Lower-Bound-Constrained Runs in Weighted Timed Automata

Patricia Bouyer[a], Kim G. Larsen[b], Nicolas Markey[a]

[a]*Lab. Spécification & Vérification, CNRS & ENS Cachan, France*
[b]*Dept. Computer Science, Aalborg University, Denmark*

Abstract

We investigate a number of problems related to infinite runs of weighted timed automata (with a single weight variable), subject to lower-bound constraints on the accumulated weight. Closing an open problem from [1], we show that the *existence* of an infinite lower-bound-constrained run is—for us somewhat unexpectedly—undecidable for weighted timed automata with four or more clocks.

This undecidability result assumes a fixed and known initial credit. We show that the related problem of *existence of an initial credit* for which there exists a feasible run is decidable in PSPACE. We also investigate the variant of these problems where only bounded-duration runs are considered, showing that this restriction makes our original problem decidable in NEXPTIME. We prove that the *universal* versions of all those problems (i.e, checking that *all* the considered runs satisfy the lower-bound constraint) are decidable in PSPACE.

Finally, we extend this study to multi-weighted timed automata: the existence of a feasible run becomes undecidable even for bounded duration, but the existence of initial credits remains decidable (in PSPACE).

1. Introduction

Weighted (or *priced*) *timed automata* [2, 3, 4] have emerged as a useful formalism for formulating a wide range of resource-allocation and optimization problems [5, 6], with applications in areas such as embedded systems [7]. In [1], a new class of resource-allocation problems was introduced, namely that of constructing infinite schedules subject to boundary constraints on the accumulation of resources.

More specifically, we proposed weighted timed automata with positive as well as negative weight-rates in locations, allowing for the modeling of systems where resources (e.g. energy) are not only consumed but also possibly produced. As a basic example, consider the two-clock weighted timed automaton $\mathcal{A}$ in Fig. 1 with infinite behaviours repeatedly delaying in ℓ_0, ℓ_1, ℓ_2 and ℓ_3 for a total duration of two time units, with one time unit spent in ℓ_0 and ℓ_3 and one time unit spend in ℓ_1 and ℓ_2 (we silently assume an invariant on all locations, imposing that clock y has to always remain below 2). The values (+2, +3 and +4) in the four locations indicate the rate by which energy is produced (or consumed, when negative), and the values (-2 and -3) on the edges indicate instantaneous updates to the energy level (there is only one weight variable in this example). Clearly, the energy remaining after a given iteration will depend not only on the initial energy but also highly on the distribution of the two time units over the four locations.

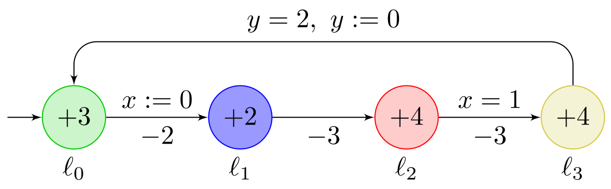


Fig. 1: A 2-clock weighted timed automaton $\mathcal{A}$ (with implicit global invariant $y \leq 2$)

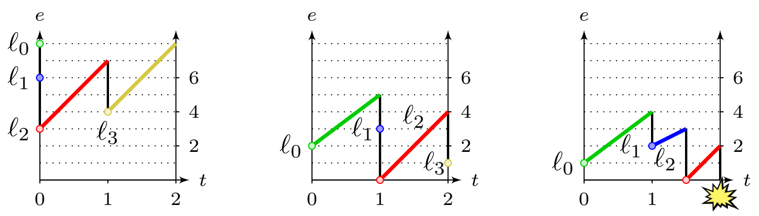


Fig. 2: Three possible behaviours (representing the evolution of the location and energy level with time) in $\mathcal{A}$ (with initial credits 8, 2 and 1, resp.)

In this paper we consider a number of problems related to infinite runs subject to lower-bound constraints on the accumulated weights (*e.g.* infinite runs where the energy level never goes below zero). In the absence of an upper bound and if there is only one weight variable, it suffices to consider runs along which the accumulated weight is maximized. Fig. 2 illustrates three such energy-maximizing behaviours of $\mathcal{A}$. For initial energy 8, the maximum energy left after one iteration is 8, thus providing an infinite lower-bound schedule. In contrast, an initial energy level of 1 does not even permit a single iteration (let alone an infinite schedule), and an initial energy level 2 leaves at maximum 1, for which we already know that no infinite lower-bound schedule exists. In this simple (non-branching) example, it can be shown that 1.5 is the least initial credit for which it is possible to come back to ℓ_0, and 5 is the minimal initial credit that allows an infinite-duration run.

For weighted timed automata with a single clock and a single weight variable, the existence of a lower-bound constrained infinite run has been shown decidable in polynomial time [1] with the restriction that no discrete updates of the accumulated weight occur on transitions. In [8], it is shown that the problem remains decidable if this restriction is lifted and even if the accumulated weight grows not only linearly but also exponentially. In contrast, the existence of *interval-constrained* infinite runs—where a simple energy-maximizing strategy does not suffice—have recently been proven undecidable for weighted timed automata with varying numbers of clocks and weight variables: *e.g.* two clocks and two weight variables [9], one clock and two weight variables [10], and two clocks and one weight variable [11]. Also, the interval-constrained problem is undecidable for weighted timed automata with one clock and one weight variable in the *game* setting [1].

Still, the general problem of *existence* of infinite lower-bounded runs for weighted timed automata has remained unsettled since [1]. In this paper we close this open problem showing that it is undecidable for weighted timed automata with four or more clocks and one weight variable. Given that this problem looks rather simple (since there is only one weight variable, it suffices to consider energy-maximizing runs), we find this result quite surprising and somewhat disappointing. Thus, we consider a number of related problems for which we show decidability and settle complexity. In particular, the undecidability result assumes a fixed and known initial energy level. We show that the related problem of *existence of an initial energy level* allowing an infinite lower-bound constrained run is decidable in PSPACE in the one-weight case. We also investigate the variant of these problems, where the lower-bound constraint is only imposed for a limited duration: for instance, for the weighted timed automaton in Fig. 1 and initial energy level of 4, we may want to settle the existence of a run along which the energy level remains non-negative during the first 4.7 time units, say. Note that the time-bounded paradigm has recently emerged as a pertinent restriction option for the verification of real-time systems [12] (in quite the same way as bounded model checking has been used for untimed systems [13]). We show that this restriction makes our original problem decidable and NEXPTIME-complete (assuming only one weight variable). Our result has to be compared with rectangular hybrid automata, for which time-bounded reachability has recently been shown decidable in EXPSPACE (no matching lower bound is provided, though), under the hypothesis that all rates are non-negative (if rates can be negative, the problem is undecidable) [14]. Our model of weighted timed automata is a special case of rectangular hybrid automata, in which all variables are clocks (rate 1) and one variable can have non-negative as well as negative rates. Therefore none of the two decidability results implies the other. We refer to Table 1 for a summary of the aforementioned results.

We also extend this study to multi-weighted timed automata, showing that the above decidability result for the existence of a time-bounded constrained run does not carry over to that multi-dimensional setting (our undecidability proof uses ten weight variables, but only one clock). Still, the existence of an initial credit in the time-bounded setting is proven to remain decidable.

Finally, we also consider the *universal* versions of all the above problems (i.e., checking that *all* the considered runs satisfy the lower-bound constraint), and prove that they all are decidable in PSPACE.

2. Definitions

2.1. Basic definitions

We write $\mathbb{R}_{\geq 0}$, $\mathbb{Q}_{\geq 0}$ and $\mathbb{N}$ respectively for the set of nonnegative reals, rationals and integers. We assume that X is a finite set of variables called *clocks*. A valuation v of the clocks is a mapping $X \to \mathbb{R}_{\geq 0}$. If v is a valuation and $t \in \mathbb{R}_{\geq 0}$, we write $v + t$ for the valuation which assigns $v(x) + t$ to clock x. If $R \subseteq X$, we write $v[R \to 0]$ for the valuation which assigns 0 to clocks in R and $v(x)$ to $x \in X \setminus R$. We write $\Phi(X)$ for the set of formulas (called clock constraints) defined by $\phi ::= \texttt{true} \mid \phi \wedge \phi \mid x \sim c$ with $x \in X$, $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$. The semantics of such formulas is given by sets of valuations and defined in a natural way.

We assume that W is a finite set of variables called *weights*. A valuation u of the weight variables is a mapping $W \to \mathbb{R}$. Given $u' \in \mathbb{R}^W$, we write $u + u'$ for the valuation mapping each $w \in W$ to $u(w) + u'(w)$. A valuation u is said *nonnegative* if $u(w) \geq 0$ for all $w \in W$.

A *multi-weighted timed automaton* (or *weighted timed automaton* in short) is a tuple $\mathcal{A} = \langle L, L_0, X, W, \mathsf{inv}, E, \mathsf{disc}, \mathsf{rate}\rangle$ consisting of a finite set L of locations, a finite set $L_0 \subseteq L$ of initial locations, a finite set X of clocks, a finite set W of weights, a location invariant mapping $\mathsf{inv}\colon L \to \Phi(X)$, a finite set $E \subseteq L \times \Phi(X) \times 2^X \times L$ of edges, and, for each weight variable w in W, functions $\mathsf{rate}_w\colon L \to \mathbb{Z}$ and $\mathsf{disc}_w\colon E \to \mathbb{Z}$, which respectively indicate how the weight variable w is to be updated when waiting in a location and when crossing an edge. In the sequel, for a location $\ell \in L$ and an edge $e \in E$, we write $\mathsf{rate}(\ell)$ and $\mathsf{disc}(e)$ for the vectors $(\mathsf{rate}_w(\ell))_{w \in W}$ and $(\mathsf{disc}_w(e))_{w \in W}$.

The semantics of a weighted timed automaton is defined as an infinite-state transition system $G_{\mathcal{A}} = \langle S, T\rangle$ with

- $S = \{(\ell, v, u) \in L \times (\mathbb{R}_{\geq 0})^X \times \mathbb{R}^W \mid v \models \mathsf{inv}(\ell)\}$,
- $T \subseteq S \times S$ contains two types of transitions:
  - delay transitions, which do not involve a change in the location:
  $$\{(\ell, v, u) \to (\ell, v', u') \mid \exists t \in \mathbb{R}_{\geq 0}.\ v' = v + t \text{ and } u' = u + \mathsf{rate}(\ell) \times t\}$$
  - action transitions:
  $$\{(\ell, v, u) \to (\ell', v', u') \mid \exists e = (\ell, g, R, \ell') \in E.\\ v \models g \text{ and } v' = v[R \to 0] \text{ and } u' = u + \mathsf{disc}(e)\}.$$

The above semantics is that of a timed automaton with extra weight variables, which evolve with rate $\mathsf{rate}(\ell)$ in location ℓ and in a discrete manner (following the disc function) when firing transitions. Notice that if $(\ell, v, u) \to (\ell', v', u') \in T$, then for all $\gamma \in \mathbb{R}^W$, there exists $\gamma' \in \mathbb{R}^W$ such that $(\ell, v, \gamma) \to (\ell', v', \gamma') \in T$. That is, the weight variables do not constrain the behaviour of the automaton. A (finite or infinite) run $(\ell_0, v_0, u_0) \to (\ell_1, v_1, u_1) \to \ldots$ of $G_{\mathcal{A}}$ will be called a *weighted run*, and we will sometimes use the underlying standard *timed run* $(\ell_0, v_0) \to (\ell_1, v_1) \to \ldots$, which forgets about the weight information. Since two consecutive delay transitions can be merged, we require that along any run, delay- and action transitions alternate (by possibly inserting zero-delay transitions between two consecutive action transitions). We define the *length* of a run as its number of action transitions. If (ℓ_0, v_0, u_0) is the first state of a run ϱ, u_0 is the *initial credit*; ϱ is said to be *initial* if $\ell_0 \in L_0$ and $v_0 = \mathbf{0}$, the valuation which assigns 0 to every clock. If ϱ is a timed run, then for every valuation u_0 of the variables in W, there is a unique corresponding weighted run with initial credit u_0.

2.2. The lower-bound constrained problems

2.2.1. The existential and universal L*-problems.*

Fix an initial credit[1] $u_0 \in \mathbb{Q}^W$. An infinite timed run $\varrho = (\ell_0, v_0) \to (\ell_1, v_1) \to (\ell_2, v_2) \ldots$ of $\mathcal{A}$ satisfies the *lower-bound constraint* $L(u_0)$ (which we write $\varrho \models L(u_0)$) if along the corresponding weighted run $(\ell_0, v_0, u_0) \to (\ell_1, v_1, u_1) \to (\ell_2, v_2, u_2) \ldots$ with initial credit u_0, u_i is nonnegative for every i. In that case we say that the run is *feasible* with initial credit u_0, or simply that the corresponding weighted run is feasible.

