# Weighted automata: model checking and games

Nicolas Markey

November 14, 2008

## Contents



# 1 Weighted automata

In the sequel, we make the classical assumption that all constants are encoded in binary. Sets, on the other hand, are given as the explicit list of their members.

## 1.1 Definitions

**Definition 1.** *A* weighted automaton *is a tuple $\mathcal{A} = \langle S, S_0, \delta \rangle$ where:*

- *$S$ is a (possibly infinite) set of states;*
- *$S_0 \subseteq S$ is a set of initial states;*
- *$\delta \subseteq S \times \mathbb{R} \times S$ is a set of weighted transitions. Unless otherwise stated, we assume that for each $s \in S$, there exists $(r, s') \in \mathbb{R} \times S$ s.t. $(s, r, s') \in \delta$.*

**Definition 2.** *A* finite weighted automaton *is a finite-state weighted automaton with $\delta \subseteq S \times \mathbb{Z} \times S$.*

*A* nonnegative (resp. positive) weighted automaton *is a weighted automaton with $\delta \subseteq S \times \mathbb{R}_{\geq 0} \times S$ (resp. $\delta \subseteq S \times \mathbb{R}_{>0} \times S$).*

*Given a finite set $AP$ of atomic propositions, an* AP-labelled weighted automaton *is a pair $\langle \mathcal{A}, \ell \rangle$ where $\ell \colon S \to 2^{AP}$ labels each state with a set of atomic propositions.*

**Definition 3.** *A* path *(or* trajectory*) in a weighted automaton $\mathcal{A} = \langle S, S_0, \delta \rangle$ is either a single state $s_0$ or a sequence $\pi = (e_i)_{0 \leq i < |\pi|}$ of transitions (we write $e_i = (s_i, w_i, s'_i)$ for each $i$) such that, for all $0 \leq i < i+1 < |\pi|$, $s'_i = s_{i+1}$. The* length *of $\pi$ is 0 if $\pi$ is a single state,and it is the value $|\pi| \in \mathbb{N} \cup \{+\infty\}$ otherwise.*

*Given a path $\pi = ((s_i, w_i, s'_i))_{0 \leq i < |\pi|}$, we write $init(\pi)$ for its first state $s_0$ and $last(\pi)$ for its last state $s'_{|\pi|}$. For a single-state path $\pi = s_0$, its first and last states are $s_0$. Given an integer $0 < j \leq |\pi|$, the* length-$j$ prefix *(or* $j$-th prefix*) of $\pi$ is the path $\pi_{<j} = ((s_i, w_i, s'_i))_{0 \leq i < j}$. The* length-$(|\pi| - j)$ suffix *(or* $j$-th suffix*) of $\pi$ is the path $\pi_{>j} = ((s_{j+i}, w_{j+i}, s'_{j+i}))_{0 \leq i < |\pi| - j}$. The 0-th prefix is $init(\pi)$, the $|\pi|$-th suffix is $last(\pi)$. Finally, the portion of $\pi$ between $j$ and $k$ is the path $\pi_{[j,k]} = ((s_{j+i}, w_{j+i}, s'_{j+i}))_{0 \leq i < k-j}$ when $j < k$; it is the single state $s_j$ when $j = k$, and is empty if $j > k$.*

*A path $\pi = ((s_i, w_i, s'_i))_{0 \leq i < |\pi|}$ is* maximal *if $|\pi| = +\infty$ or $last(\pi)$ has no outgoing edge in $\mathcal{A}$. We write $\mathsf{Path}_f(\mathcal{A})$ for the set of finite paths in $\mathcal{A}$, and $\mathsf{Path}_m(\mathcal{A}, s_0)$ for the set of maximal paths in $\mathcal{A}$ starting in $s$.*

*A path $\pi = ((s_i, w_i, s'_i))_{0 \leq i < |\pi|}$ has a* cycle *if there exists $j < k$ s.t. $s_j = s_k$. A cycle is a finite path starting and ending in the same state. A cycle is* simple *whenever it has no cycle except itself.*

**Remark.** Weighted automata *can be defined in a more abstract way, by means of an underlying semiring. This provides a way of encompassing plain automata (using the boolean semiring), weighted automata as defined above, or stochastic aspects. We refer to e.g. [DG07] for more details on this extension, but stick to the "real-valued" weighted automata in the sequel.*

## 1.2 Shortest and longest paths

In this section, we consider the problems of finding shortest paths, with the "usual" notion of weight of a path: The *weight* of a non-trivial path $\pi =$

$((s_i, w_i, s'_i))_{0 \leq i < |\pi|}$ is

$$w(\pi) = \liminf_{n \to |\pi|} \sum_{0 \leq i < n} w_i$$

The weight of a length-0 path is 0.

**Definition 4.** *We define the following problems:*

| | |
|---|---|
| ***Problem:*** | ***Shortest path*** |
| ***Input:*** | *A finite weighted automaton $\mathcal{A}$, two states $s$ and $t$, and an integer $n$;* |
| ***Question:*** | *Is there a path $\pi$ in $\mathcal{A}$ from $s$ to $t$ s.t. $w(\pi) \leq n$?* |
| ***Problem:*** | ***Shortest acyclic path*** |
| ***Input:*** | *A finite weighted automaton $\mathcal{A}$, two states $s$ and $t$, and an integer $n$;* |
| ***Question:*** | *Is there an acyclic path $\pi$ in $\mathcal{A}$ from $s$ to $t$ s.t. $w(\pi) \leq n$?* |

The corresponding *longest path* problems are defined similarly, and can be reduced to shortest path problems by considering the *dual* automaton, where weights are negated.

**Theorem 5.** *The shortest-path problem is in PTIME and NLOGSPACE-hard. The shortest acyclic path problem is NP-complete.*

*Proof.* The shortest-path problem is obviously NLOGSPACE-hard, since reachability in a directed graph already is (encoding of a non-deterministic Turing machine with logarithmic-size tape).

The shortest path problem can be solved by applying the Bellman-Ford algorithm [Bel58]. This algorithm iteratively computes the length of the shortest path of length $i$ from $s$ to each state of $\mathcal{A}$, starting with $i = 0$.

- initially, there are no path from $s$ to any other state with length 0. Thus $d_0(s) = 0$ and $d_0(q) = +\infty$ for any $q \neq s$.
- for each $i$ ranging from 0 to $n$ with $n = |S|$, apply the following procedure: first set $d_{i+1}(q)$ to $d_i(q)$ for each state $q$. Then, for each edge $(q, w, q') \in \delta$, if $d_{i+1}(q') > d_i(q) + w$, then $d_{i+1}(q')$ is set to $d_i(q) + w$.
- finally, if $d_n(q) > d_{n+1}(q)$, then $d(q)$ is set to $-\infty$. Then, for each state $q'$ reachable from $q$ with $d(q) = -\infty$, set $d(q')$ to $-\infty$. If some $d(q)$ is not set after this procedure, then $d(q) = d_n(q)$.

Obviously, this procedure runs in quadratic time (namely $O(|S| \cdot |\delta|)$). We now claim and prove that it is correct:

**Theorem 6.** *For each state $q$,*

- *if $d(q) = +\infty$, then $q$ is not reachable from $s$;*
- *if $d(q) = -\infty$, then for any $M \in \mathbb{Z}$, there is a path from $s$ to $q$ with weight less than $M$;*

- *otherwise, $d(q)$ is the weight of the shortest path from $s$ to $q$.*

*Proof.* The proof relies on the following lemma:

**Lemma 7.** *For each $i$ between 0 and $n+1$ and each $q \in S$,*

- *if $d_i(q)$ is finite, it corresponds to the weight of some path in $\mathcal{A}$ from $s$ to $q$;*
- *if there is a path from $s$ to $q$ of length at most $i$, then $d_i(q)$ is less than or equal to the weight of the shortest path from $s$ to $q$ of length at most $i$.*

*Proof.* The proof is by induction on $i$: when $i = 0$, the first claim is obvious since $d_0(q)$ is finite only when $q = s$, and $d_0(s) = 0$ is the length of the trivial path from $s$ to itself. Simlarly, there is a path from $s$ to $q$ of length 0 (if, and) only if, $q = s$; in that case, $d_0(s) = 0$ is (less than or) equal to the shortest path from $s$ to itself of length 0.

Now assume that that the result holds for some $i$ between 0 and $n$. We prove that it still holds at step $i+1$. First, if $d_{i+1}(q) = d_i(q)$, then the results follows from the induction hypothesis. Otherwise, $d_{i+1}(q)$ is set to $d_i(q') + w$ for some edge $(q', w, q)$. By induction hypothesis, there is a path from $s$ to $q'$ of length $d_i(q')$. Appending the transition $(q', w, q)$ to this path yields a path from $s$ to $q$ of weight $d_{i+1}(q)$, as required.

For the second claim, let $\pi$ be one of the shortest paths from $s$ to $q$ of length at most $i+1$, assuming that such a path exists. Let $(q', w, q)$ be its last transition, and consider the prefix $\pi_{<|\pi|-1}$, and its last state $q'$. It must be the case that $\pi_{<|\pi|-1}$ is one of the shortest path from $s$ to $q'$ of length at most $i$, since otherwise we could find a path from $s$ to $q$ of length at most $i+1$ and weight strictly less than $w(\pi)$. The induction hypothesis entails that $d_i(q')$ is less than or equal to the weight of the shortest path from $s$ to $q'$ of length at most $i$. Thus $d_{i+1}(q)$ is less than $d_i(q') + w$, which is in turn less than the weight of $\pi$. □

Assume now that $d_n(q) > d_{n+1}(q)$ for some state $q$. From the Lemma above, this means that there is a path $\pi$ from $s$ to $q$ with weight $(d_{n+1}(q))$ strictly less than the shortest paths from $s$ to $q$ of length less than or equal to $n$. Then $\pi$ contains two occurrences of some state $t$, and if we remove the corresponding cycle from $\pi$, we get a valid path of length less than or equal to $n$, hence of weight strictly larger than the weight of $\pi$. This means that the cycle we removed has negative global weight. Thus the weight of a path between $s$ and $q$ can be made arbitrarily small, by repeating this negative cycle; similarly for any state reachable from such a state $q$.

Conversely, assume that $d(q)$ is finite (*i.e.*, $q$ is reachable but $d(q)$ has not been set to $-\infty$), and that $q$ is not reachable *via* a negative cycle. From the first statement of the Lemma above, there is a path from $s$ to $q$ with weight (at most) $d(q)$. Among the paths of weight at most $d(q)$, pick a path $\pi$ with as few transitions as possible. If $\pi$ has a cycle, this cycle must be nonnegative (by hypothesis), and removing this cycle would yield a path shorter than $\pi$ (in terms

of its number of transitions) with weight at most $d(q)$. This is a contradiction. Hence $\pi$ has no cycle, so that its length is at most $n$. Since $d(q) = d_n(q)$ is less than the weight of the shortest path from $s$ to $q$ having at most $n$ transitions (Lemma 7), we get that $\pi$ has weight exactly $d(q)$, and that $d(q)$ is exactly the weight of the shortest path from $s$ to $q$.

Finally, assume that for some $q$, $d(q)$ is finite but $q$ is reachable *via* a negative cycle. Let $q'$ be a state of this negative cycle $\pi = ((q_i, w_i, q'_i))_{0 \leq i < |\pi|}$ (with $q'_{|\pi|-1} = q_0$). Since $q$ is reachable from $q'$ and $d(q)$ is finite, we also have that $d(q')$ is finite. The same holds of all the states in the negative cycle around $q'$. In particular, $d(q_0)$ is finite, which means that $d_n(q_0) \leq d_{n+1}(q_0)$. This means in particular that $d_n(q'_0) \leq d_n(q_0) + w_0$. The same holds for the other states of the cycle:

$$\begin{aligned} d_n(q'_0) &\leq d_n(q_0) + w_0 \\ d_n(q'_1) &\leq d_n(q_1) + w_1 \\ &\cdots \\ d_n(q'_{|\pi|-1}) &\leq d_n(q_{|\pi|-1}) + w_{|\pi|-1} \end{aligned}$$

Summing up these inequalities, and since $d_n(q'_i) = d_n(q_{i-1 \mod |\pi|})$, we end up with

$$\sum_{0 \leq i < |\pi|} w_i \geq 0,$$

which contradicts the fact that $\pi$ is a negative cycle, and concludes the proof of Theorem 6. □

We now turn to the acyclic-shortest-path problem. That it can be solved in NP is quite obvious, since it suffices to guess the acyclic path step-by-step. To prove hardness in NP, we reduce the Hamiltonian-path problem to our problem:

| | |
|---|---|
| **Problem:** | **Hamiltonian path** |
| **Input:** | a finite automaton $\mathcal{A}$, a state $s$; |
| **Question:** | Is there a path from $s$ visiting each state of $\mathcal{A}$ exactly once? |

This problem is known to be NP-complete [GJ79]. The reduction is rather obvious: visiting each state at most once entails that the path will be acyclic, and visiting each state at least once requires that a witnessing path should contain at least $|S|$ transitions. The only difficulty lies in the fact that the final state is not fixed in the Hamiltonian-path problem. We overcome this difficulty by adding a sink-state $s_f$ to $\mathcal{A}$, with an incoming transition from all the other states and no outgoing transition. Then there is an Hamiltonian path in $\mathcal{A}$ from $s$ iff there is one that ends in $s_f$ in the modified automaton.

Now, the reduction is achieved as follows: from $\mathcal{A}$, we add the sink-state and put weight $-1$ on each transition. We write $\mathcal{A}'$ for the resulting weighted automaton. Writing $n+1$ for the number of states of $\mathcal{A}'$ (the $n$ states of $\mathcal{A}$ plus the sink-state), there is an Hamiltonian path in $\mathcal{A}$ from $s$ if, and only if,

there is an acyclic path in $\mathcal{A}'$ from $s$ to $s_f$ with total weight less than or equal to $-n$. □

**Exercice 1** ⋆ Consider the following problem:

| | |
|---|---|
| **Problem:** | **Exact shortest path** |
| **Input:** | A finite weighted automaton $\mathcal{A}$, two states $s$ and $t$, and an integer $n$; |
| **Question:** | Is the length of the shortest path(s) from $s$ to $t$ exactly $n$? |

Prove that this problem is in PTIME.

**Exercice 2** ⋆ Consider the following problem:

| | |
|---|---|
| **Problem:** | **Exact length path** |
| **Input:** | A finite weighted automaton $\mathcal{A}$, two states $s$ and $t$, and an integer $n$; |
| **Question:** | Is there a path from $s$ to $t$ of length exactly $n$? |

Prove that this problem is NP-complete.

## 1.3 Optimal mean payoff

**Definition 8.** *Let $\pi = ((s_i, w_i, s'_i))_{0 \leq i < |\pi|}$ be a path with $|\pi| > 0$ in a weighted automaton $\mathcal{A}$. Let $\lambda \in (0, 1)$. The* $\lambda$-discounted payoff *of $\pi$ is defined as*

$$w_\lambda(\pi) = \liminf_{n \to |\pi|} \sum_{0 \leq i < n} \lambda^i w_i.$$

**Definition 9.** *Let $\pi = ((s_i, w_i, s'_i))_{0 \leq i < |\pi|}$ be a path with $|\pi| > 0$ in a weighted automaton $\mathcal{A}$. The* mean cost *(or* mean payoff*) of $\pi$ is defined as*

$$m(\pi) = \liminf_{n \to |\pi|} \frac{1}{n} \cdot \sum_{0 \leq i < n} w_i.$$

**Definition 10.** *The optimal mean-payoff path problem is defined as follows:*

| | |
|---|---|
| ***Problem:*** | ***Optimal mean-payoff path*** |
| ***Input:*** | *A weighted automaton $\mathcal{A}$, a state $s_0$, an integer $n$;* |
| ***Question:*** | *Is there a maximal path $\pi$ in $\mathcal{A}$ from $s_0$ such that $m(\pi) \leq n$?* |

**Theorem 11.** *The optimal mean-payoff path problem can be decided in PTIME (more precisely, in $O(|S|^2 \cdot |\delta|)$).*

*Proof.* We first transform the problem as follows: we consider the weighted automaton $\mathcal{A}'$ obtained from $\mathcal{A}$ by substracting $n$ to the weight of each transition. This clearly shifts the mean payoff of any (finite or infinite) path by $-n$, so that the optimal-mean-payoff-path problem can be reduced to the case where $n = 0$.

Now, assume that there is a maximal path $\pi$ achieving mean-payoff less than 0. We first consider the case where this path is finite, and we pick one

of the shortest ones (in terms of its number of transitions). Assume that it contains a cycle:

- if this cycle has nonnegative weight, then the path obtained by dropping this cycle is shorter and still has mean-payoff less than or equal to 0.
- otherwise, the cycle has negative weight, and iterating this cycle provides us with an infinite path having negative mean-payoff.

Our algorithm will then work as follows:

- first look at finite paths of length less than $|S|$: for each $k$ between 0 and $|S|$, compute the length of the shortest path of length *exactly* $k$ between $s_0$ and each state of $S$. This is achieved in time $O(|S|^2 \cdot |\delta|)$. We then exit positively if one of those paths ending in a state with no outgoing edge has weight less than or equal to 0.
- otherwise, we know that, if there is a path with non-positive mean payoff $v$, then there is an infinite one. We prove that in this case, there is a reachable simple cycle with mean-payoff less than or equal to $v$. For a contradiction, let $\pi$ be an infinite path with mean cost $v$, and assume that the smallest mean payoff $w$ of the simple cycles reachable from $s_0$ in $\mathcal{A}$ is strictly larger than $v$.

  Let $q$ be a state of $\mathcal{A}$ that occurs infinitely often along $\pi$, and consider the sequence $(\pi_k)_{k\geq 0}$ of prefixes of $\pi$ up to the $k+1$-st occurrence of $q$; write $l_k$ and $w_k$ for the total length and weight of $\pi_k$. We have that

  $$w_{k+1} \geq w_k + (l_{k+1} - l_k) \cdot w.$$

  Indeed, $\pi_{k+1}$ is obtained from $\pi_k$ by appending a cycle from $q$ back to itself, which can be decomposed as several simple cycles.

  In the end, we have that

  $$w_k \geq w_0 + (l_k - l_0) \cdot w.$$

  As $k$ tends to $+\infty$, we get that the mean-payoff of $\pi$ is larger than or equal to $w$, which is strictly larger than $v$.

  Conversely, if there is a reachable simple cycle of mean-payoff $v$, then obviously, there is an infinite path having mean-payoff $v$. Hence computing the smallest mean-payoff of a path in $\mathcal{A}$ amounts to computing the smallest mean-payoff of its reachable simple cycles. This can be done using Karp's algorithm [Kar78], which we now explain.

  For each state $q$ and each integer $k$, we define $d_k(s_0, q)$ as the minimum weight of a $k$-step path from $s_0$ to $q$ (and $+\infty$ is no such path exists). We also write $d(s_0, q)$ as $\min_k d_k(s_0, q)$.

