

Foundations of quantitative and real-time verification

Timed and Hybrid automata

Nicolas Markey

September-November 2010

Contents

Introduction	3
Why verification?	3
Basics on model-checking	3
Model-checking open systems	7
Exercises	9
1 Simply-timed automata	11
1.1 Automata with duration transitions	11
1.2 Extending temporal logics with constraints on durations	13
1.3 Semi-continuous semantics of duration automata	15
1.4 Concurrent game automata with duration transitions	17
Exercises	18
2 Timed automata	19
2.1 Discrete <i>vs</i> dense time	19
2.2 Timed automata	20
2.3 Region equivalence	21
2.4 Language-theoretic properties of timed automata	25
2.5 Timed temporal logics	28
2.5.1 Branching-time temporal logics	28
2.5.2 Linear-time temporal logics	29
2.6 Algorithmic issues	31
2.7 Timed games	35
2.7.1 Control-oriented timed games	35
2.7.2 Game-theoretic approach to timed games	37
Exercises	39
3 Hybrid automata	40
3.1 Hybrid automata: quantities beyond time	40
3.2 Rectangular hybrid automata	40
3.2.1 Decidability of initialized rectangular automata	41
3.2.2 Undecidability of rectangular automata	44
3.3 Polygonal hybrid systems	44

3.3.1	Decidability of planar polygonal hybrid systems	46
3.3.2	Undecidability of 3-dimensional polygonal hybrid systems	47
4	Timed automata with observers	49
4.1	Timed automata with linear observers	49
4.2	Optimal reachability	50
4.3	Quantitative model-checking on priced-timed automata	52
4.4	Timed game automata with linear observers	57
4.5	Energy constraints	61
	Conclusions and research directions	66
	Summary of the course	66
	Possible research directions	66

Introduction

Why verification?

- Microprocessors are everywhere: the estimation is that one billion transistors are built per person per year. CPUs in personal computers represent only 2% of all CPUs, and all CPUs are only 2% of all semiconductors.
- Bugs are everywhere: Ariane 5 blast off in June 1996 is one of the most famous examples [Wika], but we could also mention the Pentium II floating-point division bug [Wike], the loss of the Mars Climate Orbiter [Wikb], or the radiation overdoses caused by the Therac-25 radiation therapy device [Wikd]. All these problems originate in software bugs, and give prominence to the pressing need for software and hardware verification.

As another, simple and understandable, example, consider the “Zune bug”: Zune is the name of Microsoft’s MP3 player. On 31 December 2008, those devices refused to turn on. This was due to the piece of code displayed at Algorithm 1, used for computing the current year, based on the number of days elapsed since January 1st, 1980 (inclusive). While the question looks relatively easy to solve, this algorithm has a bug (find and correct it!), which caused the device to be unusable on December 31st, 2008. This witnesses the difficulty of finding even the most obvious of bugs, and the need for

Algorithm 1: Zune3.0 leap-year code

```
year = ORIGINYEAR; /* = 1980 */
```

```
while (days > 365)
{
  if (IsLeapYear(year))
  {
    if (days > 366)
    {
      days -= 366;
      year += 1;
    }
  }
  else
  {
    days -= 365;
    year += 1;
  }
}
```

formal verification of reactive and embedded systems.

Basics on model-checking

Deciding the termination of a Turing machine on any input is well-known to be undecidable. Hence there is no general technique for detecting bugs in computer programmes. Hence either consider simpler models (automata), or go for “approximate” techniques (in a wide sense).

In this course, we focus on the former direction: our aim is to find exact, terminating algorithms on restricted classes of models. The end of this section is devoted to recalling some basic facts on model-checking techniques for finite-state systems. We refer to [CGP00, BBF⁺01, BK08] for more details on model checking.

In the sequel, AP is a finite set of *atomic propositions*.

Definition 1. An automaton¹ over AP is a 3-tuple $\mathcal{A} = \langle S, T, \ell \rangle$ where

- S is the set of states;
- $T \subseteq S \times S$ is the set of transitions;
- $\ell: S \rightarrow 2^{AP}$ is the labelling function.

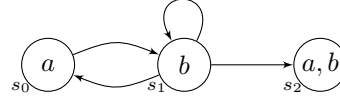
An automaton is said to be

- finite-state if S is finite;
- complete if for all $s \in S$, there is an $s' \in S$ s.t. $(s, s') \in T$.

Fig. 1 displays an example of an automaton, and its graphical representation.

Figure 1: Graphical representation of an automaton

$$\begin{aligned} \mathcal{A} = \{ & S : \{s_0, s_1, s_2\}, \\ & T : \{(s_0, s_1), (s_1, s_1), (s_1, s_0), (s_1, s_2)\}, \\ & \ell : \{s_0 \mapsto \{a\}, s_1 \mapsto \{b\}, s_2 \mapsto \{a, b\}\} \end{aligned}$$



Definition 2. Let $\mathcal{A} = \langle S, T, \ell \rangle$ be an automaton over AP. A run of \mathcal{A} is either a single state $s \in S$ (we call this a trivial run), or a non-empty sequence of transitions $(t_i)_{i \in I}$ (where $I \subseteq \mathbb{N}$ is an interval containing 0) s.t. for all $i \in I \setminus \{0\}$, writing $t_i = (s_i, s'_i)$, it holds $s_i = s'_{i-1}$.

A run is

- finite if I is a single state or a finite sequence of transitions. It is infinite otherwise;
- maximal if
 - it is infinite;
 - or it is a single state s , and s has no outgoing transition in T (i.e., for all $s' \in S$, $(s, s') \notin T$);
 - or it is non-trivial and finite, with $I = [0, m]$, and s'_m has no outgoing transition.

The length of a trivial run is 0. The length of a non-trivial run $\rho = (t_i)_{i \in I}$, written $\text{length}(\rho)$, is $\sup\{i + 1 \mid i \in I\}$.

If $\rho = s$ is a trivial run, its first and last state are both defined as being s . If $\rho = (t_i)_{i \in I}$ is non-trivial, writing $t_i = (s_i, s'_i)$ for all $i \in I$, the first state of ρ , written $\text{first}(\rho)$, is s_0 , and, if I is finite, the last state of ρ , which we denote by $\text{last}(\rho)$, is $s'_{\text{length}(\rho)-1}$. In that case, we also define the convenient notation $s_{\text{length}(\rho)} = \text{last}(\rho)$. When I is infinite, we write $\text{Inf}(\rho)$ for the set of states appearing infinitely many times along ρ :

$$\text{Inf}(\rho) = \{s \in S \mid \forall i \in \mathbb{N}. \exists j > i. s_j = s\}.$$

If ρ is an infinite run in a finite automaton, then $\text{Inf}(\rho)$ is non-empty.