We say that $\mathcal{A} \models \exists_\infty L(u_0)$ (resp. $\mathcal{A} \models \forall_\infty L(u_0)$) whenever there exists an initial infinite run ϱ s.t. $\varrho \models L(u_0)$ (resp. for every initial infinite run ϱ with initial credit u_0, it holds $\varrho \models L(u_0)$). The first (resp. second) problem is called the *existential (resp. universal)* L*-problem*, and denoted with $\exists_\infty L(u_0)$ (resp. $\forall_\infty L(u_0)$).

2.2.2. The time-bounded L*-problems.*

Fix an initial credit $u_0 \in \mathbb{Q}^W$ and a time bound $T \in \mathbb{Q}_{\geq 0}$. A timed run $\varrho = (\ell_0, v_0) \to (\ell_1, v_1) \to (\ell_2, v_2) \ldots$ satisfies the *T-time-bounded* lower-bound constraint $L(u_0)$ (which we write $\varrho \models_T L(u_0)$) whenever the following holds[2]:

- the total duration of ϱ is at most T, and can be strictly less than T only if ϱ is infinite. We let p_0 be the number of transitions along ϱ;
- if $(\ell_0, v_0, u_0) \to (\ell_1, v_1, u_1) \to (\ell_2, v_2, u_2) \ldots$ is the weighted run corresponding to ϱ with initial credit u_0, then for every $i \leq p_0$, u_i is nonnegative.

We then say that $\mathcal{A} \models \exists_T L(u_0)$ (resp. $\mathcal{A} \models \forall_T L(u_0)$) whenever there exists an initial run ϱ such that $\varrho \models_T L(u_0)$ (resp. for all initial finite runs of duration T and for all initial infinite run of duration at most T, it holds $\varrho \models_T L(u_0)$). The first (resp. second) problem is called the *existential (resp. universal) time-bounded* L*-problem* . In short we write $\exists_T L(u_0)$ (resp. $\forall_T L(u_0)$).

[1]We restrict to rationals from this point on, as considering irrational weights would lead to undecidability, as is the case with irrational guards [15].

[2]Various similar definitions could be considered instead of this one, for instance requiring that the duration is always (at least) T. Such variants could be handled by our techniques with minor amendments.

2.2.3. Existence of an initial credit.

The above four problems assume a fixed and known initial credit u_0. We are interested also in the existence (and synthesis) of an initial credit for which the previous problems can be answered positively. Formally, for $Q \in \{\exists, \forall\}$ and $\alpha \in \mathbb{Q}_{\geq 0} \cup \{\infty\}$, we write $\mathcal{A} \models \exists u_0.Q_\alpha L(u_0)$ whenever there exists an initial credit $u_0 \in \mathbb{Q}^W$ such that $\mathcal{A} \models Q_\alpha L(u_0)$. In short we denote this problem by $\exists u.\ Q_\alpha L(u)$.

2.3. Summary of our results

	fixed initial credit	existence of initial credit
∞	$\leq 1c$, $= 1w$: in EXPTIME [8] $\geq 4c$, $\geq 1w$: undecidable $\geq 1c$, $\geq 4w$: undecidable [10]	$= 1w$: in PSPACE $\geq 3c$, $= 1w$: PSPACE-c.
T	$= 1w$: in NEXPTIME $\geq 5c$, $= 1w$: NEXPTIME-c. $\geq 1c$, $\geq 10w$: undecidable	in PSPACE $\geq 3c$: PSPACE-c.

Table 1: Summary of our results (where e.g. "$\geq 3c$" refers to automata with at least three clocks, and "$= 1w$" concerns automata with exactly one weight variable.

In this paper we solve several of the various above-mentioned problems. We prove that the problem of the existence of a feasible run is undecidable in general, with the notable exception of time-bounded feasible runs in the presence of a single weight variable.

On the other hand, the problem of the existence of an initial credit for which a feasible run exists is shown decidable (in polynomial space) in most cases (we were not able to solve the case of infinite-duration runs for multi-weighted timed automata). Our results (for the case $Q = \exists$) are summarized in Table 1 (the previously known results are displayed in gray). We distinguish the time-bounded (written T) and time-unbounded (written ∞) cases. In this table, constraints such as $\geq 3c$ (resp. $\geq 1w$) refer to the number of clocks (resp. weight variables) used in our proof. When unspecified, we mean that the result holds for arbitrarily many clocks (or weights). When $Q = \forall$, we prove that all problems can be solved in polynomial space, and are PSPACE-complete as soon as the automaton has at least three clocks.

3. Undecidability of $\mathcal{A} \models \exists_\infty L(u_0)$ with one weight variable

We first prove the undecidability of the existential L-problem. While its proof is not of the most difficult, the result is quite surprising (at least to us) as the problem looks rather simple (it amounts to checking that, by maximizing the accumulated weight, we can keep it non-negative).

Theorem 1. *The existential* L*-problem is undecidable for the class of weighted timed automata with at least four clocks and one weight variable (and rates in* $\{0, 1\}$*).*

We only give a proof sketch here, which is simple to understand but requires five clocks. Another proof can be obtained by using the encoding of the hardness proof in Section 5.2.

Sketch of proof. Consider the automata depicted on Fig. 3. Writing x_0 for the value of clock x when entering ℓ_0, the effect of these automata is to add x_0 (resp. $1 - x_0$) to the weight, while preserving the value of all the clocks (provided that these values are in $[0, 1)$). Using these modules, it is then easy to enforce linear constraints between clocks: Fig. 4 is an example, in which each box is a copy of one of the automaton of Fig. 3 (after possibly exchanging the role of x and y). Discrete values between boxes represent a discrete increase or decrease of the weight variable. It is easily checked that for any run in this module with initial credit 0 and clock values x_0 and y_0 for clocks x and y, the final weight is 0 and the final values of the clocks are unchanged (assuming they are in $[0, 1)$). Moreover, such a run exists if, and only if, $y_0 \leq 2x_0$. Using such modules, we can encode reachability in a four-clock timed automaton with additive constraints (which is proven undecidable in [16]) as a reachability in a five-clock single-weight timed automaton (with z as the extra clock, used in the modules of Fig. 3). This reduction uses five clocks, but the above automata for checking additive constraints can also be used to encode Turing machines, using only four clocks (cf. Section 5.2). □

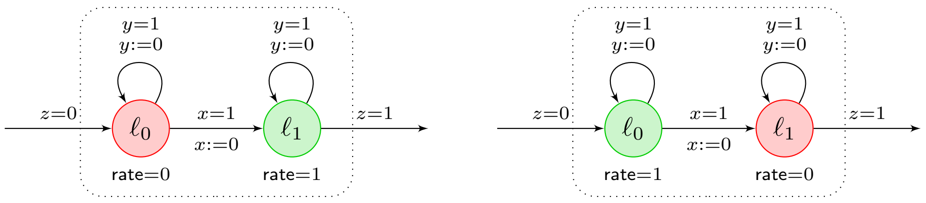


Fig. 3: Automata for crediting x_0 and $1 - x_0$

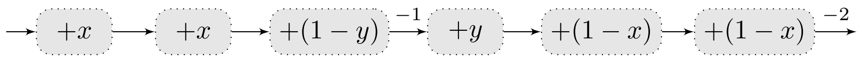


Fig. 4: Automaton for checking $y \leq 2x$

4. Undecidability of $\mathcal{A} \models \exists_T L(u_0)$ with several weight variables

In this section, we prove that when several weight variables are used, the L-problem is already undecidable when considering time-bounded paths.

Theorem 2. *The time-bounded* L*-problem is undecidable with ten weight variables and one clock.*

Proof. We follow the idea of the proof of [14] for the undecidability of the time-bounded reachability problem in rectangular hybrid automata.

We encode the behaviour of a two-counter machine as a time-bounded execution in a multi-weighted timed automaton. The classical idea to do so is to have weight variables store the values of the counters at time t, and to update those values until time $t+1$, according to the next instruction of the two-counter machine. In order to achieve such a reduction within bounded time, the timestamps will occur earlier and earlier, in such a way that time will converge. Namely, the n-th step of the run of the two-counter machine will be encoded at time $1-1/4^n$. Counters a_1 and a_2 of the two-counter machine will be encoded using two weight variables b_1 and b_2, with $b_i = 4^{-(n+a_i)}$ at the n-th step.

This way, decrementing a counter requires keeping the corresponding weight unchanged; incrementing the counter requires dividing the weight by 16, and leaving it unchanged corresponds to dividing the weight by 4. These updates will have to be performed in time $3/4^{n+1}$ between the n-th and $n+1$-st configurations. We also have to generate our special "clock" producing ticks at dates $1-1/4^n$. We now explain how we perform these computations.

A generic gadget. We first give a generic gadget, which we use several times in the reduction. This module divides the value of some weight variable, say c_1, by k^2, where $k \geq 2$ is a given integer. As we explain below, the computation is made quickly enough to fit in our reduction.

To alleviate the presentation, we first consider a slightly different problem, where both a lower-bound and an upper-bound are imposed on the weight variables: more precisely, we require that the values of the weight variables always remain between 0 and 1 during the computation. The reduction to the L-problem is then easy [10], as it suffices to associate with each weight c a weight c' whose value is kept equal to $1-c$. Then requiring that both c and c' are nonnegative amounts to require that the value of c lies in $[0,1]$. This in turn can be used to ensure that the value of c is precisely zero, using two consecutive discrete updates $+1$ and -1.

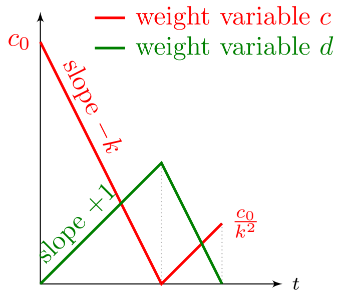


Fig. 5: Dividing c_0 by k^2

We now define our module. It uses two weight variables c and d, whose behaviour is depicted on Fig. 5. The idea is as follows: in a first phase c has rate $-k$, until it reaches zero (after c_0/k time units). At that time, d has value c_0/k, and then decreases with rate $-k$. This lasts c_0/k^2 time units, so that c (having rate 1 in this second phase) equals c_0/k^2 when $d=0$. Thus c has been divided by k^2, and the duration of this computation is $c_0\cdot(k^{-1}+k^{-2})$. The gadget achieving this computation is depicted on Fig. 6. We name it $\mathsf{Mod}(c/k^2, d, z)$ to indicate that it divides c by k^2 and involves one extra weight variable d and one clock z. If the above gadget is entered with values $c = \alpha \in [0,1]$ and $d=0$, then

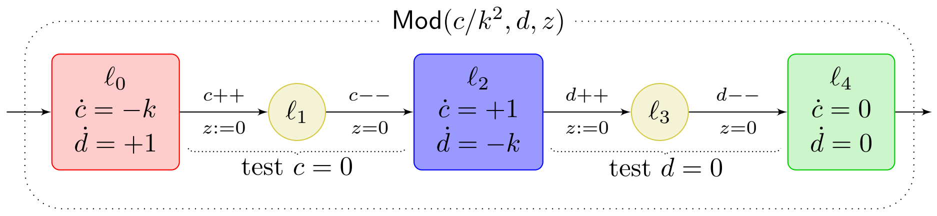


Fig. 6: Module for dividing by k^2

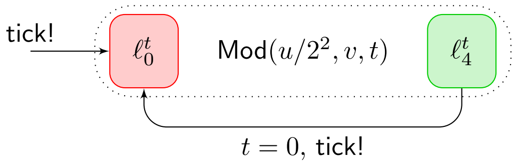


Fig. 7: Module echoing ticks at time $1 - 1/4^n$

there is a unique run traversing that module under the L-constraint, and the gadget is escaped with $c = \alpha/k^2$ and $d = 0$. The time spent before entering ℓ_4 is $\alpha \cdot (1/k + 1/k^2)$.

Generating the sequence of timestamps. We now build a gadget echoing ticks at dates $1 - 1/4^n$, for all n. These ticks are used to synchronise two or several modules: a transition labelled with tick? can only take place if a transition tick! is available (and taken) at the same time. The resulting behaviour can be captured by a timed automaton without ticks.