  Assume that $w = 0$. This implies in particular that no reachable simple cycle has negative weight: since a simple cycle has finite length, it would

yield a simple cycle with negative mean payoff. As a consequence, for any $q$, the shortest path from $s_0$ to $q$ can be achieved by an acyclic path, which entails that

$$d(s_0, q) = \min_{0 \leq k < |S|} d_k(s_0, q).$$

In particular, $d_{|S|}(s_0, q) \geq d(s_0, q)$, so that, for each $q$, there is a $k$ for which $d_{|S|}(s_0, q) \geq d_k(s_0, q)$. As a consequence, for any $q \in S$,

$$\max_{0 \leq k < |S|} \frac{d_{|S|}(s_0, q) - d_k(s_0, q)}{|S| - k} \geq 0. \tag{1}$$

Now, let $c$ be a reachable simple cycle of total weight $w = 0$. Let $s$ and $s'$ be two states visited by $c$. Let $x$ be the weight of the portion of $c$ from $s$ to $s'$. Clearly, $d(s_0, s') \leq d(s_0, s) + x$. Conversely, since $c$ has weight 0, there is a path from $s'$ to $s$ of weight $-x$, so that $d(s_0, s) \leq d(s_0, s') - x$. Hence $d(s_0, s') = d(s_0, s) + x$. Let $u$ be a state of $c$, and $\pi$ be an acyclic shortest path from $s_0$ to $u$. Then $\pi$ has length less than $|S|$, and, following $c$ for $|S| - |\pi|$ steps from $u$ yields a path $\pi'$ ending in a state $v$ such that $\pi'$ is a shortest path from $s_0$ to $v$, and has $|S|$ steps. As a consequence, $d_{|S|}(s_0, v) \leq d_k(s_0, v)$ for any $k$, which entails that

$$\max_{0 \leq k < |S|} \frac{d_{|S|}(s_0, v) - d_k(s_0, v)}{|S| - k} = 0. \tag{2}$$

Combining (1) and (2), we get

$$\min_{s \in S} \max_{0 \leq k < |S|} \frac{d_{|S|}(s_0, s) - d_k(s_0, s)}{|S| - k} = 0. \tag{3}$$

We now relax the assumption that $w = 0$. Considering the weighted automaton obtained from $\mathcal{A}$ by decreasing all weights by $w$, we get a weighted automaton with optimal mean-payoff 0, for which Equation (3) holds. We get

$$\min_{s \in S} \max_{0 \leq k < |S|} \frac{(d_{|S|}(s_0, s) - |S| \cdot w) - (d_k(s_0, s) - k \cdot w)}{|S| - k} = 0,$$

which immediately entails that

$$\min_{s \in S} \max_{0 \leq k < |S|} \frac{d_{|S|}(s_0, s) - (d_k(s_0, s)}{|S| - k} = w.$$

This provides us with a polynomial-time-computable expression of $w$. □

**Exercice 3** ⋆⋆⋆ Define the optimal-$\lambda$-discounted-payoff problem, and prove that it can be solved in PTIME (see [dAFH+05]).

## 1.4 Weighted temporal logics

Let AP be a finite set of atomic propositions.

**Definition 12.** *The* full weighted computation-tree logic *(WCTL$^*$) is the logic whose syntax is defined by the following grammar:*

$$\begin{aligned} \mathsf{WCTL}^* \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= \phi_s \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \,\mathbf{U}_I\, \phi_p \end{aligned}$$

*where $p$ ranges over AP and $I$ ranges over the set of intervals. Formulas $\phi_s$ are called* state formulas*, formulas $\phi_p$ are called* path formulas*.*

*We will consider two classical fragments of this logic:*

- *WCTL (*weighted computation-tree logic*) is the fragment of WCTL$^*$ where* path formulas *$\phi_p$ are restricted to $\phi_s \,\mathbf{U}_I\, \phi_s$ and the negation thereof;*
- *WLTL (*weighted linear-time temporal logic*) is the fragment of WCTL$^*$ where only one of $\mathbf{E}$ and $\mathbf{A}$ appears in the formula, and it is required to appears at the root of the formula.*

The semantics of these logics is defined from that of WCTL$^*$:

**Definition 13.** *Let $\langle \mathcal{A}, \ell\rangle$ be a labelled weighted automaton, $\pi = ((s_i, w_i, s'_i))_{0\le i<|\pi|}$ be a maximal path in this automaton, and $k < |\pi|$ be a position along $\pi$. The semantics of WCTL$^*$ is defined recursively as follows:*

$$\begin{aligned}
\langle \mathcal{A}, \ell\rangle, \pi, k \models p &\iff p \in \ell s_k \\
\langle \mathcal{A}, \ell\rangle, \pi, k \models \neg\phi_s &\iff \langle \mathcal{A}, \ell\rangle, \pi, k \not\models \phi_s \\
\langle \mathcal{A}, \ell\rangle, \pi, k \models \phi_s \vee \phi'_s &\iff \langle \mathcal{A}, \ell\rangle, \pi, k \models \phi_s \text{ or } \langle \mathcal{A}, \ell\rangle, \pi, k \models \phi'_s \\
\langle \mathcal{A}, \ell\rangle, \pi, k \models \mathbf{E}\phi_p &\iff \exists \pi' \in \mathsf{Path}_m(\mathcal{A}, s_k).\ \langle \mathcal{A}, \ell\rangle, \pi', 0 \models \phi_p \\
\langle \mathcal{A}, \ell\rangle, \pi, k \models \mathbf{A}\phi_p &\iff \forall \pi' \in \mathsf{Path}_m(\mathcal{A}, s_k).\ \langle \mathcal{A}, \ell\rangle, \pi', 0 \models \phi_p \\
\langle \mathcal{A}, \ell\rangle, \pi, k \models \neg\phi_p &\iff \langle \mathcal{A}, \ell\rangle, \pi, k \not\models \phi_p \\
\langle \mathcal{A}, \ell\rangle, \pi, k \models \phi_p \vee \phi'_p &\iff \langle \mathcal{A}, \ell\rangle, \pi, k \models \phi_p \text{ or } \langle \mathcal{A}, \ell\rangle, \pi, k \models \phi'_p \\
\langle \mathcal{A}, \ell\rangle, \pi, k \models \phi_p \,\mathbf{U}_I\, \phi'_p &\iff \exists k' > 0.\ \langle \mathcal{A}, \ell\rangle, \pi, k+k' \models \phi'_p \\
&\qquad \text{and } \forall k''.\ 0 < k'' < k' \Rightarrow \langle \mathcal{A}, \ell\rangle, \pi, k+k'' \models \phi_p \\
&\qquad \text{and } w(\pi_{[k,k+k']}) \in I.
\end{aligned}$$

**Exercice 4 ⋆** Prove that formula $\mathbf{A}\phi_p$ can be expressed using $\mathbf{E}\neg\phi_p$.

**Theorem 14.** *Let $\phi_s$ be a state formula of WCTL$^*$, $\langle \mathcal{A}, \ell\rangle$ be a labelled weighted automaton, $s$ be a state of $\mathcal{A}$, and $\pi$ and $\pi'$ be two maximal paths of $\mathcal{A}$ starting from $s$. Then*

$$\langle \mathcal{A}, \ell\rangle, \pi, 0 \models \phi_s \iff \langle \mathcal{A}, \ell\rangle, \pi', 0 \models \phi_s$$

**Exercice 5 ⋆** Prove Theorem 14.

**Definition 15.** *Given a labelled weighted automaton $\langle \mathcal{A}, \ell \rangle$, a state $s$ of $\mathcal{A}$ and a state formula $\phi_s$, we write $\langle \mathcal{A}, \ell \rangle, s \models \phi_s$ if there exists a maximal path $\pi$ starting in $s$ s.t. $\langle \mathcal{A}, \ell \rangle, \pi, 0 \models \phi_s$.*

**Example.** *Formula $\mathbf{E} \top \mathbf{U}_{\leq 5}\, p$ ($\mathbf{U}$ is read "until") expresses that there exists a path that reaches a $p$-state in less than $p$ time units. This formula belongs to all three logics we just defined. Since reachability properties are frequent, we define a shorthand $\mathbf{EF}_{\leq 5}\, p$ for this kind of modalities ($\mathbf{F}$ is read "eventually").*

*The dual modality of $\mathbf{F}$ is also very useful: formula $\neg(\, \mathbf{EF}_{\leq 5}\, p)$ is equivalent to $\mathbf{AG}_{\leq 5}\, \neg p$, where $\mathbf{G}$ is read "always".*

*Formula $\mathbf{EG}\, \mathbf{F}\, p$ states that there is a path along which $p$ occurs infinitely often. This formula belongs to WCTL$^*$ and WLTL, be can be shown not to be expressible in WCTL.*

*Formula $\mathbf{AG}\, (\mathbf{EF}\, p)$ states that, at any position along any path, it is possible to reach a $p$-state. This formula belongs to both WCTL and WCTL$^*$, but cannot be expressed in WLTL.*

*Finally, formula $\mathbf{AG}\, (p \Rightarrow \mathbf{EG}\, \mathbf{F}\, q)$ combines both properties. It belongs to WCTL$^*$ but is not expressible in WLTL or WCTL.*

**Exercice 6** ⋆ Using modality $\mathbf{U}$, define a shorthand modality $\mathbf{X}$ s.t. $\mathbf{AX}\, p$ means "$p$ holds in the all the 1-step successor states". What does $\neg \mathbf{X}\, \neg \phi$ mean?

**Exercice 7** ⋆⋆ Prove that $\mathbf{EG}\, \mathbf{F}\, p$ and $\mathbf{EG}\, \mathbf{EF}\, p$ are not equivalent (first define what *equivalent* means).

**Exercice 8** ⋆⋆⋆⋆ Prove the expressiveness claims above.

**Exercice 9** ⋆⋆ Prove that modalities $\mathbf{E}\, \mathbf{U}_I$ and $\mathbf{EG}_I$ are sufficient for expressing any WCTL formula for the semantics of Definition 13.

**Definition 16.** *The model-checking problem is defined as follows:*

| | |
|---|---|
| ***Problem:*** | **WCTL$^*$** ***model checking*** |
| ***Input:*** | *A WCTL$^*$ formula $\phi_s$, a labelled nonnegative weighted automaton $\langle \mathcal{A}, \ell \rangle$, a state $s$;* |
| ***Question:*** | *Does $\langle \mathcal{A}, \ell \rangle, s \models \phi_s$?* |

*We define the WCTL and WLTL model-checking problems similarly.*

**Theorem 17.** *The WCTL model-checking problem is $\Delta_2^P$-complete.*

**Remark.** *Before we proceed to the proof, let us make some remarks about $\Delta_2^P$ and the polynomial-time hierarchy. An $A$-*oracle Turing machine*, where $A$ is a decision problem, is a Turing machine having an extra write-only tape, called the* oracle tape *and three special states, which we denote with $q_?$, $q_{yes}$ and $q_{no}$. The machine runs as a classical Turing machine, except that*

- *it can write on its oracle tape;*

- *when it enters the $q_?$-state, it immediately goes to one of the states $q_{yes}$ or $q_{no}$ depending on whether the content of the oracle tape is a positive instance of $A$ or not.*

*For instance, a SAT-oracle Turing machine can be seen as a Turing machine having access to a "magic" SAT-solver. Any other NP-complete problem would define an "equivalent" Turing machine, and we generally say NP-oracle Turing machine to denote any of these equivalent Turing machines. Complexity can then be measured in terms of time and space of the computations, as well as in terms of the number of calls to the oracle. For instance, an NP-oracle Turing machine running in deterministic polynomial time defines the class $\mathsf{PTIME}^{\mathsf{NP}}$, which is precisely the class $\Delta_2^P$ of the above Theorem.*

**Exercice 10** ⋆ Prove that coNP is included in $\Delta_2^{\mathsf{P}}$, and that $\Delta_2^{\mathsf{P}}$ is included in PSPACE.

We now proceed to the proof of Theorem 17.

*Proof.* We begin with membership in $\Delta_2^{\mathsf{P}}$. The algorithm consists in labelling each state of the weighted automaton with the state-subformulas it satisfies. For each single-modality formula, we prove that the labelling procedure can be achieved in NP. Since we have to apply this labelling procedure for all the subformulas of the WCTL formula, our algorithm will require a polynomial number of calls to our NP labelling procedure, hence the membership in $\Delta_2^{\mathsf{P}}$.

- assume that a state $q$ of $\mathcal{A}$ satisfies formula $\mathbf{E}\phi_1 \mathbf{U}_{[a,b]} \phi_2$, where $b$ is bounded. This means that there is a path reaching $\phi_2$ with cost within $a$ and $b$ and visiting only $\phi_1$-states. Pick one of the shortest such paths, and assume that it contains at least $b+1$ occurrences of the same transition. Since $\mathcal{A}$ only has nonnegative costs, it must be the case that the cost between two occurrences of this transition is 0. Thus there must exist a shorter witness, which is a contradiction. We conclude that if $q$ satisfies $\mathbf{E}\phi_1 \mathbf{U}_{[a,b]} \phi_2$, then there is a witness path visiting each transition at most $b$ times. Our NP procedure then works as follows:
  - for each transition of $\mathcal{A}$, guess a number between 0 and $b$, which is intended to represent the number of times this transition is fired in the witness path;
  - check that these guesses correspond to a path: apart from the first and last states of the path, each state must be exited as many times as it is entered;
  - check that apart from the first and last states, all the states satisfy $\phi_1$;
  - check that the last state satisfies $\phi_2$;
  - check that the weight of the path lies between $a$ and $b$.

  It is clear enough that this procedure succeeds if, and only if, a witnessing path exists.

- the procedure is slightly different for the case where $b = +\infty$. for that case, we proceed in three steps:
  - guess the transition $(q, w, q')$ at which the cost will reach $a$. It must satisfy $w \geq 1$;
  - guess a path reaching $q$ with cost in $[a - w, a - 1]$;
  - prove that $q$ satisfies $\phi_2 \vee \mathbf{E}\phi_1 \,\mathbf{U}\, \phi_2$.

  Again, it is easily proved that this procedure is correct.
- for formulas of the form $\mathbf{EG}_{[a,b]}\, \phi_1$ where $b$ is bounded, the procedure is similar:
  - choose one of the following two options: either cost $a$ will never be reached, or it will. In the first case, the problem amounts to checking that $\mathbf{EG}\, \phi_1$ holds. In the second case,
    - guess the transition $(q, w, q')$ at which cost $a$ is reached;
    - guess a path reaching $q$ with cost $c$ in $[a - w, a - 1]$;
    - if $c + w > b$, then we are done. Otherwise, we have to prove that $q'$ satisfies $\phi_1 \wedge \mathbf{EG}_{[0,b-(c+w)]}\, \phi_1$. This formula can be satisfied in two ways: either via a path along which $\phi_1$ always holds, or by a path along which the cost $b - (c + w)$ is eventually reached. The first case is handled via the classical CTL model-checking algorithm. The second one can be reformulated as $\mathbf{E}\phi_1 \,\mathbf{U}_{[b-(c+w)+1,+\infty)}\, \top$.
- finally, the case where $b = +\infty$ corresponds to the first case (always $\phi_1$ once $a$ has been reached).

$\Delta_2^{\mathsf{P}}$-hardness is left as Exercice 12. □

**Exercice 11** ⋆⋆ Let $\mathsf{WCTL}_{\leq,\geq}$ be the fragment of WCTL where the intervals decorating the modalities are of the form either $[0, p]$ or $[p, +\infty)$. Prove that $\mathsf{WCTL}_{\leq,\geq}$ model checking is PTIME-complete.

**Exercice 12** ⋆⋆⋆⋆ Show that the exact-reachability problem (*"is is possible to reach state $s$ with cost exactly $c$?"*) is NP-complete. Prove that the $\Delta_2^{\mathsf{P}}$ algorithm for WCTL model-checking is optimal (see [LMS02]).

**Exercice 13** ⋆⋆ Weighted automata could be extended to allow (possibly unbounded) intervals on their transitions, as a way to encode that several weights are possible from one state to another one. Show that all our complexity results extend to this case.

**Exercice 14** ⋆ In our definition of the model-checking problems, we restricted to nonnegative weighted automata. What would be the problem with negative weights?

**Theorem 18.** *The WLTL and WCTL$^*$ model-checking problems are EXPSPACE-complete.*

*Proof.* We begin with proving that WLTL model-checking is EXPSPACE-complete. This will entail EXPSPACE-hardness of WCTL$^*$ model-checking, but also membership in EXPSPACE: it suffices to label each state of the automaton with the subformulas it satisfies, by applying the WLTL model-checking algorithm. The resulting algorithm is in PTIME$^{\text{EXPSPACE}}$, which is easily seen to equal EXPSPACE.

The EXPSPACE algorithm consists in building a so-called tableau for the WLTL formula. We first introduce some notation: given a WLTL formula $\psi$ and a nonnegative integer $n$, we define $\psi^n$ recursively as

- $\psi^0 = \psi$,
- $\psi^{n+1}$ is obtained from $\psi^n$ by replacing each interval constraint of the form $\langle a; b\rangle$ with $\langle a-1; b-1\rangle \cap (0; +\infty)$.

The set of *subformulas* $\mathsf{Subf}(\phi)$ of a formula $\phi$ of WLTL is then the smallest set of formulas containing $\phi$ and satisfying the following rules:

- $\top$ and $\bot$ belong to $\mathsf{SF}\phi$;
- $\neg\phi_1 \in \mathsf{Subf}(\phi)$ iff $\phi_1 \in \mathsf{Subf}(\phi)$, where we identify $\neg\neg\phi_1$ with $\phi_1$;
- if $\phi_1 \vee \phi_2 \in \mathsf{Subf}(\phi)$, then both $\phi_1$ and $\phi_2$ are in $\mathsf{Subf}(\phi)$;
- if $\phi_1 \,\mathbf{U}_{\langle a,b\rangle} \phi_2 \in \mathsf{Subf}(\phi)$, then $\mathbf{X}\,\phi_1$, $\mathbf{X}\,\phi_2$, $\mathbf{X}\,\phi$ and $(\phi_1 \,\mathbf{U}\, \phi_2)^n$, for all $n$ appearing on the transitions of the automaton, are in $\mathsf{Subf}(\phi)$.

It is easily seen that the size of $\mathsf{Subf}(\phi)$ is bounded by $4 \cdot |\phi| \cdot M$ where $|\phi|$ is the number of modalities and boolean operators in $\phi$ and $M$ is the product of all constants in $\phi$.

We extend $\mathsf{Subf}(\phi)$ with extra propositions $d_n$, one for each $n$ appearing in the weighted automaton, used for indicating that the latest transition has weight $n$. We write $\mathsf{Subf}^*(\phi)$ for the resulting set.

A subset $\Theta$ of $\mathsf{Subf}^*(\phi)$ is *consistent* if

- $\Theta$ contains exactly one of the $d_n$'s;
- $\top \in \Theta$;
- $\phi_1 \in \Theta$ iff $\neg\phi_1 \notin \Theta$ for all $\phi_1 \in \mathsf{Subf}(\phi)$;
- if $\phi_1 \vee \phi_2 \in \Theta$, then at least one of $\phi_1$ and $\phi_2$ is in $\Theta$;
- if $\phi_1 \,\mathbf{U}_{\langle a,b\rangle} \phi_2 \in \Theta$, then either $\mathbf{X}\,\phi_2$ is in $\Theta$, or both $\mathbf{X}\,\phi_1$ and $\mathbf{X}\,\phi$ are.

The *tableau* for $\phi$ is the automaton whose states are the consistent sets of subformulas, and such that there is a transition from $\Theta$ to $\Theta'$ of weight $n$ (which implies that $\Theta'$ contains $d_n$) iff, for all $\mathbf{X}\,\psi$ formula in $\mathsf{Subf}(\phi)$,

$$\mathbf{X}\,\psi \in \Theta \quad \Leftrightarrow \quad \psi^n \in \Theta'.$$

A $\phi$-path in the tableau is an infinite trajectory $(\Theta_i, n_i, \Theta'_i)_{0\leq i}$ starting from some state containing $\phi$, and along which all eventualities are fulfilled in time, meaning that if $\phi_1\,\mathbf{U}_{\langle a;b\rangle}\,\phi_2$ appear in some $\Theta_i$, then $\phi_2^N$ appears later in $\Theta_j$ with $N = \sum_{i<k\leq j} n_k$.

We then have the following lemma:

**Lemma 19.** *Let $\mathcal{A}$ be a nonnegative weighted automaton, and $\mathbf{E}\phi \in$ WLTL. Then $\mathcal{A}$ has a path satisfying $\phi$ iff the product of $\mathcal{A}$ with the tableau for $\phi$ has a $\phi$-path.*

It can be shown that if a $\phi$-path exists in the product, then also an ultimately periodic one exists, and it has size polynomial in the size of the product. In the end, the algorithm consists in non-deterministically guessing the ultimately periodic witness step by step, which can be achieved in exponential space.

Hardness in EXPSPACE is obtained by encoding an exponential space Turing machine (Exercice 15). □

**Exercice 15** ⋆⋆ Prove that the WLTL model-checking problem is EXPSPACE-complete.

**Exercice 16** ⋆⋆ A *unitary weighted automaton* is a finite weighted automaton with costs in $\{0, 1\}$. Prove that the WCTL model-checking problem for unitary weighted automata is PTIME-complete.

