Robust Weighted Timed Automata and Games

Patricia Bouyer, Nicolas Markey, Ocan Sankur
LSV, CNRS & ENS Cachan, France

Abstract. Weighted timed automata extend timed automata with cost variables that can be used to model the evolution of various quantities. Although cost-optimal reachability is decidable (in polynomial space) on this model, it becomes undecidable on weighted timed games. This paper studies cost-optimal reachability problems on weighted timed automata and games under robust semantics. More precisely, we consider two perturbation game semantics that introduce imprecisions in the standard semantics, and bring robustness properties w.r.t. timing imprecisions to controllers. We give a polynomial-space algorithm for weighted timed automata, and prove the undecidability of cost-optimal reachability on weighted timed games, showing that the problem is robustly undecidable.

1 Introduction

Weighted timed automata. Weighted timed automata [4, 7] are timed automata [2] enriched with observer variables whose values grow with given constant derivatives in each location. This allows one to describe systems with timing constraints while specifying the evolution of some resources, such as energy. The cost-optimal reachability problem was studied in [4, 7, 8], and the problem was shown to be PSPACE-complete [8]. The model naturally extends to timed games. Although some partial decidability results for weighted timed games have been proposed in [1, 10], the problem is undecidable in the general case [13, 9] even for a fixed number of clocks.

Robustness. Timed automata and their extensions are abstract formalisms to describe models of real-time systems. Consequently, these formalisms make idealistic assumptions, such as the perfect continuity of the clocks, their high precision, and instantaneous reaction of the system. One side effect of such idealistic semantics is that they allow easily encoding of undecidable problems. In fact, the undecidability proofs on weighted timed games of [13, 9] rely on the ability of timed automata to distinguish infinitely precise clock values, and modify these values with high precision. It is believed that some of the undecidability results in the literature can be overcome by introducing fuzziness in the semantics, making it impossible to encode complex (undecidable) languages, see for instance [3, 5, 18].

In this paper, we investigate how adding imprecisions to the semantics affects the (un)decidability of the cost-optimal reachability problem in weighted timed automata and games. We consider two prominent perturbation semantics from [15, 11, 23], where perturbations are modeled as a game between a controller which

This work was partly supported by ANR project ImpRo (ANR-10-BLAN-0317), by ERC Starting grant EQualIS (308087) and by European project Cassting (FP7-ICT-601148).

chooses delays and edges, and an environment which perturbs the chosen delay by a (parametrized) bounded amount. The problem consists in deciding whether the controller has a winning strategy ensuring a given objective for a sufficiently small value of the parameter. Such a winning strategy is then *robust*, since it can ensure winning even if the moves it suggests are perturbed by a bounded amount. Two variants have been considered: in the *excess-perturbation semantics*, the controller is only required to suggest delays and edges whose guards are satisfied after the delay, while the guard can be violated after perturbations. This semantics allows one to design systems with simple timing constraints (for instance, using equalities), and synthesize robust strategies afterwards, taking into account new behaviors due to imprecisions. The *conservative perturbation semantics* requires the guards to be satisfied after *any* perturbation. This semantics appears naturally for instance in applications where lower and upper bounds on task execution times are given and need to be respected strictly.

For timed automata, robust reachability is EXPTIME-complete for the excess-perturbation semantics [11], while robust reachability and safety are PSPACE-complete for the conservative perturbation semantics [23]. Here, we apply both semantics to weighted timed automata and games and study the problem of deciding an upper bound on the *limit-cost* of a winning strategy for reachability objectives for the controller. The limit-cost refers to the limit of the cost achieved by a given strategy, when the bound δ on the perturbations goes to zero. In fact, the cost of a given path can slightly increase (or decrease) due to perturbations on the delays, but only by an amount proportional to δ. Thus, the limit-cost allows us to concentrate on the bound that could be achieved if this effect is discarded. On weighted timed automata, we prove that the problem is PSPACE-complete in the conservative perturbation semantics, by adapting the corner-point abstraction [8]. In the excess-perturbation semantics, we show, perhaps surprisingly, that the problem becomes undecidable on weighted timed automata. On weighted timed games, our results are negative: in both excess- and conservative perturbation semantics, cost optimal reachability remains undecidable on weighted timed games, even when the number of clocks is fixed and the constants are bounded. We also prove the undecidability of the problem for fixed parameter δ on weighted timed games. Hence, similarly to "robust undecidability" results of [20] on timed automata, we establish that cost-optimal reachability on weighted timed games is robustly undecidable, for the considered semantics.

Related Work. The work [19] attempted to introduce fuzziness in the semantics of timed automata, via a topological semantics, with the hope of extending the decidability results. However timed language inclusion turned out to be still undecidable [20]. Another related line of work is that of [21, 17, 16], which consists in modeling imprecisions by *enlarging* all clock constraints of the automaton by some parameter δ, transforming each constraint of the form $x \in [a, b]$ into $x \in [a - \delta, b + \delta]$. One analyzes the resulting system with a worst case approach. The game semantics we consider allow the system to observe and react to perturbations. The dual notion of *shrinking* was considered in [22] in order to study whether any significant behavior is lost when guards are shrunk.

The long version [12] of this paper contains the detailed proofs.

2 Preliminaries

Game structures. A *(two-player) game structure* is a tuple $\mathcal{T} = (S, s_0, M_1, M_2, T_1, T_2, \mathsf{jt})$, where S is a set of *states*, $s_0 \in S$ is the *initial state*, M_i is the set of *moves* of Player i, $T_i \subseteq S \times M_i$ is the *enabling condition* for Player i, and $\mathsf{jt}\colon S \times M_1 \times M_2 \to S$ the *joint transition function*. We assume that each M_i contains a distinguished element $\bot$ called the *empty move*, and that $\mathsf{jt}(s, \bot, \bot) = s$ for any $s \in S$. A *run* of $\mathcal{T}$ is a finite or infinite sequence $q_1 e_1 q_2 e_2 \dots$ of alternating states $q_i \in S$, and pairs of moves $e_i = (m_1, m_2) \in M_1 \times M_2$, such that for all $i \geq 1$, we have $(q_i, m_\iota) \in T_\iota$ for $\iota \in \{1, 2\}$, and $q_{i+1} = \mathsf{jt}(q_i, e_i)$. For any finite run ρ, let $|\rho|$ denote its *length*, that is, the number of states it contains. For any natural number $1 \leq i \leq |\rho|$, let $\mathsf{state}_i(\rho)$ the i-th state of ρ, and $\mathsf{trans}_i(\rho)$ its i-th transition. We let $\mathsf{first}(\rho) = \mathsf{state}_1(\rho)$, and $\mathsf{last}(\rho) = \mathsf{state}_{|\rho|}(\rho)$. We also denote by $\rho_{i\dots j}$ the sub-run of ρ between states of indices i and j.

A *strategy* for Player i is a function f that maps each finite run h to a move M_i, such that $(\mathsf{last}(h), f(h)) \in T_i$. A run ρ is *compatible* with strategies f and g of Players 1 and 2, if $\mathsf{state}_{i+1}(\rho) = \mathsf{jt}(\rho_{1\dots i}, (f(\rho_{1\dots i}), g(\rho_{1\dots i})))$ for all $i \geq 1$. Given strategies f and g for Players 1 and 2 resp., the *outcome of the pair* (f, g) *in* $\mathcal{T}$, denoted by $\mathsf{Outcome}_{\mathcal{T}}(f, g)$ is the unique infinite run that is compatible with both strategies. Let $S_i(\mathcal{T})$ denote the set of the strategies of Player i in $\mathcal{T}$.