The tick module involves a weight variable u, which has value $1/4^n$ when echoing the n-th tick (*i.e.*, at time $1 - 1/4^n$). Hence it has to be divided by 4 between two consecutive ticks. This is achieved by using Mod$(u/2^2, v, t)$, which at step n runs in time $1/4^n \cdot (1/2 + 1/2^2)$, *i.e.*, $3/4^{n+1}$, which is precisely the duration between the n-th and $n+1$-st ticks. The corresponding module is depicted at Fig. 7. Assume the gadget is entered at time $t_n = 1 - 1/4^n$ with $u = 1/4^n$ and $v = 0$. As explained above, location ℓ_4^t is entered after $3/4^{n+1}$ time units (with clock t being zero). It is left immediately, echoing a tick, so that the module is re-entered at time $t_{n+1} = 1 - 1/4^{n+1}$, with $u = 1/4^{n+1}$ and $v = 0$.

Encoding the instructions. Let p be an instruction. We first assume that p decrements counter a_i, and then goes to instruction q. This is the easiest case to handle, as we have to keep the weights unchanged. The module of Fig. 8 achieves this. Notice that there are no timing constraints for entering and exiting this module, but instead it has to synchronise on the ticks.

Now assume that instruction p increments counter a_i before moving to q. We need then a gadget that, when entered with weight value $\frac{1}{4^{n+\alpha}}$ (for every n and for every α) is exited with weight value $\frac{1}{4^{n+1+\alpha+1}}$, which corresponds to dividing the weight by 16. This is achieved by using module Mod$(b_i/4^2, c_i, z_i)$,

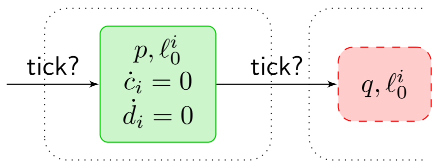


Fig. 8: Module for decrementing

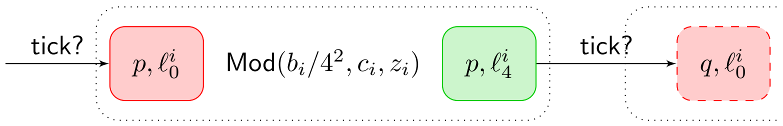


Fig. 9: Module for incrementing

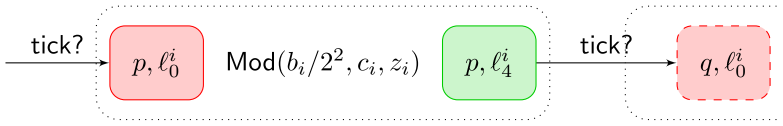


Fig. 10: Module keeping a counter unchanged

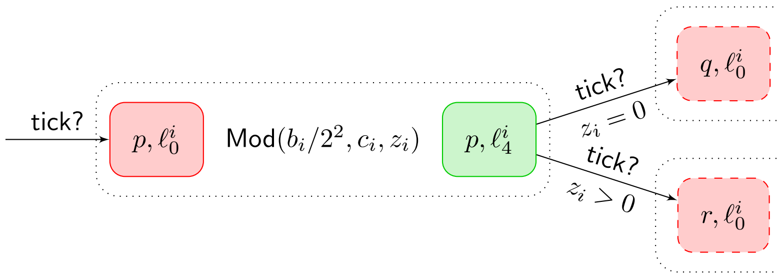


Fig. 11: Module for zero-tests

in which b_i is the counter we use to encode counter a_i and c_i is an auxiliary weight associated with counter a_i. The gadget is then mostly $\mathsf{Mod}(b_i/4^2, c_i, z_i)$, decorated with ticks on its incoming and outgoing edges, as displayed on Fig. 9. One can check that the computation is over before the exiting tick is emitted.

At each step, one of the counters is unchanged, which is encoded by dividing the value of its associated weight variable by 4. This is achieved using the module of Fig. 10. Again, one can check that we have enough time to perform this computation before the next tick: the time taken to divide $b_i = 1/4^{n+a_i}$ by 2^2 is $(1/4^{n+a_i}) \cdot (3/4)$, which is less than $3/4^{n+1}$.

Finally, assume instruction p is a zero-test, going to instruction q if counter a_i is zero and to r otherwise. Our module will divide by 2^2 (as we want to keep counters unchanged). As we just explained above, this takes time $(1/4^{n+a_i}) \cdot (3/4)$. As a consequence, we can synchronise on tick immediately if, and only if, $a_i = 0$. Hence our module is as depicted on Fig. 11.

Global reduction. Each instruction p of the two-counter machine requires updating both weight variables, due to our encoding. We thus have to consider the asynchronous product of the two modules updating weights b_1 and b_2. This in turn has to run in parallel with our special timer echoing ticks, with synchronisation on those signals.

It remains to plug the resulting modules together, according to the instructions of the two-counter machine. Finally, we drop edges going out of states corresponding to the halting state of the two-counter machine, and replace them with a self-loop decrementing one of the counters by one. This way, the halting state of the two-counter machine is reached if, and only if, there is no feasible run of duration less than one in our multi-weighted timed automaton.

To conclude, notice that our reduction makes use of three clocks and six weight variables: indeed, each counter requires one weight variable for storing its value, and one additional weight variable and one clock for performing the instructions. Then the automaton generating the ticks also involves two more weight variables and one clock. However, our modules use lower- and upper-bound constraints; using only L-constraints, we have to duplicate each weight variable, hence using twelve weights.

Remark 3. *The global reduction could be achieved slightly differently, by updating the weights encoding the counters one after the other. This requires adapting the encoding in the following ways:*

- *make the* timer *alternatively issue two kinds of ticks, namely tick_1 and tick_2;*
- *change the encoding of the counters: counter a_1 would be encoded with weight value $1/4^{2n+a_1}$, while a_2 would be encoded as $1/4^{2n+1+a_2}$. Updating the values of the counters would then amount to dividing the weights by* 4*,* 16 *or* 64*, which we can easily achieve.*

This way, the modules managing both counters can share their auxiliary weight variable and their auxiliary clock, thus reducing to ten weight variables and two clocks.

In order to spare one more clock, we notice that the clock t used for the generation of the ticks might be omitted, since it is only used to enforce urgency*, i.e., that no time elapses in some of the states. This can be enforced using weight variables,* e.g. *by assuming rate $+1$ for u in location ℓ_1^t, and rate $+1$ for v in location ℓ_3^t, and rate -1 for v in ℓ_4^t. Thanks to the constraints on the weights, this will enforce delay zero in those locations, without using clock t. The global reduction then uses ten weight variables and one clock.* □

5. Decidability of $\mathcal{A} \models \exists_T L(u_0)$ with one weight variable

In this section, we show the decidability of the existential problem, when we restrict to the time-bounded setting and a single weight variable. We also characterize the exact complexity of this problem. We assume basic knowledge about *regions*, a classical technique to prove various results in timed automata [17].

Theorem 4. *The existential time-bounded* L*-problem is* $\mathsf{NEXPTIME}$*-complete for weighted timed automata involving five clocks or more and with one weight variable.*

5.1. Upper bound

We fix a weighted timed automaton $\mathcal{A}$ with a single weight variable, and we assume that it has an extra clock u that is never reset. We fix a time bound $T \in \mathbb{Q}_{\geq 0}$ and an initial credit $c_0 \in \mathbb{Q}_{\geq 0}$ for the unique weight variable.

From a run witnessing that $\mathcal{A} \models \exists_T L(c_0)$, our approach consists in building an exponential-size witness. The new witness may have the same duration as the original one, or it may reach a configuration from which there is a feasible zero-delay cycle with non-negative effect on the accumulated weight. We detail the proof for the case where the automaton only has non-strict clock constraints, and assume that the initial witnessing run is finite (Sections 5.1.1 to 5.1.5). We then explain how to handle infinite runs in Section 5.1.6. Handling strict guards is much more technical, and is presented in the technical appendix Appendix A.

5.1.1. Zero-delay cycles.

A zero-delay cycle on state (ℓ, v) is a path starting and ending in (ℓ, v) in which all delay transitions are zero-delay. In particular, such a cycle cannot visit a resetting transition, unless the reset clock already has value 0 in v.

Our first task is to detect those configurations from which a zero-delay cycle is feasible. For a state (ℓ, v) and a non-negative real c, we set the Boolean predicate $\mathsf{profit}((\ell, v), c)$ to true iff there is a zero-delay cycle from (ℓ, v, c) back to (ℓ, v, c') for some $c' \geq c$, along which the lower-bound constraint is satisfied. Such a cycle is called a profitable zero-delay cycle. Note that $\mathsf{profit}((\ell, v), c) = \mathsf{profit}((\ell, v'), c)$ whenever v and v' belong to the same region r (because no time elapses), so that the region-based predicate $\mathsf{profit}((\ell, r), c)$ is well-defined (in the obvious way). Moreover, detecting zero-delay cycles can be achieved efficiently:

Lemma 5. *Given* (ℓ, r)*, we can compute in polynomial time the smallest* c *such that* $\mathsf{profit}((\ell, r), c)$ *holds, if any. Furthermore it is a natural number.*

Proof. It suffices to consider the subgraph of the region automaton where only region r appears. This is a weighted graph whose size is at most $|L|$, the number of locations of the automaton. Using a modified Bellman-Ford algorithm [1], we can compute the least c for which $\mathsf{profit}((\ell, r), c)$ holds, if any. This algorithm runs in polynomial time, and returns a natural number (because discrete weights are integers). $\square$

We use this predicate to abstract weighted runs: from a state (ℓ, v, c) satisfying $\mathsf{profit}((\ell, v), c)$, there is an infinite run (whose total duration is finite, a.k.a. a Zeno run) satisfying the lower-bound constraint[3]. Furthermore as we shall

[3]Notice that, using similar techniques, we could easily handle different variants of our time-bounded problem, for instance if it is required that the witness run must have duration

prove in Section 5.1.6, any (Zeno) infinite witness will visit a state satisfying $\mathsf{profit}((\ell, v), c)$. Hence we have reduced our problem to that of finding a finite witness, either with duration T or with duration less than T but ending in a state which carries a profitable zero-delay cycle.

5.1.2. Reduction scheme.

We now claim and explain our main technical lemma. Our NEXPTIME-membership result, as stated in Theorem 4, follows from this lemma, as the algorithm simlpy consists in nondeterministically constructing the path step-by-step.

Lemma 6. *If $\mathcal{A}$ has a feasible run ϱ from some (ℓ_0, v_0, c_0) to some (ℓ, v, c) of duration T, then $\mathcal{A}$ has also a feasible run ϱ' of length N in*[4] *$O(T \cdot |X|^3 \cdot |L|^2)$, starting from (ℓ_0, v_0, c_0) and ending in (ℓ', v', c') such that:*

- *either $(\ell', v') = (\ell, v)$ and $c' \geq c$, and $v'(u) = v(u)$;*
- *or from (ℓ', v', c') there is a profitable zero-delay cycle.*

In order to bound the length of the witness run, we split it in segments in which the integral part of clock u is constant. We then show that such segments can be made short enough, while still satisfying the lower-bound constraint and possibly improving the final accumulated weight.

For the rest of the proof we assume we are given a finite weighted run $\varrho\colon (\ell_0, v_0, c_0) \xrightarrow{\delta(t_0)} (\ell_0, v_0 + t_0, c'_0) \xrightarrow{e_1} (\ell_1, v_1, c_1) \xrightarrow{\delta(t_1)} (\ell_1, v_1 + t_1, c'_1) \ldots$ witnessing the fact that $\mathcal{A} \models \exists_T L(c_0)$.

The $[i..j]$-segment of ϱ, denoted $\varrho[i..j]$, is defined as $(\ell_i, v_i, c_i) \to \cdots \to (\ell_j, v_j, c_j)$. Segment $\varrho[i..j]$ is said *flexible* whenever for every clock $x \in X \setminus \{u\}$, there exist an integer $0 \leq d_x \leq M$ (where M is the maximal constant of $\mathcal{A}$) and an index $i < h_x \leq j+1$ such that

- clock x is not reset along $\varrho[i..h_x - 1]$;
- for every $i \leq k < h_x$, it holds $d_x \leq v_k(x) < d_x + 1$ (we assume $M + 1 = \infty$);
- and for every $h_x \leq k \leq j$, it holds $0 \leq v_k(x) \leq 1$.