Given a non-trivial run $\rho = (t_i)_{i \in I}$, and two integers j and k with $j < k \leq \text{length}(\rho)$, the segment of ρ between positions j and k is the run $(t_{j+i})_{i \in [0, k-j-1]}$, which we denote by $\rho_{[j, k]}$. When $j = k \leq \text{length}(\rho)$, the segment of ρ between positions j and k is the trivial run s_j . Given $j < \text{length}(\rho)$, the j -th suffix of ρ is the run $\rho_{\geq j} = (t_{j+i})_{i \geq 0, i+j \in I}$. The $(\text{length}(\rho))$ -th suffix is the single state $s_{\text{length}(\rho)}$. Similarly, given $0 < j \leq \text{length}(\rho)$, the j -th prefix of ρ is the run $\rho_{\leq j} = (t_i)_{i \in [0, j-1]}$, and the 0-th prefix is the single state s_0 . Trivial runs only admit a 0-th suffix and a 0-th prefix, both being equal to the run itself.

¹Sometimes also called *transition system* or *Kripke structure* in the literature. In this course, *automata* can have infinite (possibly uncountable) state space.

For all $s \in S$, we write $\text{Runs}_{\mathcal{A}}(s)$ (or sometimes $\text{Runs}(s)$, when the underlying automaton is clear from the context) for the set of maximal runs of \mathcal{A} with first state s , and $\text{FinRuns}_{\mathcal{A}}(s)$ (or $\text{FinRuns}(s)$) for the set of finite runs of \mathcal{A} from s . We refer to the sets of all maximal (resp. finite) runs by omitting to mention s in the above notations.

Definition 3. The reachability problem is defined as follows:

Problem: *Reachability*

Input: *An automaton \mathcal{A} , and two states s and s' ;*

Question: *Does there exist a finite run starting in s and ending in s' ?*

Theorem 4. The reachability problem is decidable in deterministic polynomial time. It is NLOGSPACE-complete.

Sketch of proof. Based on the notion of predecessors:

Definition 5. Let $\mathcal{A} = \langle S, T, \ell \rangle$ be an automaton, and $A \subseteq S$. The set of predecessors of A is the set

$$\text{Pre}(A) = \{s \in S \mid \exists t \in A. (s, t) \in T\}.$$

For any set A , the sequence $(C_n)_{n \in \mathbb{N}}$ defined by $C_0 = \emptyset$ and $C_{n+1} = \text{Pre}(A \cup C_n)$ is non-decreasing and bounded (since S is finite), hence converges after finitely many iterations. It is easily seen that “finitely” here can be bounded by $|S|$.

Also, a state s is in $A \cup C_k$ iff there is a run of length at most k starting in s and ending in a state in A . \square

Definition 6. CTL^* formulas over AP are formulas built on the following grammar:

$$\begin{aligned} CTL^* \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= \phi_s \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \mathbf{U} \phi_p. \end{aligned}$$

where p ranges over AP . A formula of the form ϕ_s is called a state formula, while ϕ_p is a path formula.

Given a run ρ of an automaton \mathcal{A} and a CTL^* formula ϕ , that ϕ holds along ρ in \mathcal{A} , denoted by $\mathcal{A}, \rho \models \phi$, is defined inductively as follows²:

$$\begin{aligned} \mathcal{A}, \rho \models p &\Leftrightarrow p \in \ell(\text{first}(\rho)) \\ \mathcal{A}, \rho \models \neg\phi_s &\Leftrightarrow \mathcal{A}, \rho \not\models \phi_s \\ \mathcal{A}, \rho \models \phi_s \vee \phi'_s &\Leftrightarrow \mathcal{A}, \rho \models \phi_s \text{ or } \mathcal{A}, \rho \models \phi'_s \\ \mathcal{A}, \rho \models \mathbf{E}\phi_p &\Leftrightarrow \exists \rho' \in \text{Runs}(\text{first}(\rho)). \rho' \models \phi_p \\ \mathcal{A}, \rho \models \mathbf{A}\phi_p &\Leftrightarrow \forall \rho' \in \text{Runs}(\text{first}(\rho)). \rho' \models \phi_p \\ \mathcal{A}, \rho \models \neg\phi_p &\Leftrightarrow \mathcal{A}, \rho \not\models \phi_p \\ \mathcal{A}, \rho \models \phi_p \vee \phi'_p &\Leftrightarrow \mathcal{A}, \rho \models \phi_p \text{ or } \mathcal{A}, \rho \models \phi'_p \\ \mathcal{A}, \rho \models \phi_p \mathbf{U} \phi'_p &\Leftrightarrow \exists j \in [1, \text{length}(\rho)]. \mathcal{A}, \rho_{\geq j} \models \phi'_p \text{ and } \forall i \in [1, j]. \mathcal{A}, \rho_{\geq i} \models \phi_p. \end{aligned}$$

The linear-time temporal logic LTL is the fragment of CTL^* defined by the following grammar:

$$\begin{aligned} LTL \ni \phi_s &::= \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= p \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \mathbf{U} \phi_p. \end{aligned}$$

The computation-tree logic CTL is the fragment of CTL^* defined by the following grammar:

$$\begin{aligned} CTL \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= \phi_s \mathbf{U} \phi_s. \end{aligned}$$

Their semantics follow from the semantics of CTL^* .

²Notice that the semantics of the “until” modality \mathbf{U} is *strict*, meaning that it does not take the first state into account. Other semantics can be defined.

Definition 7.

$$\mathbf{F} \phi \stackrel{\text{def}}{=} \text{true} \mathbf{U} \phi \qquad \mathbf{G} \phi \stackrel{\text{def}}{=} \neg \mathbf{F} \neg \phi$$

(where *true* can be seen as a special atomic proposition which holds in every state, or can be defined as $p \vee \neg p$ for some atomic proposition p).

Lemma 8. Given two runs ρ and ρ' of an automaton \mathcal{A} with $\text{first}(\rho) = \text{first}(\rho')$, it holds

$$\forall \phi_s \in \text{CTL}^*. \quad \mathcal{A}, \rho \models \phi_s \Leftrightarrow \mathcal{A}, \rho' \models \phi_s.$$

Proof. By induction on the structure of ϕ_s . □

Definition 9. Given a state s of an automaton \mathcal{A} , and a state formula ϕ_s of CTL^* , we write $\mathcal{A}, s \models \phi_s$ when $\mathcal{A}, \rho \models \phi_s$ for some $\rho \in \text{Runs}(s)$.

The model-checking problem can then be expressed as follows:

Problem: *Model-checking*

Input: A finite automaton \mathcal{A} , a state s of \mathcal{A} , and a formula ϕ of some temporal logic;

Question: Is it the case that $\mathcal{A}, s \models \phi$?

Theorem 10 ([Pnu77, CE82, QS82, ES84]). The model-checking problem is decidable for CTL (PTIME-complete), LTL (PSPACE-complete) and CTL^* (PSPACE-complete).