Weighted timed automata and games. Given a finite set of clocks $\mathcal{C}$, we call *valuations* the elements of $\mathbb{R}^{\mathcal{C}}_{\geq 0}$. For a subset $R \subseteq \mathcal{C}$ and a valuation ν, $\nu[R \leftarrow 0]$ is the valuation defined by $\nu[R \leftarrow 0](x) = \nu(x)$ for $x \in \mathcal{C} \setminus R$ and $\nu[R \leftarrow 0](x) = 0$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$ and a valuation ν, the valuation $\nu + d$ is defined by $(\nu + d)(x) = \nu(x) + d$ for all $x \in \mathcal{C}$. We extend these operations to sets of valuations in the obvious way. We write $\mathbf{0}$ for the valuation that assigns 0 to every clock.

An atomic clock constraint is a formula of the form $k \preceq x \preceq' l$ or $k \preceq x - y \preceq' l$ where $x, y \in \mathcal{C}$, $k, l \in \mathbb{Z} \cup \{-\infty, \infty\}$ and $\preceq, \preceq' \in \{<, \leq\}$. A *guard* is a conjunction of atomic clock constraints. A valuation ν satisfies a guard g, denoted $\nu \models g$, if all constraints are satisfied when each $x \in \mathcal{C}$ is replaced with $\nu(x)$. We write $\Phi_{\mathcal{C}}$ for the set of guards built on $\mathcal{C}$.

Definition 1. *A* weighted timed game *(WTG) $\mathcal{A}$ is a tuple $(\mathcal{L}, \ell_0, \mathcal{C}, \mathcal{I}, E_1, E_2, \mathcal{S})$, where $\mathcal{L}$ is a finite set of locations, $\mathcal{C}$ is a finite set of clocks, $\mathcal{I} : \mathcal{L} \to \Phi_{\mathcal{C}}$ assigns an invariant to every location, $E_1, E_2 \subseteq \mathcal{L} \times \Phi_{\mathcal{C}} \times 2^{\mathcal{C}} \times \mathcal{L}$ are sets of edges, $\ell_0 \in \mathcal{L}$ is the initial location, and $\mathcal{S}\colon \mathcal{L} \to \mathbb{Z}$ is the slope function*[1]*. For any edge $e = (\ell, g, R, \ell')$, g is the guard of the edge, and R its reset set. An edge $e = (\ell, g, R, \ell')$ is also written as $\ell \xrightarrow{g,R} \ell'$. A* weighted timed automaton *(WTA) is a WTG of the form $(\mathcal{L}, \ell_0, \mathcal{C}, \mathcal{I}, E_1, \emptyset, \mathcal{S})$.*

[1] We do not introduce discrete weights on transitions, but all our results would also hold in that setting.

Weighted timed games define game structures similarly to timed games [6]: the guards of the edges enable or disable the transitions, and the reset set determines the update after the transition is taken by resetting the clocks belonging to the set. Intuitively, the edges E_i are controlled by Player i. Moreover, the state space contains the value of a cost variable, denoted cost, which grows with derivative $\mathcal{S}(\ell)$ at any location ℓ.

In this paper, we assume that all clocks are bounded above by a constant, *i.e.*, the invariant at each location imposes some upper bound on all clocks.

Formally, the *exact semantics* of a WTG $\mathcal{A}$ is a game structure $[\![\mathcal{A}]\!] = (S, s_0, M_1, M_2, T_1, T_2, \mathsf{jt})$. The state space is $S = \{(\ell, \nu, c) \mid \ell \in \mathcal{L}, \nu \in \mathbb{R}^{\mathcal{C}}_{\geq 0}, c \in \mathbb{R}, \nu \models \mathcal{I}(\ell)\}$. The initial state is $s_0 = (\ell_0, \mathbf{0}, 0)$. The moves are defined by $M_i = \mathbb{R} \times E_i \cup \{\bot\}$ whose components are pairs of a delay and a Player-i edge. A pair $(d, e) \in \mathbb{R} \times E_i$ is said enabled at $(\ell, \nu) \in \mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$ whenever, writing $e = (\ell_1, g, R, \ell_2)$, we have $\ell = \ell_1$, $\nu \models \mathcal{I}(\ell_1)$, $\nu + d \models \mathcal{I}(\ell_1)$, $\nu + d \models g$, and $(\nu + d)[R \leftarrow 0] \models \mathcal{I}(\ell_2)$. The enabling condition $T_i((\ell, \nu, c), (d, e))$ holds if, and only if (d, e) is enabled at (ℓ, ν). Note that $T_i((\ell, \nu, c), \bot)$ holds at any state (ℓ, ν, c). The joint transition function is defined as follows. Given $d_i \geq 0$, and edges $e_i = (\ell, g_i, R_i, \ell_i) \in E_i$, we have $\mathsf{jt}((\ell, \nu, c), (d_1, e_1), (d_2, e_2)) = (\ell_{i_0}, (\nu + d_{i_0})[R_{i_0} \leftarrow 0], c + d_{i_0}\mathcal{S}(\ell))$, where $i_0 = 1$ if $d_1 \leq d_2$, and $i_0 = 2$ otherwise. Moreover, $\mathsf{jt}((\ell, \nu, c), (d_1, e_1), \bot) = (\ell_1, (\nu + d_1)[R_1 \leftarrow 0], c + d_1\mathcal{S}(\ell))$, and symmetrically.

Example 1. Figure 1 displays an example of a weighted timed game. Plain (resp. dashed) arrows are for Player 1 (resp. Player 2) edges. The slopes are indicated above each state. A strategy for Player 1 is to suggest a delay of 1 and choose the edge from ℓ_1 to ℓ_2. This prevents Player 2 from going down to location ℓ_5, where the cost of accepting is $12 - o(\delta)$. From location ℓ_2, Player 1 can go to ℓ_4, from where a target location is reached with cost 7.

Regions, Vertices. We assume familiarity with *regions* (see [2]). We recall that the *region automaton* of a timed automaton $\mathcal{A}$ is a finite automaton, denoted $\mathcal{R}(\mathcal{A})$, with states (ℓ, r), where ℓ is a location, and r a region. Let us write $\mathsf{loc}((\ell, r)) = \ell$. There is a transition $(\ell, r) \xrightarrow{\mathsf{delay}} (\ell, r')$ iff some valuation in r' can be reached by a time delay from some valuation in r. We have $(\ell, r) \xrightarrow{e} (\ell', r')$, for an edge $e = (\ell, g, R, \ell')$ iff all valuations of r satisfy g, and $r' = r[R \leftarrow 0]$. A path of $\mathcal{R}(\mathcal{A})$ is a sequence $q_1 t_1 q_2 t_2 \ldots$ where for all $i \geq 1$, $q_i = (\ell_i, r_i)$ for some location ℓ_i and region r_i, and t_i is either an edge or delay. We say that a run ρ of $\mathcal{A}$ follows a path $\pi = q_1 t_1 \ldots$ of $\mathcal{R}(\mathcal{A})$ if for any $i \geq 1$, if we write $(\ell_i, r_i) = \mathsf{state}_i(\pi)$,

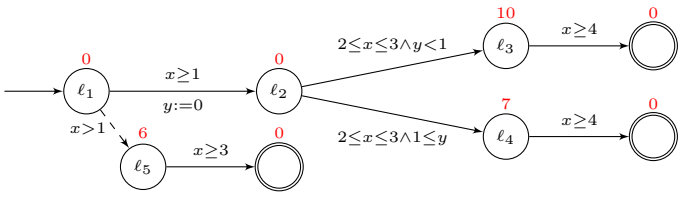


Fig. 1. Example of a WTG

then $\mathsf{state}_i(\rho) = (\ell_i, \nu_i)$ for some $\nu_i \in r_i$, and moreover, $\mathsf{trans}_i(\pi) = \mathsf{delay}$ implies that $\mathsf{trans}_i(\rho) \in \mathbb{R}_{\geq 0}$, and $\mathsf{trans}_i(\pi) = \mathsf{trans}_i(\rho)$ otherwise.