Our semantics for weighted automata uses atomic steps for the transitions. In order to get closer to the semantics of timed automata, we could define a *semi-continuous* semantics, where a step of weight $n$ would correspond to $n$ successive steps fo weight 1. The modified semantics is defined as follows: now, a configuration of the automaton is a pair containing the state and the "time" elapsed in the present state. As for timed automata, we have two kinds of transitions:

- action transitions: $((q, i), 0, (q', 0))$ when $(q, 0, q') \in \delta$, and $((q, i), 1, (q', 0))$ when $(q, i + 1, q') \in \delta$;
- delay transitions: $((q, i), 1, (q, i + 1))$, when $i$ is less than the maximal constant appearing in the automaton.

A state $q$ satisfies a formula $\phi$ of some temporal logic iff the correpsonding state $(q, 0)$ in the weighted automaton above satisfies the same formula.

**Example.** *Figure 1 explains the intuition behing the semi-continuous semantics on a small example. It is easily noticed that the semi-continuous is really different from the discrete one, even when no quantitative constrinat is used: for*

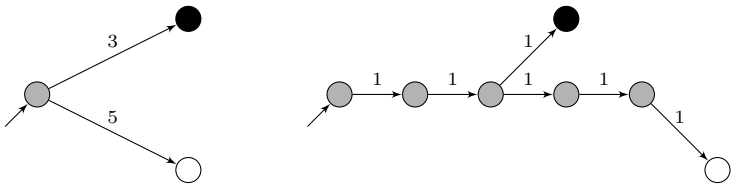


Figure 1: The continuous semantics for weighted automata

*instance, formula* $\mathbf{EF}\,(\mathbf{EF}\,\mathit{white} \wedge \neg\,\mathbf{EF}\,\mathit{black})$ *is true in the semi-continuous semantics on our example, while it does not hold under the discrete semantics.*

This modified semantics comes with a blow-up in the complexity:

**Theorem 20.** *The WCTL model-checking problem in the semi-continuous semantics is PSPACE-complete.*

*Proof.* The PSPACE algorithm is easily derived from the algorithm for TCTL model-checking on classical timed automata. We prove that this is optimal by reduction of the QBF problem:

| | |
|---|---|
| **Problem:** | **Quantified boolean formula** |
| **Input:** | A boolean formula $\phi(x_1, ..., x_{2n})$; |
| **Question:** | Does the formula $\exists x_1.\forall x_2 \ldots \exists x_{2n-1}\forall x_{2n}.\phi(x_1, ..., x_{2n})$ hold true? |

This problem is well-known to be PSPACE-complete. The reduction is achieved as follows: we consider the weighted automaton depicted on Figure 2. Notice that this automaton only depends on the number of variables in the QBF instance, and not on the boolean formula itself: this formula will be transformed into a WCTL formula to be checked on the automaton.

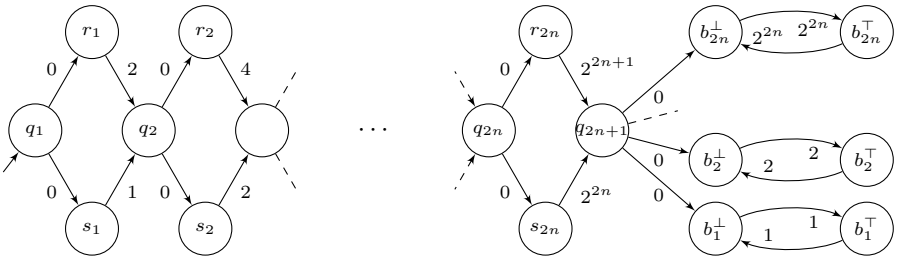


Figure 2: Reduction from QBF

In this automaton, a trajectory from $q_1$ to $q_{2n+1}$ can be seen as a valuation of the variables appearing in the QBF formula: setting $x_{2i-1}$ to true is encoded by visiting $r_{2i-1}$ (and not $s_{2i-1}$), and setting $x_{2i}$ to true is encoded by visiting $r_{2i}$ (and not $s_{2i}$).

Let $S_i$ be the set of configurations reachable from $q_0$ with weight exactly $2^i - 1 = \sum_{j=0}^{i-1} 2^j$. It is easily shown that

$$S_i = \{(q_i, 0)\} \cup \{(r_{i-1}, \alpha) \mid 1 \leq \alpha \leq 2^{i-1}\} \cup \{(s_{i-1}, \alpha) \mid 1 \leq \alpha \leq 2^{i-1} - 1\}.$$

Notice that $|S_i| = 2^i$. In fact, any configuraton in $S_i$ has exactly two successors in $S_{i+1}$: after reaching $q_i$, it can go *via* $r_i$ or *via* $s_i$, and spend the remaining credit on the transition to $q_{i+1}$. As a consequence, for each $s \in S_{2n+1}$, there exists exactly one possible trajectory reaching configuration $s$ in time $2^{2n+1} - 1$. Moreover, there exists only one path between $s$ and $q_{2n+1}$, and the length of this path, which lies between 0 and $2^{2n+1} - 1$, uniquely characterizes $s$. This allows us to associate with each $s$ in $S_{2n+1}$ a valuation of the variables $(x_i)_i$, letting $x_1 x_2 \dots x_{2n}$ be the binary notation of the distance between $s$ and $q_{2n+1}$. It is easily observed that the valuation $v_s$ defined by a state $s$ of $S_{2n+1}$ has the following property:

$$v_s(x_i) = \top \quad \Leftrightarrow \quad s \models \mathbf{EF}_{=2^{2n}-1}\, b_i^\top.$$

Replacing each occurrence of $x_i$ with $\mathbf{EF}_{=2^{2n}-1}\, b_i^\top$ in $\phi$, we get a formula that holds true in $s$ iff the valuation $v_s$ satisfies $\phi$. It remains to encode the quantification over the variables. This is achieved by replacing each quantification $\exists x_{2i+1}$ with $\mathbf{EF}_{=2^{2i}}$ and each quantification $\forall x_{2i}$ with $\mathbf{AF}_{=2^{2i-1}}$ in the original QBF instance. Since each state in $S_i$ has exactly two successors at distance $2^i$ in $S_{i+1}$, this quantification is sound and properly encodes the original problem. □

**Exercice 17** ⋆⋆⋆⋆ Show that $\mathsf{WCTL}_{\leq,\geq}$ (see Ex. 11) can be checked in PTIME on weighted automata under the semi-continuous semantics. *(Hint: define* $\mathsf{Sat}(q, \phi)$ *as the set of integer values* $i$ *s.t.* $(q, i) \models \phi$*, for all states* $q$ *and all subformulas* $\phi$*. Prove that this set can always be written as the union of a bounded number of intervals, which can be computed in polynomial time [LMS02].)*

## 1.5 Energy constraints

**Definition 21.** *Let* $\mathcal{A} = \langle S, S_0, \delta \rangle$ *be a weighted automaton,* $c \in \mathbb{R}$*,* $b \in \mathbb{R} \cup \{+\infty\}$*, and let* $\pi = ((s_i, w_i, s_i'))_{0 \leq i < n}$ *be a finite path in* $\mathcal{A}$*. The* accumulated weight with initial credit $c$ under weak upper bound $b$ *is* $p_{c\downarrow b}(\pi) = a_n$*, where* $(a_i)_{0 \leq i \leq n}$ *is defined inductively by* $a_0 = \min(c, b)$ *and* $a_{i+1} = \min(a_i + w_i, b)$*.*

These definitions of accumulated weight lead to the definition of the following problems:

**Definition 22.**

| | |
|---|---|
| ***Problem:*** | ***Infinite path with energy constraint*** |
| ***Input:*** | *A weighted automaton* $\mathcal{A} = \langle S, S_0, \delta \rangle$*, a state* $s_0$*, an initial credit* $c \in \mathbb{N}$*, a weak upper bound* $b \in \mathbb{N} \cup \{+\infty\}$*, and two bounds* $l \in \mathbb{N}$ *and* $u \in \mathbb{N} \cup \{+\infty\}$*;* |
| ***Question:*** | *Is there a maximal path* $\pi$ *in* $\mathcal{A}$ *from* $s_0$ *all of whose prefixes* $\pi'$ *satisfy* $l \leq p_{c\downarrow b}(\pi') \leq u$*?* |

*A* universal *version of the problem can be defined similarly, where the question is whether all paths satisfy the energy constraint.*

**Theorem 23.** *The infinite-path-with-energy-constraint problem (in its existential version) can be decided in* PTIME *if* $u$ *is* $+\infty$*. It is in* PSPACE *and* NP*-hard otherwise. The universal version of the problem is in* PTIME *in both cases.*

*Proof.* First assume that $u = +\infty$. For each trajectory $\gamma$, we write $\mathsf{MinCr}(\gamma)$ for the minimal credit that allows $\gamma$ to continuously satisfy the weight constraint (it is $+\infty$ when no iniital credit fulfills this requirement, in which case $\gamma$ is said to be *infeasible*).

In each state of the weighted automaton, we compute the value $\mathsf{MinCr}(s)$ of the minimal initial credit that allows to have an infinite trajectory. It will then suffice to check that this value is less than $c$.

We begin with an easy lemma:

**Lemma 24.** *Let* $u = +\infty$*. Let* $\gamma$ *be a maximal trajectory in* $\mathcal{A}$ *from* $s$*. Write* $\mathsf{MinCr}(\gamma)$ *for the least initial credit that allows* $\gamma$ *to always fulfill the lower bound constraint (or* $-\infty$ *if no such finite value exists). Then there exists an ultimately periodic trajectory* $\lambda$ *starting in* $s$ *such that* $\mathsf{MinCr}(\lambda) \leq \mathsf{MinCr}(\gamma)$*.*

*Proof.* First assume that there is a cycle $\pi$ in $\gamma$ with nonnegative weight, and write $\gamma = \gamma_1 \cdot \pi \cdot \gamma_2$. Then the ultimately periodic trajectory $\gamma_1 \cdot \pi^\omega$ is an infinite trajectory in $\mathcal{A}$ from $s$ and clearly enough, $\mathsf{MinCr}(\gamma_1 \cdot \pi^\omega) \leq \mathsf{MinCr}(\gamma)$.

Now, if there are no nonnegative cycle along $\gamma$, then we can write $\gamma$ as $\gamma_1 \cdot \gamma_2$ in such a way that $\gamma_2$ only contains states that appear infinitely many times along $\gamma$. □

**Lemma 25.** *Assume* $u = +\infty$*. For finite weighted automata,* $\mathsf{MinCr}(s_0)$ *is computable in polynomial time.*

*Proof.* We first assume that $b = +\infty$. We use a slightly modified version of the Bellman-Ford algorithm: the algorithm computes, from a given state $s_0$ with initial credit $c_0$, the maximal (without positive loops for the moment) remaining credit to reach each state of the automaton without dropping below 0. This is achieved as in the Bellman-Ford loop:

---

```
1: procedure INIT(s_0, c_0);
2: for all s ∈ S do
3:    C(s) := −∞
4: end for
5: C(s_0) := c_0; b :=FALSE;
6:
7: procedure ROUND(s_0);
8: for i := 1 to |S| − 1 do
9:    for each edge s −p→ s' in T do
10:      if C(s') ≤ C(s) + p and C(s) + p ≥ 0 then
11:         C(s') := C(s) + p
```

```
12:          if s' = s_0 then
13:            b :=TRUE
14:          end if
15:        end if
16:      end for
17:    end for
```

A second call to ROUND allows us to detect the nonnegative loops. That the maximal remaining credit is increased in some state in the second phase means that there is a positive feasible loop, and our algorithm returns positively. Otherwise, we repeat the same computation from each state, with the maxmial possible remaining credit after arriving from $s_0$ with initial credit $c_0$. We then detect whether it is possible to come back to this state with at least the same credit, in which case we return positively as we have found a feasible lasso.

The case where $b$ is finite is handled similarly, except that the remaining credit is bounded above by $b$.

We now turn to the universal problems (still with $u = +\infty$). When $b = +\infty$, the dual problem (*i.e.*, exhibiting a finite path with negative accumulated cost) can be solved in deterministic polynomial time by applying the Bellman-Ford shortest path algorithm. Hence the polynomial time algorithm for the lower-bound problem.

Similarly, when $b$ is finite, the dual problem is solved in deterministic polynomial time by adapting the Bellman-Ford algorithm as follows: whenever the weight of some node should be updated to a larger value than is the given weak upper bound $b$, we update it only to $b$. □

The second part of the proof, where $u$ is finite, is easier: NP-hardness is achieved by encoding the SUBSET-SUM problem:

| | |
|---|---|
| **Problem:** | **Subset sum** |
| **Input:** | A finite sequence of integers $(a_i)_{1 \leq i \leq n}$, an integer $b$; |
| **Question:** | Is there a subset $I \subseteq [1, n]$ s.t. $\sum_{i \in I} a_i = b$? |

Given an instance of this problem, we build the weighted automaton $\mathcal{A}$ made of $n + 2$ states $q_1$ to $q_{n+2}$. For each $i \leq n$, there are two transitions from $q_i$ to $q_{i+1}$, one with weight 0 and one with weight $a_i$. Finally, there is a transition $(q_{n+1}, -b, q_{n+2})$, and a self-loop $(q_{n+2}, 0, q_{n+2})$. Clearly enough, there is an infinite path in this structure if, and only if, the original instance of the subset-sum problem is positive.

Regarding the membership in PSPACE: clearly enough, if there is an infinite path with cost between $l$ and $u$, then at least one state $q$ appears infinitely often, and this path can be written as $\pi_0 \cdot \pi_1 \cdot \pi_2$ where $\pi_1$ is a non-trivial path starting and ending in $q$ and having global total weight 0, and where the length of $\pi_0$ and $\pi_1$ is at most $(u - l) \cdot |S|$. Then $\pi_0 \cdot \pi_1^\omega$ is a possible witness of an infinite path whose cost remains between $l$ and $u$. Our algorithm non-deterministically guesses $\pi_0$ and $\pi_1$, checking on-the-fly that the cost constraints

are fulfilled. It suffices to store the current state, the initial state of $\pi_1$, and the accumulated cost along the path guessed thus far. This information can be stored in polynomial space.

Finally, we tackle the case of the universal version of the problem when $u$ is finite. In this case, the dual problem consists in finding a trajectory that violates either the upper- or the lower-bound constraint. This can be achieved with two runs of the Bellman-Ford algorithm, one for testing if the lower bound can be violated, and another one for the upper bound. □

**Exercice 18** ⋆⋆ Write the details of the above proof.

# 2 Weighted games

## 2.1 Definitions

**Definition 26.** *A* weighted game automaton *is a tuple* $\mathcal{G} = \langle S, S_0, \delta, \mathbb{A}, \mathbb{M}, \mathsf{Ch}, \mathsf{Edg} \rangle$ *where*

- $\langle S, S_0, \delta \rangle$ *is a weighted automaton;*
- $\mathbb{A} = \{A_1, ..., A_p\}$ *is a finite set of agents;*
- $\mathbb{M}$ *is a set of moves;*
- $\mathsf{Ch} \colon S \times \mathbb{A} \to 2^{\mathbb{M}} \setminus \varnothing$ *lists the set of moves allowed for each agent in each state;*
- $\mathsf{Edg} \colon S \times \mathbb{M}^{|\mathbb{A}|} \to \delta$ *is a transition table. It is such that, for any state* $s$ *and any set of moves* $(m_i)_{A_i \in \mathbb{A}}$*, the value* $\mathsf{Edg}(s, (m_i)_{A_i \in \mathbb{A}})$*, if defined, is a transition from* $s$*.*

*A game automaton is* turn-based *if there is a mapping* $p \colon S \to \mathbb{A}$ *s.t. for each state* $s$*, the set* $\mathsf{Ch}(s, a)$ *is a singleton when* $a \neq p(s)$*. In that case, for each state* $s$*, there is a correspondance between the set of outgoing transitions of* $s$ *and the set of choices of* $p(s)$*. We thus use the syntax* $\langle (S_c)_{A_c \in \mathbb{A}}, S_0, \delta \rangle$*, with* $p(s) = c$ *for each* $s \in A_c$*.*

A path in a weighted game automaton is a path in the underlying weighted automaton.

**Definition 27.** *Let* $\mathcal{G}$ *be a weighted game automaton, and* $A \in \mathbb{A}$ *be one of its agents. A* strategy *for* $A$ *is a partial mapping* $\sigma_A \colon \mathsf{Path}_f(\mathcal{G}) \to \mathbb{M}$ *such that for any finite path* $\pi$*, we have* $\sigma_A(\pi) \in \mathsf{Ch}(last(\pi), A)$*.*

*A strategy for a coalition* $C \subseteq \mathbb{A}$ *of players is a set of strategies* $(\sigma_c)_{A_c \in C}$*, one for each player.*

*A strategy is* memoryless *(or* positional*) if it only depends on the last state of the history.*

**Definition 28.** *Let $\mathcal{G}$ be a weighted game automaton, $C \subseteq \mathbb{A}$ be a coalition, $(\sigma_c)_{A_c \in C}$ be a strategy for $C$, and $\pi$ be a finite path in $\mathcal{G}$. The set of possible transitions after $\pi$ according to $(\sigma_c)_{A_c \in C}$, denoted with $\mathsf{Next}(\pi, (\sigma_c)_{A_c \in C})$, is defined as*

$$\mathsf{Next}(\pi, (\sigma_c)_{A_c \in C}) = \{d \in \delta \mid \exists (m_i)_{A_i \in \mathbb{A}} \in \mathbb{M}^{\mathbb{A}}.\ d = \mathsf{Edg}(q, (m_i)_{A_i \in \mathbb{A}}) \text{ and } m_c = \sigma_c(\pi) \text{ for each } A_c \in C \text{ s.t. } \sigma_c(\pi) \text{ is defined}\}.$$

*In the sequel,* $\mathsf{Next}$ *will mainly be used for designating the set of possible successors of a given state under a partial choice of the agents: we abusively write* $\mathsf{Next}(q, (m_c)_{A_c \in C})$ *for*

$$\mathsf{Next}(q, (\sigma_c)_{A_c \in C}) \text{ with } \sigma_c(q) = m_c \text{ for each } A_c \in C.$$

*A path $\pi = ((s_i, w_i, s'_i))_{0 \leq i < |\pi|}$ is* compatible *with $(\sigma_c)_{A_c \in C}$ after position $j$ if, for each $i$ s.t. $j \leq i < |\pi|$, we have $(s_i, w_i, s'_i) \in \mathsf{Next}(\pi_{<i}, (\sigma_c)_{A_c \in C})$.*

*Given a finite path $\pi$, we write $\mathsf{Out}(\pi, (\sigma_c)_{A_c \in C})$ for the set of maximal paths having $\pi$ as a prefix and compatible with $(\sigma_c)_{A_c \in C}$ after position $|\pi|$.*

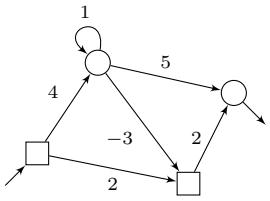


Figure 3: Example of a weighted game automaton

**Example.** *Figure 3 displays an example of a finite, turn-based*[1] *two-player weighted game automaton. The optimal weight player $A$ (controlling circle states) can ensure to reach the exit state is 4 when the discounter factor is 1 as well as when the discounted factor is 0. It is 3 when the discounted factor is 0.5.*

**Definition 29.** *A* winning objective $\Omega$ *is a set of finite or infinite paths, with the requirement that if a finite path $\pi$ belongs to $\Omega$, then any prolongation $\pi \cdot \pi'$ of this path is also in $\Omega$.*

**Example.** *A* reachability objective *is the set of all finite and infinite paths containing a given state.*

**Definition 30.** *A strategy $\sigma_A$ for coalition $A$ is winning for an objective $\Omega$ from a finite history $\pi$ if $\mathsf{Out}(\pi, \sigma_A) \subseteq \Omega$.*

**Definition 31.** *A game is* determined *for an objective $\Omega$ if, for any coalition $A$, either coalition $A$ has a winning coalition for the objective $\Omega$, or coalition $\overline{A}$ has a strategy for the objective $\overline{\Omega}$.*

[1] As is usual for turn-based games, the shape of the node encodes the value of the function $p$ indicating which player controls a given state.