The index h_x corresponds to the first transition where x is reset. Roughly, a flexible segment can be split into at most $2|X|+1$ parts (with the first and last resets (if any) of each clock as the boundaries). In each part, we can "move" the delays (while preserving the total duration of each part), allowing us to make this witness shorter while preserving the initial and final clock valuations and augmenting the value of the weight variable.

at least T: when a state (ℓ, v, c) with $\mathsf{profit}((\ell, v), c)$ is visited, the weight variable can be made arbitrarily large, and the lower-bound constraint is lifted. It just remains to decide the existence of a run along which a sufficient amount of time elapses, with no weight constraint.

[4]Precisely, $N = \big(1 + (|X|+1) \cdot (1 + (|L|+1) \cdot |L| \cdot (|X|+1)^2)\big) \cdot (\lfloor T \rfloor + 1)$, where $\lfloor T \rfloor$ is the integral part of T.

The following lemma states that a segment of duration less than 1 can be decomposed into a small number of flexible subsegments.

Lemma 7. *Let $\varrho[i..j]$ be a segment of duration less than 1. Then $\varrho[i..j]$ can be split into at most $|X|+1$ many flexible segments.*

Proof. Since $\varrho[i..j]$ has duration less than 1, there are at most $|X|$ many delay transitions along which some clock reaches the upper bound of the unit interval to which it belongs (that is, changes integral part with no reset operation). This defines at most $|X|+1$ segments, which clearly are flexible. $\square$

Consider a flexible segment $\varrho[i..j] = (\ell_i, v_i, c_i) \xrightarrow{\delta(t_i)} \dots \xrightarrow{e_j} (\ell_j, v_j, c_j)$ along which clock u has constant integral part. We define, for every $i < h \leq j$, $\delta_h = \sum_{k=i}^{h-1} t_k$. For every clock x that is reset along $\varrho[i..j]$ we write m_x (resp. n_x) for the index of the first (resp. last) edge along which x is reset. The run $\varrho[i..j]$ is depicted on Fig. 12.

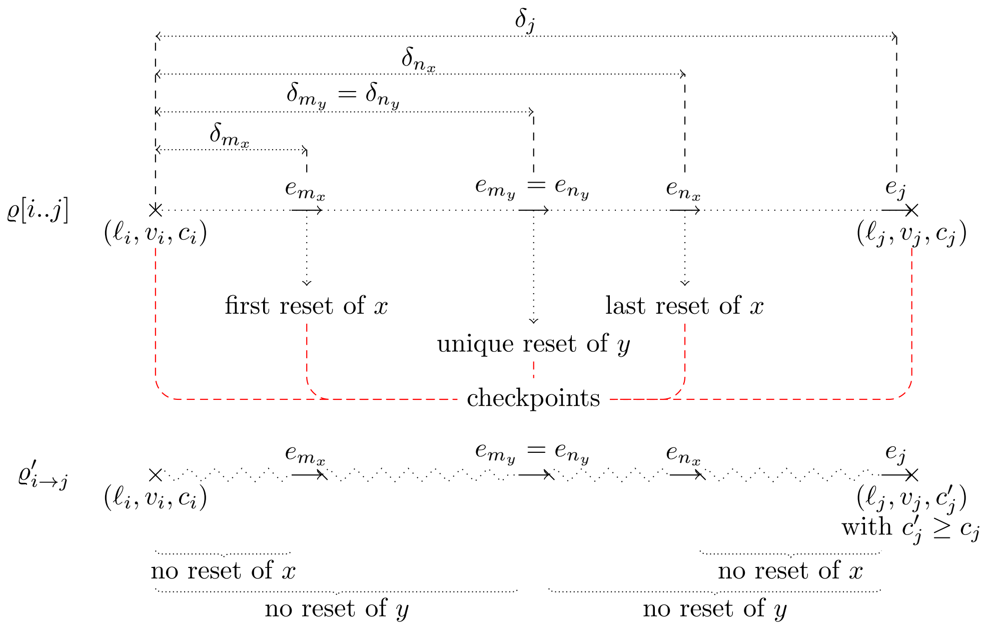


Fig. 12: A flexible segment

We transform $\varrho[i..j]$ into a shorter run $\varrho'_{i\to j}$ that starts in (ℓ_i, v_i, c_i) and fires edges e_j (resp. e_{m_x}, e_{n_x}) after exactly δ_j (resp. δ_{m_x}, δ_{n_x}) time units. Furthermore that run will not reset clock x before firing e_{m_x} after δ_{m_x} time units, and it will not reset clock x after having fired e_{n_x} after δ_{n_x} time units (see Figure 12). This enforces that the clock constraints are satisfied along the new run, and that the final state of ϱ' is the same as that of ϱ. Edges e_{m_x},

e_{n_x} (for $x \in X$) and e_j act as checkpoints. Between two checkpoints we will "optimize" the accumulated weight (which will ensure that $c'_j \geq c_j$) and shorten the path (which will prove the existence of a short witness).

5.1.3. Transformation of a segment between two checkpoints.

Let us first discuss how the fractional parts of the clocks compare to that of the universal clock u. A valuation v is said Y*-small* (for some $Y \subseteq X$) whenever for every $y \in Y$, it holds $0 \leq \mathsf{frac}(v(y)) \leq \mathsf{frac}(v(u))$ (where $\mathsf{frac}(q)$ is the fractional part of q). Notice that for any $t < 1 - \mathsf{frac}(v(u))$, any valuation reached from v within t time units is Y-small.

We now focus on one segment between two checkpoints. It is characterized by a constraint described by a tuple $\gamma = ((\ell, v, c), \alpha, e, B, R, c')$ where:

(a) (ℓ, v, c) is the initial state of that segment;

(b) the duration $\alpha \in \mathbb{R}_{>0}$ of that segment, assuming that $\mathsf{frac}(v(u)) + \alpha < 1$;

(c) the final edge e of that segment;

(d) the set $B \subseteq X$ of clocks that should not be reset along that segment, except possibly on the last edge e; we require that v is $(X \setminus B)$-small;

(e) the set $R \subseteq X$ of clocks that should have value 0 at the end of the segment (when firing e);

(f) a lower bound $c' \in \mathbb{R}_{\geq 0}$ for the accumulated weight at the end of the segment.

We write $R_\gamma(\mathcal{A})$ for the set of runs satisfying constraint γ (defined in an obvious way). Intuitively, if we replace a segment of a feasible run with another segment satisfying the same constraint γ, then the newly constructed run is still feasible.

The following lemma shows that any constraint γ can be fulfilled by a short segment.

Lemma 8. *Let $K = (|L| + 1) \cdot (|L| \cdot (|X| + 1))$. The following two properties are equivalent:*

(i) *there is a finite run ϱ in $R_\gamma(\mathcal{A})$;*

(ii) *there is a run of length at most K that either is in $R_\gamma(\mathcal{A})$ or reaches a state (ℓ', v', c') from which there is a profitable zero-delay cycle.*

Roughly, the idea is to postpone any delay in a location with negative rate to the latest appearance of that location along the run, and to transfer all delays in a location with positive rate to its first appearance along the run. This transformed run is in $R_\gamma(\mathcal{A})$, delays in at most $|L|$ locations, and still satisfies the lower-bound constraint on the rate. Then we can remove the zero-delay cycles (unless they are profitable).

Proof of Lemma 8. Pick a finite run ϱ in $R_\gamma(\mathcal{A})$, and modify it as follows: for every location ℓ_1 in which a positive delay is spent along ϱ, do the following:

- if $\mathsf{rate}(\ell_1) \geq 0$, then transfer all delays spent in such a location to the first occurrence of ℓ_1 along ϱ where some positive delay is already spent;
- if $\mathsf{rate}(\ell_1) < 0$, transfer all delays spent in such a location to the last occurrence of ℓ_1 where some positive delay is already spent.

We can do so since none of the guards along ϱ is violated because of this change: indeed, if $x \in B$, from the first positive delay on, x is in $(d_x; d_x + 1)$ all along ϱ (as ϱ is flexible and x is not reset along ϱ). If $x \notin B$, then v is $\{x\}$-small. Assume that some transition has a guard $x = c$ (with $c > 0$) along ϱ. Then no delay is spent before this transition, or no delay is spent after. Hence there is no transfer of delay from one side of this transition to the other, so that the guard is still fulfilled in the new run. Similarly, if there is a guard $x = 0$ in the original run, then no time is elapsed since the previous reset of x (or since the beginning if x is zero at the beginning), and the guard is still fulfilled in the new run. Finally, if there is a guard $x \in [c; c + 1]$, then transferring delays to the beginning of the run (before a reset of x) may increase the value of x, but as $v(x) < v(u)$ and $\mathsf{frac}(v(u)) < 1$ all along ϱ, then $x \in [c; c + 1]$ is still fulfilled along the new run. Transferring delay after a reset of x would still keep $x \in [0; 1]$.

The modified run delays in at most $|L|$ locations, the lower-bound constraint is still satisfied (because the accumulated weight in any location of the new run is at least as high as in the original run).

We name ϱ' the new run, which is a new witness for (i). Consider a subrun of ϱ' that is taken in zero-delay: if a state (ℓ_1, v_1) is visited twice along that subrun and if the corresponding cycle has a non-positive effect, then we can remove that part of the run, the overall weight will be larger and we still have a witness for (i); if such a cycle has a positive effect, then $\mathsf{profit}((\ell_1, v_1), c_1)$ is true (c_1 is the credit at the first visit), and the rest of the run can be dropped.

The resulting run delays in at most $|L|$ locations, and is acyclic in the zero-delay segments. In a zero-delay segment, once a clock is reset, it remains equal to 0: therefore, the length of such an acyclic zero-delay segment is at most $|L| \cdot (|X| + 1)$. The length of the resulting run is thus bounded by $(|L| + 1) \cdot (|L| \cdot (|X| + 1))$. This run is a witness for (ii). $\square$

We now formalize an equivalence between constraints.

Lemma 9. *Assume we are given a constraint $\gamma = ((\ell, v, c), \alpha, e, B, R, c')$. Fix $Y \subseteq X$ such that v is Y-small, and fix another Y-small valuation $\widehat{v}$ that agrees with v on $X \setminus Y$, and s.t. for every $x \in Y$, $v(x) = 0$ implies $\widehat{v}(x) = 0$. We define the constraint $\widehat{\gamma}$ as $((\ell, \widehat{v}, c), \alpha, e, B, R, c')$. Then:*

1. *If $R_\gamma(\mathcal{A}) \neq \emptyset$, then $R_{\widehat{\gamma}}(\mathcal{A}) \neq \emptyset$. Furthermore, if there is a run in $R_\gamma(\mathcal{A})$ with final accumulated weight e, then there is one in $R_{\widehat{\gamma}}(\mathcal{A})$ with the same final accumulated weight.*
2. *If $\varrho \in R_\gamma(\mathcal{A})$ and $\widehat{\varrho} \in R_{\widehat{\gamma}}(\mathcal{A})$, writing v' (resp. $\widehat{v}'$) for the valuation at the*

end of ϱ (resp. $\widehat{\varrho}$), we have that:

$$\begin{cases} \forall x \in (X \setminus Y) \cap B,\ v'(x) = \widehat{v}'(x) \\ \forall x \in R,\ v'(x) = \widehat{v}'(x) = 0 \\ v' \text{ and } \widehat{v}' \text{ are } ((X \setminus B) \cup Y)\text{-small} \end{cases}$$

Proof. Pick a flexible run ϱ' in $R_\gamma(\mathcal{A})$. We explain why this run can be mimicked (with the same delay- and action transitions) from $(\ell, \widehat{v}, c)$, yielding a flexible run in $R_{\widehat{\gamma}}(\mathcal{A})$. Write $\widehat{\varrho'}$ for the run (we prove below that all guards are fulfilled, so that this is really a run in $\mathcal{A}$) obtained by mimicking ϱ' from $(\ell, \widehat{v}, c)$. First, all valuations along ϱ' are Y-small, since $v(u) = \widehat{v}(u)$ and both ϱ' and $\widehat{\varrho'}$ have duration α. Furthermore, for $y \in Y$, if $v(y) = 0$ then $\widehat{v}(y) = 0$. Therefore for clocks in Y, all constraints that are satisfied along ϱ' are satisfied along $\widehat{\varrho'}$. For clocks not in Y, v and $\widehat{v}$ agree, so that all conditions are fulfilled for being a flexible run in $R_{\widehat{\gamma}}(\mathcal{A})$.