Sketch of proof. We only propose sketches of proof here, and refer to the profuse literature on the subject, e.g. [VW86, KVV00, CGP00, BK08], for detailed proofs.

CTL model-checking can be achieved in polynomial time (both in the size of the formula and in the size of the automaton) by an inductive algorithm labelling states with the state subformulas they satisfy. We briefly explain how to compute the set of states of $\mathcal{A} = \langle S, T, \ell \rangle$ satisfying $\mathbf{E}a \mathbf{U} b$, assuming that we know the set of states $\llbracket a \rrbracket$ satisfying a and the set of states $\llbracket b \rrbracket$ satisfying b . Let $F: S \rightarrow S$ be the function mapping a set of states Q to

$$\{s \in S \mid \exists t \in S. (s, t) \in T \text{ and } [\mathcal{A}, t \models b \text{ or } (\mathcal{A}, t \models a \text{ and } t \in Q)]\}$$

which can be written equivalently as

$$\text{Pre}\left(\llbracket b \rrbracket \cup (\llbracket a \rrbracket \cap Q)\right).$$

Clearly, F is non-decreasing: if $Q \subseteq Q'$, then $F(Q) \subseteq F(Q')$. Hence the sequence defined by $S_0 = \emptyset$ and $S_{i+1} = F(S_i)$ is non-decreasing, and converges to a fixpoint in finitely many steps (because S is finite). Write R for the limit of $(S_i)_i$. We claim that R is the least fixpoint of F : indeed, if R' is another fixpoint, then it is easily checked that $F(R \cap R') \subseteq F(R) \cap F(R') \subseteq R \cap R'$. Moreover, $S_0 \subseteq R \cap R'$, and by induction, $S_i \subseteq R \cap R'$ for all i . Hence $R \subseteq R \cap R'$, which means that $R \subseteq R'$.

Now, again by induction, any state in some S_i satisfies $\mathbf{E}a \mathbf{U} b$. Conversely, if a state s satisfies $\mathbf{E}a \mathbf{U} b$, then there is a run (which we can assume contains no loop) reaching a b -state and visiting only a -states inbetween. Obviously, all the states involved in this run belong to some S_i , hence to R .

Finally, to compute the set of states satisfying $\mathbf{E}a \mathbf{U} b$, it suffices to compute the limit of the sequence $(S_i)_i$ defined above, which can be achieved in polynomial time. Similar algorithms

Hardness in PTIME is proved by encoding (a simple variant of) the CIRCUIT-VALUE problem. We leave it to the reader as an exercise.

Several algorithms exist for LTL model-checking. The classical algorithm consists in building the so-called *tableau* of an LTL formula ϕ : it is an automaton whose states are the sets of maximally consistent subformula ϕ , and whose transitions are also consistent (for instance, from a state containing subformula $\mathbf{X}\psi$, we can only reach states containing ψ). This results in an exponential-size automaton (which we actually build on-the-fly), with a fairness condition to ensure that all right-hand side of “untils” are eventually satisfied. Taking the product of this tableau with the automaton under study, we get an automaton which contains a fair path if, and only if, ϕ holds in the underlying automaton. It can be proved that this is equivalent to the existence of a lasso-shaped fair path of exponential size, which can be guessed step-by-step using only polynomial space.

Hardness in PSPACE is proved by encoding the computations of a linear-bounded (deterministic) Turing machine. We leave it to the interested reader.

Finally, CTL* model-checking can be done by a labelling algorithm similar to the CTL algorithm. However, state subformulas may be more complicated here (they may belong to LTL), so that this algorithm may also take exponential time and polynomial space in the size of the formula, and polynomial time in the size of the automaton. \square

Definition 11. Let $\mathcal{A} = \langle S, T, \ell \rangle$ be an automaton over AP. A relation $\mathcal{R} \subseteq S \times S$ is a bisimulation iff the following two properties are met:

- if $(s, s') \in \mathcal{R}$, then $\ell(s) = \ell(s')$;
- if $(s, s') \in \mathcal{R}$ and $(s, t) \in T$, then there exists a state $t' \in S$ s.t. $(s', t') \in \mathcal{R}$ and $(t, t') \in \mathcal{R}$;
- if $(s, s') \in \mathcal{R}$ and $(s', t') \in T$, then there exists a state $t \in S$ s.t. $(s, t) \in \mathcal{R}$ and $(t, t') \in \mathcal{R}$.

Two states s and s' are bisimilar if there is a bisimulation relation \mathcal{R} s.t. $(s, s') \in \mathcal{R}$.

A simulation is a relation where only the first two requirements above hold.

Proposition 12. Let $\mathcal{A} = \langle S, T, \ell \rangle$ be an automaton over AP. Let s and s' be two bisimilar states of S . Then for any $\phi \in \text{CTL}$, it holds $\mathcal{A}, s \models \phi$ iff $\mathcal{A}, s' \models \phi$.

Proof. By induction on ϕ . \square

Definition 13.

- Problem:** *Bisimulation checking*
Input: *An automaton $\mathcal{A} = \langle S, T, \ell \rangle$, and two states s and s' of S ;*
Question: *Are s and s' bisimilar?*

Proposition 14. *Bisimulation checking is PTIME-complete.*

Proof. The proof goes by a characterization of the largest bisimulation relation in terms of a fixpoint. Let $F: 2^{S \times S} \rightarrow 2^{S \times S}$ be the mapping defined as

$$F(R) = \{(s, s') \in S \times S \mid \ell(s) = \ell(s') \text{ and } \forall (s, t) \in T. \exists t' \in S. (s', t') \in T \text{ and } (t, t') \in R \\ \text{and } \forall (s', t') \in T. \exists t \in S. (s, t) \in T \text{ and } (t, t') \in R\}.$$

A relation R is a bisimulation iff it satisfies $R \subseteq F(R)$. Moreover, $F(R \cup R') \supseteq F(R) \cup F(R')$, so that F is non-decreasing, and has a fixpoint. Consider the sequence $S_0 = S \times S$, and $S_{i+1} = F(S_i)$. Clearly, $S_1 \subseteq S_0$ and, by induction, $S_{i+1} \subseteq S_i$ as F is non-decreasing. Since $S \times S$ is finite, the sequence converges to a fixpoint P such that $P = F(P)$, which is a bisimulation (by the first remark above). Moreover, if R is another bisimulation, then $P \cup R$ is also a bisimulation: indeed, it holds $P \cup R \subseteq F(P) \cup F(R) \subseteq F(P \cup R)$. Now, $P \cup R \subseteq S_0$, and by induction, $P \cup R \subseteq S_i$ for all i . Hence also $P \cup R \subseteq P$, which means that $R \subseteq P$. In other terms, P is the (unique) largest bisimulation: if two states s and s' are bisimilar, then $(s, s') \in P$. Checking bisimilarity hence amounts to computing P , which is achieved by computing the sequence $(S_i)_i$ defined above, and converges in polynomial time.