**Example.** *For* reasonnable *objectives (see [Mar75]), turn-based games are determined. General concurrent games are not: for instance, the game depicted on Fig. 4 is not determined if the objective is to reach the black state.*

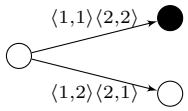


Figure 4: A non-determined game

## 2.2 Optimal mean payoff

**Definition 32.** *The optimal mean-payoff strategy problem is defined as follows:*

| | |
|---|---|
| ***Problem:*** | ***Optimal mean-payoff game*** |
| ***Input:*** | *A weighted game automaton $\mathcal{G}$, a state $s_0$, a coalition $C$, a rational $n$;* |
| ***Question:*** | *Is there a strategy $(\sigma_c)_{A_c \in C}$ all of whose outcomes $\pi$ from $s_0$ satisfy $m(\pi) \leq n$?* |

We only deal with the turn-based case in this section.

**Theorem 33.** *The optimal-mean-payoff-game problem for turn-based games is in NP ∩ coNP, and PTIME-hard. It can be solved in pseudo-polynomial time*[2]*.*

*Proof.* Hardness in PTIME is easily obtained by encoding the circuit-value problem [GHR95], adding a self-loop of weight $n-1$ on the positive leave and $n+1$ on the negative leave.

In order to prove membership in NP and coNP, we prove that mean-payoff *turn-based* games are *memoryless determined*, meaning that either coalition $C$ has a winning strategy, or coalition $\overline{C}$ has one (this is *determinacy*), and, in both cases, the strategy can be chosen memoryless.

Clearly enough, the turn-based optimal-mean-payoff-game problem can be transformed into an equivalent two-player game, where one of the player controls all the agents in $C$ and the second player controls the other agents. Thus, let $\mathcal{G} = \langle S_C, S_{\overline{C}}, S_0, \delta \rangle$ be a two-player turn-based game. In order to prove memoryless determincay of this game, we consider another game played on the same board: the game is played according to the same rules, but stops as soon a some state is visited twice. The aim of Player $C$ is then to minimize the mean payoff *over the cycle* that has been formed during the play. We call *finite*

[2] *Pseudo-polynomial* means "polynomial if the constants were encoded in unary". Constants are generally assumed to be encoded in binary, which entails an exponential blowup in the complexity.

*mean-payoff game* this game:

| | |
|---|---|
| **Problem:** | **Optimal finite mean-payoff game** |
| **Input:** | A weighted game automaton $\mathcal{G}$, a state $s_0$, a coalition $C$, a rational $n$; |
| **Question:** | Is there a strategy $(\sigma_c)_{A_c \in C}$ all of whose finite outcomes $\pi$ from $s_0$, ending as soon as a cycle $\pi'$ is formed, satisfy $m(\pi') \leq n$? |

**Lemma 34.** *Finite mean-payoff games are memoryless determined.*

*Proof.* We begin with proving that those games are determined: either player $C$ has a strategy for making the value of the game less than or equal to $n$, or Player $\overline{C}$ has a strategy for making the value larger than $n$. Indeed, consider the finite tree of all the possible outcomes of the game, replace each leaf of this tree with the mean-payoff of the corresponding cycle (*i.e.*, with the value of the corresponding play), and think of Player $C$'s states as "minimum" gates and Player $\overline{C}$'s states as "maximum" gates. It is easily seen that, in each node of this tree, the value $\nu$ of this node is such that Player $C$ has a strategy for making the value of the play less than $\nu$ while Player $\overline{C}$ can make it larger than or equal to $\nu$. In the end, there is a value $\nu_0$ s.t. Player $C$ can ensure that the value of the game will be less than or equal to $\nu$, and Player $\overline{C}$ can ensure that the value will be larger than or equal to $\nu$. If $\nu \leq n$, then Player $C$ has a winning strategy, otherwise Player $\overline{C}$ has one.

We now prove that in both case, considering memoryless strategies is enough: to this aim, we define the set of winning states for each player, which we write $W_C$ and $W_{\overline{C}}$. We begin with two lemmas:

**Lemma 35.** *If Player $C$ (resp. $\overline{C}$) has a positional strategy winning from some state $u$, and there is a state $v \in S_C$ (resp. $v \in S_{\overline{C}}$) s.t. $(v, u) \in \delta$, then $v \in W_C$ (resp. $v \in W_{\overline{C}}$).*

*Proof.* Consider the automaton $\mathcal{G}_\sigma$ obtained from $\mathcal{G}$ by applying strategy $\sigma$: in each state of $S_C$ (resp. $S_{\overline{C}}$), we only keep the transition indicated by $\sigma$. Since $v \in S_C$ (resp. $v \in S_{\overline{C}}$), $\sigma(v)$ is defined.

If $\sigma(v) = u$, then the set of reachable states in $\mathcal{G}_\sigma$ from $u$ and $v$ are the same, except possibly $v$. In any case, the set of reachable loops are the same, so that if $\sigma$ is winning from $u$, it is also winnign from $v$.

Now, if $\sigma(v) \neq u$, we again consider two cases:

- if $v$ is reachable from $u$ in $\mathcal{G}_\sigma$, then all loops reachable from $v$ are reachable from $u$, so that all of them must be winning.
- otherwise, if $v$ is not reachable from $u$, we define $\sigma'$ from $\sigma$ by letting $\sigma'(s) = \sigma(s)$ when $s \neq v$, and $\sigma'(v) = u$. Clearly enough, $v$ is still not reachable from $u$ in $\mathcal{G}_{\sigma'}$, so that $\sigma'$ is still winning from $u$. Also, the set of reachable states, and loops, from $u$ and $v$ in $\mathcal{G}_{\sigma'}$ are the same, and $\sigma'$ is also winning from $v$. □

**Lemma 36.** *Assume that $\sigma_C$ and $\sigma_{\overline{C}}$ are two positional strategies such that $\sigma_C$ (resp. $\sigma_{\overline{C}}$) is winning for Player $C$ (resp. Player $\overline{C}$) from any state in $W_C$ (resp. $W_{\overline{C}}$). Then any play from $W_C$ (resp. $W_{\overline{C}}$) following strategy $\sigma_C$ (resp. $\sigma_{\overline{C}}$) only visits states in $W_C$ (resp. $W_{\overline{C}}$).*

*Proof.* Since both results are symmetric, we only prove the result for $C$: consider the game $\mathcal{G}_{\sigma_C}$: this automaton can be seen as a game where all states belong to Player $\overline{C}$. Any path in $\mathcal{G}_{\sigma_C}$ starting from a state in $W_C$ is winning for Player $C$. We have to prove that $\mathcal{G}_{\sigma_C}$ contains no transition $(v, u)$ with $v \in W_C$ and $u \in W_{\overline{C}}$. This immediately follows from Lemma 35 applied to the game $\mathcal{G}_{\sigma_C}$. □

We now prove that memoryless strategies are sufficient:

**Lemma 37.** *Let $\mathcal{G} = \langle S_C, S_{\overline{C}}, S_0, \delta \rangle$ be a weighted game automaton. There exists a positional strategy $\sigma_C$ (resp. $\sigma_{\overline{C}}$) for Player $C$ (resp. Player $\overline{C}$) that is winning from any state in $W_C$ (resp. $W_{\overline{C}}$).*

*Proof.* The proof is by induction on the difference $|\delta| - |S|$ in the game (which is the number of possible "choices" in the game). This value is nonnegative since we require that each state has at least one outgoing transition. The base case is when $|\delta| = |S|$. This case is straightforward since any strategy is positional: since Player $C$ has a winning strategy from any state in $W_C$, then he has a memoryless one.

Assume the result holds in any game having $|\delta| - |S|$ less than or equal to some integer $i \geq 0$, and consider a game in which $|\delta| - |S| = i+1$. We let $S^*_C \subseteq S_C$ and $S^*_{\overline{C}} \subseteq S_{\overline{C}}$ be the set of states having at least two outgoing transitions. We consider three cases:

- If $S^*_C \cap W_C \neq \varnothing$: this indicates that there is a state $s$ winning for Player $C$ in which this player has at least two possible choices. Consider a strategy $\sigma_{C,s}$ winnign for Player $C$ from $s$, and let $e$ be its first move $\sigma_{C,s}(s)$. Consider the game $\mathcal{G}'$ obtained from $\mathcal{G}$ by removing all transitions out of $s$, except $e$. From the induction hypothesis, both players have memoryless winning strategies $\sigma'_C$ and $\sigma'_{\overline{C}}$ from their respective winning states $W'_C$ and $W'_{\overline{C}}$ in $\mathcal{G}'$. These strategies can be applied in the original game, where they are still memoryless. We prove that $W_C = W'_C$ and $W_{\overline{C}} = W'_{\overline{C}}$, and that the memoryless strategies are winning in $\mathcal{G}$.

  Pick a state $q \in S_C \cup S_{\overline{C}}$, assuming that it is winning for Player $C$ in $\mathcal{G}'$ by playing according to $\sigma'_C$. Any play from $v$ following $\sigma'_C$ can be played in $\mathcal{G}'$. Since it is winning in $\mathcal{G}'$ for $C$, it must be winning in $\mathcal{G}$ for $C$. Hence $W'_C \subseteq W_C$.

  Now assume that $q \in W'_{\overline{C}}$. In that case, we must have $q \neq s$. Indeed, $s$ is a winning state for Player $C$ in $\mathcal{G}'$: it is winning in $\mathcal{G}$ after playing transition $e$, and playing the same transition in $\mathcal{G}'$ from $s$ yields the very same situation (the play stops if it goes back to $s$). Then from Lemma 36, if Player $\overline{C}$ plays according to the memoryless strategy $\sigma'_{\overline{C}}$ from state $q$ in $\mathcal{G}$, the play cannot reach $x$. But then, any play not reaching $x$ in $\mathcal{G}$

is also possible in $\mathcal{G}$', where it must be winning for Player $\overline{C}$. Hence $W'_{\overline{C}} \subseteq W_{\overline{C}}$.

This proves that the winning sets are equal, and that the memoryless strategies $\sigma'_C$ and $\sigma'_{\overline{C}}$ are winning.

- If $S^*_{\overline{C}} \cap W_{\overline{C}} \neq \varnothing$, the argument is symmetric.
- If $S^*_C \cap W_C = S^*_{\overline{C}} \cap W_{\overline{C}} = \varnothing$, then we can assume w.l.o.g. that Player $C$ controls all the states in $W_{\overline{C}}$ and Player $\overline{C}$ controls all the states in $W_C$.

  Assume, for a contradiction, that there is an edge $e$ from $u \in W_C$ to $v \in W_{\overline{C}}$. Player $C$ has a winning strategy from $u$, and since he does not control that state, the strategy must be winning if the play goes from $u$ to $v$ *via* $e$. If, from $v$, Player $C$ does not have a strategy to go back to $u$, then he should lose because the play stops as soon as a cycle has been formed. Thus from $v$, Player $C$ has a strategy to go to $u$, forming a winning cycle for himself. But now, state $v$ is winning for Player $\overline{C}$: in particular, if Player $C$ forces the play to reach $u$, Player $\overline{C}$ must have a way to avoid going back to $v$, since this would form a cycle winning for Player $C$. Thus there must exist another edge $e'$ out of $u$.

  Now, consider the game $\mathcal{G}'$ where edge $e$ is removed. Then Player $\overline{C}$ has less choices in $\mathcal{G}'$ than in $\mathcal{G}$, so that it must be the case that $W'_{\overline{C}} \subseteq W_{\overline{C}}$. From the induction hypothesis, there must exist memoryless strategies $\sigma'_C$ and $\sigma'_{\overline{C}}$ winning from $W'_C$ and $W'_{\overline{C}}$ for Players $C$ and $\overline{C}$ resp. As in $\mathcal{G}$, all the states of $\mathcal{G}'$ in $W'_{\overline{C}}$ can be assumed to belong to Player $C$. According to Lemma 35, no edge can leave $W'_{\overline{C}}$. This entails that $v$ belongs to $W'_C$: if this were not the case, then Player $\overline{C}$ would win from $u$ by moving to $v$ *via* $e$ and would then not have left $W'_{\overline{C}}$, thus forming a loop that would have been winning for Player $\overline{C}$.

  This means that state $v$ is losing for Player $C$ when transition $e$ is present, and it is winning for Player $C$ when $e$ has been removed. This means that, in $\mathcal{G}'$, Player $C$ must be able to force the play from $v$ to $u$ in order to win, since any other play would have been possible in $\mathcal{G}$, and is thus losing for Player $C$. In $\mathcal{G}$, if Player $C$ forces the play from $v$ to $u$, Player $\overline{C}$ can win by playing an edge other than $e$. The same play is possible in $\mathcal{G}'$, which means that Player $C$ cannot win from $v$ in $\mathcal{G}'$. This is a contradiction, meaning that there is no edge from $\mathcal{G}$ to $\mathcal{G}'$. Symmetrically, there is also no edge from $\mathcal{G}'$ to $\mathcal{G}$.

  As a consequence, any strategy is winning, since no player has several possible moves in his own winning set. □

It remains to establish a correspondance between those finite mean-payoff games and the infinite ones. This is the role of the following lemma:

**Lemma 38.** *A strategy $\sigma_C$ for Player $C$ (resp. $\sigma_{\overline{C}}$ for Player $\overline{C}$) in a finite mean-payoff game can be modified into a strategy $\sigma'_C$ (resp. $\sigma'_{\overline{C}}$) in the corresponding infinite mean-payoff game so that both strategies ensure the same value*

*(in the worst case). Moreover, if $\sigma_C$ (resp. $\sigma_{\overline{C}}$) is memoryless, then so is $\sigma'_C$ (resp. $\sigma'_{\overline{C}}$).*

*Proof.* We prove the claim for Player $C$. The proof is symmetric for the other player.

Given a finite trajectory $\pi = ((s_i, w_i, s'_i))_{0 \leq i < |\pi|}$, we define the trajectory $\pi'$ obtained from $\pi$ by repeatedly removing the *earliest* cycle. For instance, if the sequence of states is $s_0 \to s_1 \to s_2 \to s_3 \to s_1 \to s_3 \to s_2$, we get $s_0 \to s_1 \to s_3 \to s_2$. We then define $\sigma'_C(\pi) = \sigma_C(\pi')$. Notice that the last states of $\pi$ and $\pi'$ are always the same, so that if $\sigma_C$ is memoryless, then $\sigma'_C$ is the same as $\sigma_C$. We now prove that if $\sigma_C$ ensures a mean-payoff $\nu$ for Player $C$ in the finite game, then $\sigma'_C$ ensures a mean-payoff $\nu$ for Player $C$ in the infinite game.

Indeed, let $\pi$ be an infinite play in the infinite game corresponding to Player $C$ applying strategy $\sigma'_C$. Each finite prefix of $\pi$ can be split into a sequence of simple cycles $(\pi_j)_{1 \leq j \leq m}$ of mean-payoff less than $\nu$, plus a finite acyclic trajectory $\pi_0$, possibly of mean-payoff larger than $\nu$. Indeed, it suffices to detect the first cycle formed along $\pi$, which is obtained by following strategy $\sigma_C$ and thus has mean-payoff less than $\nu$. by recursively applying the same argument to the trajectory obtained by removing this cycle, we get the required set of trajectories.

We then have

$$\frac{1}{n}\sum_{i=0}^{n-1} w_i = \frac{1}{n} \cdot \left( \sum_j \sum_{w_i \in \pi_j} w_i \right) \leq \frac{1}{n} \left( \sum_j \nu \cdot |\pi_j| - \nu \cdot |\pi|_0 + \sum_{w_i \in \pi_0} w_i \right)$$
$$\leq \nu + \frac{1}{n} \cdot \left( \sum_{w_i \in \pi_0} w_i - \nu \cdot |\pi|_0 \right)$$

Now, $\pi_0$ is acyclic, so that the term in parentheses is finite. In the end, the mean-payoff of $\pi$ is less than $\nu$. □ □

This immediately provides us with an NP algorithm: if Player $C$ should be winning, it suffices to guess a memoryless strategy, prune the game automaton according to this strategy, and compute the worst achievable mean-payoff in the resulting weighted automaton, which can be achieved in PTIME following the lines of the proof of Theorem 11. Membership in coNP is proved similarly, because if Player $C$ has no winning strategy, then Player $\overline{C}$ has a memoryless one for the opposite objective.

We now explain how we compute the mean-payoff in pseudo-polynomial time: to this aim, we again change the rules of the game: the players will now play for a total of $k$ steps, with Player $C$ trying to minimize the total weight of the resulting trajectory. Write $\nu_k(s)$ for the best weight Player $C$ can achieve in $k$ steps from state $s$.

**Lemma 39.** *The value $\nu_k(s)$ can be computed in time $O(k \cdot |\delta|)$.*

*Proof.* Clearly enough, we have

$$\nu_k(s) = \begin{cases} \max_{(s,w,s')\in\delta} w + \nu_{k-1}(s') & \text{if } s \in S_C \\ \min_{(s,w,s')\in\delta} w + \nu_{k-1}(s') & \text{if } s \in S_{\overline{C}} \end{cases}$$

with initially $\nu_0(s) = 0$. □

**Lemma 40.** *Write $W$ for the maximal constant appearing in $\mathcal{G}$. For every $s \in S$ and any $k \in \mathbb{N}$, writing $\nu(s)$ for the best mean-payoff that Player $C$ can ensure, we have*

$$k \cdot \nu(s) - 2|S|W \leq \nu_k(s) \leq k \cdot \nu(s) + 2|S|W$$

*Proof.* Let $\sigma_C$ be an positional strategy for Player $C$ that is optimal when starting from $s$ (thus achieving mean payoff $\nu(s)$). Consider a $k$-step outcome played according to $\sigma_C$. Since $\sigma_C$ is memoryless, each cycle must have mean weight less than or equal to $\nu$, so that a cycle of length $l$ has weight at most $l \cdot \nu$. We easily get that

$$\nu_k(s) \leq (k - |S|) \cdot \nu + |S| \cdot W \leq k \cdot \nu + 2|S|W.$$

Symmetrically, given an optimal memoryless strategy for Player $\overline{C}$, and considering a $k$-step outcome, we get

$$k \cdot \nu - 2|S|W \leq (k - |S|) \cdot \nu - |S|W \leq \nu_k(s).$$

□

From Lemma 38, the value of $\nu$ is a rational with denominator at most $|S|$. Letting $k = 4|S|^3 \cdot W$, we have

$$\left| \nu(s) - \frac{\nu_k(s)}{k} \right| \leq \frac{1}{2n^2}$$

Now, since the distance between two rationals with denominator less than $|S|$ is less than $1/(|S|^2)$, there is a unique possible candidate at distance less than $1/(2|S|^2)$ from $\nu_k(s)/k$. □

**Exercice 19** ⋆ The pseudo-polynomial algorithm above computes the exact value of the mean-payoff game. Prove that the *decision* problem can be solved in time $O(|S|^2 \cdot |\delta| \cdot W)$.

**Open Problem 1** Can the optimal-mean-payoff-game problem (for turn-based games) be solved in polynomial time?

**Exercice 20** ⋆⋆ Show that for general concurrent games, the optimal-mean-payoff-game problem is in NP. Is it in coNP?

## 2.3 Shortest and longest paths

**Definition 41.**

| | |
|---|---|
| ***Problem:*** | ***Optimal reachability strategy*** |
| ***Input:*** | *A finite weighted game automaton $\mathcal{G}$, two states $s$ and $t$, a coalition $C$, and an integer $n$;* |
| ***Question:*** | *Does there exist a strategy $(\sigma_c)_{A_c \in C}$ s.t. any outcome of $(\sigma_c)_{A_c \in C}$ from $s$ reaches $t$ with total cost less than $n$?* |

**Exercice 21** ⋆ Show that reachability games (without weight) can be solved are PTIME-complete.

**Exercice 22** ⋆⋆ Show that reachability objectives can be achieved with memoryless strategies. *(Hint: assuming that there is a winning strategy $\sigma$, consider the resulting execution tree, and define a fuction $\nu$ assigning an integer to each node:*

- *leaves (corresponding to the target state) are assigned the value* 0,
- *the other nodes are assigned $\nu(n) = 1 + \max_{n' \in succ(n)} \nu(n')$.*

*Define the strategy $\sigma'$ as follows: $\sigma'(q)$ is the values of $\sigma$ at one of the $q$-node having least value for $\nu$.)*

**Exercice 23** ⋆⋆ Show that the optimal-mean-payoff-game problem can be reduced to the optimal-reachability-strategy problem.

