(\lambda)$.



Direct bounded window objective

Correctness of the reduction

- The chain of inclusions $\mathsf{DFTW}(\lambda) \subseteq \mathsf{DBTW} \subseteq \mathsf{RR}(\mathcal{R}(p))$ implies the similar chain for the sets of winning states.
- To show that if a state is winning for $\mathsf{RR}(\mathcal{R}(p))$, then it must be for $\mathsf{DFTW}(\lambda)$, we proceed by contradiction.
- We fix a finite-memory region winning strategy for $\mathsf{RR}(\mathcal{R}(p))$ with $4 \cdot (\lfloor \frac{\mathsf{D}}{2} \rfloor + 1)$ states and assume by contradiction that it is not winning for $\mathsf{DFTW}(\lambda)$.



Contradiction: we have derived an outcome that is not winning for $\mathsf{RR}(\mathcal{R}(p))$.

Table of contents

Indirect bounded window objective

Algorithm

- To solve timed games with the BTW objective, we repeatedly solve games with request-response objectives, starting with $\mathsf{RR}(\mathcal{R}(p))$.
- At each step, we add state regions that are declared winning to all response sets. We stop whenever no new state is declared winning.

Illustration of the algorithm

- From any computed state, there is a winning strategy of $\mathcal{P}_1$ for $\mathsf{FTW}(8 \cdot |L| \cdot |\mathsf{Reg}| \cdot (\lfloor \frac{\mathsf{D}}{2} \rfloor + 1) + 3)$.

Indirect bounded window objective

Algorithm

- To solve timed games with the BTW objective, we repeatedly solve games with request-response objectives, starting with $\mathsf{RR}(\mathcal{R}(p))$.
- At each step, we add state regions that are declared winning to all response sets. We stop whenever no new state is declared winning.

Illustration of the algorithm

- From any computed state, there is a winning strategy of $\mathcal{P}_1$ for $\mathsf{FTW}(8 \cdot |L| \cdot |\mathsf{Reg}| \cdot (\lfloor \frac{\mathsf{D}}{2} \rfloor + 1) + 3)$.

Indirect bounded window objective

Algorithm

- To solve timed games with the BTW objective, we repeatedly solve games with request-response objectives, starting with $\mathsf{RR}(\mathcal{R}(p))$.
- At each step, we add state regions that are declared winning to all response sets. We stop whenever no new state is declared winning.

Illustration of the algorithm

- From any computed state, there is a winning strategy of $\mathcal{P}_1$ for $\mathsf{FTW}(8 \cdot |L| \cdot |\mathsf{Reg}| \cdot (\lfloor \frac{\mathsf{D}}{2} \rfloor + 1) + 3)$.

Indirect bounded window objective

Algorithm

- To solve timed games with the BTW objective, we repeatedly solve games with request-response objectives, starting with $\mathsf{RR}(\mathcal{R}(p))$.
- At each step, we add state regions that are declared winning to all response sets. We stop whenever no new state is declared winning.

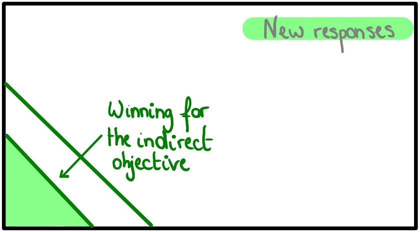


Illustration of the algorithm

- From any computed state, there is a winning strategy of $\mathcal{P}_1$ for $\mathsf{FTW}(8 \cdot |L| \cdot |\mathsf{Reg}| \cdot (\lfloor \frac{\mathsf{D}}{2} \rfloor + 1) + 3)$.

Indirect bounded window objective

Algorithm

- To solve timed games with the BTW objective, we repeatedly solve games with request-response objectives, starting with $\mathsf{RR}(\mathcal{R}(p))$.
- At each step, we add state regions that are declared winning to all response sets. We stop whenever no new state is declared winning.

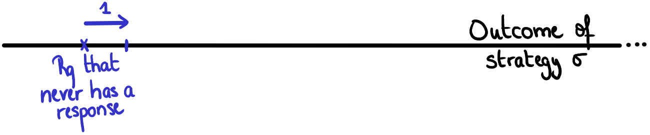
Illustration of the algorithm

- From any computed state, there is a winning strategy of $\mathcal{P}_1$ for $\mathsf{FTW}(8 \cdot |L| \cdot |\mathsf{Reg}| \cdot (\lfloor \frac{\mathsf{D}}{2} \rfloor + 1) + 3)$.

Indirect bounded window objective

Correctness of the algorithm

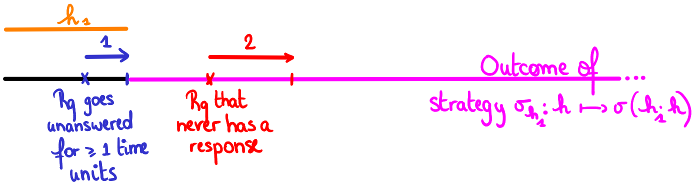
- There is no winning strategy of $\mathcal{P}_1$ for BTW from states outside the output of the algorithm.