We omit the proof of hardness in PTIME, which can be found *e.g.* in [Saw03]. \square

Model-checking open systems

Open systems are systems interacting with their environment (other systems, users, ...). Automata are not adequate to model this interaction. They have been extended to *multi-agent automata*, or *concurrent game automata*.

Definition 15. An concurrent game automaton over AP is a 7-tuple $\mathcal{G} = \langle S, T, \ell, \mathbb{A}, \mathbb{M}, \text{Ch}, \text{Edg} \rangle$ where

- $\langle S, T, \ell \rangle$ is a complete automaton;
- \mathbb{A} is a finite set of agents (sometimes also called players);
- \mathbb{M} is a finite, non-empty set of possible actions for the agents;

- $Ch: S \times \mathbb{A} \rightarrow 2^{\mathbb{M}} \setminus \{\emptyset\}$ indicates the set of allowed moves for a given agent in a given location;
- $Edg: S \times \mathbb{M}^{\mathbb{A}} \rightarrow T$ returns the transition corresponding to the selected actions of the agents, with the requirement that if $Edg(s, \langle m_A \rangle_{A \in \mathbb{A}}) = (s', s'')$, then $s' = s$.

An concurrent game automaton is said to be

- turn-based if for all $s \in S$, there exists an agent $owner(s) \in \mathbb{A}$ s.t. $Ch(s, a)$ is a singleton if $a \neq owner(s)$.

Figure 2: A concurrent game automaton

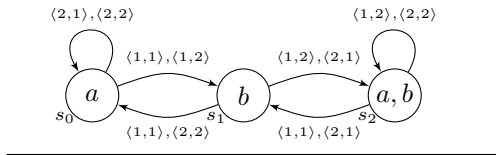


Fig. 2 displays an example of a concurrent game automaton. It has two agents A and B , and two actions 1 and 2, always allowed to both agents. The Edg function is represented by decorating transitions with the set of actions they correspond to. For instance, $Edg(s_0, \langle 1, 2 \rangle)$ is the transition from s_0 to s_1 : in other terms, if the current location is s_0 , and agent A selects action 1 and agent B selects 2, then the transition from s_0 to s_1 will be fired.

Definition 16. A (finite, infinite, maximal) run in a concurrent game automaton $\mathcal{G} = \langle S, T, \ell, \mathbb{A}, \mathbb{M}, Ch, Edg \rangle$ is a (finite, infinite, maximal) run in the underlying automaton $\mathcal{A}_{\mathcal{G}} = \langle S, T, \ell \rangle$.

Definition 17. Let $C \subseteq \mathbb{A}$ be a subset of agents (sometimes called a coalition), and $s \in S$. A move vector for C from s is a mapping $m_C: C \rightarrow \mathbb{M}$ s.t. for all $A \in C$, it holds $m_C(A) \in Ch(s, A)$. Given such a move vector, we define

$$Next(s, m_C) = \{Edg(s, m_{\mathbb{A}}) \mid m_{\mathbb{A}} \text{ is a move vector for } \mathbb{A} \text{ and } \forall A \in C. m_{\mathbb{A}}(A) = m_C(A)\}$$

The important notion in game automata is the notion of strategies:

Definition 18. Let $\mathcal{G} = \langle S, T, \ell, \mathbb{A}, \mathbb{M}, Ch, Edg \rangle$ be a concurrent game automaton, and $A \in \mathbb{A}$ be an agent. A strategy for A is a mapping $\sigma_A: FinRuns \rightarrow \mathbb{M}$ such that for any $\rho \in FinRuns$, it holds $\sigma_A(\rho) \in Ch(last(\rho), A)$. We write $Strat_{\mathcal{G}}(A)$ for the set of strategies of A in \mathcal{G} , and $Strat_{\mathcal{G}}$ for the set of all strategies of all agents (and omit the subscript when it is clear from the context).

Given a subset $C \subseteq \mathbb{A}$, a strategy for C is a mapping $\sigma_C: C \rightarrow Strat$ associating with each agent $A \in C$ a strategy $\sigma_C(A)$ in $Strat(A)$.

A strategy σ is said to be memoryless if, for all finite runs ρ and ρ' with $last(\rho) = last(\rho')$, it holds $\sigma(\rho) = \sigma(\rho')$.

When an agent commits to a strategy, she restricts the set of possible runs of the automaton:

Definition 19. Let $\mathcal{G} = \langle S, T, \ell, \mathbb{A}, \mathbb{M}, Ch, Edg \rangle$ be a concurrent game automaton, $C \subseteq \mathbb{A}$ be a coalition. Let σ_C be a strategy for C . A finite run $\rho = (t_i)_i$ is compatible with σ_C if for all $j < length(\rho)$, it holds $t_j \in Next(last(\rho_{\leq j}), \sigma_C(A)(\rho_{\leq j}))$. An infinite run ρ is compatible with σ_C if all its finite prefixes are.

Given a state $s \in S$ and a strategy σ_C for some coalition C , we write $Out_{\mathcal{G}}(s, \sigma_C)$ (or $Out(s, \sigma_C)$ when the underlying automaton is clear from the context) for the set of maximal runs of \mathcal{G} with first state s and compatible with σ_C , and $FinOut_{\mathcal{G}}(s, \sigma_C)$ (or $FinOut(s, \sigma_C)$) for the set of finite runs of \mathcal{G} from s compatible with σ_C . We refer to the sets of all maximal (resp. finite) runs by omitting to mention s in the above notations

Often, the aim of a strategy is to win the game. This can be defined as follows:

Definition 20. Let \mathcal{G} be a game automaton. A winning condition Ω is a subset of the set of maximal runs of \mathcal{G} . A strategy is said to be winning for Ω from a state s if all its outcomes from s belong to Ω .

As for plain automata, reachability is the basic property to be checked of a concurrent game automaton:

Definition 21. The reachability problem in games is defined as follows:

Problem: *Reachability in games*

Input: *An game automaton \mathcal{A} , a coalition A , and two states s and s' ;*

Question: *Is there a strategy for coalition A all of whose outcomes starting from s eventually visit s' ?*

Theorem 22. *Reachability in games can be solved in deterministic polynomial time. It is PTIME-complete.*

Proof. We extend the notion of *predecessors* to *A-controllable predecessors*:

Definition 23. *Let $\mathcal{A} = \langle S, T, \ell \rangle$ be an automaton, $A \subseteq S$ and $C \subseteq \mathbb{A}$. The set of controllable predecessors of A by C is the set*

$$\text{CPre}(A, C) = \{s \in S \mid \text{there is a move vector } m_C: C \rightarrow \mathbb{M} \text{ s.t. } \text{Next}(s, m_C) \subseteq \{s\} \times A\}.$$

In other terms, a state s is a *controllable predecessor* of A if coalition C can force the automaton into A in at most one step. Again, the sequence defined by $C_0 = \emptyset$ and $C_{n+1} = \text{CPre}(A \cup C_n)$ is non-decreasing, hence it converges to a fixpoint, which is the set of states from which coalition C can force the automaton to eventually reach A , whatever the agents not in C play. \square

Definition 24. *ATL* formulas over AP are formulas built on the following grammar:*

$$\begin{aligned} \text{ATL}^* \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \langle\langle C \rangle\rangle \phi_p \mid \llbracket C \rrbracket \phi_p \\ \phi_p &::= \phi_s \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \mathbf{U} \phi_p. \end{aligned}$$

where p ranges over AP, and C ranges over $2^{\mathbb{A}}$. A formula of the form ϕ_s is called a state formula, while ϕ_p is a path formula.