**Exercice 24** ⋆ Prove that the optimal-reachability-strategy problem is PTIME-complete for *nonnegative* weigthed games.

**Exercice 25** ⋆⋆ What would be the equivalent game problem to the shortest *acyclic* path problem? Show that this problem is NP-complete.

**Open Problem 2** Is the optimal-reachability-strategy problem in PTIME in the general case?

## 2.4 Weighted temporal logics

Let AP be a finite set of atomic propositions.

**Definition 42.** *The* full weighted alternating-time temporal logic *(WATL$^*$) is the logic whose syntax is defined by the following grammar:*

$$\begin{aligned} \mathsf{WATL}^* \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \langle\!\langle C \rangle\!\rangle \phi_p \\ \phi_p &::= \phi_s \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \, \mathbf{U}_I \, \phi_p \end{aligned}$$

*where $p$ ranges over AP, $C$ ranges over the set of coalitions, and $I$ ranges over the set of intervals.*

*We consider the classical fragment* WATL *(*alternating-time temporal logic*) where* path formulas $\phi_p$ *are restricted to $\phi_s \, \mathbf{U}_I \, \phi_s$ and negation thereof;*

**Definition 43.** *The semantics of WATL$^*$ extends the semantics of WCTL$^*$ with the following rule:*

$$\langle \mathcal{G}, \ell \rangle, \pi, k \models \langle\!\langle C \rangle\!\rangle \phi_p \quad \iff \quad \exists (\sigma_c)_{A_c \in C}.\ \forall \pi' \in \mathit{Out}(\pi_{<k}, (\sigma_c)_{A_c \in C}).\ \langle \mathcal{G}, \ell \rangle, \pi', 0 \models \phi_p$$

**Exercice 26** ⋆ What does $\neg \langle\!\langle C \rangle\!\rangle \neg \phi$ mean? How can it be expressed in turn-based games (provided that $\phi$ defines *reasonnable* objectives, see Example following Def. 31)?

**Definition 44.** *The model-checking problem is defined as follows:*

| | |
|---|---|
| ***Problem:*** | **WATL** ***model checking*** |
| ***Input:*** | *A WATL formula $\phi_s$, a labelled positive weighted game automaton $\langle \mathcal{G}, \ell \rangle$, a state $s$;* |
| ***Question:*** | *Does $\langle \mathcal{G}, \ell \rangle, s \models \phi_s$?* |

**Theorem 45.** *The WATL model-checking problem is EXPTIME-complete.*

*Proof.* As for the WCTL model-checking problem, our procedure consists in labelling states with the subformulas they satisfy. For a formula of the form $\langle\!\langle C \rangle\!\rangle \phi_1 \mathbf{U}_{[a,b]} \phi_2$, this is achieved by computing a big weighted game $\mathcal{G}'$ containing $b+1$ copies of $\mathcal{G}$, and in which each state explicitely contains the cost since the beginning. We also label the states having cost between $a$ and $b$ with a fresh atomic proposition ✓. Then

$$\mathcal{G}, q \models \langle\!\langle C \rangle\!\rangle \phi_1 \mathbf{U}_{[a,b]} \phi_2 \quad \Leftrightarrow \quad \mathcal{G}', (q, 0) \models \langle\!\langle C \rangle\!\rangle \phi_1 \mathbf{U} (\phi_2 \wedge ✓).$$

It remains to apply the classical ATL model-checking procedure, which runs in polynomial time, on this exponential structure. Since this has to be done a polynomial number of times (once for each state and each subformula), the whole procedure runs in exponential time.

We now prove that the problem is EXPTIME-complete. This is achieved by encoding the countdown-game problem. Such a game is played on a positive weighted game automaton $\mathcal{A}$: from state $s$ and with credit $c > 0$, one turn of the game proceeds as follows:

- Player 1 chooses an integer $d \leq c$ for which there exists a state $s'$ s.t. $(s, d, s') \in \delta$;
- Player 2 chooses one of the states $s'$ s.t. $(s, d, s') \in \delta$;
- the game continues from $s'$ with credit $c - d$.

An instance of the countdown game is a positive weighted automaton, an initial state $s$ and an initial credit $c > 0$. Each play is finite, because the credit

strictly decreases at each step. Player 1 wins if the play reaches a configuration where $c = 0$. The decision problem is as follows:

| | |
|---|---|
| **Problem:** | **countdown game** |
| **Input:** | a positive weighted automaton $\mathcal{A}$, an initial state $s$ and an initial credit $c$; |
| **Question:** | is there a winning strategy for Player 1 in the countdown game $\mathcal{A}$ from initial configuration $(s, c)$? |

It is rather clear that the countdown-game problem can be solved in exponential time (and in pseudo-polynomial time): a countdown game can be transformed into an exponential turn-based game, where each state represents a possible configuration of the countdown game. The full proof is left as Exercice 27. Hardness in EXPTIME is proved by encoding an EXPTIME Turing machine [JLS07].

Finally, it remains to prove that we can encode the countdown-game problem as a WATL model-checking problem. Again, this is rather straightforward and left to the reader as Exercice 28. □

**Exercice 27** ⋆ Write the full proof that the countdown-game problem can be solved in pseudo-polynomial time.

**Exercice 28** ⋆ Write the complete reduction of countdown games to WATL model-checking. Explain how the reduction can be adapted to turn-based games.

**Exercice 29** ⋆⋆⋆ Prove that $\mathsf{WATL}_{\leq,\geq}$ model-checking is PTIME complete.

**Exercice 30** ⋆⋆ The definition of the WATL model-checking requires strictly positive weights. What is the problem with transitions of weight 0?

## 2.5 Energy constraints

**Definition 46.**

| | |
|---|---|
| ***Problem:*** | ***Winning strategy with energy constraint*** |
| ***Input:*** | *A weighted game automaton* $\mathcal{G} = \langle S, S_0, \delta, \mathbb{A}, M, \mathbb{M}, \mathsf{Edg}\rangle$*, a state* $s_0$*, a coalition* $C$*, an initial credit* $c \in \mathbb{N}$*, a weak upper bound* $b \in \mathbb{N} \cup \{+\infty\}$*, and two bounds* $l \in \mathbb{Z} \cup \{-\infty\}$ *and* $u \in \mathbb{Z} \cup \{+\infty\}$*;* |
| ***Question:*** | *Is there a strategy* $(\sigma_c)_{A_c \in C}$ *such that all the outcomes of* $(\sigma_c)_{A_c \in C}$ *from* $s_0$ *have the property that all of their prefixes* $\pi'$ *satisfy* $l \leq p_{c\downarrow b}(\pi') \leq u$*?* |

**Theorem 47.** *When* $u = +\infty$*, the winning-strategy-with-energy-constraint problem is in* NP*, and is* PTIME*-hard. It is* EXPTIME*-complete when* $u$ *is finite.*

*Proof.* We begin with the case when $u = +\infty$. Hardness can be obtained *e.g.* by encoding the (monotone) circuit-value problem: the game is played on the circuit, where or-nodes are controlled by Player 1 and and-nodes are controlled

by Player 2. The circuit is positive if, and only if, Player 1 has a strategy for reaching state 1. It now suffices to assign weight 0 to each edge in the circuit, and to add self-loops on state 0 and 1, with respective weights $-1$ and 0, in order to have a reduction to our problem.

In order to prove membership in NP, we prove that games with lower-bound constraint are memoryless determined: either coalition $C$ has a memoryless strategy all of whose outcomes satisfy the cost constraint, or the opponent coalition $\overline{C}$ has a memoryless strategy to eventually have the cost go below $l$.

Assume that coalition $C$ has a strategy $\sigma$ for maintaining the cost above $l$. For each state $s$ reachable under this strategy, let $C_s$ be the set of integers $c'$ such that there is an outcome of $\sigma$ from $s_0$ with initial credit $c$ reaching $s$ with credit $c'$. As $\sigma$ is winning, it must be the case that $c_s = \min\{c'_s \mid c'_s \in C_s\} \geq l$. For each state $s$ reachable from $s_0$ under $\sigma$, we pick an outcome $\rho_s$ arriving in $s$ with credit $c_s$, and define $\sigma'(s) = \sigma(\rho_s)$. For states $s$ not reachable under $\sigma$, the value of $\sigma'(s)$ is irrelevant. The strategy $\sigma'$ is memoryless. We prove that it satisfies the cost constraint, by proving that the accumulated cost of any outcome when visiting a state $s$ is larger than or equal to $c_s$. This is achieved by induction on the number of steps of the path:

- this is obvious in the initial state, since the initial credit is necessarily larger than $c_{s_0}$;
- assuming that the result for any $k$-step outcome, let $\pi$ be a $k+1$-step outcome. Write $q$ for the penultimate state $last(\pi_{<|\pi|-1})$ of $\pi$, and $q'$ for its last state. According to the induction hypothesis, the accumulated cost in $q$ is larger than or equal to $c_q$, hence it is larger than the accumulated cost in $q$ along $\rho_q$. Let $(q, w, q') \in \mathsf{Next}(q, \sigma')$. By definition of $\sigma'$, we also have $(q, w, q') \in \mathsf{Next}(\rho_q, \sigma)$. Thus $c_q + w \geq l$. This entails that the accumulated cost in $q'$ along $\pi$ is larger than or equal to $c_{q'}$.

It now suffices to guess a memoryless strategy, prune the transitions that are not compatible with this strategy, and check that the cost of any path in the resulting weighted automaton remains above $l$. This is achieved by a shortest path computation, which is in PTIME.

When $u$ is finite, an obvious EXPTIME algorithm is obtained by considering the (exponential-size) game where the cost explicitly appears in the states. A state in this game is then a pair $(q, c)$ where $q$ is a state in the original game and $c$ is the accumulated weight so far. It is sufficient to consider only states for which $c \in [0, u]$, and to add a sink state for the case where $c$ would get out of this interval. The winning-strategy-with-energy-constraint problem is then equivalent to finding a strategy for avoiding this sink state.

Hardness in EXPTIME is proved by encoding the countdown-game problem. □

**Exercice 31** ⋆⋆ Prove that the optimal mean-cost strategy problem and the winning strategy with lower-bound energy constraint problem are logspace-reducible to each other.

# 3 Weighted timed automata

## 3.1 Definitions

Given a finite set $X$ of variables, we define $\mathsf{Constr}(X)$ as the set of clock constraints, defined by the following grammar:

$$\mathsf{Constr}(X) \ni g ::= \top \mid x \sim c \mid g \wedge g$$

where $x$ ranges over $X$, $\sim$ over $\{<, \leq, =, \geq, >\}$ and $c$ over $\mathbb{Z}$. That a valuation $v\colon X \to \mathbb{R}$ of the variables of $X$ satisfy a constraint $g$ of $\mathsf{Constr}(X)$ is defined in the obvious way, and is denoted by $v \models g$.

**Definition 48.** *A* weighted timed automaton *[ALP01, BFH+01] is a tuple* $\mathcal{T} = \langle S, S_0, X, \mathsf{Inv}, \delta, r_s, r_t \rangle$ *where:*

- $\langle S, S_0, X, \mathsf{Inv}, \delta \rangle$ *is a timed automaton,*
- $r_s\colon S \to \mathbb{R}$ *assigns a weight to each state,*
- $r_t\colon \delta \to \mathbb{R}$ *assigns a weight to each transition.*

The semantics of a weighted timed automaton is the semantics of the underlying timed automaton. Weights are only *observers* labelling each transition, but they do not change the semantics. Thus the semantics of a weighted timed automaton can be defined as an infinite weighted automaton:

**Definition 49.** *With a weighted timed automaton* $\mathcal{T} = \langle S, S_0, X, \mathsf{Inv}, \delta, r_s, r_t \rangle$ *we associate an infinite weighted automaton* $\mathcal{A} = \langle Q, Q_0, R \rangle$ *defined as follows:*

- $Q = \{(s, v) \mid s \in S, v\colon X \to \mathbb{R}_{\geq 0} \text{ s.t. } v \models \mathsf{Inv}(q)\}$,
- $Q_0 = S_0 \times \{\mathbf{0}\}$,
- $((s, v), d, (s', v')) \in R$ *iff both* $(s, v)$ *and* $(s', v')$ *are in* $Q$ *and one of the following two cases holds:*
  - $s = s'$ *and there exists* $t \in \mathbb{R}_{\geq 0}$ *s.t.* $v'(x) = v(x) + t$ *for all* $x \in X$ *and* $d = t \times r_s(s)$ *(in this case, the transition is a* delay transition*),*
  - *there is a transition* $e = (s, g, r, s') \in \delta$ *s.t.* $v \models g$ *and* $v'(x) = v(x) \cdot \mathbb{1}_r(x)$*, and* $d = r_t(e)$ *(this is an* action transition*).*

A path in a weighted timed automaton is a path in the associated weighted automaton. The definition of the weight of a path follows; in the sequel, we will only consider the weight with discounting factor 1: discounting according to the number of steps is not very meaningful. We will define the time-discounted weight in the sequel (see Remark 3.3).

**Example.** *Fig. 5 displays an example of a weighted timed automaton (where we assume the global invariant* $x \leq 4 \wedge y \leq 2$*). The trajectory delaying for* 0.5 *t.u. in the initial location, then going to the upper branch and delaying there for* 1.5 *t.u. and finally reaching the rightmost state has total cost* 5*.*

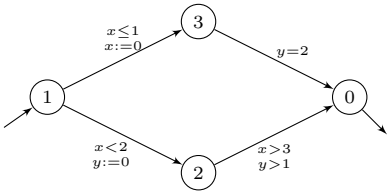


Figure 5: A weighted timed automaton

## 3.2 Shortest and longest paths

**Definition 50.** *We define the following problems:*

| | |
|---|---|
| ***Problem:*** | ***Optimal reachability in weighted timed automata*** |
| ***Input:*** | *A nonnegative weighted timed automaton $\mathcal{T}$, two locations $s$ and $t$, and an integer $n$;* |
| ***Question:*** | *Is there a path $\pi$ in $\mathcal{T}$ from $s$ to $t$ s.t. $w(\pi) \leq n$?* |

**Theorem 51.** *The optimal-reachability-in-weighted-timed-automata problem is PSPACE-complete.*

*Proof.* We begin with membership in PSPACE. The intuition is the following: if there is a path satisfying the conditions of the problem, then there is one that performs transitions only *in the neighbourhood* of punctual regions (*i.e.*, regions where all the clocks have integer values). This will allow us to build a modified version of the region graph in which we will be able to solve our problem. Since the proof is long and intricate, we only sketch its important steps, without formally proving them. The detailed proof is available in [BBBR07].

We begin with the case where the set of regions to be visited is fixed.

**Example.** *We illustrate the construction on the example of Fig. 5. Assume that we fix the following sequence of regions to be visited (the states are named after their cost):*

$$(1, x = y = 0) \to (1, 0 < x = y < 1) \to (3, x = 0, 0 < y < 1) \to \\ (3, 1 < x < 2, y = 2) \to (0, 1 < x < 2, y = 2).$$

*This particular trajectory in the region graph depends on two parameters: the time $t_1$ elapsed in state 1, and the time $t_3$ elapsed in state 3. We must have $0 < t_1 < 1$, $1 < t_3 < 2$, and $t_1 + t_3 = 2$. Moreover, the cost of the corresponding trajectory is $t_1 + 3 \cdot t_3$. The problem thus amounts to computing*

$$\min(t_1 + 3 \cdot t_3) \text{ under constraints } 0 < t_1 < 1 \text{ and } t_1 + t_3 = 2.$$

*This is a linear programming problem, which is solved in polynomial time. This method also allows to decide whether the optimal cost is realizable. In the example above, the optimal cost is 4, but it cannot be realized because of the strict inequalities (the "optimal" value is reached for $t_1 = 1 - \varepsilon$ and $t_3 = 1 + \varepsilon$).*

The result depicted in this example extends to the general case.

**Definition 52.** *Let $\mathcal{T}$ be a weighted timed automaton, and $\varepsilon \in (0, 1/2]$. We define its* $\varepsilon$-semantics *as the weighted graph $\mathcal{A}^\varepsilon = \langle Q, R^\varepsilon \rangle$ where $Q$ is the set of possible configurations of $\mathcal{T}$ (as in Def. 49) and $R^\varepsilon$ contains only action transitions and delay transitions* with duration $t$ s.t. $t$ is at distance at most $\varepsilon$ from an integer.

We then have the following result:

**Lemma 53.** *Given a trajectory $\rho_R$ in the region graph, the optimal cost of the trajectories following $\rho_R$ can be approached to $\varepsilon$ using a trajectory in the $\varepsilon$-semantics of $\mathcal{T}$.*

We now define $\varepsilon$-regions:

**Definition 54.** *Let $\varepsilon \in (0, 1/2]$ and $M \in \mathbb{N}$. Two clock valuations $v$ and $v'$ are* $\varepsilon, M$-equivalent *if they satisfy the following conditions*[3]*:*

- *they are $M$-equivalent (in the classical sense of region equivalence):*
  - *for all $x \in X$, $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ or $v(x) > M$ and $v'(x) > M$;*
  - *for all $x, y \in X$ with $v(x) \leq M$ and $v(y) \leq M$, we have $\overline{v(x)} \leq \overline{v(y)}$ iff $\overline{v'(x)} \leq \overline{v'(y)}$;*
  - *for all $x \in X$ with $v(x) \leq M$, we have $\overline{v(x)} = 0$ iff $\overline{v'(x)} = 0$;*
- *for all $x \in X$ with $v(x) \leq M$, we have $\overline{v(x)} \leq \varepsilon$ iff $\overline{v'(x)} \leq \varepsilon$, and $\overline{v(x)} > 1 - \varepsilon$ iff $\overline{v'(x)} > 1 - \varepsilon$.*

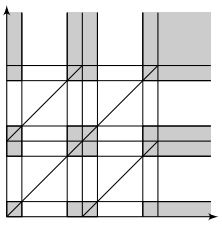

Figure 6: $\varepsilon$-regions

**Example.** *Figure 6 displays the set of $\varepsilon$-regions in two dimentions for $M = 2$.*

Given a configuration $(s, v)$ of a timed automaton $\mathcal{T}$, we write $[(s, v)]^\varepsilon$ for the $\varepsilon$-region it belongs to. As mentionned above, for the purpose of computing optimal trajectories we only need to consider $\varepsilon$-regions that are close to punctual regions (*i.e.*, those in grey on Fig. 6).

[3] In the sequel, $\overline{a}$ represents the fractional part of $a$, and $\lfloor a \rfloor$ represents its integer part.