Indirect bounded window objective

Correctness of the algorithm

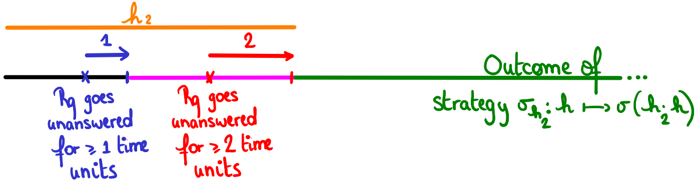
- There is no winning strategy of $\mathcal{P}_1$ for BTW from states outside the output of the algorithm.



Indirect bounded window objective

Correctness of the algorithm

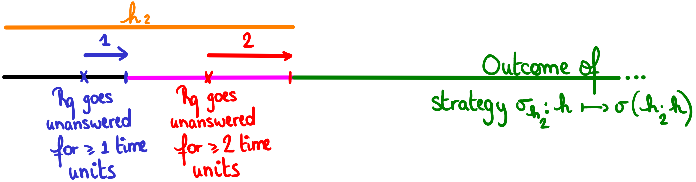
- There is no winning strategy of $\mathcal{P}_1$ for BTW from states outside the output of the algorithm.



Indirect bounded window objective

Correctness of the algorithm

- There is no winning strategy of $\mathcal{P}_1$ for BTW from states outside the output of the algorithm.



Theorem

Let $\mathsf{D} = \max_{\ell \in L}(p(\ell)) + 1$ *and* $\lambda = 8 \cdot |L| \cdot |\mathsf{Reg}| \cdot (\lfloor \frac{\mathsf{D}}{2} \rfloor + 1) + 3$.

- *The sets of winning states for the objectives* $\mathsf{FTW}(\lambda)$ *and* BTW *coincide.*
- *There exists a finite-memory region strategy that is simultaneously winning for these two objectives from any winning state.*

Table of contents

Multi-dimensional objectives and complexity

- The algorithms for both variants of bounded window parity objectives can be used to handle conjunctions of direct bounded objectives or indirect bounded objectives.
- The request-response objective used for a conjunction described from priority functions $p_1, \ldots, p_k$ is $\mathsf{RR}(\bigcup_{1 \leq i \leq k} \mathcal{R}(p_i))$.
- The algorithms for multi-dimensional objectives run in time:
  - polynomial in the size of the region abstraction;
  - polynomial in the number of priorities;
  - exponential in the number of dimensions.
- The realizability problem for BTW and DBTW can be shown EXPTIME-hard by a reduction from the realizability problem for safety objectives.

Theorem

The realizability problem for direct and indirect bounded timed window parity objectives is EXPTIME-complete.

Verification of timed automata

In addition to games, we have also studied the verification problem for bounded timed window objectives.

Verification problem for timed automata

Given an objective, check whether all time-divergent paths of the timed automata satisfy the objective.

We have shown the following.

Theorem

The verification problem for direct and indirect bounded timed window parity objectives is PSPACE-complete.

Complexity overview

Complexity summary for all variants of window parity objectives

	Single dimension	Multiple dimensions
Timed automata	PSPACE-complete	PSPACE-complete
Timed games	EXPTIME-complete	EXPTIME-complete
Games (untimed) [BHR16][6]	P-complete	EXPTIME-complete

[6] Bruyère, Hautem, and Randour, "Window parity games: an alternative approach toward parity games with time bounds", GandALF 2016.

References I

Rajeev Alur and David L. Dill. "A Theory of Timed Automata". In: Theor. Comput. Sci. 126.2 (1994), pp. 183–235. DOI: 10.1016/0304-3975(94)90010-8. URL: https://doi.org/10.1016/0304-3975(94)90010-8.

Luca de Alfaro et al. "The Element of Surprise in Timed Games". In: CONCUR 2003 - Concurrency Theory, 14th International Conferen Ed. by Roberto M. Amadio and Denis Lugiez. Vol. 2761. Lecture Notes in Computer Science. Springer, 2003, pp. 142–156. DOI: 10.1007/978-3-540-45187-7_9. URL: https://doi.org/10.1007/978-3-540-45187-7_9.

References II

Véronique Bruyère, Quentin Hautem, and Mickael Randour. "Window parity games: an alternative approach toward parity games with time bounds". In: Proceedings of the Seventh International Symposium on Games, A
Ed. by Domenico Cantone and Giorgio Delzanno. Vol. 226. EPTCS. 2016, pp. 135–148. DOI: 10.4204/EPTCS.226.10. URL: https://doi.org/10.4204/EPTCS.226.10.

James C. A. Main, Mickael Randour, and Jeremy Sproston. "Time Flies When Looking out of the Window: Timed Games with Window Parity Objectives". In: 32nd International Conference on Concurrency Theory, CONCUR 2
Ed. by Serge Haddad and Daniele Varacca. Vol. 203. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 25:1–25:16. DOI: 10.4230/LIPIcs.CONCUR.2021.25.