A valuation with integer coordinates is called a *vertex*. For any region r, let us denote by $\mathcal{V}(r)$ the set of vertices that belong the topological closure of r. A region is *non-punctual* if for some $\nu \in r$, and $\epsilon > 0$, $\nu + [-\epsilon, \epsilon] \subseteq r$. It is *punctual* otherwise. A *non-punctual path* of the region automaton is a path of $\mathcal{R}(\mathcal{A})$ where all regions reached after a delay are non-punctual.

3 Robust Cost-Optimal Reachability

We now define two perturbed semantics for weighted timed games. These semantics were first studied in [14, 11] for timed automata and games in order to synthesize robust strategies. We adapt these here to WTG and formalize cost-optimal reachability problems.

Perturbed semantics. We will call Player 1 Controller, and Player 2 Perturbator. The idea behind the perturbed semantics is to give Perturbator the additional power of perturbing the delays chosen by Controller by some bounded amount δ (in that sense, the perturbed semantics becomes asymmetric). In this setting, a winning strategy for Controller is then robust to perturbations in the time delays. Informally, at any state (ℓ, ν, c), both players suggest a delay and an edge. If Perturbator suggests a shorter delay, then the suggested delay and edge is taken. If Controller suggests the shorter delay d and edge e, then the system moves to an intermediate state (ℓ, ν, c, d, e), from which Perturbator chooses $\epsilon \in [-\delta, \delta]$, and the edge e is taken after a delay of $d + \epsilon$; the cost then increases by $(d + \epsilon) \cdot \mathcal{S}(\ell)$. We require both players to only suggest delays no smaller than δ, to model the fact that the system is not infinitely fast. We will formally define two perturbed semantics based on the above ideas; they will differ on the satisfaction or not of the guards after the delay has been perturbed by ϵ.

Formally, given $\delta > 0$, the *δ-excess perturbation semantics* of a WTG $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \mathcal{I}, E_1, E_2, \mathcal{S})$ is a game structure $\mathcal{G}^e_\delta(\mathcal{A}) = (S', s'_0, M'_1, M'_2, T'_1, T'_2, \mathsf{jt}')$, where $S' = S \cup S \times \mathbb{R}_{\geq 0} \times E_1$, with $S = \{(\ell, \nu, c) \mid \ell \in \mathcal{L}, \nu \in \mathbb{R}^{\mathcal{C}}_{\geq 0}, c \in \mathbb{R}, \nu \models \mathcal{I}(\ell)\}$. The initial state is $s'_0 = (\ell_0, \mathbf{0}, 0)$. We have $M'_1 = [\delta, \infty) \times E_1$, and $M'_2 = [\delta, \infty) \times E_2 \cup [-\delta, \delta]$. The enabling conditions T'_i are as follows. For any $i \in \{1, 2\}$, from any state $(\ell, \nu, c) \in S'$, we have $((\ell, \nu, c), d, e) \in T'_i$ for any $d \geq \delta$ and $e \in E_i$ whenever (d, e) is enabled at (ℓ, ν). We also have $(\ell, \nu, c), \bot) \in T'_i$. For states $(\ell, \nu, c, d, e) \in S'$, we have $((\ell, \nu, c, d, e), \bot) \in T'_1$ and $((\ell, \nu, c, d, e), \epsilon) \in T'_2$ for any $\epsilon \in [-\delta, \delta]$. The joint transition function δ respects the shorter delay:

$$\mathsf{jt}'((\ell, \nu, c), (d_1, e_1), (d_2, e_2)) = \begin{cases} (\ell, \nu, c, d_1, e_1) & \text{if } d_1 \leq d_2, \\ (\ell_2, (\nu + d_2)[R' \leftarrow 0], c + d_2 \cdot \mathcal{S}(\ell)) & \text{if } d_1 > d_2. \end{cases}$$

Moreover, in case one player plays $\bot$, we let $\mathsf{jt}'((\ell, \nu, c), (d, e_1), \bot) = (\ell, \nu, c, d, e_1)$ and $\mathsf{jt}'((\ell, \nu, c), \bot, (d', e_2)) = (\ell_2, (\nu + d')[R' \leftarrow 0], c + d' \cdot \mathcal{S}(\ell))$, as expected. Last, we let $\mathsf{jt}'((\ell, \nu, c, d, e_1), \bot, \epsilon) = (\ell_1, (\nu + d + \epsilon)[R \leftarrow 0], c + (d + \epsilon) \cdot \mathcal{S}(\ell))$.

Thus, the cost variable grows with derivative $\mathcal{S}(\ell)$ at location ℓ, and the sojourn time is either the delay suggested by Perturbator if it is shorter, or the delay suggested by Controller and perturbed by Perturbator otherwise. Notice that, in this semantics the guard of an edge that is taken need not be satisfied after a perturbation (hence the term *excess*).

We also consider another natural semantics for perturbation, which we call the *δ-conservative perturbation semantics* and denote $\mathcal{G}^c_\delta(\mathcal{A})$. This semantics is defined similarly with the only difference that the enabling condition for Controller from states (ℓ, ν, c) are defined as follows. From any state $(\ell, \nu, c) \in S'$, we have $((\ell, \nu, c), d, e) \in T'_1$ for any $d \geq \delta$ and $e \in E_1$ whenever $(d+\epsilon, e)$ is enabled at (ℓ, ν) for every $\epsilon \in [-\delta, \delta]$. In other terms, Controller should only suggest delays and edges whose guards are enabled under any perturbation of the delay. Consequently, this semantics forbids equality constraints, since these are never enabled.

Example 2. Figure 2 explains the differences between our two perturbation semantics: Controller has to suggest a delay such that the resulting valuation does not end up in the grey area; Perturbator can then shift this delay by $[-\delta, \delta]$. In the conservative semantics, no new behaviors are added, because the final delay chosen by Perturbator will satisfy the guard; in the excess semantics, new behaviors may occur because neighboring regions can be reached.

Example 1 (Cont'd). We come back to the WTG of Fig. 1, to illustrate the differences between the two perturbed semantics. Under the excess-perturbation semantics, as in the exact case, Controller can suggest a delay of 1 and choose the edge from ℓ_1 to ℓ_2. The location ℓ_5 can thus be avoided. Now, one can see that the move of Perturbator determines the next location to be visited: if Perturbator adds a positive perturbation (*i.e.* if the delay is in $[1, 1+\delta]$), then only location ℓ_3 is reachable. Conversely, a negative perturbation enables only location ℓ_4. To maximize the cost, Perturbator will force the play to ℓ_3, so Controller can only ensure a cost of $10 + \Theta(\delta)$.