**Definition 55.** *Given a timed automaton $\mathcal{T}$, $0 < \varepsilon \leq 1/2$ and $M \in \mathbb{N}$, the set of* useful $\varepsilon, M$*-regions is defined as*

$$S^\varepsilon = \{[q]^\varepsilon \mid \forall x \in X.\ v(x) \leq M \Rightarrow (v(x) < \varepsilon \text{ or } v(x) > 1 - \varepsilon)\}.$$

**Definition 56.** *Let $\mathcal{T}$ be a weighted timed automaton, $M$ be the maximal constant appearing in $\mathcal{T}$, and $0 < \varepsilon \leq 1/2$. The $\varepsilon$-region automaton $\mathcal{R}^\varepsilon = (S^\varepsilon, T)$ is the finite automaton whose states are the useful $\varepsilon$-regions, with a transition between two regions $r$ and $r'$ iff there exist $q \in r$ and $q' \in r'$ s.t. there is a transition from $q$ to $q'$ in $\mathcal{T}$.*

The relation between $\mathcal{R}^\varepsilon$ and $\mathcal{A}^\varepsilon$ is given by the following two lemmas:

**Lemma 57.** *Let $\mathcal{T}$ be a weighted timed automaton, and $\varepsilon \in (0, 1/3]$. Given a run in $\mathcal{R}^\varepsilon$ starting in a region of the form $(s, \mathbf{0})$, there exists a run in $\mathcal{A}^\varepsilon$ visiting the same sequence of regions.*

Conversely:

**Lemma 58.** *Let $\mathcal{T}$ be a weighted timed automaton, and $m \in \mathbb{N}$, and $\varepsilon \in (0, 1/(2m+2)]$. Any trajectory of length at most $m$ in $\mathcal{A}^\varepsilon$ can be projected as a trajectory in $\mathcal{R}^{(m+1)\cdot\varepsilon}$.*

Now that we have a relationship between the $\varepsilon$-optimal trajectories and the $\varepsilon$-region automaton, we abstract away the value $\varepsilon$.

**Lemma 59.** *Given two values $\varepsilon$ and $\varepsilon'$ in $(0, 1/3]$, the graphs $\mathcal{R}^\varepsilon$ and $\mathcal{R}^{\varepsilon'}$ are isomorphic.*

Also, as we only consider *useful* $\varepsilon$-regions, we have the following result:

**Lemma 60.** *Let $\mathcal{T}$ be a timed automaton, and $\varepsilon \in (0, 1/6]$. With each delay transition $r \to r'$ in the $\varepsilon$-region automaton $\mathcal{R}^\varepsilon$, we can associate an integer $N$ s.t. the duration $\tau$ of any transition in $\mathcal{T}$ corresponding to $r \to r'$ is s.t. $|N - \tau| \leq 2\varepsilon$.*

This is illustrated on Fig. 7. Notice that this result would not hold for the standard definition of timed automaton: inside a region where no clock has an integral value, some transitions have duration almost 1 while some others have duration almost 0.

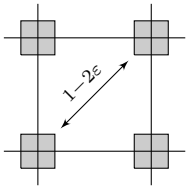


Figure 7: Illustration of Lemma 60

This lemma allows us to decorate the $\varepsilon$-region automaton with weights corresponding to the weight of the original weighted timed automaton:

**Definition 61.** *Let $\mathcal{T}$ be a weighted timed automaton. We define the* weighted corner automaton *of $\mathcal{T}$, which we denote by $\mathcal{A}$, as the* 1/6*-region automaton whose transitions are decorated with the following weights:*

- *for each delay transition, we consider the integer $N$ defined in Lemma 60, and assign to this transition the weight $C \cdot N$ where $C$ is the cost of the location corresponding to the source location;*
- *action transitions are decorated with the weight of the corresponding transition in $\mathcal{T}$.*

Notice that we have chosen $\varepsilon = 1/6$ in this definition, but any smaller (yet positive) value would fit. We now state that the above construction allows us to solve our original problem:

**Lemma 62.** *Let $\mathcal{T}$ be a weighted timed automaton, and $r, r'$ be two regions of $\mathcal{T}$. Then the optimal weight of all the trajectories from $r$ to $r'$ is the optimal weight in the weighted corner automaton.*

The PSPACE algorithm is obtained by non-deterministically guessing a trajectory in the weighted corner automaton. Hardness is a direct consequence of the fact that reachability is PSPACE-complete. □

## 3.3 Optimal mean payoff

Similarly to the untimed case, we can define the *mean-cost* of a trajectory by deviding the weight of a path with its duration.

**Definition 63.** *Let $\mathcal{T}$ be a weighted timed automaton, and $\pi$ be an infinite trajectory in $\mathcal{T}$. For any length-$k$ prefix $\pi_k$ of $\pi$, we let $w(\pi_k)$ be its total weight, and $t(\pi_k)$ be its total duration (defined as a cost having derivative 1 in each location and discrete value 0 on any transition). The* mean cost *of $\pi$ is defined as*

$$m(\pi) = \liminf_{k \to \infty} w(\pi_k)/t(\pi_k).$$

*We define the optimal mean-cost problem as follows:*

| | |
|---|---|
| ***Problem:*** | ***Optimal mean-cost in a weighted timed automaton*** |
| ***Input:*** | *a weighted timed automaton $\mathcal{T}$, a value $n$;* |
| ***Question:*** | *Is there an infinite path $\pi$ in $\mathcal{T}$ s.t. $m(\pi) \leq n$?* |

**Theorem 64.** *Let $\mathcal{T}$ be a bounded*[4] *weighted timed automaton in which any trajectory is time-diverging. Then the optimal-mean-cost problem is decidable, and it is PSPACE-complete.*

[4] This is a technical requirement meaning that along any execution, the values of all the clocks remain bounded.

*Proof.* Again, we only sketch the main steps of the proof. The details can be found in [BBL04]. We consider the corner region automaton defined at Def. 61.

With a trajectory $\pi$ in $\mathcal{T}$, we associate the set $\mathsf{proj}(\pi)$ of its *projections* over the corner region autamaton as follows: $\mathsf{proj}(\pi)$ contains all the trajectories in the corner region automaton visiting the same sequence of regions.

**Lemma 65.** *Let $\pi$ be a trajectory in $\mathcal{T}$. Then there exists a trajectory $\gamma$ in $\mathsf{proj}(\pi)$ s.t. $m(\gamma) \leq m(\pi)$.*

As a consequence, the optimal mean cost computed in the corner region automaton is less than or equal to the optimal mean cost of the weighted timed automaton. Conversely:

**Lemma 66.** *let $\pi$ be a trajectory in the corner region automaton of $\mathcal{T}$. For any $\varepsilon > 0$, there is a trajectory $\pi'$ in $\mathcal{T}$ s.t. $|m(\pi) - m(\pi')| \leq \varepsilon$.* □

**Remark.** *We could also define the $\lambda$-discounted weight of a trajectory: in that case, the discount depends on the duration elapsed since the beginning of the trajectory. For instance, if a trajectory begins with letting $k$ t.u. elapse in location $s$, this would have cost $\int_{t=0}^{k} \lambda^t \cdot w(s) \underset{!}{t}$, that is $w(s) \cdot \frac{\lambda^t - 1}{\ln \lambda}$. Of course, if such a transition occurs along a trajectory after an initial prefix of duration $t'$, the corresponding weight should be multiplied by $\lambda^{t'}$. Similarly, action transitions are discounted by the same amount depending on the duration of the prefix.*

*Again, computing the optimal discounted weight for timed automata can be solved using the corner region automaton (see [FL08]).*

## 3.4 Weighted temporal logics

The WCTL model-checking problem can naturally be extended to weighted *timed* automata.

**Theorem 67.** *The WCTL model-checking problem for weighted timed automata is undecidable (as soon as the automaton has three clocks).*

*Proof.* The proof consists in reducing the reachability problem for two-counter machines (which is known to be undecidable [Min61]). Let $\mathcal{M}$ be such a two-counter machine. We build a weighted timed automaton $\mathcal{T}_{\mathcal{M}}$ (with three clocks and one stopwatch cost) and a WCTL-formula $\Phi$ such that given $q_0$, a well-chosen state of $\mathcal{T}_{\mathcal{M}}$, we have that $\mathcal{M}$ halts if, and only if, $q_0 \models \Phi$. The two counters $c_1$ and $c_2$ will be encoded alternately by three clocks $x$, $y$ and $z$. The value of $c_1$ is encoded by $x_1 = 1/2^{c_1}$ (with $x_1 \in \{x, y, z\}$) whereas the value of $c_2$ is encoded by $x_2 = 1/3^{c_2}$ (with $x_2 \in \{x, y, z\}$). To each instruction will be associated six modules, one for each injective function $\{x_1, x_2\} \to \{x, y, z\}$.

We first explain how we increment a counter. Consider the following instruction of the two-counter machine:

$$p_i : \ c_1 := c_1 + 1; \ \texttt{goto}\ p_j.$$

We assume that the initial value of $c_1$ is stored in clock $x$ whereas that of $c_2$ is stored in $y$. To $p_i$, we associate the automaton $\mathsf{Aut}^i_{1,+}(x,y,z)$ as in Fig. 8. In that figure (and in all the other ones), costs which are omitted are equal to zero. The subscript $1,+$ is a remainder that instruction $p_i$ deals with counter stored in the first clock ($x$ here) and is an incrementation (we might omit it when it is not necessary), the tuple $(x,y,z)$ indicates which clocks encode counters $c_1$ and $c_2$: here, $c_1$ is initially stored in $x$ and $c_2$ is initially stored in $y$. At the end of this module, the new values of $c_1$ and $c_2$ are stored in $z$ and $y$, resp.; that's why we swap $x$ and $z$ when leaving the module (transition from $D^i_{x,y,z}$ to $A^j_{z,y,x}$).

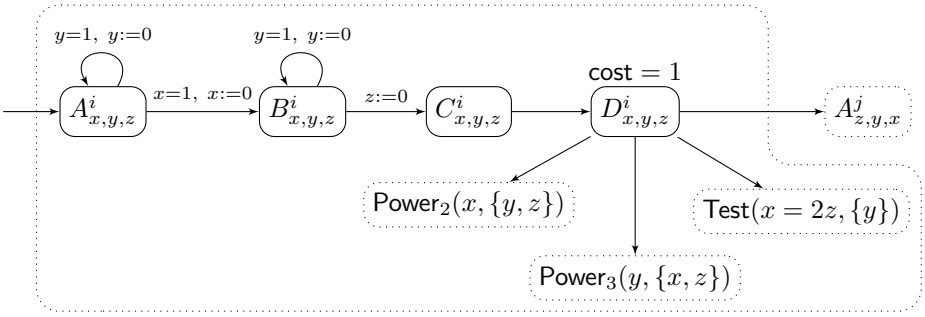


Figure 8: Incrementing a counter

For that automaton to really increment the first counter, we will enforce the following requirements:

1. the delay between arrival in $A^i_{x,y,z}$ and arrival in $D^i_{x,y,z}$ is 1 t.u.,
2. when entering $D^i_{x,y,z}$, $z$ equals $x/2$ and
3. the delay elapsed in $D^i_{x,y,z}$ is 0.

The last point will be ensured through a global WCTL-formula stating that no cost is accumulated in location $D^i_{x,y,z}$. The second point is obtained by a module $\mathsf{Test}(x=2z,\{y\})$, together with a WCTL-formula $\phi_1$. Finally, according to Lemma 68 below, the first point is enforced by checking that the values of $x$ and $y$ when entering $D^i_{x,y,z}$ are $1/2^n$ and $1/3^m$ for some integers $n$ and $m$. Those conditions are ensured by modules $\mathsf{Power}_2$ and $\mathsf{Power}_3$ and the associated formulas $\phi_2$ and $\phi_3$.

**Lemma 68.** *If a run enters location $A^i_{x,y,z}$ with $x = 1/2^{c_1}$, $y = 1/3^{c_2}$ and enters location $D^i_{x,y,z}$ $t$ time units later with the value of $x$ of the form $1/2^n$ for some $n$, and the value of $y$ of the form $1/3^m$ for some $m$, then $t = 1$, $n = c_1$ and $m = c_2$.*

This lemma can easily be proved using elementary arithmetical manipulations. It plays a crucial role in our reduction: it explains how comparing clocks

to powers of 1/2 and 1/3 gives a way to measure exactly 1 t.u., and thus why we encode the counters as powers of 1/2 and 1/3. Note that 2 and 3 could be replaced by any two relatively prime numbers.

Similar ideas can be used for designing an automaton $\mathsf{Aut}^i_{2,+}(x,y,z)$ that increments the second counter (*i.e.* ends up with $z = y/3$, while $x$ returns to its original value), involving module $\mathsf{Test}(x = 3z, \{y\})$.

We now treat instruction:

$$p_i : \texttt{ if } (c_1 > 0) \texttt{ then } c_1 := c_1 - 1; \texttt{ goto } p_j \texttt{ else goto } p_k.$$

We only give the construction of automaton $\mathsf{Aut}^i_{1,-}(x,y,z)$, which is a slight variation of the previous construction. This automaton implements the decrementation of the first counter, initially stored in $x$, unless it equals zero.

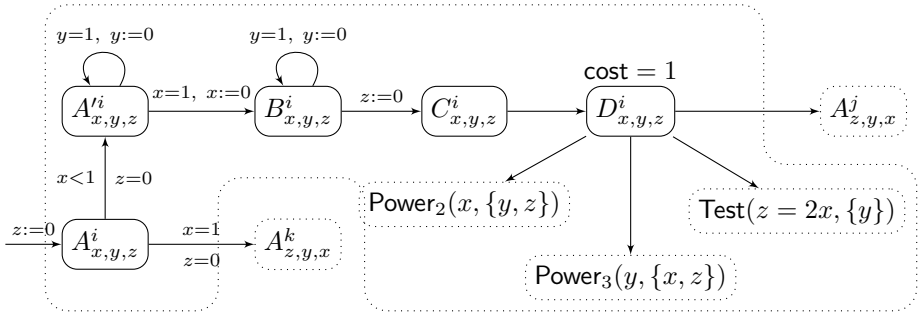


Figure 9: Incrementing a counter

In the global reduction, we will enforce the following properties:

1. the values of $x$ and $y$ when entering $D^i_{x,y,z}$ are $1/2^n$ and $1/3^m$ for some $n$ and $m$,
2. when entering $D^i_{x,y,z}$, $z$ equals $2x$ and
3. the delay in $D^i_{x,y,z}$ is 0.

As previously, we can prove that these three conditions express correctness of the construction. Lemma 68 clearly also holds for $\mathsf{Aut}^i_{1,-}(x,y,z)$. Automaton $\mathsf{Aut}^i_{2,-}(x,y,z)$ is built in the same way.

It then suffices to plug the different modules into each other following the initial two-counter machine. It remains us to explain how the test modules ($\mathsf{Power}_2$, $\mathsf{Power}_3$ and $\mathsf{Test}$) are built.

As a first step, we build modules for adding the value of a clock to a cost variable: this is achieved using modules $\mathsf{Add}^+(x,\{z\})$ and $\mathsf{Add}^-(x,\{z\})$, displayed on Fig. 10 and 11. Those automata clearly satisfy the following Lemma:

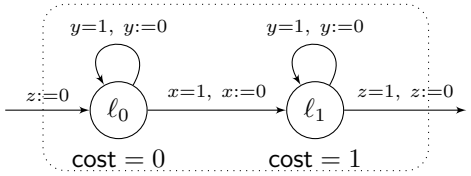


Figure 10: Automaton $\mathsf{Add}^+(x, \{z\})$

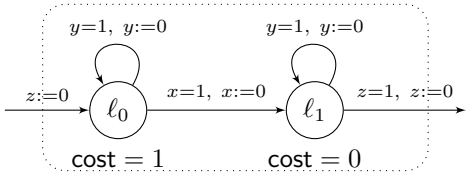


Figure 11: Automaton $\mathsf{Add}^-(x, \{z\})$

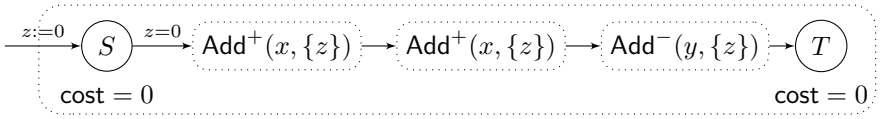


Figure 12: Automaton $\mathsf{Test}(y = 2x, \{z\})$

**Lemma 69.** *If a run enters location $\ell_0$ of $\mathsf{Add}^+(x, \{z\})$ (resp. $\mathsf{Add}^-(x, \{z\})$) with $x = \alpha_0 \in [0, 1]$, $y = \beta_0 \in [0, 1]$ and $\mathsf{cost} = \gamma_0$, it then leaves location $\ell_1$ with the same values for $x$ and $y$, and with $\mathsf{cost} = \gamma_0 + \alpha_0$ (resp. $\mathsf{cost} = \gamma_0 + 1 - \alpha_0$).*

Module $\mathsf{Test}(y = 2x, \{z\})$ is the (deterministic) automaton displayed on Fig. 12. It sets the cost to $2x + 1 - y$. Let $\varphi_1 = S \wedge \mathbf{EF}_{\leq 1} T \wedge \mathbf{EF}_{\geq 1} T$. The following Lemma clearly holds:

**Lemma 70.** *Formula $\varphi_1$ holds in $S$ along module $\mathsf{Test}(y = 2x, \{z\})$ with $x = \alpha_0 \in [0, 1]$ and $y = \beta_0 \in [0, 1]$ if, and only if, $\beta_0 = 2\alpha_0$.*

This construction can easily be adapted for other tests, especially for building a module $\mathsf{Test}(y = 3x, \{z\})$ testing if $y = 3x$.

Module $\mathsf{Power}_2(x, \{y, z\})$ is displayed on Fig. 13. Note that it requir es two auxiliary clocks. Note also that it uses an update "$x := y$", instead of classical resets. This is for the sake of simplicity, as the module could be adapted (by duplicating the periodic part, involving no extra clock) in order to only have standard resets.

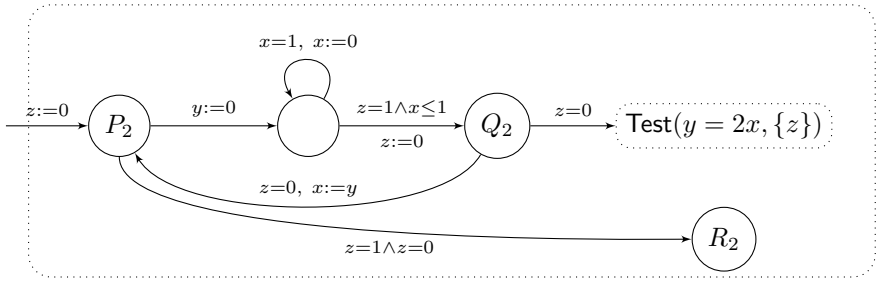


Figure 13: Automaton $\mathsf{Power}_2(x, \{y, z\})$

We let $\varphi_2 = P_2 \wedge \mathbf{E}((Q_2 \rightarrow \mathbf{E}(Q_2 \,\mathbf{U}\, \varphi_1)) \,\mathbf{U}\, R_2)$. We have the following Lemma:

**Lemma 71.** *Formula $\varphi_2$ holds in $P_2$ in module $\mathsf{Power}_2(x, \{y, z\})$ with $x = \alpha_0 \in (0, 1]$ if, and only if, there exists a non-negative integer $d$ s.t. $\alpha_0 = 1/2^d$.*

It is easy to adapt this construction in order to build a module $\mathsf{Power}_3(x, \{y, z\})$ and a formula $\varphi_3$ that check if $x$ is of the form $1/3^d$, for some integer $d$.

In order to conclude the construction, we label each $D^i_{x,y,z}$ state with an atomic proposition $D$, and consider the formula

$$\Phi = \mathbf{E}[(D \rightarrow \phi) \,\mathbf{U}_{\leq 0}\, \mathsf{Halt}] \quad \text{where} \quad \phi = \bigwedge_{i=1,2,3} \mathbf{E}(D \,\mathbf{U}_{\leq 0}\, \phi_i).$$

Then:

**Lemma 72.** *$\mathcal{T}_{\mathcal{M}}, q_0 \models \Phi$ iff the two-counter machine $\mathcal{M}$ has a halting computation.* □

We can recover decidability by restricting to one-clock timed automata:

**Theorem 73.** *The WCTL model-checking problem is decidable (in PSPACE) for nonnegative one-clock weighted timed automata.*

*Proof.* The idea of the proof is to refine the regions. We begin with an example showing that this is necessary.