The semantics of ATL* is obtained from that of CTL*, with the following two additions:

$$\begin{aligned} \mathcal{A}, \rho \models \langle\langle C \rangle\rangle \phi_p &\Leftrightarrow \exists \sigma_C \in \text{Strat}(C). \forall \rho' \in \text{Out}(\text{last}(\rho), \sigma_C). \rho' \models \phi_p \\ \mathcal{A}, \rho \models \llbracket C \rrbracket \phi_p &\Leftrightarrow \forall \sigma_C \in \text{Strat}(C). \exists \rho' \in \text{Out}(\text{last}(\rho), \sigma_C). \rho' \models \phi_p \end{aligned}$$

Theorem 25 ([AHK02]). *The model-checking problem is decidable for ATL (PTIME-complete) and ATL* (2EXPTIME-complete).*

Proof. We explain the algorithm for ATL: it follows the same ideas as for CTL, using CPre instead of Pre: the function F is then defined as

$$F(Q) = \text{CPre}\left(\llbracket b \rrbracket \cup (\llbracket a \rrbracket \cap Q)\right).$$

The arguments are similar to prove that this function is non-decreasing, has a least fixpoint which can be computed in polynomial time, and which corresponds to the set of states satisfying $\langle\langle A \rangle\rangle (a \mathbf{U} b)$. \square

Exercises

Exercise 1 ★★★ Prove that the problem of deciding whether a Turing machine halts on any input is undecidable.

Exercise 2 ★★ Write algorithms for solving the following problems:

Problem: **Reachability**

Input: An automaton \mathcal{A} , and two states s and s' ;

Question: Does there exist a finite run starting in s and ending in s' ?

Problem: **Safety**

Input: An automaton \mathcal{A} , and two states s and s' ;

Question: Does there exist a maximal run starting in s and never visiting s' ?

Problem: **Repeated reachability**

Input: An automaton \mathcal{A} , and two states s and s' ;

Question: Does there exist an infinite run starting in s and visiting s' infinitely often?

Problem: CTL model-checking

Input: An automaton \mathcal{A} , a state s and a CTL formula ϕ ;

Question: Is it the case that $\mathcal{A}, s \models \phi$?

Prove that your algorithms terminate and returns the correct solution, and analyze their worst-case complexity.

Exercise 3 ★★ Using the “Until” modality, define the modality “Next”, written \mathbf{X} , whose semantics is as follows:

$$\mathcal{A}, \rho \models \mathbf{X} \phi \Leftrightarrow \mathcal{A}, \rho_{\geq 1} \models \phi.$$

Define a “non-strict” semantics for the “Until” modality, and prove that it is strictly less expressive than the “strict” semantics (first define what *strictly less expressive* means).

Exercise 4 ★★ Is it possible (briefly explain why) to express the following requirements in LTL or CTL:

- there exists an infinite run;
- there exists an infinite run along which b occurs infinitely often;
- there exists a finite run having its first and last state labelled with the same set of atomic propositions;
- there exists a run having every even position labelled with a (only), and every odd position labelled with b (only);
- there exists a run having every even position labelled with a (only).

Exercise 5 ★ Describe the behaviour of the game automaton depicted on Fig. 2. Is there a strategy for agent A to visit an a -state infinitely often? A b -state? What about agent B ?

Exercise 6 ★★ Write algorithms for solving the following problems:

Problem: Game reachability

Input: A concurrent game automaton \mathcal{G} , a coalition C , and two states s and s' ;

Question: Does coalition C have a strategy to eventually reach s' from s ?

Problem: Game safety

Input: A concurrent game automaton \mathcal{G} , a coalition C , and two states s and s' ;

Question: Does coalition C have a strategy whose outcomes from s never visit s' ?

Problem: Game repeated reachability

Input: A concurrent game automaton \mathcal{G} , a coalition C , and two states s and s' ;

Question: Does coalition C have a strategy whose outcomes from s visit s' infinitely often?

Problem: ATL model-checking

Input: A game automaton \mathcal{A} , a state s and an ATL formula ϕ ;

Question: Is it the case that $\mathcal{A}, s \models \phi$?

Prove that your algorithms terminate and returns the correct solution, and analyze their worst-case complexity.

Exercise 7 ★★★ Let \mathcal{G} be a concurrent game automaton, s be one of its states, and C be a coalition. Let $\phi \in \text{LTL}$ be a formula. Propose an algorithm for checking that coalition C has a strategy for which all the outcomes from s satisfy ϕ . What is its worst-case complexity?

Exercise 8 ★★ Let $\mathcal{A} = \langle S, T, \ell \rangle$ be an automaton, s and s' be two states of S . Prove that the problem of deciding bisimilarity between s and s' can be reduced in logarithmic space to a safety game problem.

1. Simply-timed automata

1.1 Automata with duration transitions

Definition 26. A duration automaton over AP is a 3-tuple $\mathcal{A} = \langle S, T, \ell \rangle$ where:

- S is a (possibly infinite) set of states;
- $T \subseteq S \times \mathbb{N} \times S$ is a set of transitions, each carrying a duration³;
- $\ell: S \rightarrow 2^{AP}$ labels states with atomic propositions.

The notion of a run is directly obtained from the same notion on plain automata.

Definition 27. The duration of a trivial run is zero; for a non-trivial finite run $\rho = (t_i)_{i \in [0, m]}$, it is defined as follows: writing $t_i = (s_i, d_i, s'_i)$ for all $i \in [0, m]$,

$$\text{dur}(\rho) = \sum_{i \in [0, m]} d_i.$$

For an infinite run $\rho = (t_i)_{i \in \mathbb{N}}$, we let

$$\text{dur}(\rho) = \lim_{n \rightarrow \infty} \text{dur}(\rho_{\leq n}).$$

Notice that, since weights are nonnegative, this limit exists, and is finite if, and only if, all but a finite number of transitions have weight zero.

Definition 28.

Problem: *Shortest path*

Input: *A finite weighted automaton \mathcal{A} , two states s and s' , and an integer n (given in binary notation);*

Question: *Is there a path π in \mathcal{A} from s to s' s.t. $w(\pi) \leq n$?*

Theorem 29. *The shortest-path problem is solvable in PTIME*⁴.