We now prove the second point. Along both ϱ and $\widehat{\varrho}$, clocks in B are not reset (except possibly for those also in R). Therefore as v and $\widehat{v}$ initially agree on $X \setminus Y$, we get the two first lines. The third point is straightforward. □

5.1.4. *Flexible segments.*

We come back to our flexible segments $\varrho[i..j] = (\ell_i, v_i, c_i) \xrightarrow{\delta(t_i)} \dots \xrightarrow{e_j} (\ell_j, v_j, c_j)$, and consider the checkpoints defined earlier. We write $I = \{i, j\} \cup \{m_x, n_x \mid x \in X\}$ for the indices corresponding to checkpoints, and we assume that it is ordered as $I = \{i_0 < i_1 < \cdots < i_p\}$. For each $0 \le h < p$, we consider the constraint

$$\gamma_h = ((\ell_{i_h}, v_{i_h}, c_{i_h}), \delta_{i_{h+1}} - \delta_{i_h}, e_{i_{h+1}}, B_h, R_h, c_{i_{h+1}})$$

where $B_h = \{x \in X \mid i_h < m_x \text{ or } i_h \ge n_x\}$ and $R_h = \{x \in X \mid v_{i_{h+1}}(x) = 0\}$. The set B_h is the set of clocks that should not be reset (except possibly on the last edge). We first argue why this is a proper constraint. Condition (b) is by assumption on clock u whereas condition (d) requires some arguments that we give now. Pick $y \notin B_h$: it means that $m_y \le i_h < n_y$. We have that $v_{i_h}(y) \le \delta_{i_h} - \delta_{m_y}$ since y was reset at edge m_y (and also possibly later). In the meantime $\mathsf{frac}(v_{i_h}(u)) = \mathsf{frac}(v(u)) + \delta_{i_h} \ge \delta_{i_h}$. Therefore $0 \le v_{i_h}(y) \le \mathsf{frac}(v_{i_h}(u))$, which implies condition (d).

We build a run ϱ solving the constrained problem γ_0, then γ'_1, then γ'_2, etc. Constraints γ'_h will only differ from γ_h in the initial valuation (which will still satisfy the hypotheses of Lemma 9). We proceed by induction on h. First note that for every h, v_{i_h} is $\left(\bigcup_{k<h}(X \setminus B_k)\right)$-small. Assume that v'_{i_h} is related to v_{i_h} as in the hypotheses of Lemma 9 (with $Y_h = \bigcup_{k<h}(X \setminus B_k)$). We build a run in $R_{\gamma'_h}(\mathcal{A})$ (γ'_h is same as γ_h except for the initial valuation which is v'_{i_h} instead of v_{i_h}). The run $\varrho[i_h..i_{h+1}]$ is in $R_{\gamma_h}(\mathcal{A})$. Applying the first part of Lemma 9 we get that $R_{\gamma'_h}(\mathcal{A})$ is non-empty. Applying Lemma 8, we select a run $\varrho'_{i_h \to i_{h+1}}$ of length bounded by K in $R_{\gamma'_h}$ (or ending in a state carrying a profitable zero-delay

cycle, which concludes the proof). Applying the second part of Lemma 9, we get that the final valuation of $\varrho'_{i_h \to i_{h+1}}$, say $v'_{i_{h+1}}$, is $((X \setminus B_h) \cup Y_h)$-small. By construction, for all clocks x such that $v_{i_{h+1}}(x) = 0$ (this is R_h), $v'_{i_{h+1}}(x) = 0$. Valuation $v'_{i_{h+1}}$ therefore satisfies the hypotheses of Lemma 9.

Initially we assume $v_0 = v'_0$. We have thus constructed by induction runs $\varrho'_{i_h \to i_{h+1}}$ that can be glued together, yielding a run $\varrho'_{i \to j}$ of duration δ. The final state of $\varrho'_{i \to j}$ is (ℓ_j, v_j, c'_j) (by construction), and $c'_j \geq c_j$. Furthermore the length of $\varrho'_{i \to j}$ is at most $(2|X| + 1) \cdot K$. We have proved:

Proposition 10. *Take a flexible segment $\varrho[i..j]$ of ϱ, such that clock u has constant integral part along $\varrho[i..j]$. We can construct a feasible run $\varrho'_{i \to j}$ from (ℓ_i, v_i, c_i) and of length at most $(2|X| + 1) \cdot K$, such that either it ends in a configuration from which a profitable zero-delay cycle can be taken, or the following holds:*

- *its duration is that of $\varrho[i..j]$;*
- *if (ℓ_j, v'_j, c'_j) is its final configuration, then $v'_j = v_j$, and $c'_j \geq c_j$.*

5.1.5. Complexity.

The NEXPTIME upper bound is obtained by guessing a path (that is, a sequence of transitions) of length at most N in $\mathcal{A}$ and then checking by some linear programming resolution whether there exists a witness along that path which satisfies the lower-bound constraint and the time constraint. This is in NEXPTIME (assuming that T is given in binary).

5.1.6. Case where the witness ϱ is infinite.

We now assume that $\varrho = (\ell_0, v_0, c_0) \to \dots (\ell_n, v_n, c_n) \dots$ is an infinite witness of total duration no more than T. We will show that there is a finite run which satisfies the lower-bound constraint $L(c_0)$ and which ends in a state from which it is possible to follow a zero-delay cycle while satisfying the lower-bound constraint and whose accumulated weight is non-negative (we can define a predicate $\mathsf{profit}^{\geq 0}$ similar to profit, but where the accumulated weight is non-negative instead of positive).

There exists n such that the tail $\varrho[n..\infty)$ of the witness is flexible (and clock u lies within one time unit): indeed, since ϱ has finite duration, there is a position after which the integral parts of all the clocks are constant. For all locations ℓ_1 with $\mathsf{rate}(\ell_1) > 0$ in which some delay is spent along $\varrho[n..\infty)$, we transfer all delays in location ℓ_1 to its first occurrence where some delay is spent. We remove all delays spent in some location ℓ_2 with $\mathsf{rate}(\ell_2) \leq 0$. Since all clocks have constant integral part along the tail, the clock constraints are preserved, and this yields a new witness ϱ', such that $\varrho'_{m \to +\infty}$ is zero-delay, for some m: we can extract from that tail a zero-delay cycle which satisfies the lower-bound constraint and has a non-negative accumulated weight.

5.2. Lower bound

In this section, we prove the lower bounds of Theorem 4. We reuse the modules defined in the proof of Theorem 1 for checking linear constraints between clocks. These modules can be used to check constraints of the form $\alpha x + \beta y + \gamma \geq 0$, for integers α, β and γ. If we allow integer rates (instead of only 0 and 1), we can assume the total time spent to check such a linear constraint is constant (2 time units).

5.2.1. Encoding a rule of a Turing machine.

We encode a (non-deterministic) Turing machine over alphabet $\{0, 1\}$ as follows. Assume the tape content is $w_1(q, a)w_2$ where $w_1, w_2 \in \{0, 1\}^*$, q is a state of the Turing machine and $a \in \{0, 1\}$ is the content of the cell which is being read. We encode this with a location (q, a) and with the values of two clocks x_1 and x_2, whose binary representation will be $\mathsf{enc}(x_1) = 0.\overline{w_1}$ (we write $\overline{w_1}$ for the word obtained from w_1 by reading from right to left) and $\mathsf{enc}(x_2) = 0.w_2$.

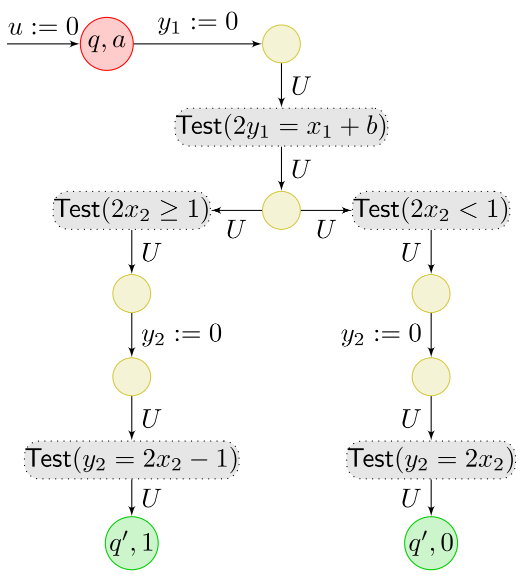
We assume that $(b, \rightarrow, q') \in \delta(q, a)$ is a possible transition from (q, a) in the Turing machine. In order to be able to mimic that transition, we want to go to state (q', c) with two clocks[5] y_1 and y_2 such that $\mathsf{enc}(y_1) = 0.b \cdot \overline{w_1}$ and $\mathsf{enc}(y_2) = 0.w_2'$ where $w_2 = c \cdot w_2'$. The symbol $c \in \{0, 1\}$ is the content of the new cell which is pointed. In terms of value, we have to enforce:

$$2y_1 = x_1 + b \qquad \text{and} \qquad y_2 = 2x_2 - c.$$

The module is depicted on Figure 13. We write U on a transition instead of $u = 1, u := 0$. On every location, for every clock x that is not already used in an outgoing transition, there is a self-loop $x = 1, x := 0$, which is omitted on the figure (for readability). The module works as follows:

- in the first part (until module $\mathsf{Test}(2y_1 = x_1 + b)$), the automaton non-deterministically resets clock y_1, and checks that the new value has the property that after exactly two time units (i.e. after the second U-transition), it holds $2y_1 = x_1 + b$.
- then, non-deterministically again, the automaton jumps to one of the two branches:
  - the left branch corresponds to the case where $c = 1$: this is checked by the module $\mathsf{Test}(2x_2 \geq 1)$. We then have to set y_2 accordingly, which is achieved by non-deterministically resetting y_2 and checking that $y_2 = 2x_2 - 1$.
  - the right branch is for the case where $c = 0$: the encoding is similar to that used in the upper branch, adapted to the fact that $c = 0$. Notice that we have to test a strict inequality, namely $2x_2 < 1$. This is

[5]For the sake of readability, our reduction involves six clocks. However, clock y_2 needs not to be fresh, and we will be able to use x_1 instead since it is not used later in the module.



$u := 0$
q, a
$y_1 := 0$
U
$\mathsf{Test}(2y_1 = x_1 + b)$
U
$\mathsf{Test}(2x_2 \geq 1)$
U
U
$\mathsf{Test}(2x_2 < 1)$
U
U
$y_2 := 0$
$y_2 := 0$
U
U
$\mathsf{Test}(y_2 = 2x_2 - 1)$
$\mathsf{Test}(y_2 = 2x_2)$
U
U
$q', 1$
$q', 0$

Fig. 13: Mimicking instruction $(\beta, \rightarrow, q') \in \delta(q, \alpha)$ (self-loops omitted)

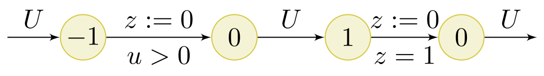


Fig. 14: Automaton for testing that the energy level is positive

achieved by crediting the variable with $1 - 2x_2$ and checking that this value is positive. This in turn can be tested by the module depicted on Figure 14: this module checks positiveness of the weight variable while preserving the values of the weight and of all clocks, except possibly the auxiliary clock z.[6]

We assume that the total time needed to traverse this module is the same along both branches (even if it means adding some more states). We write d_1 for that constant.

5.2.2. The global reduction.

We fix a non-deterministic Turing machine $\mathcal{M}$ and an input w of length n. We use the encoding of instructions of a non-deterministic Turing machine as in the previous section, and we glue all modules together. We add an initialization module which leads to (q_0, w_1), set x_1 to 0 and x_2 to $\sum_{i=2}^{n} w_i 2^{-(i-1)}$; this can easily be achieved using module $\mathsf{Test}(2^{n-1}x_2 - \sum_{i=2}^{n} w_i 2^{n-i} = 0)$, in total

[6]Similarly as for y_2, clock z needs not be fresh, and it can be replaced by clock x_1, which is not re-used later in the module.

time $d_2 = n-1$ (using rates 2^{n-1}, 2^{n-2}). We parametrize $\mathcal{A}$ with an integer K, yielding $\mathcal{A}[K]$. In $\mathcal{A}[K]$, a new clock t is reset when reaching the initial state of $\mathcal{M}$ (after the initialization phase), and is used in an invariant $t \leq d_1 K$ in all locations (except the initialization phase and the halting location). Furthermore from every location (except locations of the initialization phase and the halting location) we can go to a sink location with discrete weight -4 when $t \geq d_1 K$ (we call these transitions the bad transitions). Notice that the maximal value of the accumulated weight within a module while mimicking the instructions of $\mathcal{M}$ never exceeds 3: hence, if this transition to the sink state is taken, then the lower-bound constraint will be violated. The halting location has no outgoing transition, and we can wait there as long as we want, with rate zero.