We now focus on the conservative semantics. The above strategy is no more valid since in this case, Controller can only suggest delays of at least $1 + \delta$. Then Perturbator can force the play to ℓ_5. Here, the cost of winning is $12 + \Theta(\delta)$.

Cost-optimal reachability. We define cost-optimal reachability problems that take into account the perturbed semantics. We are interested in computing strategies for reachability which minimize the cost when the parameter δ goes to 0.

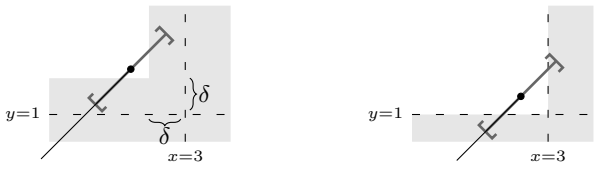


Fig. 2. The conservative- (left) and excess semantics (right) for a transition guarded with $x \leq 3 \wedge y \geq 1$. The grey area corresponds to forbidden delays, the bullet corresponds to the choice of Controller, and the segment indicates the possible choices for Perturbator.

First notice that $S_1(\mathcal{G}^e_\delta(\mathcal{A}))$ does not depend on δ, since Controller only has to suggest moves that satisfy the guards (a *winning* strategy will depend on δ though). In contrast, $S_1(\mathcal{G}^e_\delta(\mathcal{A}))$ does depend on δ since Controller is required to satisfy the guards even after perturbations. It is easy to see that $S_1(\mathcal{G}^c_\delta(\mathcal{A})) \subseteq S_1(\mathcal{G}^c_{\delta'}(\mathcal{A}))$ for any $\delta' < \delta$. We denote $S_1(\mathcal{G}^c(\mathcal{A})) = \bigcup_{\delta>0} S_1(\mathcal{G}^c_\delta(\mathcal{A}))$.

Let us write $(\ell, \nu, c)|_{\mathsf{cost}} = c$, and $(\ell, \nu, c, d, e)|_{\mathsf{cost}} = c$, the projection of a state to the cost value. For any run ρ of the exact or perturbed semantics, we define the *cost of ρ w.r.t. a location ℓ* as, $\mathsf{cost}^\ell(\rho) = \inf\{\mathsf{state}_i(\rho)|_{\mathsf{cost}} \mid 1 \leq i < |\rho| + 1, \mathsf{loc}(\mathsf{state}_i(\rho)) = \ell\}$ (Note that the definition includes the case where $|\rho| = \infty$). Hence, if ℓ is never reached, then the cost is ∞. Otherwise it is the infimum of the costs observed at location ℓ. Given $\delta > 0$, a pair of strategies $(\sigma, \sigma') \in S_1(\mathcal{G}^e_\delta(\mathcal{A})) \times S_2(\mathcal{G}^e_\delta(\mathcal{A}))$, and location ℓ, we define $\mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}^e_\delta(\mathcal{A})) = \mathsf{cost}^\ell(\mathsf{Outcome}_{\mathcal{G}^e_\delta(\mathcal{A})}(\sigma, \sigma'))$. We define similarly $\mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}^c_\delta(\mathcal{A}))$. Given a strategy $\sigma \in S_1(\mathcal{G}^e_\delta(\mathcal{A}))$, we define the *limit-cost of σ* as $\mathsf{lim\text{-}cost}^{\mathrm{exs}}_\sigma(\mathcal{A}, \ell) = \lim_{\delta\to 0} \sup_{\sigma' \in S_2(\mathcal{G}^e_\delta(\mathcal{A}))} \mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}^e_\delta(\mathcal{A})))$. The limit is well defined since strategy σ is valid for any $\delta > 0$. Similarly, for $\sigma \in \mathcal{G}^c_{\delta_0}(\mathcal{A})$, we let $\mathsf{lim\text{-}cost}^{\mathrm{cons}}_\sigma(\mathcal{A}, \ell) = \lim_{\substack{\delta\to 0 \\ 0<\delta<\delta_0}} \sup_{\sigma' \in S_1(\mathcal{G}^c_\delta(\mathcal{A}))} \mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}^c_\delta(\mathcal{A}))$. Here, we take the limit for $0 < \delta < \delta_0$, such that $\sigma \in S_2(\mathcal{G}^c_{\delta_0}(\mathcal{A}))$ so that the strategy is valid for any considered δ. Thus, the limit-cost of a Controller's strategy σ is the cost it guarantees in the limit, against any strategy of Perturbator, when δ goes to 0.

We are interested in deciding whether Controller has a strategy that guarantees an upper bound on the limit-cost for a reachability objective.

Definition 2. *The* limit strategy (strict) upper-bound problem for the excess perturbation semantics *asks, given a weighted timed game $\mathcal{A}$, a location ℓ, and a rational λ, whether there exists a strategy $\sigma \in S_1(\mathcal{G}^e_\delta(\mathcal{A}))$ such that $\mathsf{lim\text{-}cost}^{exs}_\sigma(\mathcal{A}, \ell) \leq \lambda$ (resp. $\mathsf{lim\text{-}cost}^{exs}_\sigma(\mathcal{A}, \ell) < \lambda$). Similarly, we define the* limit strategy (strict) upper-bound problem for the conservative perturbation semantics.

We define the *limit-value* as the infimum of the limit-cost that can be guaranteed by Controller: $\mathsf{lim\text{-}value}^{\mathrm{exs}}(\mathcal{A}, \ell) = \inf_{\sigma \in S_1(\mathcal{G}^e_\delta(\mathcal{A}))} \mathsf{lim\text{-}cost}^{\mathrm{exs}}_\sigma(\mathcal{A}, \ell)$, and $\mathsf{lim\text{-}value}^{\mathrm{cons}}(\mathcal{A}, \ell) = \inf_{\sigma \in S_1(\mathcal{G}^c(\mathcal{A}))} \mathsf{lim\text{-}cost}^{\mathrm{cons}}_\sigma(\mathcal{A}, \ell)$. We also consider deciding upper bounds on values:

Definition 3. *The* limit value upper-bound problem for the excess (resp. conservative) perturbation semantics *asks whether given a weighted timed automaton $\mathcal{A}$, a target location ℓ, and a rational λ, it holds $\mathsf{lim\text{-}value}^{exs}(\mathcal{A}, \ell) \leq \lambda$ (resp. $\mathsf{lim\text{-}value}^{cons}(\mathcal{A}, \ell) \leq \lambda$)?*

Notice that the strategy strict upper-bound problem is equivalent to deciding whether the strict upper bound holds for the *value*, that is the infimum of the limit cost over all possible strategies. We will consider the restriction of both problems on WTA.

Example 1 (Cont'd). We come back to the WTG of Fig. 1. We have seen the impact of the choice of the semantics on the possible behaviors of the system. In particular, the limit-optimal cost in the conservative (resp. excess) semantics is equal to 12 (resp. 10).