Proof. One of the classical algorithms for solving this problem is the Bellman-Ford algorithm [Bel58] (which we prove is also valid for automata having “negative durations”, as it detects cycles). This algorithm iteratively computes the duration of the shortest run of length i from s to each state of \mathcal{A} , starting with $i = 0$. When no such run exists, the duration is $+\infty$.

- initially, there are no path from s to any other state with length 0. Thus $d_0(s) = 0$ and $d_0(q) = +\infty$ for any $q \neq s$.
- for each i ranging from 0 to n with $n = |S|$, apply the following procedure: first set $d_{i+1}(q)$ to $d_i(q)$ for each state q . Then, for each edge $(q, w, q') \in \delta$, if $d_{i+1}(q') > d_i(q) + w$, then $d_{i+1}(q')$ is set to $d_i(q) + w$.

³Equivalently, edges could be labelled with nonnegative rationals, without much changes to the following results. Additionally, they could also be labelled with intervals. This would come with better succinctness (quadratic), and the ability to have transitions with arbitrarily high duration. Our algorithms could easily be adapted to handle these interval-labelled automata.

⁴Notice that the result holds even if the transitions of the automaton are labelled with intervals, as the shortest path is obviously obtained by considering only the lower bound of the interval.

- finally, if $d_n(q) > d_{n+1}(q)$, then $d(q)$ is set to $-\infty$. Then, for each state q' reachable from q with $d(q) = -\infty$, set $d(q')$ to $-\infty$. If some $d(q)$ is not set after this procedure, then $d(q) = d_n(q)$.

Obviously, this procedure runs in quadratic time (namely $O(|S| \cdot |\delta|)$). Notice that the last step is not needed when durations are nonnegative.

By induction, we can prove the following invariants:

Lemma 30. *For each i between 0 and $n + 1$ and each $q \in S$,*

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

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

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

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

Using this intermediate result, we now prove the following statements, which entail the correctness of the algorithm:

Proposition 31. *For each state q ,*

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

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

Conversely, assume that $d(q)$ is finite (*i.e.*, q is reachable but $d(q)$ has not been set to $-\infty$), and that q is not reachable *via* a negative cycle. From the first statement of the Lemma above, there is a path from s to q with weight (at most) $d(q)$. Among the paths of weight at most $d(q)$, pick a path π with as few transitions as possible. If π has a cycle, this cycle must be nonnegative (by hypothesis), and removing this cycle would yield a path shorter than π (in terms of its number of transitions) with weight at most $d(q)$. This is a contradiction. Hence π has no cycle, so that its length is at most n . Since $d(q) = d_n(q)$ is less than the weight of the shortest path from s to q having at most n transitions, we get that π has weight exactly $d(q)$, and that $d(q)$ is exactly the weight of the shortest path from s to q .

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

around q' . In particular, $d(q_0)$ is finite, which means that $d_n(q_0) \leq d_{n+1}(q_0)$. This means in particular that $d_n(q'_0) \leq d_n(q_0) + w_0$. The same holds for the other states of the cycle:

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

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

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

which contradicts the fact that π is a negative cycle, and concludes the proof. \square

Definition 32.

Problem: *Exact length path*

Input: *A finite weighted automaton \mathcal{A} , two states s and s' , and an integer n ;*

Question: *Is there a path from s to s' of length exactly n ?*

Theorem 33. *The exact-length-path problem can be solved in deterministic exponential time, and is NP-complete (assuming binary notation).*

Proof. Membership in NP is proved by guessing a witnessing path. However, such a path *could* have length exponential in n (for instance if \mathcal{A} only has transitions of duration 1). Instead of guessing the sequence of transitions, we guess the number of times each transition will be fired along the witnessing run. This information can be guessed and stored in polynomial time. More precisely, the procedure works as follows:

- for each transition of \mathcal{A} , guess a number between 0 and n , which is intended to represent the number of times this transition is fired in the witness path;
- check that these guesses correspond to a run: apart from s and s' (unless $s = s'$), each state must be exited as many times as it is entered, and the set of visited states (*i.e.*, entered a positive number of times) must be connected (*i.e.*, there must be a sequence of states entered a positive number of times from s to each visited state).
- check that the weight of the run equals n .

All these conditions can be checked in polynomial time.

Hardness is obtained by encoding the SUBSET-SUM problem [GJ79]: given a set of nonnegative integers $(a_i)_{i \leq k}$ and a nonnegative integer n , we build the weighted automaton (S, T) where $S = \{0, 1, \dots, k\}$ and $T = \{(i, a_i, i + 1), (i, 0, i + 1) \mid 0 \leq i < k\}$. There is a path from state 0 to state k of weight exactly n iff there is a solution to the original instance of the subset-sum problem. \square

1.2 Extending temporal logics with constraints on durations

Definition 34. *The logic DCTL* (for Duration CTL*) is the extension of CTL* where path formula can additionally be built using the $\mathbf{U}_{[a,b]}$ modalities (with $a \in \mathbb{N}$ and $b \in \mathbb{N} \cup \{+\infty\}$), whose semantics is as follows:*

$$\begin{aligned} \mathcal{A}, \rho \models \phi_p \mathbf{U}_{[a,b]} \phi'_p &\Leftrightarrow \exists j \in [1, \text{length}(\rho)]. \mathcal{A}, \rho_{\geq j} \models \phi'_p \text{ and } \text{dur}(\rho_{\leq j}) \in [a, b] \\ &\text{and } \forall i \in [1, j - 1]. \mathcal{A}, \rho_{\geq i} \models \phi_p. \end{aligned}$$

The logics DCTL and DLTL are the restrictions of DCTL similar to the restrictions CTL and LTL of CTL*.*

Several useful shorthand can be built on this modality:

$$\begin{aligned} \phi \mathbf{U}_{[a,b]} \phi' &\stackrel{\text{def}}{=} \phi \mathbf{U}_{[a,b+1]} \phi' & \mathbf{F}_{[a,b]} \phi &\stackrel{\text{def}}{=} \text{true } \mathbf{U}_{[a,b]} \phi & \mathbf{G}_{[a,b]} \phi &\stackrel{\text{def}}{=} \neg \mathbf{F}_{[a,b]} \neg \phi \\ \phi \mathbf{U}_{\leq b} \phi' &\stackrel{\text{def}}{=} \phi \mathbf{U}_{[0,b]} \phi' & \phi \mathbf{U}_{=a} \phi' &\stackrel{\text{def}}{=} \phi \mathbf{U}_{[a,a]} \phi' & \phi \mathbf{U}_{\geq a} \phi' &\stackrel{\text{def}}{=} \phi \mathbf{U}_{[a,+\infty]} \phi' \end{aligned}$$

Definition 35. We define the fragments $DCTL_{\leq, \geq}^*$, $DCTL_{\leq, \geq}$, and $DLTL_{\leq, \geq}$ as the respective fragments of $DCTL^*$, $DCTL$ and $DLTL$ involving only modalities $\mathbf{U}_{\leq a}$ and $\mathbf{U}_{\geq a}$.