Our reduction involves seven clocks: four clocks are used for simulating the instructions of the Turing machine, and three auxiliary clocks u, z and t are used in the construction. However, as already mentioned, in both branches below $\mathsf{Test}(2y_1 = x_1 + b)$, clock x_1 is not used anymore and can be used to play the role of z (in the test for strict inequalities) and of y_2.

Proposition 11. *$\mathcal{M}$ halts on input w with a computation of at most K steps iff $\mathcal{A}[K] \models \exists_{(d_1 K + d_2)+1} L(0)$.*

Proof. Assume $\mathcal{M}$ has a halting computation of length k (with $k \leq K$) on w. Then in $\mathcal{A}$, we can first safely initialize the two clocks x_1 and x_2, and reach the initial state of the Turing machine. The initialization phase takes d_2 time units. Then the instructions of the computation can also be safely mimicked, and each instruction takes d_1 time units. Hence globally the execution takes then $d_2 + d_1 k$ time units before reaching the halting location, and we can wait there enough time to meet the requirement.

Assume that $\mathcal{A}[K] \models \exists_{(d_1 K + d_2)+1} L(0)$. Then it means that we have first safely initialized the clocks, and then mimicked properly instructions. And then as we have not violated the lower-bound constraint, it means that we have reached a halting location at the latest when $t = pK$ (since otherwise we should have taken the bad transition to the sink state, violating the lower-bound constraint). □

If K is exponential (in n), the binary encoding of K, and therefore the binary encoding of $d_1 K + d_2$, is polynomial. The above reduction implies $\mathsf{NEXPTIME}$-hardness for the existential time-bounded L-problem when the time bound is given in binary.

Remark 12. *Notice that if we lift the time bound (hence we don't need clock t), we encode a (generic) non-deterministic Turing machine. This provides another proof of Theorem 1, which uses only four clocks.*

6. The universal L-problems

In this section we show that the universal L-problems are much easier to solve than the existential L-problems. Interestingly, this is also the case when considering several weight functions.

Theorem 13. *The universal and the universal time-bounded* L*-problems are* PSPACE*-complete, even when considering several weight variables.*

The PSPACE lower bounds for single-weight timed automata are straightforward reductions from the reachability problem in timed automata: if there is a run reaching the final location, then there is a run of duration at most $T \overset{\text{def}}{=} |\mathcal{R}(\mathcal{A})|$, where $\mathcal{R}(\mathcal{A})$ is the region automaton of $\mathcal{A}$ (see [17]). Therefore by giving a negative rate to accepting locations and rate zero to other locations, and setting all discrete updates to zero, the non-reachability problem is equivalent to the universal L-problem, be it time-bounded (by T) or not.

The PSPACE upper bound can be proven using the corner-point abstraction $\mathcal{R}_{cp}(\mathcal{A})$ of $\mathcal{A}$; the corner-point automaton is a weighted finite automaton that refines the region abstraction. Intuitively, a *corner point* is a pair (r, v) made of a region and of one of its *extremal points* (notice that v does not necessarily belong to r, but it belongs to its closure). These form the state space of the corner-point abstraction. Transitions are computed in the natural way: there is a "delay" transition from (r, v) to $(r, v + 1)$ when both v and $v + 1$ belong to the closure of r, and from (r, v) to (r', v) when v belongs to the closures of both r and r', and r' is the immediate time successor of r. The former kind of transition corresponds to elapsing almost one time unit in a region, and thus is decorated with the rate (or vector of rates) of the corresponding location. The latter kind of transitions have duration almost zero, and thus come with no weight. Finally, action transitions are computed in the natural way. We refer to [18] for more details on the corner-point abstraction. The result then relies on the following lemma.

Lemma 14. $\mathcal{A} \models \forall_\infty L(w_0)$ *if, and only if,* $\mathcal{R}_{cp}(\mathcal{A}) \models \forall_\infty L(w_0)$.

Proof. Assume that there is an infinite run $\varrho = (\ell_0, v_0, w_0) \to (\ell_1, v_1, w_1) \ldots$ which does not satisfy the lower-bound constraint for one of the weight variables, say for variable c. Take a prefix of ϱ, say $\varrho[0..n]$, which violates the lower-bound constraint: there is $0 \le i \le n$ such that $w_i(c) < 0$. Prop. 3 of [18] (rephrased in our setting) states that there is a finite path π in $\mathcal{R}_{cp}(\mathcal{A})$ with the weight being smaller: hence this path violates the constraint as well.

Conversely take an infinite path π in $\mathcal{R}_{cp}(\mathcal{A})$ which violates the lower-bound constraint for weight variable c, and take a violating prefix. Then Prop. 6 of [18] states that for every $\varepsilon > 0$, there exists a real path whose weight is ε-close to that of π, hence becomes negative (when ε is taken small enough). □

Lemma 15. $\mathcal{R}_{cp}(\mathcal{A}) \not\models \forall_\infty L(w_0)$ *if, and only if, one of the following conditions is satisfied:*

(1) *there is a reachable cycle with a negative effect for some weight variable* c*;*

(2) *there is an acyclic path from the initial state, which can be extended into an infinite path, that yields a negative value for some weight variable* c*.*

Proof. Take a path which violates the lower-bound constraint. If it contains a cycle with negative effect, then (1) holds. Otherwise consider a prefix which violates the lower-bound constraint. We can remove from that prefix all cycles (since they have non-negative effect), and we still get a counter-example, which furthermore satisfies (2). $\square$

The above conditions can be checked in PSPACE (the size of $\mathcal{R}_{cp}(\mathcal{A})$ is exponential, and a path in $\mathcal{R}_{cp}(\mathcal{A})$ can be guessed using polynomial space).

This algorithm can be adapted to handle the time-bounded problem, by adding a clock u, which is never reset, but is used in an invariant $u \leq T$ on every location. From every location we add a transition constrained by $u = T$ leading to a sink location where the weight remains constant. This yields an automaton $\mathcal{B}$ such that

$$\mathcal{A} \models \forall_T L(w_0) \;\Leftrightarrow\; \mathcal{B} \models \forall_\infty L(w_0).$$

We apply the previous algorithm to $\mathcal{B}$. The size of $\mathcal{R}(\mathcal{B})$ is exponential in the size of $\mathcal{A}$ and T. This yields a PSPACE algorithm for deciding $\mathcal{A} \models \forall_\infty L(c_0)$.

7. When the initial credit is not known

We prove here that if the initial credit is not known, then most of the problems can be solved in PSPACE. More precisely we prove the following theorem.

Theorem 16. *It is possible to decide the existence of an initial credit for which*

- *a time-bounded feasible run exists in a multiple-weight timed automaton;*
- *a feasible run exists in a single-weight timed automaton;*
- *all runs are feasible in a multiple-weight timed automaton.*

We first focus on the upper bounds, and then turn to the lower bounds.

7.1. PSPACE *upper bound in the time-bounded case*

In order to prove the upper bounds for the time-bounded problems, the idea is to reduce the size of possible witnesses using a construction similar to that in Lemma 6, which allows to bound the weight variation along a possible witness and therefore gives information on how much the initial credit has to be for satisfying the lower-bound constraint along the witness. We first focus on the universal problem. The result relies on the following characterisation, which can be checked in polynomial space.

Lemma 17. *Let $\mathcal{A}$ be a weighted timed automaton. $\mathcal{A} \not\models \exists w.\forall_T L(w)$ iff there is a finite initial timed run of duration at most T, which ends in some configuration (ℓ, v) s.t. from (ℓ, v), there is a zero-delay cycle with negative accumulated weight for one of the variables.*

Proof. For every weight variable c of $\mathcal{A}$, we write $\mathcal{B}_c$ for the weighted timed automaton corresponding to $\mathcal{A}$ with unique weight variable c. Then, it is obvious that $\mathcal{A} \models \exists w.\forall_T L(w)$ iff for every weight variable c, $\mathcal{B}_c \models \exists w_c.\forall_T L(w_c)$. It is therefore sufficient to prove the above statement under the assumption that $\mathcal{A}$ has a single weight variable.

The right-to-left implication is obvious.

Assume $\mathcal{A} \not\models \exists w.\forall_T L(w)$, fix a timed run ϱ of duration at most T (those will be the only possible witnesses contradicting $\forall_T L(w)$). We transform ϱ as in the proof of Lemma 6 but we minimize the weight instead of maximizing it (and we do not require the lower-bound on the weight be satisfied). This yields an alternative finite run ϱ' which satisfies the following conditions (N is the uniform bound of Lemma 6):

- its duration is at most T;
- its length is at most N;
- at any point in time, its accumulated weight is smaller than that of ϱ;
- additionally,
  - (i) either it reaches a state (ℓ, v), from which there is a zero-delay cycle with a negative accumulated weight,
  - (ii) or its duration is that of ϱ.

Assume we are in case (ii). If we start with initial credit 0, the accumulated weight never goes below $-(N+T)\cdot R$ where R is the maximal absolute value for a rate or a weight decorating a location or an edge of the automaton. Therefore by setting $w_0 = (N+T)\cdot R$, we get that $\varrho' \models L(w_0)$. By construction, ϱ' has smaller accumulated weight than ϱ at any point in time, which implies $\varrho \models L(w_0)$.

Assume towards a contradiction that case (i) never happens. This means that for all runs ϱ of duration at most T, $\varrho \models L(w_0)$. This contradicts the assumption that $\mathcal{A} \not\models \exists w.\forall_T L(w)$. Therefore there is a run ϱ such that its corresponding ϱ' satisfies (i). □

Then, we handle the case of the existential problem. The two conditions of the lemma below can be checked in polynomial space.

Lemma 18. *$\mathcal{A} \models \exists w.\exists_T L(w)$ iff there is a finite initial run ϱ such that:*

(i) *either its duration is T;*

(ii) *or its duration is no more than T, and it ends in configuration (ℓ, v) s.t. from (ℓ, v), there is a zero-delay cycle which is almost-profitable*[7] *for every weight variable.*

[7]Almost-profitable is the same as profitable, except that we require non-negativity instead of positivity.

Proof. Assume $\mathcal{A} \models \exists w.\exists_T L(w)$, and take a witness run ϱ. If ϱ is finite, then (i) holds. If ϱ is infinite, we apply a construction similar to the proof of Section 5.1.6, except that we remove all delays in all locations once we are in the last time-unit of the witness. We compensate the loss due to the removal of (small) delays by increasing the initial values of the weight variables. We get a finite run which satisfies the condition (ii) (the cycle needs not be simple).

Assume that either (i) or (ii) holds. If (i) holds, then we can choose a large enough initial credit to compensate any decrease in the weight along ϱ: if the length of ϱ is ℓ, then if we start with initial credit 0 for weight variable c, the value of c never goes below $-(\ell + T) \cdot R_c$, where R_c is the maximal absolute value for a rate or a weight decorating a location or an edge of $\mathcal{A}$ for variable c. Therefore by setting $w_0(c) = (\ell + T) \cdot R_c$ for every weight variable c, we get that $\varrho \models L(w_0)$. If (ii) holds, then we easily get an infinite witness for the property. The initial credit will only be used to compensate any loss along the finite run leading to the cycle. Therefore $\mathcal{A} \models \exists w.\exists_T L(w)$. □

7.2. PSPACE *upper bound in the time-unbounded case*

In this section, if $\mathcal{A}$ is a weighted timed automaton, we write $\mathcal{R}_{cp}(\mathcal{A})$ for its corner-point abstraction (see [18]).

We first focus on the problem $\exists w.\exists_\infty L(w)$. The result only holds for weighted timed automata with a single weight variable. The following lemma immediately yields the PSPACE upper bound.