Consider the 1-clock weighted timed automaton depicted on Fig. 14. Then

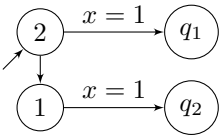


Figure 14: A one-clock weighted timed automaton

formula $\mathbf{EF}_{\leq 1}\, q_1$ only holds in the initial state if clock $x$ is larger than or equal to $1/2$. Similarly, formula $\mathbf{E}(\neg\, \mathbf{EF}_{\leq 1}\, q_1) \,\mathbf{U}_{\geq 1}\, q_2$ only holds in the initial state for values of $x$ less than $1/4$.

On the other hand, we have the following proposition:

**Lemma 74.** *Let $\Phi$ be a WCTL formula and let $\mathcal{T}$ be a one-clock weighted timed automaton. Then there exist finitely many constants $0 = a_0 < a_1 < \ldots < a_n < a_{n+1} = +\infty$ s.t. for every location $q$ of $\mathcal{T}$, for every $0 \leq i \leq n$, the truth of $\Phi$ is uniform over $\{(q, x) \mid a_i < x < a_{i+1}\}$. Moreover,*

- *$\{a_0, ..., a_n\}$ contains all the constants appearing in clock constraints of $\mathcal{T}$;*

- *the constants are integral multiples of* $1/C^{|\Phi|}$ *where* $|\Phi|$ *is the* constrained temporal height *of* $\Phi$*, i.e. the maximal number of nested constrained modalities in* $\Phi$*, and* $C$ *is the lcm of all positive costs labeling a location of* $\mathcal{T}$*;*
- $a_n$ *equals the largest constant* $M$ *appearing in the guards of* $\mathcal{T}$*;*
- $n \leq M \cdot C^{|\Phi|} + 1$.

This provides us with a finite abstraction of the one-clock weighted automaton in which we can check our WCTL formula. It can be shown that polynomial space is sufficient for that. □

**Exercice 32** ⋆⋆⋆ Prove that WCTL$^*$ model-checking is undecidable on one-clock weighted timed automata [BLM06].

## 3.5 Energy constraints

The problem of energy constraints in weighted timed automata has only been solved for the simple case of one-clock automata with no discrete costs ($r_t(\delta) = \{0\}$).

**Theorem 75.** *The infinite-path-with-energy-constraints problem with lower-bound constraint is decidable (in* PTIME*) on one-clock weighted timed automata* provided that there are no discrete costs*.*

*Proof.* The proof again relies on the corner region automaton.

**Lemma 76.** *Let* $A$ *be a one-clock weighted timed automaton with* $r_t \equiv 0$.

Completeness: *Let* $\gamma$ *be an infinite run in* $A$ *from* $(q_0, 0)$ *which is* $c$*-feasible (*i.e.*, it satisfies the cost constraints if starting with initial credit* $c$*) for some* $c < +\infty$*. Then there exists a* $c$*-feasible infinite run* $\gamma'$ *from* $(q_0, \langle\{0\}, 0\rangle)$ *in the corner region automaton.*

Soundness: *Let* $\gamma'$ *be an infinite run in the corner region automaton from* $(q_0, \langle\{0\}, 0\rangle)$ *which is* $c$*-feasible for some* $c < +\infty$*. Then, for any* $\varepsilon > 0$*, there exists a* $(c + \varepsilon)$*-feasible infinite run* $\gamma$ *from* $(q_0, 0)$ *in* $A$.

We first prove *completeness:* $\gamma'$ is obtained from $\gamma$ by essentially pushing delay transitions within the same region $\rho = (a, b)$ towards the most profitable end-point. Consider some maximal sub-sequence of $\gamma$ of alternating delay and switch transitions within $\rho$:

$$\gamma : \cdots \to (q_1, v_1) \xrightarrow{d_1} (q_1, v_2) \xrightarrow{e_1} (q_2, v_2) \xrightarrow{d_2} (q_2, v_3) \xrightarrow{e_2} \cdots \xrightarrow{d_n} (q_n, v_{n+1}) \to \cdots$$

where $e_i$ are non-resetting switch transitions, and $a < v_1 < v_2 < \cdots < v_{n+1} < b$. Now with $j \in \{1, \ldots, n\}$ maximizing the rate $r_s(q_j)$, we construct the following sub-sequence of $\gamma'$:

$$\gamma' : \cdots \to (q_1, \langle \rho, a \rangle) \to (q_2, \langle \rho, a \rangle) \to \cdots \to$$
$$(q_j, \langle \rho, a \rangle) \to (q_j, \langle \rho, b \rangle) \to (q_{j+1}, \langle \rho, b \rangle) \to \cdots \to (q_n, \langle \rho, b \rangle) \to \cdots$$

Given an initial credit, it is easy to see that the remaining credit at a state in $\gamma'$ is always greater than or equal to the remaining credits at the corresponding states in $\gamma$.

We now turn to *soundness:* Let $K$ be the maximum of the absolute values of weight-rates of $A$. If $(q, \langle \{a\}, a \rangle) \to (q, \langle (a, b), a \rangle)$ is the $n$-th transition in $\gamma'$, the corresponding $n$-th transition of $\gamma$ is taken to be $(q, a) \to (q, a + \varepsilon(2^{n+1}K)^{-1})$. Similarly, if $(q, \langle (a, b), a \rangle) \to (q, \langle (a, b), b \rangle)$ is the $n$-th transition of $\gamma'$, the corresponding $n$-th transition of $\gamma'$ is taken to be $(q, a + \varepsilon(2^{n+1}K)^{-1}) \to (q, b - \varepsilon(2^{n+1}K)^{-1})$. □

**Exercice 33** ⋆⋆ Show that the requirement about the absence of discrete costs is compulsory in the result above.

**Open Problem 3** Is the infinite-path-with-energy-constraint problem decidable in the presence of discrete weights? For automata with more than one clocks? With interval-constraints?

# 4 Weighted timed game automata

## 4.1 Definitions

**Definition 77.** *A* weighted timed game automaton *is a tuple* $\mathcal{G} = \langle S, S_0, X, \mathsf{Inv}, \delta_1, \delta_2, r_s, r_t \rangle$ *where* $\langle S, S_0, X, \mathsf{Inv}, \delta_1 \cup \delta_2, r_s, r_t \rangle$ *is a weighted timed automaton, Transitions in* $\delta_1$ *are controlled by Player* 1*, those in* $\delta_2$ *belong to Player* 2*.*

**Remark.** *In our setting, a timed game automaton is a timed automaton in which the set of transitions is split into* controllable *and* uncontrollable *ones (considering that we are Player* 1*). We only define weighted timed game automata in a two-player setting. Defining general concurrent n-player timed games involves many technical details that are out of the scope of this course.*

A trajectory in a weighted timed game automaton is a trajectory in the underlying weighted timed automaton. We now define *timed* strategies:

**Definition 78.** *Let* $\mathcal{T} = \langle S, S_0, X, \mathsf{Inv}, \delta_1, \delta_2, r_s, r_t \rangle$ *be a weighted timed game automaton. A strategy for Player i (*$i \in \{1, 2\}$*) is a partial function* $\sigma$ *mapping each finite trajectory of* $\mathcal{T}$ *to* $\delta_i \cup \{wait\}$*, where* $wait \notin \delta_1 \cup \delta_2$ *is the "wait" action. For a finite trajectory* $\gamma$ *ending in* $(q, v)$*, it is required that*

- *if* $\sigma(\gamma) = wait$ *then* $v + t \models \mathsf{Inv}(q)$ *for some* $t > 0$*,*

- *if* $\sigma(\gamma) = e$ *then* $e$ *is of the form* $(q, g, r, q')$ *and* $v \models g$ *and* $v[r \to 0] \models \mathsf{Inv}(q')$.

The outcomes of a strategy are the set of trajectories that are compatible with that strategy:

**Definition 79.** *Let* $\mathcal{T} = \langle S, S_0, X, \mathsf{Inv}, \delta_1, \delta_2, r_s, r_t \rangle$ *be a weighted timed game automaton,* $s \in S$ *be a state and* $v$ *be a clock valuation, and* $\sigma_i$ *be a strategy for Player* $i$ *(*$i \in \{1, 2\}$*). The set of finite outcomes of* $\sigma_i$ *from* $(s, v)$*, written* $\mathsf{Out}_f((s, v), \sigma_i)$*, is defined inductively as follows:*

- $(s, v)$ *is the only outcome of length* $0$*;*
- *an execution* $\rho \cdot ((s', v'), d, (s'', v''))$ *is in* $\mathsf{Out}_f((s, v), \sigma_i)$ *iff* $\rho \in \mathsf{Out}_f((s, v), \sigma_i)$ *and one of the following three cases holds:*
  - *there is a transition* $e = (s', g, r, s'') \in \delta_{3-i}$ *s.t.* $v' \models g$ *and* $v''(x) = v'(x) \cdot \mathbb{1}_r(x)$ *and* $d = r_t(e)$*;*
  - $\sigma_i(\rho) = (s', g, r, s'') \in \delta_i$ *and* $v' \models g$ *and* $v''(x) = v'(x) \cdot \mathbb{1}_r(x)$ *for all* $x$*, and* $d = r_t(e)$*;*
  - *there exists* $t \in \mathbb{R}_{>0}$ *s.t. for all* $x$*,* $v''(x) = v(x) + t$ *and* $s' = s''$ *and* $d = t \cdot r_s(s')$ *and for all* $0 \leq t' < t$*,* $\sigma_i(\rho \cdot ((s', v'), r_s(s') \cdot t', (s', v' + t'))) = \mathit{wait}$*.*

*The set of* maximal outcomes *of* $\sigma_i$ *from* $(s, v)$ *is the union of*

- *the set of finite outcomes that cannot be extended into a longer outcome;*
- *the set of infinite trajectories that are limits of a sequence of finite outcomes.*

## 4.2 Shortest and longest paths

In this section, we consider reachability objectives: we assume that we are given a goal location $s_g$ which has no outgoing transitions. The winning objective will be made of all the finite trajectories containing $s_g$. A strategy $\sigma$ is winning from an initial state $s_0$ if all its outcomes belong to the objective.

**Definition 80.** *Let* $\mathcal{G}$ *be a weighted timed game automaton,* $(s_0, v_0)$ *be an initial configuration, and* $\sigma_1$ *be a strategy for Player* $1$*. The* cost *of* $\sigma_1$ *is* $+\infty$ *if* $\sigma_1$ *is not winning from* $(s_0, v_0)$*, and it is*

$$\mathsf{cost}((s_0, v_0), \sigma_1) = \sup\{\mathit{cost}(\rho) \mid \rho \in \mathsf{Out}((s_0, v_0), \sigma_1)\}$$

*if* $\sigma_1$ *is winning.*

**Definition 81.** *Let* $\mathcal{G}$ *be a weighted timed game automaton,* $(s_0, v_0)$ *an initial configuration of the game. The* optimal cost of winning *from* $(s_0, v_0)$ *is defined as*

$$\mathsf{OptCost}(s_0, v_0) = \inf\{\mathsf{cost}((s_0, v_0), \sigma_1) \mid \sigma_1 \text{ winning strategy}\}.$$

**Example.** *We begin with a simple example showing that the region equivalence is not sufficient for dealing with weighted timed game. Consider the weighted timed game depicted on Fig. 15. In this (turn-based) game, Player* 2 *only con-*

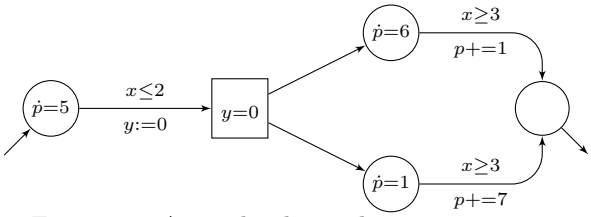

Figure 15: A weighted timed game automaton

*trols one state, and time cannot elapse in that state. The only role of Player* 2 *is to decide whether the game should go in the upper or in the lower branch. Let us compute the optimal price to pay for reaching the final state for Player* 1*:*

- *in the initial state, she can wait for a delay* $t \leq 2$*, and pay* $5 \cdot t$*;*
- *if the game goes in the upper branch, it is rather clear that the optimal strategy is to wait until* $x = 3$ *and then immediately go to the final state. This means waiting for* $3 - t$ *time units, and paying* $6 \cdot (3 - t)$*.*
- *similarly in the lower branch: the optimal strategy is to wait for* $3 - t$ *and pay* $1 \cdot (3 - t)$*.*

*In the end, computing the optimal strategy in this example amounts to computing the value of* $t$ *that realizes the following infimum:*

$$\inf_{0 \leq t \leq 2} 5 \cdot t + \max\{6 \cdot (3 - t) + 1, 1 \cdot (3 - t) + 7\}.$$

*A simple computation gives* $t = 1.8$*, resulting in a cost* 17.2*.*

As can be expected, this quickly leads to undecidability:

**Theorem 82.** *The optimal-reachability-strategy problem is undecidable on weighted timed game automata (having at least three clocks).*

**Exercice 34** ⋆⋆ Write the proof of Theorem 82.

Restricting to one-clock *turn-based* weighted timed game automata allows us to recover decidability:

**Theorem 83.** *Let* $\mathcal{G}$ *be a one-clock turn-based weighted timed game automaton, with one special goal state* $s_g$*. Then for each* $s \in S$*, the function* $x \mapsto \mathsf{OptCost}_{\mathcal{G}}(q, x)$ *(assigning to each initial value of the clock the optimal cost of reaching the goal state) is piecewise affine. Moreover, it can be computed exactly, and for any* $\varepsilon > 0$*, we can compute a memoryless strategy* $\sigma$ *achieving cost* $\mathsf{OptCost}(q, x) + \varepsilon$*.*


Figure 16: No optimal memoryless strategy

**Example.** *Before turning to the proof, we first depict (see Fig. 16) examples where the optimal strategy does not exist or is not memoryless. Still, on both cases, there is an almost-optimal memoryless strategy. In the first game, the optimal cost of winning is* 1*, but it is obviously not achievable. In the second game, the optimal cost of winning is* 2*, and there is an optimal winning strategy. But this strategy must remember the date at which the transition to the lower state has been taken in order to achieve cost less than* 2*.*

*Proof.* We only sketch the proof, whose details can be found in [BLMR06]. It relies on an extension of the definition of weighted timed game automata with *outside cost functions* and *urgency*:

- an *urgent* state is a state where time cannot elapse. This can be modelled by adding an extra clock which is reset upon entering an urgent state, which must also be decorated with an invariant requiring that this clock must be 0.
- an *outside cost function* is a function assigning to each value of the clock the cost of winning when starting with that value. This can be used to "simplify" the game, as depicted on Fig. 17: in that figure, the game on

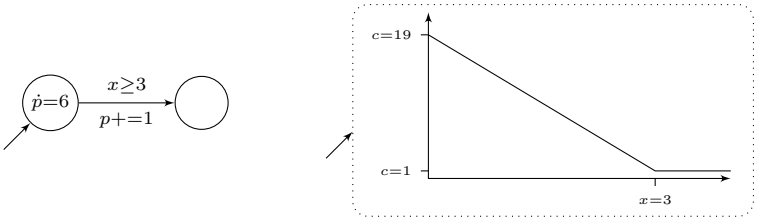


Figure 17: Outside cost functions

the left is "equivalent" to the game on the right, which is reduced to the function representing the optimal cost of winning when entering that state as a function of the value of $x$.

**Example.** *We revisit our previous example of Fig. 15. This example has two clocks, but clock* $y$ *can be dropped after making the second state urgent. We solve this problem be recursively computing the outside cost functions corresponding*

*to each state, as depicted on Fig. 18. In the end, we obtain the optimal cost of winning from the initial state for any initial value of clock $x$. We (hopefully) obtain the same values as computed before: the optimal cost of winning is* 17.2*, and the optimal strategy consists in switching when* $x = 1.8$.

Of course, this example only depicts a simple case where there are no cycles. In the general case, the algorithm works as follows:

- we first transform the game in such a way that the value of the clock is bounded by 1 in each state. This is achieved by taking several copies of the game, one for each integer value of $x$ below the maximal constant $M$. This entails an exponential blowup of the game;
- we remove resetting transitions from strongly connected components: since there is only one clock, we can assume that a resetting transition is fired at most once in each winning play. Again, we take as many copies of the game as its number of resetting transitions.
- we apply an algorithm for handling strongly connected components (see below). This algorithm replaces a strongly connected component with its equivalent outside cost function. Applying this procedure repeatedly, we end up with the cost function in the initial state.

Strongly connected components are handled by repeatedly removing one of their states. More precisely, we prove by induction that the following result holds:

**Lemma 84.** *Let* $\mathcal{G} = \langle S_1, S_2, S_0, X, \mathsf{Inv}, \delta, r_s, r_t \rangle$ *be a turn-based one-clock weighted timed game automaton without reset, with global invariant "*$x \leq 1$*", and whose underlying graph is strongly connected (or contains a single state). Let* $U \subseteq S_2$ *be the set of urgent locations (which are required to be uncontrollable). Let* $f$ *map some of the states of* $\mathcal{G}$ *to outside cost functions (those states are assumed to have no outgoing edge). Then*

- *for every state* $s$, $\mathsf{OptCost}_{\mathcal{G}}(s, x)$ *is computable for* $x \in [0, 1]$*;*
- *this function is piecewise affine whose finitely many segments either have slope* $-c$ *for some* $c \in r_s(S)$ *or are fragments of some outside cost function of* $f$*;*
- *there exists a finite partition* $(I_i)_i$ *of* $[0, 1]$ *s.t., for any* $\varepsilon > 0$*, we can compute a memoryless* $\varepsilon$*-optimal strategy* $\sigma$ *that is constant on each interval* $I_i$*.*

The proof is by induction on the number of non-urgent (uncontrollable) locations in $\mathcal{G}$.

- if all locations are urgent, and if there are at least two states, then Player 2 can avoid the play to reach the final state. If there is only one state, then time cannot elapse and computing the optimal cost is straightforward;

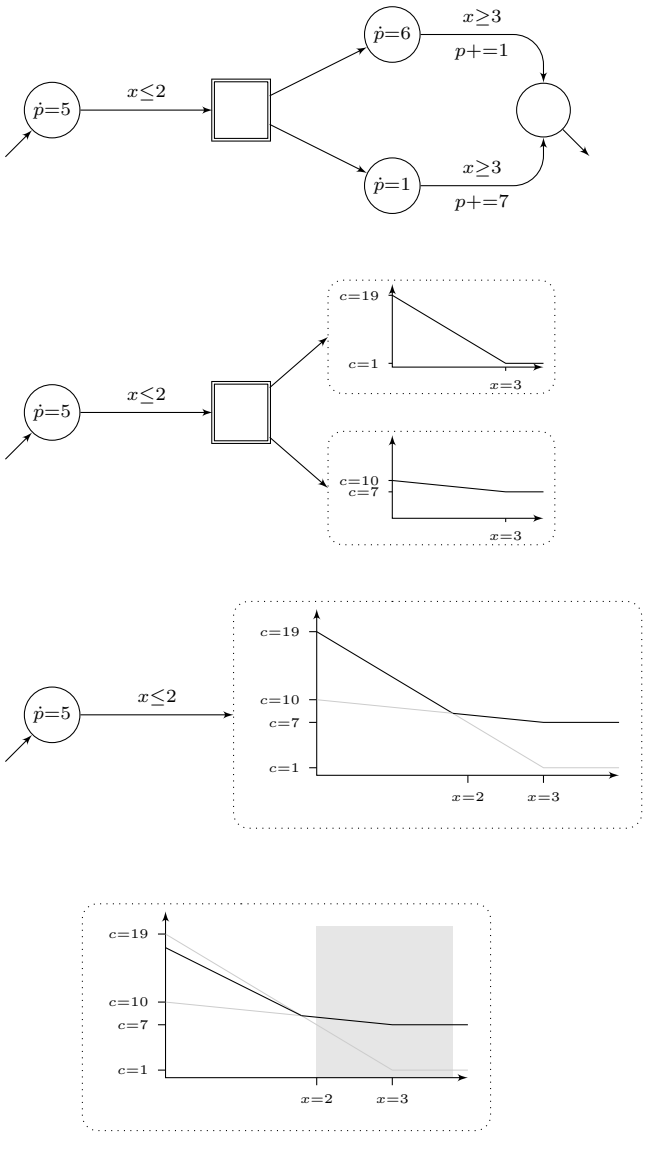


Figure 18: Solving the example of Fig. 15

- if there is one controllable location, then the result is easily obtained by considering $m\colon x \mapsto \min_{s\to s'} f(s')(x)$. The optimal cost is then

$$\mathsf{OptCost}(s,x) = \inf_{x\le x'\le 1} r_s(s)\cdot(x'-x) + m(x').$$

  An almost-optimal strategy is easily derived, which is composed of finitely many different moves (see Fig. 18 for an example);

- for the induction step, we consider one of the non-urgent locations $s_{\min}$ having least cost $r_s(s_{\min})$.