It is easily checked that Lemma 8 still holds for those quantitative temporal logics, so that the model-checking problem can be extended to duration automata and duration temporal logics.

Theorem 36. The model-checking problem for $DCTL^*$ (and its fragments) is decidable, with the following complexities:

- $DCTL_{\leq, \geq}$ model-checking is PTIME-complete;
- $DCTL$ model-checking is Δ_2^P -complete;
- $DLTL_{\leq, \geq}$ and $DCTL_{\leq, \geq}^*$ model-checking is PSPACE-complete;
- $DLTL$ and $DCTL^*$ model-checking is EXPSpace-complete.

Remark. Before we proceed to the proof, let us make some remarks about Δ_2^P and the polynomial-time hierarchy. An A -oracle Turing machine, where A is a decision problem, is a Turing machine having an extra write-only tape, called the oracle tape and three special states, which we denote with $q_?$, q_{yes} and q_{no} . The machine runs as a classical Turing machine, except that

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

For instance, a SAT-oracle Turing machine can be seen as a Turing machine having access to a “magic” SAT-solver. Any other NP-complete problem would define an “equivalent” Turing machine, and we generally say NP-oracle Turing machine to denote any of these equivalent Turing machines. Complexity can then be measured in terms of time and space of the computations, as well as in terms of the number of calls to the oracle.

For instance, an NP-oracle Turing machine running in deterministic polynomial time defines the class PTIME^{NP} , which is precisely the class Δ_2^P of the above Theorem. Obviously, a problem solvable in NP is also solvable in PTIME^{NP} : it suffices to ask the oracle. Similarly, a problem solvable in coNP is also solvable in PTIME^{NP} : just ask the oracle for the complement problem (which is in NP), and return the opposite answer. This entails that Δ_2^P contains both NP and coNP, hence it is strongly believed to differ from NP. On the other hand, it is easily seen to be included in PSPACE, since NP is.

Sketch of proof. • For $DCTL_{\leq, \geq}$, we propose a labelling algorithm, extending the algorithm for CTL with modalities $\mathbf{U}_{\leq a}$ and $\mathbf{U}_{\geq a}$.

- for $\mathbf{E}\phi \mathbf{U}_{\leq a} \phi'$, assuming that the states satisfying ϕ and those satisfying ϕ' have been labelled (inductively), it suffices to restrict to the states labelled with ϕ or with ϕ' , and find the length of the shortest run reaching ϕ' . This can be achieved in polynomial time (adapting the algorithm of Theorem 29).
- for $\mathbf{E}\phi \mathbf{U}_{\geq a} \phi'$, we distinguish between two ways of satisfying such a formula: either by an acyclic run, or *via* a cycle with positive duration, which can be used to make the duration as long as needed. The former case can be handled by adapting the algorithm for $\mathbf{E}\phi \mathbf{U}_{\leq a} \phi'$. For the latter case, we first label the automaton with a new atomic proposition p indicating those states that lie on a strictly connected component with positive duration. This can obviously be done in polynomial time. It then suffices to compute those states satisfying $\mathbf{E}(\phi \mathbf{U}(p \wedge \mathbf{E}\phi \mathbf{U} \phi'))$, which is a CTL formula.

Hardness in PTIME follows from that of the CTL model-checking problem.

- for $DCTL$, we begin with proving membership in Δ_2^P : Δ_2^P is the class of problems solvable by a polynomial-time algorithm having access to an NP oracle. The algorithm is similar to the previous one: it aims at labelling states with the subformula they satisfy. However, deciding whether a formula of the form $\mathbf{E}\phi \mathbf{U}_{[a,b]} \phi'$ is in NP, and is where we make use of the NP oracle.

Hardness in Δ_2^P is achieved by encoding the SNSAT problem. We refer to [LMS01] for a detailed proof.

- We omit the proof for DCTL and $\text{DCTL}_{\leq, \geq}$. The main ideas can be found in [AH94, AFH96].
- For DCTL^* and $\text{DCTL}_{\leq, \geq}^*$, we combine the CTL algorithm, labelling states with subformulas they satisfy, with the DCTL algorithm for checking path formulas. \square

In the special case where durations are in $\{0, 1\}$ (weighted automata having this property are said to be *unitary*), the algorithm for DCTL can be optimized:

Theorem 37. *DCTL model-checking restricted to unitary weighted automata can be achieved in polynomial time.*

Proof. We explain the algorithm for the modality $\mathbf{EaU}_{=k}b$: assume the formula to be checked is $\mathbf{EaU}_{=k}b$, with $k > 0$, and that we already know which states satisfy subformulas a and b . We define the following $|S|^2$ -matrices T_a and $T_{a,b}$ as follows:

- $T_a = \{(s, s') \mid s \models \mathbf{EaU}_{=1}(s' \wedge a)\}$;
- $T_{a,b} = \{(s, s') \mid s \models \mathbf{EaU}_{=1}(s' \wedge b)\}$;

Both matrices can be computed in PTIME. Now, computing the sequence T_a^2, T_a^4 up to $T_a^{2^m}$ for $m = \lceil \log_2(k-1) \rceil$ can be achieved in polynomial time as well, and so do the product $T_a^{k-1} \cdot T_{a,b}$. It is easy to prove, by induction on k , that the resulting matrix contains a positive number in a line iff the corresponding state satisfies $\mathbf{EaU}_k b$:

- when $k = 1$: this is by definition of $T_{a,b}$;
- if the result holds up to k , we pick a state for which the corresponding line in $T_a^k \cdot T_{a,b}$ contains a positive number. This matrix is the result of the product of T_a and $T_a^{k-1} \cdot T_{a,b}$. Clearly enough, all the cells contain nonnegative numbers, so that a cell in the line corresponding to s contains a positive number iff there is two states s' and s'' s.t. $T_a(s, s')$ and $[T_a^{k-1} \cdot T_{a,b}](s', s'')$ are positive. This entails the result.