Lemma 19. *Assume $\mathcal{A}$ has a single weight variable. Then, $\mathcal{A} \models \exists w.\exists_\infty L(w)$ iff $\mathcal{R}_{cp}(\mathcal{A}) \models \exists w.\exists_\infty L(w)$.*

Proof. Let ϱ be a witness for $\mathcal{A} \models \exists w.\exists_\infty L(w)$. We project ϱ on $\mathcal{R}_{cp}(\mathcal{A})$, yielding an infinite weighted tree. For each index i there is a branch π'_i of length i of the tree, with overall weight better than that of $\varrho[0..i]$ ([18, Prop. 6]). Since the tree is finitely branching, applying König's lemma, there is an infinite branch π of the tree such that π'_i coincide with $\pi[0..i]$. Let $i_1, \dots, i_n, \dots$ the sequence of such positions i's. W.l.o.g. we assume that the accumulated weight along $(\pi[0..i])$ for $i \to \infty$ converges, say to γ (which might be infinite). If γ is finite, this is a stationary sequence (since $\mathcal{R}_{cp}(\mathcal{A})$ is a weighted finite automaton with integral values), and we can therefore find a cycle with overall weight zero: this yields a witness for the property (we can then easily compute a bound for the initial credit). If $\gamma = \infty$, then we can find an increasing subsequence, and we therefore exhibit a cycle with positive weight, yielding a witness.

Assume now that π is a witness for $\mathcal{R}_{cp}(\mathcal{A}) \models \exists w.\exists_\infty L(w)$. If clock constraints are non-strict, π is a real timed run in $\mathcal{A}$, which witnesses the property. With strict constraints, we can just notice that the witness in $\mathcal{R}_{cp}(\mathcal{A})$ can be chosen as a reachable simple cycle that will be profitable in the sense of Appendix Appendix A, and to build a feasible infinite run over an iterate of that cycle, as made in the proof of Lemma 22. □

We now turn to the universal problem. As an obvious consequence of Lemma 14, we have that:

Lemma 20. *$\mathcal{A} \models \exists w.\forall_\infty L(w)$ iff $\mathcal{R}_{cp}(\mathcal{A}) \models \exists w.\forall_\infty L(w)$.*

This condition can also be checked in polynomial space by detecting cycles with negative weight in $\mathcal{R}_{cp}(\mathcal{A})$.

7.3. PSPACE *lower bounds*

The lower bounds will be proven by reduction from the reachability problem in timed automata, which is known to be PSPACE-hard already when there are three clocks [19].

We first focus on the universal problems. We fix a timed automaton $\mathcal{A}$ and we build the weighted timed automaton $\mathcal{B}$ with a single weight variable, by assigning weight and rate zero everywhere in $\mathcal{A}$, and from the final location of $\mathcal{A}$ we go to a sink location, with a self-loop labelled with weight -1. We define $T = |\mathcal{R}(\mathcal{A})|$. The following four properties are then equivalent:

(i) the final location of $\mathcal{A}$ is reachable;

(ii) the final location of $\mathcal{A}$ is reachable in no more than T time units;

(iii) $\mathcal{B} \not\models \exists w.\forall_\infty L(w)$;

(iv) $\mathcal{B} \not\models \exists w.\forall_T L(w)$.

For the existential case, the argument is similar to the previous one: automaton $\mathcal{B}$ is now obtained from $\mathcal{A}$ by assigning weight and rate -1 everywhere: for any initial credit the weight of any run will decrease to infinity unless we allow to escape to a rate-zero location. From the final location of $\mathcal{A}$ we go to a sink location, with a self-loop of weight zero. The very same equivalent properties can be stated for this case.

8. Conclusion

Weighted timed automata with energy constraints are a nice formalism for modeling resource consumption in real-time systems. Unfortunately (and surprisingly), we proved that reachability in this framework is undecidable (though it was proven decidable for one-clock automata).

Still, we have been able to identify variants of the problem where algorithms exist: in particular, following [14], we believe that the time-bounded variant of the problem is interesting, and we would like to push it further by analyzing the decidability of various problems that are undecidable in general. One of the problems of interest is the existence of a path that satisfies both a lower-bound and an upper-bound constraint. On another direction, some variants of our models, for instance allowing one to set the weight to some value, are also on our agenda for future work.

Acknowledgments. This work has been partly supported by ERC Starting grant EQualIS (308087) and by European project Cassting (FP7-ICT-601148).

References

[1] P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, J. Srba, Infinite runs in weighted timed automata with energy constraints, in: FORMATS'08, vol. 5215 of *LNCS*, Springer, 33–47, 2008.

[2] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, F. Vaandrager, Minimum-cost reachability for priced timed automata, in: HSCC'01, vol. 2034 of *LNCS*, Springer, 147–161, 2001.

[3] R. Alur, S. La Torre, G. J. Pappas, Optimal paths in weighted timed automata, in: HSCC'01, vol. 2034 of *LNCS*, Springer, 49–62, 2001.

[4] P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, Quantitative analysis of real-time systems using priced timed automata, Communications of the ACM 54 (9) (2011) 78–87.

[5] G. Behrmann, K. G. Larsen, J. I. Rasmussen, Optimal scheduling using priced timed automata, SIGMETRICS Performance Evaluation Review 32 (4) (2005) 34–40.

[6] K. G. Larsen, J. I. Rasmussen, Optimal conditional reachability for multi-priced timed automata, in: FoSSaCS'05, vol. 3441 of *LNCS*, Springer, 234–249, 2005.

[7] M. Woehrle, K. Lampka, L. Thiele, Segmented state space traversal for conformance testing of cyber-physical systems, in: FORMATS'11, vol. 6919 of *LNCS*, Springer, 193–208, 2011.

[8] P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, Timed automata with observers under energy constraints, in: HSCC'10, ACM Press, 61–70, 2010.

[9] K. Quaas, On the interval-bound problem for weighted timed automata, in: LATA'11, vol. 6638 of *LNCS*, Springer, 452–464, 2011.

[10] U. Fahrenberg, L. Juhl, K. G. Larsen, J. Srba, Energy games in multi-weighted automata, in: ICTAC'11, vol. 6916 of *LNCS*, Springer, 95–115, 2011.

[11] N. Markey, Verification of embedded systems – Algorithms and complexity, Mémoire d'habilitation, École Normale Supérieure de Cachan, France, 2011.

[12] J. Ouaknine, J. Worrell, Towards a theory of time-bounded verification, in: ICALP'10, vol. 6199 of *LNCS*, Springer, 22–37, 2010.

[13] A. Biere, A. Cimatti, E. M. Clarke, Y. Zhu, Symbolic model checking without BDDs, in: TACAS'99, vol. 1579 of *LNCS*, Springer, 193–207, 1999.

[14] Th. Brihaye, L. Doyen, G. Geeraerts, J. Ouaknine, J.-F. Raskin, J. Worrell, On reachability for hybrid automata over bounded time, in: ICALP'11, vol. 6756 of *LNCS*, Springer, 416–427, 2011.

[15] A. Puri, An undecidable problem for timed automata, Discrete Event Dynamic Systems 9 (2) (1999) 135–146.

[16] B. Bérard, C. Dufourd, Timed automata and additive clock constraints, Information Processing Letters 75 (1-2) (2000) 1–7.

[17] R. Alur, D. L. Dill, A theory of timed automata, Theoretical Computer Science 126 (2) (1994) 183–235.

[18] P. Bouyer, E. Brinksma, K. G. Larsen, Staying alive as cheaply as possible, Formal Methods in System Design 32 (1) (2008) 2–23.

[19] C. Courcoubetis, M. Yannakakis, Minimum and maximum delay problems in real-time systems, Formal Methods in System Design 1 (4) (1992) 385–415.

Appendix A. Proof of Theorem 4 in the presence of strict guards

The proof of Theorem 4 assumes timed automata have only non-strict constraints. However Theorem 4 is still valid if we relax this hypothesis. In this appendix, we explain how the algorithm can be modified for handling strict constraints. It basically impacts three points in the proof, as we now explain.

► *First the notion of profitable zero-delay cycle has to be extended to cycles that can be taken in arbitrary small but possibly non-zero delays..* This is for instance the case of a simple loop constrained by $x > 0$ and which resets clock x. Such a cycle might be used to generate feasible infinite runs "for free", even though time does not diverge.

A finite sequence τ of edges can be taken arbitrarily fast from (ℓ, v) whenever there exists $\epsilon > 0$ such that for every $0 < \epsilon' \leq \epsilon$, there exists a run from (ℓ, v) along τ of duration at most ϵ'. A particular case is when the sequence of edges can be taken in zero-delay.

Let τ be a cyclic sequence of edges in the region automaton which can be taken arbitrarily fast, and assume that the initial and target region-state of that cycle is (ℓ, r). Let $c \geq 0$. We say that τ generates a profitable cycle with arbitrary small delays from (ℓ, v, c) with $v \in r$ whenever (i) there exists $\epsilon > 0$ such that for every $0 < \epsilon' \leq \epsilon$, there is a feasible run from (ℓ, v, c) along τ, of duration at most ϵ'; and (ii) the sum of all discrete costs along τ is nonnegative. If this is the case, we set the predicate $\mathsf{profit}_\tau((\ell, v), c)$ to true. First notice that valuations in the same region cannot be distinguished by predicate profit_τ:

Lemma 21. *If* $\mathsf{profit}_\tau((\ell, v), c)$ *and* $v' \in [v]$*, then* $\mathsf{profit}_\tau((\ell, v'), c)$*.*

Proof. Let ϱ' be a run along τ from (ℓ, v'), and assume it has positive duration. Let $\epsilon > 0$ be smaller than any positive delay along ϱ', and let $0 < \epsilon' \leq \epsilon$. Let ϱ be a feasible run along τ from (ℓ, v, c) of duration at most ϵ'. The same sequence of delays can be made from (ℓ, v', c), and this yields a feasible run along τ of duration at most ϵ': $\mathsf{profit}_\tau((\ell, v'), c)$ is set to true.

Assume that there is a single run along τ from (ℓ, v'), which has duration zero. Then there is also a single run from (ℓ, v) along τ, which has duration zero (since this is a property of the underlying regions). If played from (ℓ, v, c), this run is feasible, and so is the one from (ℓ, v', c). Hence the expected result. □

The interest of profitable cycles lies in the following lemma:

Lemma 22. *If* $\mathsf{profit}_\tau((\ell, v), c)$*, then there is an infinite feasible run from* $((\ell, v), c)$ *along* τ^ω*. Note that even though the run is infinite, its duration might be finite.*

Proof. Write $\tau = e_1 e_2 \dots e_p$, and c_i for the discrete cost of edge e_i.

- Assume there is some location along τ with a positive rate and in which some positive delay can be spent. Then it is not hard to be convinced that there is some feasible run from (ℓ, v, c) to (ℓ, v', c') with $c' \geq c + \sum_{i=1}^{p} c_i \geq c$ (start from a feasible run, and transfer most of the delays in this cost-interesting location). Now it holds that $\mathsf{profit}_\tau((\ell, v'), c)$ and therefore $\mathsf{profit}_\tau((\ell, v'), c')$ since $c' \geq c$. We can then iterate the argument.

- Assume that the sum of discrete costs along τ is positive. This is easy to see that there is a feasible run along τ from (ℓ, v, c) to some (ℓ, v', c') such that $c' \geq c$. Since $\mathsf{profit}_\tau((\ell, v'), c)$, this implies that there is an infinite feasible run from (ℓ, v, c) (obtained as previously by iterating the sequence τ).

- Assume now that all locations along τ have nonpositive rate and that the sum $\sum_{i=1}^{p} c_i$ is zero. Either along τ, some delay has to be spent in a location with negative rate (and this is then the case from every (ℓ, v') with $v' \in [v]$), or no delay has to be spent in such locations. In the second case, it is possible to build a feasible run along τ from (ℓ, v, c) to some (ℓ, v', c) (spend zero time in all locations with negative rates), in which case it is easy to iterate the process.

  In the first case, for every feasible run from (ℓ, v, c) along τ, the target state (ℓ, v', c') is such that $c' < c$. However delays can be made as small as we want, hence for every $\delta > 0$, we can find a feasible run from (ℓ, v, c) to (ℓ, v', c') with $c - \delta < c' < c$ such that cost 0 is never hit along that run. We can then mimick such a run from (ℓ, v', c') with very small delays, and we get that $\mathsf{profit}_\tau((\ell, v'), c')$ is true. We can iterate the process from (ℓ, v', c'), and we get an infinite feasible run as expected.

In all cases we build an infinite run along τ^ω which is feasible. □

If there exists τ such that $\mathsf{profit}_\tau((\ell, v), c)$, then we set predicate $\mathsf{profit}_+((\ell, v), c)$ to true: this means that there exists some cycle that can be iterated in such a way that an infinite feasible run can be generated from (ℓ, v, c). Every predicate $\mathsf{profit}_\tau((\ell, \bullet), c)$ is uniform by region (Lemma 21), it is therefore also the case for $\mathsf{profit}_+((\ell, \bullet), c)$: we define the predicate $\mathsf{profit}_+((\ell, r), c)$ where r is a region.