Theorem 1. *The limit strategy upper-bound problem for WTA (and WTG) is undecidable under the excess perturbation semantics, for a fixed number of clocks.*

Theorem 1 is a rather surprising result. It reveals that adding perturbations can render problems intractable, which is the opposite of a common belief [5, 18]. In this case, optimal reachability is PSPACE-complete for weighted timed automata under the exact semantics, but becomes undecidable under the excess perturbation semantics.

The conservative robust semantics is more restrictive than the excess perturbation semantics. In timed automata, reachability under the conservative semantics is PSPACE-complete [23], in contrast with the EXPTIME-completeness under the excess perturbation semantics [11]. For weighted timed automata, the conservative semantics renders the problem tractable; the limit value upper-bound problem is PSPACE-complete:

Theorem 2. *The limit value upper-bound problem is* PSPACE*-complete on WTA under the conservative perturbation semantics.*

The algorithm is based on the corner-point abstraction [8], but requires eliminating *punctual* regions, following the ideas in [23]. However the conservative semantics does not allow the treatment of weighted timed games:

Theorem 3. *The limit strategy strict upper-bound problem is undecidable for WTG under the conservative perturbation semantics, for a fixed number of clocks.*

The undecidability also holds in both semantics on WTGs when δ is fixed:

Theorem 4. *The following problem is undecidable: For any fixed* $0 \leq \delta \leq \frac{1}{3}$*, given a WTG* $\mathcal{A}$*, a target location* ℓ*, and a rational* λ*, decide whether it holds* $\inf_{\sigma \in S_1(G)} \sup_{\sigma' \in S_2(G)} \mathsf{cost}^{\ell}_{\sigma,\sigma'}(G) < \lambda$*, where* G *denotes either* $\mathcal{G}^e_\delta(\mathcal{A})$ *or* $\mathcal{G}^c_\delta(\mathcal{A})$*.*

In Section 4, we present our results on WTA, that is, the algorithm of Theorem 2 and the undecidability result of Theorem 1. The long version [12] of this paper contains the detailed proofs of these results, and of Theorems 3 and 4.

4 Weighted Timed Automata

4.1 Algorithm in the Conservative Semantics

We present a polynomial-space algorithm for the limit value upper-bound problem, based on a variant of the corner-point abstraction [8]. The idea behind our algorithm is that Perturbator can always avoid punctual regions by adding an infinitesimal perturbation. Thus, one needs to remove punctual delay transitions from the corner-point abstraction. It turns out that the resulting construction suffices to solve the limit-cost value for a given WTA.

A *finite weighted automaton* over $(\mathbb{Z}, +)$ is a tuple $\mathcal{F} = (S, s_0, \Sigma, T, W)$, where S is the set of states, $s_0 \in S$ the initial state, $T \subseteq S \times \Sigma \times S$ the set of transitions, and $W \colon E \to \mathbb{Z}$ is the *weight function*. A *path* (or *run*) of a finite

weighted automaton is a (finite or infinite) sequence $q_1 t_1 q_2 t_2 \dots$ alternating states and transitions and such that for all $i \geq 1$, $t_i = (q_i, \sigma_i, q_{i+1})$ for some σ_i. We write $\mathsf{Runs}(\mathcal{F})$ for the set of runs of $\mathcal{F}$ starting in the initial state s_0. The length, first and last states and sub-runs are defined in the same way as for runs of a game structure. A finite weighted automaton then associates to any finite path the sum of the weights of the edges it visits. Given any path $\pi = q_1 t_1 q_2 \dots q_n$, the *weight of* π is defined as $W(\pi) = \sum_{1 \leq i < n} W(t_i)$.

Let us consider a weighted timed automaton $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, E_1, \emptyset, \mathcal{S})$. Notice that, following Def. 1, we write it as a weighted timed games with no edges belonging to Player 2. The *corner-point abstraction* of $\mathcal{A}$ is a finite weighted automaton, denoted $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$. The states of $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$ are triples (ℓ, r, v), where ℓ is a location, r a region, and $v \in \mathcal{V}(r)$ a vertex of r. Edges are defined as follows: we have $(\ell, r, v) \xrightarrow{\mathsf{delay}} (\ell, r', v')$ if $(\ell, r) \xrightarrow{\mathsf{delay}} (\ell, r')$ in the region automaton, and $v' = v + k$ for some natural number k. In other terms, v' is a time-successor of v, and is a vertex of region r'. The weight associated to this transition is $k \times \mathcal{S}(\ell)$. Further, we have an edge $(\ell, r, v) \xrightarrow{e} (\ell', r', v')$ if $e = (\ell, g, \sigma, R, \ell')$ is an edge of $\mathcal{A}$ such that $r \models g$, and $r' = r[R \leftarrow 0]$, $v' = v[R \leftarrow 0]$. Such an edge has weight 0. Observe that $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$ is finite since all clocks are assumed to be bounded. Notice that a path in the corner-point abstraction corresponds to a path of the WTA that runs arbitrarily close to vertices of the regions it visits.

Let the *non-punctual corner-point abstraction*, denoted $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$, be the finite weighted automaton obtained from the corner-point abstraction by removing any transition of the form $(\ell, r, v) \xrightarrow{\mathsf{delay}} (\ell, r', v')$, where r' is punctual. Thus, any path in the non-punctual corner-point abstraction corresponds to a non-punctual path in the region automaton.

For any path π of the region automaton $\mathcal{R}(\mathcal{A})$ of $\mathcal{A}$, we denote by $\mathsf{Runs}(\pi)$, the set of runs of $[\![\mathcal{A}]\!]$ that follow π. If π is a path of the corner-point abstraction $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$, then we say that a run follows π if it follows the path projected to $\mathcal{R}(\mathcal{A})$ (that is, obtained by removing vertices in each state). We extend the notation $\mathsf{Runs}(\pi)$ to paths π of the corner-point abstraction. For any path π of $\mathcal{R}(\mathcal{A})$ or $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$, let us define $\bar{\pi}$ obtained from π by replacing all regions by their topological closures. We will consider $\mathsf{Runs}(\bar{\pi})$ which is the set of runs visiting the topological closures of the regions of π. In other terms, this is the topological closure of the set $\mathsf{Runs}(\pi)$.

We define $\mathsf{value}(\mathcal{F}, s)$ for a finite weighted automaton $\mathcal{F}$ and state s as the cost of the shortest path from the initial state to s. Formally, for any finite weighted automaton $\mathcal{F} = (S, s_0, \Sigma, T, W)$, and $s \in S$, we let $\mathsf{value}(\mathcal{F}, s) = \inf\{W(\pi) \mid \pi \in \mathsf{Runs}(\mathcal{F}), \mathsf{last}(\pi) = s\}$. For corner-point abstractions, we extend this notation to locations: $\mathsf{value}(\mathcal{R}_{\mathsf{cp}}(\mathcal{A}), \ell) = \inf\{W(\pi) \mid \pi \in \mathsf{Runs}(\mathcal{R}_{\mathsf{cp}}(\mathcal{A})), \mathsf{loc}(\mathsf{last}(\pi)) = \ell\}$.

In the exact semantics, results of [8] show that the infimum of the cost of the runs of a WTA following a given path π of the corner-point abstraction is achieved by a run that follows $\bar{\pi}$, and only visits vertices. Hence, to compute the infimum cost, it suffices to compute the value of the corner-point abstraction. In the conservative perturbed case, we prove that the same algorithm can be applied, once we discard punctual paths.