  – if it is controllable, then we can prove that there is no need to delay twice in $s_{\min}$: since it has the least cost in $\mathcal{G}$, it is more profitable to elapse in $s_{\min}$ as long as possible. We can then consider the game $\mathcal{G}'$ made of two copies of each location (except $s_{\min}$). We enter the second copy after visiting $s_{\min}$. Then both games are equivalent w.r.t. the optimal cost, and from an almost-optimal strategy in $\mathcal{G}'$, we can obtain an almost-optimal one in $\mathcal{G}$ preserving the nice properties (memorylessness, piecewise constant) of the original one.

  – if it is uncontrollable, we make it urgent, but give Player 2 an extra transition going to an outside cost function corresponding to the optimal cost of winning in $s_{\min}$, which is computed iteratively (from $x = 1$). □

## 4.3 Energy constraints

**Theorem 85.** *The winning-strategy-with-energy-constraint problem with upper and lower bounds is undecidable for one-clock weighted timed automata.*

*Proof.* This is achieved by simulating a two-counter machine. If $c_1$ and $c_2$ are the values of the counters, the accumulated weight along runs of the simulating weighted timed game will be

$$E(c_1,c_2) = 5 - \frac{1}{2^{c_1}3^{c_2}},$$

and we will work with an upper bound of 5. We start with accumulated weight 4 (encoding that the two counters are initialized to 0).

The instructions of the two-counter machine will be encoded as modules which we describe below. All our modules have invariant "$x \le 1$". For the purpose of this section, a *configuration* is a triple $(\ell, x, W)$ where $\ell$ is a discrete location, $x$ is the value of the clock, and $W$ is the accumulated weight.

**Module ok.** Our first module is the "accepting" (ok) module, which is depicted on Fig. 19. Clearly enough, entering module ok with accumulated weight in $[0,5]$ is winning.

A strategy for Player 1 which is defined on a module and has the property that any outcome either reaches a module ok or exits the module while always

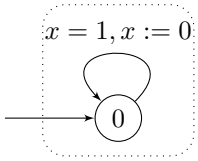


Figure 19: Module ok

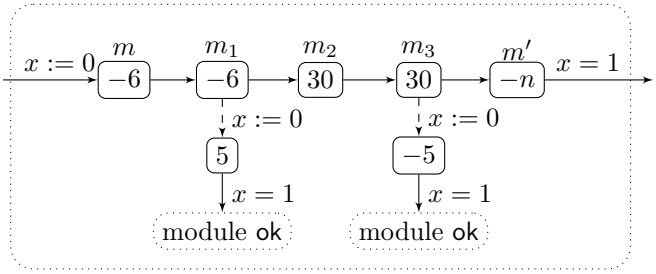


Figure 20: Generic module $\mathsf{Mod}_n$

satisfying the lower- and upper-bound conditions will be said to be *locally safe* in that module.

**Generic module $\mathsf{Mod}_n$** Consider the generic module $\mathsf{Mod}_n$, as depicted on Figure 20, for some given value of $n$ (which will be fixed later among $\{2, 3, 12, 18\}$), and assume that $0 \leq ne \leq 30$. Consider the following strategy $\sigma_n$, depicted on Fig. 21:

- from location $(m, 0, 5-e)$, it delays during $(5-e)/6$ time units in $m$, then leaves to $m_1$ (thus to configuration $(m_1, (5-e)/6, 0)$);
- from $m_1$, it directly goes to $m_2$;
- from $(m_2, (5-e)/6, 0)$, it waits for $1/6$ time units and then leaves to $m_3$ (hence to configuration $(m_3, 1-e/6, 5)$);
- from $(m_3, 1-e/6, 5)$, it directly goes to $m'$;
- in $m'$, it delays until $x = 1$, and fires the last transition to the next module.

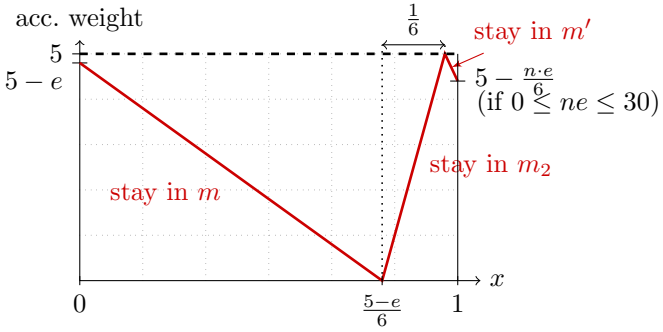


Figure 21: The effect of module $\mathsf{Mod}_n$

The only memory that is needed to define strategy $\sigma_n$ is the accumulated weight so far.

**Lemma 86.** *For $0 \leq ne \leq 30$, the strategy $\sigma_n$ is locally safe in $\mathsf{Mod}_n$ and has the following three outcomes, all of which adhere to the lower and upper bound conditions:*

- *one where Player 2 takes her move down from location $m_1$;*
- *one where Player 2 takes her move down from location $m_3$;*
- *one that leaves the module to the right.*

*Moreover, any other strategy for Player 1 is not locally safe.*

*Proof.* It is clear that $\sigma_n$ satisfies the properties above. To prove the last claim, we exhaustively list the possible other strategies:

- The maximal delay that can be waited in $m$ before violating the lower-bound constraint is $(5-e)/6$ time units. Assume that we wait strictly less than $(5-e)/6$ time units in $m$ before leaving to $m_1$; then the accumulated weight when reaching $m_1$ is positive, and the second player can immediately go down and increase it above 5.
- If the strategy waits for $(5-e)/6$ time units in $m$ before reaching $m_1$, then waiting any positive delay in $m_1$ will also take the accumulated weight above 5.
- The cases of $m_2$ and $m_3$ are similar to the cases of $m$ and $m_1$, respectively (but instead, the lower bound will be violated).
- Last, waiting until $x = 1$ is the only possible move in $m'$. □

We have shown the following:

**Lemma 87.** *Starting from configuration $(m, 0, 5-e)$ in $\mathsf{Mod}_n$, with $0 \leq ne \leq 30$, Player 1 has a unique locally safe strategy. Under that strategy, the only outcome that visits $m'$ exits the module with accumulated weight $5 - ne/6$.*

In the sequel, we consider the modules $\mathsf{Inc}(c_1) = \mathsf{Mod}_3$, $\mathsf{Inc}(c_2) = \mathsf{Mod}_2$, $\mathsf{Dec}(c_1) = \mathsf{Mod}_{12}$, and $\mathsf{Dec}(c_2) = \mathsf{Mod}_{18}$. Note that when starting with accumulated weight $5 - e$ in module $\mathsf{Inc}(c_1)$, the accumulated weight when leaving this module is $5 - e/2$, hence $\mathsf{Inc}(c_1)$ increments the counter $c_1$. Similarly, $\mathsf{Inc}(c_2)$ increments $c_2$, and $\mathsf{Dec}(c_1)$ and $\mathsf{Dec}(c_2)$ decrement the respective counters. In the last two modules, no check is performed whether the counter to be decremented actually has a positive value. We now describe modules that will test the values of the counters (to implement an instruction of the form "`if` $c_1 = 0$ `then goto` $p_1$ `else goto` $p_2$"), and for the global reduction we will assume that a decrementing instruction is always preceded by such an instruction.

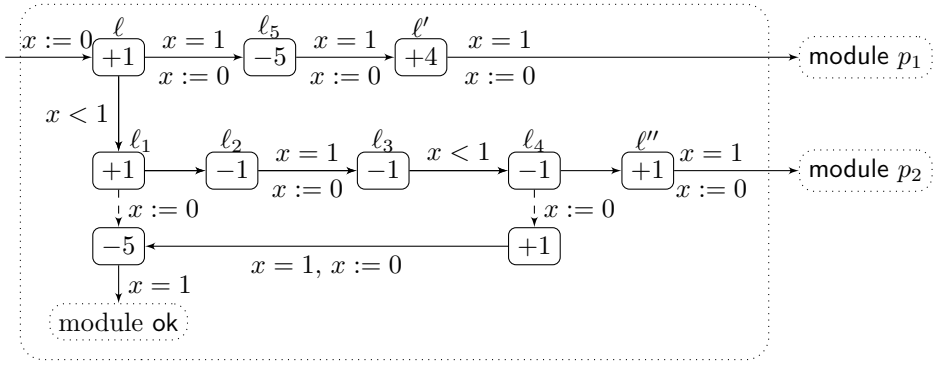


Figure 22: Module $\mathsf{Test}(c_1 = c_2 = 0)$

**Module for checking $c_1 = c_2 = 0$.** As a first step, we construct a module implementing the instruction "`if` $c_1 = c_2 = 0$ `then goto` $p_1$ `else goto` $p_2$" where $p_1$ and $p_2$ are instructions of the two-counter machine. This module is depicted on Fig. 22.

The following result is easily proved:

**Lemma 88.** *Starting from configuration $(\ell, 0, 5 - e)$ in module $\mathsf{Test}(c_1 = c_2 = 0)$, Player 1 has a locally safe strategy if and only if $e \in [0, 1]$. In that case, the strategy is unique and has three outcomes, two of which end in module* ok*, and*

- *if $e = 1$, the third outcome exits the module to module $p_1$ via configuration $(\ell', 1, 4)$;*
- *otherwise, the third outcome exits to $p_2$ via $(\ell'', 1, 5 - e)$.*

**Module for testing $c_1 = 0$.** In this module, Player 1 will have a winning strategy if, and only if, the value of $c_1$ is zero when entering the module (*i.e.*, the accumulated weight is $5 - 1/3^k$ for some $k \in \mathbb{N}$). This is achieved by repeatedly decrementing $c_2$ until $c_1 = c_2 = 0$, as implemented in the module depicted on Fig. 23. A similar module can be constructed for counter $c_2$. Note that in this module, and also in module $\mathsf{Test}(c_1 > 0)$, we might decrement a counter which already has value 0. We show in the proof of the next lemma that this is not a problem.

**Lemma 89.** *Starting from configuration $(\ell, 0, 5 - e)$ in module $\mathsf{Test}(c_1 = 0)$, Player 1 has a locally safe strategy in this module if and only if $e = 1/3^k$ for some $k \in \mathbb{N}$. In that case, the strategy is unique.*

*Proof.* The reverse direction of the equivalence is immediate, applying the strategies from Prop. 87 and 88.

Conversely, from Prop. 88, if $e > 1$ then Player 1 has no winning strategy. We now prove by induction that if $3^{-k} < e < 3^{1-k}$ for some $k \in \mathbb{N}$, then there is

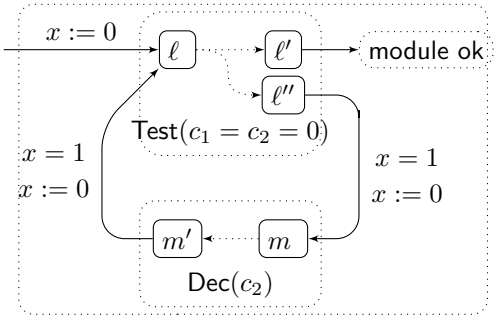


Figure 23: Module $\mathsf{Test}(c_1 = 0)$

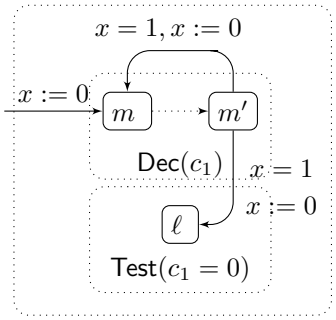


Figure 24: Module $\mathsf{Test}(c_1 > 0)$

no safe strategy. The base case ($k = 0$) has been handled above. Now, if $k > 0$, then from Prop. 88 there is only one safe strategy to exit submodule $\mathsf{Test}(c_1 = c_2 = 0)$, yielding an outcome entering module $\mathsf{Dec}(c_2)$ in configuration $(m, 0, 5 - e)$. Prop. 87 applies, and Player 1 has a unique strategy to safely exit that module. One of the outcomes then re-enters module $\mathsf{Test}(c_1 = c_2 = 0)$ in configuration $(\ell, 0, 5 - 3e)$. From the induction hypothesis, there is no safe strategy from that configuration, which concludes the proof. □

**Module for checking $\mathbf{c_1 > 0}$.** The module $\mathsf{Test}(c_1 > 0)$, depicted on Fig. 24, uses the previous one: it first lets Player 1 decrement counter $c_1$ a positive number of times, and then checks whether $c_1 = 0$ using the previous module. The following lemma, whose proof is similar to the previous one, expresses the correctness of this module:

**Lemma 90.** *Starting from configuration $(\ell, 0, 5 - e)$ in module $\mathsf{Test}(c_1 > 0)$, Player 1 has a locally safe strategy in this module if and only if $e = 1/(2^k \cdot 3^{k'})$ for some $k, k' \in \mathbb{N}$ with $k > 0$. In that case, the strategy is unique.*

**Modules for conditional instructions.** We now implement conditional instructions of the form "`if` $c_1 = 0$ `then goto` $p_1$ `else goto` $p_2$" of the two-counter machine. The corresponding module gives the choice to Player 1 to go either to $p_1$ or to $p_2$, and lets Player 2 check that the value of $c_1$ meets the requirement before proceeding to module $p_1$ or $p_2$. It is depicted on Fig. 25.

**Lemma 91.** *Starting from configuration $(t, 0, 5 - e)$ in module $\mathsf{Cond}(c_1 = 0)$, Player 1 has a locally safe strategy in this module if and only if $e = 1/(2^{k_1} \cdot 3^{k_2})$ for some $k_1, k_2 \in \mathbb{N}$. In that case, the strategy is unique, and*

- *if $k_1 = 0$, one of the outcomes of the strategy visits module $p_1$, and all the other ones eventually reach module ok;*
- *if $k_1 > 0$, one of the outcomes of the strategy visits module $p_2$, and all the other ones eventually reach module ok.*

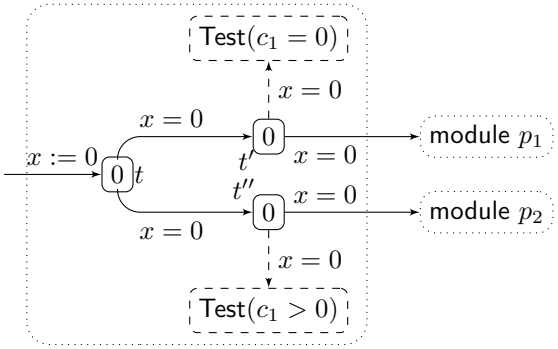


Figure 25: Module $\mathsf{Cond}(c_1 = 0)$

*Proof.* Letting time elapse in $t$ would block the game in that location. Thus, from $t$, Player 1 has only two possible strategies, visiting $t'$ or $t''$. If $e$ is not of the form $1/(2^{k_1}3^{k_2})$ with $k_1, k_2 \in \mathbb{N}$, then both strategies are losing since Player 2 will be able to enter one of the Test modules with that incorrect value.

If $e = 1/(2^{k_1}3^{k_2})$ with $k_1, k_2 \in \mathbb{N}$, then if $k_1 = 0$, the strategy visiting $t'$ is locally safe, while if $k_1 > 0$, the other strategy is locally safe. □

**Simulation of the two-counter machine.** Let $\mathcal{M}$ be a two-counter machine. We simulate an increment operation of the first (resp. second) counter by $\mathsf{Inc}(c_1)$ (resp. $\mathsf{Inc}(c_2)$). We simulate a decrement operation of the form "`if` $c_1 = 0$ `then goto` $p_i$ `else decrement` $c_1$`; goto` $p_j$" by plugging the corresponding modules on module $\mathsf{Cond}(c_1 = 0)$, and similarly for counter $c_2$. From the results above, we have the following lemma, entailing undecidability of the 1-clock interval-bound game problem.

**Lemma 92.** *Let $\mathcal{M}$ be a two-counter machine. We can construct a one-clock weighted timed game $G_{\mathcal{M}}$ which is a positive instance of the interval-bound problem if and only if $\mathcal{M}$ has an infinite computation.* □

# 5 Summary

Table 1 summarizes the results presented in this course.

# References

[ALP01] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovani-Vincentelli, editors, *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture*

| | | optimal reachability | mean-cost/discounted | temporal logics | energy constraints |
|---|---|---|---|---|---|
| untimed | model-checking | PTIME | PTIME | PTIME-c. ($\leq, \geq$), $\Delta_2^{\mathsf{P}}$-c. (general case) | PTIME-c. (lower constr.), NP/PSPACE (interval) |
| | games | PTIME-c. (costs $\geq 0$) | PTIME/NP $\cap$ coNP | PTIME-c. ($\leq, \geq$), EXPTIME-c. (general case) | PTIME/NP $\cap$ coNP (lower), EXPTIME-c. (interval) |
| timed | model-checking | PSPACE-c. | PSPACE-c. | undecidable ($\geq 3$ clocks), PSPACE-c. (1 clock) | PTIME (1 cl.) |
| | games | PTIME/3EXPTIME (1 cl.), undecid. ($\geq 3$ clocks) | | | undecidable (interval) |

Table 1: Summary of our results

*Notes in Computer Science*, pages 49–62. Springer-Verlag, March 2001.

[BBBR07] Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, October 2007.

[BBL04] Patricia Bouyer, Ed Brinksma, and Kim Gulstrand Larsen. Staying alive as cheaply as possible. In Rajeev Alur and George J. Pappas, editors, *Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control (HSCC'04)*, volume 2993 of *Lecture Notes in Computer Science*, pages 203–218. Springer-Verlag, March-April 2004.

[Bel58] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Gulstrand Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovani-Vincentelli, editors, *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer-Verlag, March 2001.

[BLM06] Patricia Bouyer, Kim Gulstrand Larsen, and Nicolas Markey. Model checking one-clock priced timed automata. In Helmut Seidl, editor, *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'07)*, volume 4423 of *Lecture Notes in Computer Science*, pages 108–122. Springer-Verlag, March 2006.

[BLMR06] Patricia Bouyer, Kim Gulstrand Larsen, Nicolas Markey, and Jacob Illum Rasmussen. Almost optimal strategies in pne-clock priced timed automata. In S. Arun-Kumar and Naveen Garg, editors, *Proceedings of the 26th Conferentce on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer-Verlag, December 2006.

[dAFH+05] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. Model checking discounted temporal properties. *Theoretical Computer Science*, 345(1):139–170, November 2005.

[DG07] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1-2):69–86, June 2007.

[FL08] Uli Fahrenberg and Kim Gulstrand Larsen. Discount-optimal infinite runs in proced timed automata. In Peter Habermehl and Tomáš Vojnar, editors, *Proceedings of the 10th International Workshop on Verification of Infinite-State Systems (INFINITY'08)*, Electronic Notes in Theoretical Computer Science. Elsevier Science, August 2008.

[GHR95] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, 1995.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979.

[JLS07] Marcin Jurdziński, François Laroussinie, and Jeremy Sproston. Model checking probabilistic timed automata with one or two clocks. In Orna Grumberg and Michael Huth, editors, *Proceedings of the 13th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lecture Notes in Computer Science*, pages 170–184. Springer-Verlag, March 2007.

[Kar78] Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, September 1978.

[LMS02] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. On model checking durational Kripke structures (extended abstract). In Mogens Nielsen and Uffe Engberg, editors, *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'02)*, volume 2303 of *Lecture Notes in Computer Science*, pages 264–279. Springer-Verlag, April 2002.

[Mar75] Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, September 1975.

[Min61] Marvin L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, November 1961.