It remains to prove that such profitable regions can be computed, by extending Lemma 5 as follows:

Lemma 23. *Given (ℓ, r), we can compute in polynomial space the set $\{c \mid \mathsf{profit}_+((\ell, r), c)$ holds$\}$. If this set is non-empty, its infimum is a natural number.*

Proof. It suffices to consider the subgraph of the region automaton where only states that can be reached from (ℓ, r) by arbitrary small delays appear, and to indicate for each transition whether it is taken in zero delay, or whether it requires a positive delay. The transitions are furthermore decorated by discrete costs, and each location is labelled by its rate. This is an extended weighted graph, which can be analyzed by standard graph algorithms to compute the desired values. The graph might have exponential size, but can be computed on-the-fly; The algorithm of Lemma 5 can then be adapted here, computing the infimum of $\{c \mid \mathsf{profit}_+((\ell, r), c)$ holds$\}$ and whether this infimum can be reached. ☐

In the rest of the algorithm the notion of profitable zero-delay cycle has to be replaced by the notion of profitable cycle with arbitrary small delays.

► *The second point where strict constraints do have an impact is in the proof of Lemma 8, where runs are transformed by setting delay zero in some locations.* Delays spent in a given location are postponed to the first or the latest appearance of this location along the run (depending on the cost of this location), and in all other occurrences the delay spent is set to zero. In the more general context of strict constraints, this is not possible, since some positive delay might have to elapse before a transition can be fired (due to strict constraints).

However we can twist the transformation to take into account strict constraints: the idea is to replace the zero-delay transitions by arbitrary small delays. We call the original run ϱ, and let γ be the sequence of edges underlying ϱ, and $(\ell_i)_{0 \le i \le p}$ be the sequence of locations which are visited. A run along γ is characterized by the sequence of delays spent in each location. We write $(d_i)_{0 \le i \le p}$ for the sequence of delays characterizing ϱ. We apply the transfer mentioned in the original proof and this yields a sequence of delays $(d'_i)_{0 \le i \le p}$. If $d'_i > 0$, we say the corresponding location ℓ_i is full-delay. If $d'_i = 0$, the corresponding location ℓ_i is said zero-delay. There are at most $|L|$ full-delay locations.

Unfortunately, due to strict constraints, the sequence of delays $(d'_i)_{0 \le i \le p}$ may not define a real run of the automaton. The idea is then to only transfer partially the delays in the "full-delay" locations, and we realize that whatever the partial transfer, it defines a real run of the system which is also feasible (same proof as the original). In the following the run defined by $(d'_i)_{0 \le i \le p}$ is called the abstract run. It represents a family of concrete runs obtained by partial transfer, as explained above.

Assume that there exists some concrete run obtained by partial transfer, which visits some configuration (ℓ, v, c) with $\mathsf{profit}_+((\ell, v), c)$. Then pick the first position i_0 (with $0 \le i_0 \le p$) from which there is a concrete run visiting such a profitable configuration at position i_0. We will bound the length of the run until position i_0, instead of bounding the total length of the run. Therefore, from now on, w.l.o.g. we assume that no concrete run visits a state from which a profitable cycle can be taken.

Let κ be a maximal segment (sequence of edges) of τ along which all locations are zero-delay (that is, $d'_i = 0$). We will analyze portions of concrete runs along κ, and we write abusively $i \in \kappa$ for the set of indices defining κ. A concrete realization $(d''_i)_{i\in\kappa}$ over κ is said fast from (ℓ, v, c) whenever it can be arbitrarily compressed: for every $0 < \lambda \leq 1$, there is a sequence $(d'''_i)_{i\in\kappa}$ such that $\sum_{i\in\kappa} d'''_i \leq \lambda \cdot \Big(\sum_{i\in\kappa} d''_i\Big)$, and such that the concrete realization $(d'''_i)_{i\in\kappa}$ from (ℓ, v, c) visits the same regions as that defined by $(d''_i)_{i\in\kappa}$. Note that the abstraction of κ is zero-delay, this is why we will be interested in fast realizations of κ. Note also that such fast concrete realizations always exist.

We fix the initial configuration (ℓ, v, c) to enter κ, and consider a fast (feasible) realization $(d''_i)_{i\in\kappa}$. We let (ℓ', v', c') be the final configuration of that realization. We aim at modifying that realization into another which is feasible, shorter and ends up at the same valuation, with a cost which is larger than or equal to c'. There are several positions along κ which are crucial: the edges where some clock is reset for the first time, and the edges where some clock is reset for the last time. If we respect these edges, and the delay between those edges, even though we remove some fragments of κ, we will end up at the same state (possibly with a different cost), and the removal will be harmless for the rest of the concrete run, if the final cost is larger than or equal to c'. We therefore mark all those edges in κ, and we will "fix" those positions together with the first and the last, and we will reduce the length of the segment between two such positions.

Let κ' be a segment between two fixed positions. Assume $\kappa' = e_1 \dots e_k$. Initialize $Y_0 = \emptyset$, and define for $0 \leq i < k$, $Y_{i+1} = Y_i \cup Y(e_i)$ where $Y(e_i)$ is the set of clocks which is reset by edge e_i. We now consider along κ' a maximal sequence κ'' along which Y_i is constant, and for every location which appears along κ'', we mark the first occurence of a location with positive rate where some time elapse and the last occurence of a location with negative rate where some time elapse. Between two such marked locations, if a location ℓ is visited twice, then the corresponding cycle has nonpositive accumulated discrete cost (otherwise a profitable cycle could be taken, which is assumed not to be the case). We can therefore remove that part, and transfer all delays into the marked versions of the locations. This yields a (shorter) concrete run which is feasible (same argument as with the transfer in the original proof). The new final cost is larger than or equal to the original final cost, since it is obtained after removal of cycles with nonpositive discrete cost (all costs spent in some location are preserved). Finally, due to the construction, it is not difficult to check that the final valuation after κ is preserved. The construction is illustrated on Figure A.15.

This results in a bound for κ of $(2|X|+1) \cdot (|X|+1) \cdot (|L|+1) \cdot |L|$. Globally we can therefore bound the length of ϱ by:

$$(|L|+1) \cdot \Big((2|X|+1) \cdot (|X|+1) \cdot (|L|+1) \cdot |L|\Big).$$

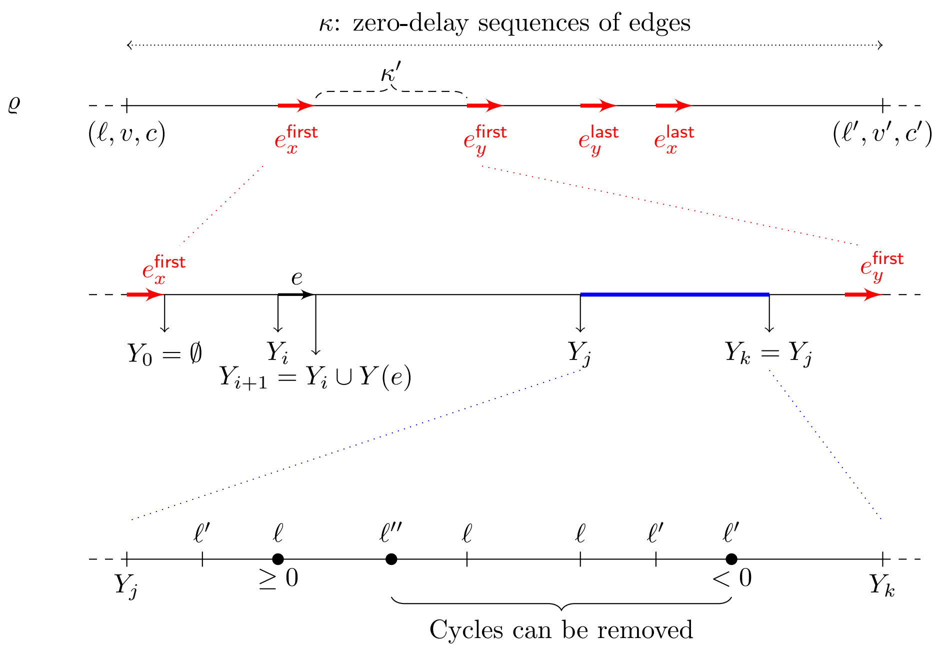


Fig. A.15: Scheme of the proof for strict constraints

► *The case of infinite runs is quite easy to handle, by adapting the original proof.* We first restrict to a suffix ϱ' of the run ϱ which is flexible (and clock u lies within one time unit). Along that tail, a region-state is visited infinitely often, say at configurations $(\ell_{i_j}, v_{i_j}, c_{i_j})_{j\geq 0}$. Write $\varrho^{(j)}$ for the portion of the run between $(\ell_{i_j}, v_{i_j}, c_{i_j})_{j\geq 0}$ and $(\ell_{i_{j+1}}, v_{i_{j+1}}, c_{i_{j+1}})$. The accumulated discrete costs along the $\varrho^{(j)}$'s cannot be all negative, otherwise the run ϱ would not be feasible. Pick $\varrho^{(j)}$ whose accumulated discrete cost is nonnegative. Due to the flexibility of the suffix, the delays along $\varrho^{(j)}$ can be made as short as we want while keeping the feasibility: we have detected a feasible profitable cycle. We have therefore transformed the infinite feasible run into a finite feasible run which reaches a configuration (ℓ, v, c) with $\mathsf{profit}_+((\ell, v), c)$ set to true.

Appendix B. Biographies of authors

Patricia Bouyer-Decitre holds a PhD in Computer Science from ENS Cachan (2002). She is a CNRS researcher since 2002 at Laboratoire Spécification et Vérification (LSV, CNRS & ENS Cachan, France). She has held visiting positions at Aalborg University (Denmark) in 2002 and Oxford University (UK) in 2007.

Patricia Bouyer-Decitre is the principal investigator of ERC Starting Grant project EQualIS, whose aim is to enhance the design and verification of interacting systems, by providing quantitative analysis methods for such systems. Her main research topics are timed systems, model checking, games for synthesis and quantitative aspects of verification.

She was the recipient of a Marie Curie fellowship in 2006, of the Bronze medal of CNRS in 2007 and of the Presburger Award given by the EATCS in 2011.

Kim Guldstrand Larsen holds an MSc in Mathematics and Computer Science from Aalborg University (1982) and a PhD in Computer Science from Edinburgh University (1986). He is a full professor in Computer Science at Aalborg University since 1993, and has held visiting appointments at research centers like ENS Cachan (France), SICS (Sweden), Uppsala University (Sweden), and Carnegie-Mellon University (U.S.A).

Kim Guldstrand Larsen is Honary Doctor (Honoris Causa) at Uppsala University (1999) and Ecole Normal Superieure de Cachan, Paris (2007). He is honary member of the Academia Europaea (2012). He is member of the Royal Danish Academy of Sciences and Letters and member of the Danish Academy of Technical Sciences. He received in 2005 the Danish Citation Laureates Award, Thomson Scientific Award as the most cited Danish Computer Scientist in the period 1990-2004. He is Knight of the Order of Dannebrog (2007).

He is member of the board of the Danish Independent Research Councils, and newly reappointed Danish National Expert for the ICT Programme under EU. He is director of the Danish-Chinese research center IDEA4CPS – Foundations of Cyber-Physcial Systems funded by the Danish National Research Foundation (2011-2017). He is director of InfinIT, the Danish ICT Innovation Network funded by the Danish Council for Technology and Innovation. The key research areas of Kim G. Larsen include concurrency theory, model checking, real-time and hybrid systems, stochastic systems, games and synthesis. In 2013 he received the CAV Award for his long-term effort on the tool UPPAAL.

Nicolas Markey holds a PhD in Computer Science from Orléans University (2003). He is a CNRS researcher at Laboratoire Spécification & Vérification (LSV) of École Normale Suprieure de Cachan (ENS Cachan, France) since 2004. He was a post-doctoral researcher at Université Libre de Bruxelles (ULB, Belgium) in 2003-2004.

Nicolas Markey is the leader of the team TEMPO at LSV, which focuses on the verification, optimization and synthesis of complex computerized systems in which quantitative aspects (e.g. real-time) play an important role. He is the coordinator of the European project FP7-Cassting, where the aim is to study non-zero-sum games in order to develop algorithms for synthesizing complex, multi-agent systems. His main research topics are model checking, temporal logics, quantitative aspects of verification, and games for synthesis.