Lemma 1. *For any weighted timed automaton $\mathcal{A}$ and target location ℓ, we have* $\mathit{lim\text{-}value}^{cons}(\mathcal{A}, \ell) = \mathit{value}(\mathcal{R}^{np}_{cp}(\mathcal{A}), \ell)$.

Theorem 2 follows from the previous lemma. In fact, to compute the optimal cost on $\mathcal{A}$, it suffices to consider the finite weighted automaton $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$, and find the shortest path to location ℓ. To decide whether the limit value is less than some given constant λ, one can guess a path in $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$ in polynomial-space (such a path can be constructed on-the-fly in polynomial space, see *e.g.* [8]), and check whether its weight is less than or equal to λ. Note that the problem is PSPACE-hard since it already is in the unweighted case.

4.2 Undecidability Under Excess Perturbation

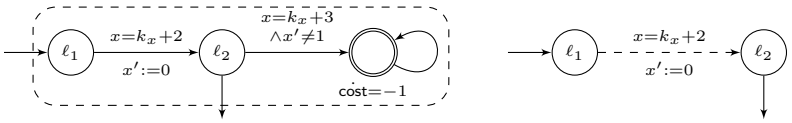
We present the proof of Theorem 1, showing the undecidability of the limit strategy upper-bound problem for WTA with excess perturbation. Our proof is based on a reduction from the halting problem of Minsky machines, following the encoding of [9]. Compared to the reductions of earlier work [13, 9], special care needs to be taken when dealing with perturbations, since the present semantics disables precise moves.

We consider a Minsky machine with counters c_1 and c_2, and a list of instructions $I_1, \ldots, I_n$. Here, each instruction I_i, for $1 \leq i \leq n-1$, is an *incrementation for* c_b as, $c_b = c_b + 1$; `goto` I_j, for $b = 1$ or 2, or a *decrementation with zero-test for* c_b as, `if` $(c_b = 0)$ `goto` I_j `else` $c_i = c_i - 1$; `goto` $I_{j'}$. The instruction I_n is the *ending instruction*, that is, the final state. The halting problem asks whether the instruction I_n is reachable, starting from the configuration $c_1 = 0$, $c_2 = 0$, at instruction I_1.

Our reduction uses 10 clocks $x, x', y, y', u, u', t, t', z, z'$. A counter of a Minsky machine with value n will be encoded by a pair of clocks x, x' with values $k_x + \frac{1}{2^n}$ and $k_{x'} + \frac{1}{2^n}$ for some integers $k_x, k_{x'}$. Here, k_x is called the *shift* of x. If α denotes the clock x', we let $\alpha' = x$, and $\alpha'' = x'$, and similarly for other clocks. A configuration of a Minsky machine with counter values $n, m \geq 0$, is entirely encoded by four clocks:

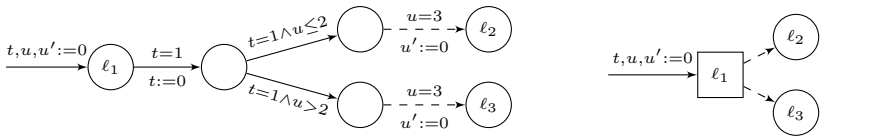
$$x = k_x + \frac{1}{2^n} \qquad x' = k_{x'} + \frac{1}{2^n} \qquad y = k_y + \frac{1}{2^m} \qquad y' = k_{y'} + \frac{1}{2^m} \tag{1}$$

for some shifts $k_x, k_{x'}, k_y, k_{y'}$. The redundancy in this encoding is necessary to cope with perturbations; this will be clear in the constructions. We denote by $\boldsymbol{k}$ the vector of shifts, for all clocks, and by $\mathsf{code}_{\boldsymbol{k}}(n, m)$ the set of valuations satisfying (1). We also define $\mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n, m)$ the set of valuations ν such that $\nu + \eta \in \mathsf{code}_{\boldsymbol{k}}(n, m)$, where $|\eta(\alpha)| \leq \epsilon$, $\eta(\alpha) = \eta(\alpha')$ for all clocks α, and moreover $\eta(x) = \eta(x') = 0$ whenever $n = 0$, and $\eta(y) = \eta(y') = 0$ whenever $m = 0$. In other terms, $\mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n, m)$ is the set of valuations that encode a configuration with an error bounded by ϵ, except that the encoding of the counter value 0 is exact. Given shifts $\boldsymbol{k}$ and a valuation $\nu \in \mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n, m)$, we denote $\mathsf{fracp}(\nu) = \nu - \boldsymbol{k}$. This gives the fractional part of the clocks, except when $n = 0$, in which case the



(a) An unperturbed edge. (b) A simpler representation of that edge.

Fig. 3. Unperturbed edges. Perturbator has interest in not perturbing these transitions, since otherwise Controller can go to the target location and win with cost $-\infty$.



(a) Letting Perturbator decide. (b) A simpler representation.

Fig. 4. Module that lets Perturbator decide a successor among ℓ_2 and ℓ_3.

components x and x' are equal to 1, and similarly for y. We say that a valuation ν encodes a configuration (n, m) of the machine if it satisfies (1) for some $\boldsymbol{k}$.

We define modules for incrementation and decrementation with zero-test instructions, which will, once combined, yield the reduction. The modules will be defined on a given list of clocks. For instance, if we describe a module $M(x, y, z, u, t)$ that uses the clocks x, y, z, u, and t in its definition, then $M(z, y, x, t, u)$ is obtained simply by exchanging x and z, and u and t.

Unperturbed edges. Let us first present a construction that prevents Perturbator from perturbing the delays along an edge. The construction only applies to deterministic transitions (with equality constraints) and requires resetting one of the two clocks used in the encoding of a counter. Consider the timed automaton of Fig. 3(a). At ℓ_2, Controller can go to the accepting state where the cost decreases to $-\infty$ if, and only if Perturbator has perturbed by a nonzero amount the transition from ℓ_1 to ℓ_2. Thus, Perturbator does not have interest in perturbing since its objective is to maximize the cost. If there has been no perturbation, the clock values are only increased or decreased by some integers. More precisely, the shifts of all clocks but x' increase by 2, and the shift of x' becomes 0. In the rest, we will use this trick extensively to construct our modules. For better readability, we will represent such *unperturbed edges* by dashed arrows; when clear from context, we may omit the edges leading to accepting sink states (see Fig. 3(b)).

Ask-Perturbator module. In weighted timed automata, unlike in weighted timed games, Perturbator cannot suggest moves since it controls no edges. However, a special construction allows letting Perturbator decide the successor location. We describe in Fig. 4(a) such a construction, which also ensures that the configuration is preserved, up to shifts.

The first edge is deterministic, and Perturbator *can* add any perturbation. Controller then distinguishes between positive and negative perturbations, and only has one possible move accordingly. We disallow perturbations (using un-

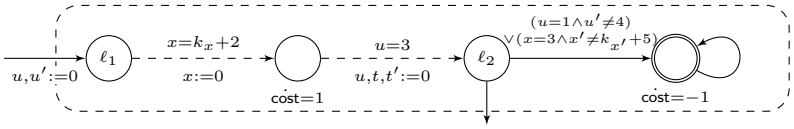


Fig. 5. Module $\mathsf{Add}_{\boldsymbol{k}}^{1+x}(x,u,t)$.

perturbed edges) at the edges leading to ℓ_2 or ℓ_3, so that the configuration is preserved up to shifts. More precisely, the shifts of all clocks x, x', y, y' increase by 3. To simplify the presentation of more complex modules, we will represent this module more compactly as in Fig. 4(b).

Reduction module. In the above modules, we have seen that configurations are preserved up to shifts, but shifts could grow. We present a module that reduces the shifts of all clocks. The module $\mathsf{Reduce}_{\boldsymbol{k}}(x,y,u,t)$ is constructed for each shift vector $\boldsymbol{k}$ (there will be a finite number of these), is deterministic, and constructed using unperturbed edges. The definition of the module is omitted (see [12]); the following lemma summarizes its properties.

Lemma 2. *Let $\delta < \frac{1}{2}$. Assume $\mathsf{Reduce}_{\boldsymbol{k}}(x,y,u,t)$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n,m)$ for some $\epsilon < \frac{1}{2}$. Controller has a strategy to either go to the target location with cost $-\infty$ or to reach location ℓ_2 with valuation ν' satisfying $\nu' = \mathsf{fracp}(\nu) + \boldsymbol{k'}$ where $\boldsymbol{k'}$ is defined as follows: $k'_x = 6, k'_{x'} = 2, k'_y = 5, k'_{y'} = 1$.*

Test Module. In order to verify the incrementation and decrementation, we use the cost variable. We first show how one can add $1{+}\mathsf{fracp}(x)$ and $2{-}\mathsf{fracp}(x)$ to the cost variable, without changing the configuration. The construction is similar to [9]; we adapt it using unperturbed edges. The module $\mathsf{Add}_{\boldsymbol{k}}^{1+x}(x,u,t)$ depicted in Fig. 5 adds $1 + \mathsf{fracp}(x)$ to the cost, leaving the configuration unchanged (up to shifts).

Lemma 3. *Let $\delta < \frac{1}{2}$. Assume module $\mathsf{Add}_{\boldsymbol{k}}^{1+x}(x,u,t)$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n,m)$ for some $\epsilon < \frac{1}{2}$. Controller has a strategy that ensures either reaching a target location with cost $-\infty$ or location ℓ_2 with valuation ν' satisfying $\nu' = \mathsf{fracp}(\nu) + \boldsymbol{k'}$ where $k'_x = 1$, and $k'_\alpha = k_\alpha + 3$ for all $\alpha \in \{x', y, y', z, z'\}$, while the cost increases by $1 + \mathsf{fracp}(x)$.*

We define similarly a module $\mathsf{Add}_{\boldsymbol{k}}^{2-x}(x,u,t)$ that adds $2 - \mathsf{fracp}(x)$ to the cost variable. The module is similar to the one of Fig. 5, except that cost only increases (with slope 1) at location ℓ_1.

Lemma 4. *Let $\delta < \frac{1}{2}$. Assume module $\mathsf{Add}_{\boldsymbol{k}}^{2-x}(x,u,t)$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n,m)$ for some $\epsilon < \frac{1}{2}$. Controller has a strategy that ensures either reaching a target location with cost $-\infty$ or location ℓ_2 with valuation ν' satisfying $\nu' = \mathsf{fracp}(\nu) + \boldsymbol{k'}$ where $k'_x = 1$, and $k'_\alpha = k_\alpha + 3$ for all $\alpha \in \{x', y, y', z, z'\}$, while the cost increases by $2 - \mathsf{fracp}(x)$.*

Concatenating modules $\mathsf{Add}_{\boldsymbol{k}}^{1+x}(x,u,t)$, $\mathsf{Add}_{\boldsymbol{k'}}^{2-x}(z,t,u)$ and $\mathsf{Add}_{\boldsymbol{k''}}^{2-x}(z,u,t)$, we obtain the module $\mathsf{Add}_{\boldsymbol{k}}^{5+x-2z}(x,z,u,t)$, which adds $5 + \mathsf{fracp}(x) - 2\mathsf{fracp}(z)$

to the cost and leads to a target location (we make the last location accepting). Similarly, using concatenation, we define $\mathsf{Add}_{\boldsymbol{k}}^{4+2z-x}(x,z,u,t)$, which increases the cost by $4+2\mathsf{fracp}(z)-\mathsf{fracp}(x)$. One can append at the end of module $\mathsf{Add}_{\boldsymbol{k}}^{4+2z-x}(x,z,u,t)$ an edge that increments the cost by 1, which yields module $\mathsf{Add}_{\boldsymbol{k}}^{5+2z-x}(x,z,u,t)$. Finally, module $\mathsf{Test}^{2z=x}(x,z,u,t)$ is defined by letting Perturbator choose whether to go to $\mathsf{Add}_{\boldsymbol{k'}}^{5+2z-x}(x,z,t,u)$ or to $\mathsf{Add}_{\boldsymbol{k'}}^{5+x-2z}(x,z,t,u)$ (here the new shift vector $\boldsymbol{k'}$ is due to the shift added by the ask-Perturbator module). Note that in both cases, the run ends in a target location. The following property follows from Lemmas 3 and 4.

Lemma 5. *Let $\delta<\frac{1}{2}$. If module $\mathsf{Test}^{2z=x}(x,z,u,t)$ is entered with valuation $\nu\in\mathsf{code}_{\boldsymbol{k}}^{\epsilon}(n,m)$ for some $\epsilon<\frac{1}{2}$, Controller has a strategy to ensure reaching a target location with cost at most $5+|2\mathsf{fracp}(z)-\mathsf{fracp}(x)|$.*

Incrementation Module $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_1,+}$. We define module $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_1,+}$, given in Fig. 6, which simulates the incrementation of counter c_1. We assume first that there are no perturbation. When the module is entered with a valuation in $\mathsf{code}_{\boldsymbol{k}}(n,m)$, we expect Controller to choose the delays so that $z=1+\frac{\mathsf{fracp}(x)}{2}$ at location D. From this point on, the clocks z and z' will switch roles with x and x'. Thus, this corresponds to incrementing the counter c_1 by 1. At location D, Perturbator can either decide to "test" the incrementation has been correctly performed by going to the test module, or to continue the simulation by first passing through the reduction module. Here, $\mathsf{Instr}_{\boldsymbol{k''}}^{j}$ refers to a module among $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_b,+}$, $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_b,-}$ for $b\in\{1,2\}$ (to be defined next) according to the instruction I_j. Now, in the presence of perturbations, Perturbator can perturb the value of z chosen by Controller by δ. So at D, if Perturbator goes to the test module the cost is $5+O(\delta)$, provided that Controller has played correctly. Otherwise, the simulation carries on with $\left|z-\frac{\mathsf{fracp}(x)}{2}\right|\leq\delta$. The following lemma states this formally.

Lemma 6. *Let $\delta<\frac{1}{2}$. Assume module $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_1,+}$ is entered with valuation $\nu\in\mathsf{code}_{\boldsymbol{k}}^{\epsilon}(n,m)$ for some $\epsilon>0$ with $\delta+\epsilon/2<\frac{1}{2}$, and cost 0. Then, Controller has a strategy that ensures that either module $\mathsf{Instr}_{\boldsymbol{k}}{}^{j}$ is entered with a valuation $\nu'\in\mathsf{code}_{\boldsymbol{k'}}^{\delta+\epsilon/2}(n+1,m)$ for some $\boldsymbol{k'}$, or the target location is reached with cost at most $5+2\delta$.*

A decrementation module can be defined following the same ideas.

Lemma 7. *Assume module $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_1,-}$ is entered with valuation $\nu\in\mathsf{code}_{\boldsymbol{k}}^{\epsilon}(n,m)$ and cost 0. Then,*

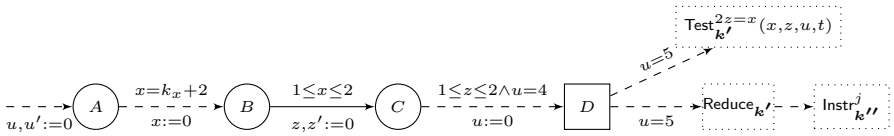


Fig. 6. Module $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_1,+}(x,y,z,u,t)$. The module $\mathsf{Reduce}_{\boldsymbol{k'}}$ refers to $\mathsf{Reduce}_{\boldsymbol{k'}}(z,y,u,t)$, and the module $\mathsf{Instr}_{\boldsymbol{k''}}^{j}$ to $\mathsf{Instr}_{\boldsymbol{k''}}^{j}(z,y,x,u,t)$.

- *If $n = 0$, Controller has a strategy to ensure reaching either $\mathsf{Instr}_{\boldsymbol{k}}{}^{j}$ with the same configuration up to shifts and cost, or an accepting location with cost 0.*
- *If $n \geq 1$, and $\epsilon < \frac{1}{2}$, the play cannot reach $\mathsf{Instr}_{\boldsymbol{k}}{}^{j}$.*
- *If $n \geq 1$, Controller has a strategy that ensures that either module $\mathsf{Instr}_{\boldsymbol{k}}{}^{j'}$ is reached with a valuation $\nu' \in \mathsf{code}_{\boldsymbol{k'}}^{\delta+2\epsilon}(n-1, m)$ for some $\boldsymbol{k'}$, or the target location is reached with cost at most $5 + \epsilon + 2\delta$. Moreover, if $n = 1$, then Controller can ensure that $\nu'(x) = k'_x + 1$.*

Complete reduction. To construct the complete reduction, we define for each instruction I_j of the Minsky machine, a module $\mathsf{Instr}_{\boldsymbol{k}}{}^{j}$ as one of the incrementation or decrementation modules according to the type of I_i. We mark the first location of $\mathsf{Instr}_{\boldsymbol{k}}{}^{1}$ as the initial location. The halting state $\mathsf{Instr}_{\boldsymbol{k}}{}^{n}$ of the machine is an accepting location of the timed automaton. For any machine M, let $\mathcal{A}_M$ denote the weighted timed automaton constructed in this manner, and let ℓ denote the target location obtained by merging all target locations presented in the above construction. Theorem 1 follows from the following proposition.

Proposition 1. *The Minsky machine M halts if, and only if, there is a strategy $\sigma \in S_C(\mathcal{G}^{\mathsf{exs}}(\mathcal{A}_M))$ such that $\mathsf{lim\text{-}cost}_{\sigma}^{exs}(\mathcal{A}_M, \ell) \leq 5$.*

Discussion. One can argue that the undecidability in the excess perturbation game semantics is due to the ability of Controller to test clock values with precision using equality constraints, and in particular in detecting perturbations. This allows for instance letting Perturbator make a discrete choice, as in the above reduction. Hence, this ability and the possibility of disallowing perturbations on some edges make the semantics of weighted timed automata somehow close to that of two-player weighted timed games in the exact semantics for which the optimal-cost reachability is undecidable.

The conservative perturbation game semantics disallows both abilities since Controller is required to suggest delays whose perturbations satisfy the guard of the chosen edge. This excludes equality constraints from guards. Therefore, one cannot encode unperturbed edges nor define the *ask-Perturbator* module as previously. The decidability proof presented in Section 4 confirms these intuitions.

5 Conclusion

In this paper, we showed "robust undecidability" results for weighted timed games: optimal reachability problems remain undecidable under perturbation game semantics. Moreover, the problem even becomes undecidable for weighted timed automata in the excess perturbation game semantics. The undecidability in both cases is due to the ability of either of the players to play precisely, and the other one to check previous delays with precision. We conclude that game semantics does not introduce enough "fuzziness" in the semantics to avoid encoding undecidable languages.

We did not study the value upper-bound problems for the excess perturbation game semantics; we conjecture that it should be undecidable. We believe we could also recover decidability by restricting to closed guards, since then players would not be able to check the non-equality of the clock values.

References

1. R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In ICALP'04, LNCS, p. 122–133. Springer, 2004.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. R. Alur, S. La Torre, and P. Madhusudan. Perturbed timed automata. In HSCC'05, LNCS 3414, p. 70–85. Springer, 2005.
4. R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In HSCC'01, LNCS 2034, p. 49–62. Springer, 2001.
5. E. Asarin and A. Bouajjani. Perturbed Turing machines and hybrid systems. In LICS'01, p. 269–278. IEEE Comp. Soc. Press, 2001.
6. E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In SSC'98, p. 469–474. Elsevier Science, 1998.
7. G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In HSCC'01, LNCS 2034, p. 147–161. Springer, 2001.
8. P. Bouyer, Th. Brihaye, V. Bruyère, and J.-F. Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, 2007.
9. P. Bouyer, Th. Brihaye, and N. Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, 2006.
10. P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In FSTTCS'04, LNCS 3328, p. 148–160. Springer, 2004.
11. P. Bouyer, N. Markey, and O. Sankur. Robust reachability in timed automata: a game-based approach. In ICALP'12, LNCS 7392, p. 128–140. Springer, 2012.
12. P. Bouyer, N. Markey, and O. Sankur. Robust weighted timed automata and games. Research Report LSV-13-10, Laboratoire Spécification et Vérification, ENS Cachan, France, 2013. 25 pages.
13. Th. Brihaye, V. Bruyère, and J.-F. Raskin. On optimal timed strategies. In FORMATS'05, LNCS 3829, p. 49–64. Springer, 2005.
14. K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. *Information and Computation*, 208(6):677–693, 2010.
15. K. Chatterjee, T. A. Henzinger, and V. S. Prabhu. Timed parity games: Complexity and robustness. *Logicical Methods in Computer Science*, 7(4), 2011.
16. M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 33(1-3):45–84, 2008.
17. M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. *Formal Aspects of Computing*, 17(3):319–341, 2005.
18. M. Fränzle. Analysis of hybrid systems: an ounce of realism can save an infinity of states. In CSL'99, LNCS 1862, p. 126–139. Springer, 1999.
19. V. Gupta, T. A. Henzinger, and R. Jagadeesan. Robust timed automata. In HART'97, LNCS 1201, p. 331–345. Springer, 1997.
20. T. A. Henzinger and J.-F. Raskin. Robust indecidability of timed and hybrid systems. In HSCC'00, LNCS 1790, p. 145–159. Springer, 2000.
21. A. Puri. Dynamical properties of timed systems. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.
22. O. Sankur, P. Bouyer, and N. Markey. Shrinking timed automata. In FSTTCS'11, LIPIcs 13, p. 375–386. Leibniz-Zentrum für Informatik, 2011.
23. O. Sankur, P. Bouyer, N. Markey, and P.-A. Reynier. Robust controller synthesis in timed automata. In CONCUR'13, LNCS. Springer, 2013. To appear.