The other modalities can be handled in a similar way. \square

1.3 Semi-continuous semantics of duration automata

Our semantics for weighted automata uses atomic steps for the transitions. In order to get closer to the semantics of timed automata, we could define a *semi-continuous* semantics, where a step of weight n would correspond to n successive steps of weight 1: in other terms, a weighted automaton \mathcal{A} in the semi-continuous semantics is a representation of a (possibly infinite) unitary weighted automaton $\mathfrak{C}(\mathcal{A})$. This automaton is defined as follows: a state is a pair containing a state of \mathcal{A} and the “time” elapsed in the present state. We then have two kinds of transitions:

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

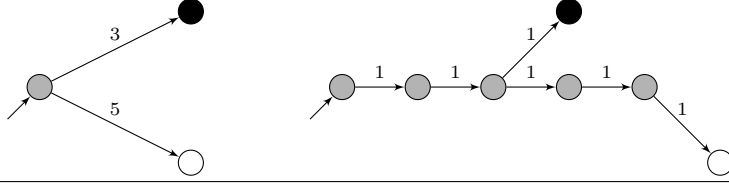
The labeling is preserved, *i.e.*, $\ell'((q, i)) = \ell(q)$. A state q satisfies a formula ϕ of some temporal logic iff the corresponding state $(q, 0)$ in the unitary weighted automaton above satisfies the same formula.

Example. *Figure 3 explains the intuition behind the semi-continuous semantics on a small example. It is easily noticed that the semi-continuous is really different from the discrete one, even when no quantitative constraint is used: for instance, formula $\mathbf{EF}(\mathbf{EF}\bigcirc \wedge \neg \mathbf{EF}\bullet)$ is true in the semi-continuous semantics on our example, while it does not hold under the discrete semantics.*

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

Theorem 38. *The DCTL model-checking problem in the semi-continuous semantics is PSPACE-complete.*

Figure 3: The continuous semantics for weighted automata



Proof. Membership in PSPACE is proved by designing a recursive non-deterministic algorithm for checking that a configuration (*i.e.*, a state in the unitary duration automaton) satisfies a formula. Instead of storing this information (which would require exponential space, since the number of configurations is exponential), we recompute the information when we need it. This way, we reduce the space consumption to polynomial (with an increase in the time complexity).

Notice that a configuration can be stored using polynomial size in the size of the input. Then checking that a configuration satisfies $\mathbf{EpU}_{[a,b]} q$ is achieved by guessing the sequence of configurations witnessing this formula, and recursively checking that they all satisfy the requirements.

PSPACE-hardness is proved by encoding the QBF problem:

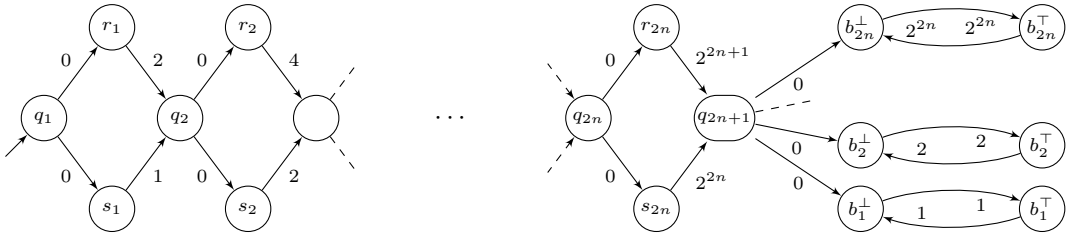
Problem: Quantified boolean formula

Input: A boolean formula $\phi(x_1, \dots, x_{2n})$;

Question: Does the formula $\exists x_1. \forall x_2. \dots \exists x_{2n-1}. \forall x_{2n}. \phi(x_1, \dots, x_{2n})$ hold true?

This problem is well-known to be PSPACE-complete. The reduction is achieved as follows: we consider the weighted automaton depicted on Figure 4. Notice that this automaton only depends on the number of variables in the QBF instance, and not on the boolean formula itself: this formula will be transformed into a DCTL formula to be checked on the automaton.

Figure 4: Reduction from QBF



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

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

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

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

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

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

1.4 Concurrent game automata with duration transitions

Definition 39. A concurrent game automaton with durations is a 7-tuple $\mathcal{G} = \langle S, T, \ell, \mathbb{A}, \mathbb{M}, Ch, Edg \rangle$ where

- $\langle S, T, \ell \rangle$ is a duration automaton;
- \mathbb{A} is a finite set of agents;
- \mathbb{M} is a set of actions;
- $Ch: S \times \mathbb{A} \rightarrow 2^{\mathbb{M}} \setminus \emptyset$ lists the set of moves allowed for each agent in each state;
- $Edg: S \times \mathbb{M}^{\mathbb{A}} \rightarrow T$ is a transition table, with the requirement that, for any state s and any set of moves $(m_i)_{A_i \in \mathbb{A}}$, $Edg(s, (m_i)_{A_i \in \mathbb{A}})$ is a transition out of s .

Given a state s and a move vector $(m_A)_{A \in \mathbb{A}}$, assuming that $Edg(s, (m_A)_{A \in \mathbb{A}}) = (s, t, s')$, we let $Edg_\tau(s, (m_A)_{A \in \mathbb{A}}) = t$ and $Edg_\ell(s, (m_A)_{A \in \mathbb{A}}) = s'$ for the duration and the target state of the selected transition.

The usual notions of runs, strategy, outcomes are directly derived from the same notions on plain game automata. Again, in this quantitative setting, the interesting question is not only to find a winning strategy (e.g. for reaching a target state), but to find an *optimal* one, minimizing the total duration of reaching the target in the worst case. More precisely, given a target state t , the *duration* of a path π before visiting t is defined as

$$\text{dur}_t(\pi) = \min\{\text{dur}(\rho) \mid \rho \text{ prefix of } \pi \text{ visiting } t\},$$

assuming $+\infty$ if π never visits t . Then, given a state s and a strategy σ , the *duration of σ from s before visiting t* is

$$\text{dur}_t(s, \sigma) = \max\{\text{dur}_t(\pi) \mid \pi \in \text{Out}(s, \sigma)\}.$$

Definition 40.

- Problem:** *Optimal reachability strategy*
Input: *A finite weighted game automaton \mathcal{G} , two states s and t , a coalition C , and an integer n ;*
Question: *Does there exist a strategy σ_C for coalition C s.t. $\text{dur}_t(s, \sigma) \leq n$?*

Theorem 41. *The problem of optimal reachability strategy is PTIME-complete.*

Proof. Hardness in PTIME directly follows from the fact that finding a winning strategy for plain reachability is PTIME-hard. Our PTIME algorithm recursively compute, for each state, the optimal duration of winning in at most n steps, with n ranging from 0 to $|S|$. It runs as follows:

- first set $v_0(t) = 0$ for the target, and $v_0(s) = +\infty$ for any $s \neq t$;
- then, for $n \geq 1$, set $v_n(t) = 0$ and

$$v_n(s) = \min_{m_C \in \text{Ch}(s, C)} \left(\max_{m_{\bar{C}} \in \text{Ch}(s, \bar{C})} \{d + v_{n-1}(q) \mid Edg(s, m_{\mathbb{A}}) = (s, d, q) \text{ with } m_{\mathbb{A}} = m_C \cup m_{\bar{C}}\} \right)$$

We claim that for all integers n and d , and all state q , it holds $d \geq v_n(q)$ iff there is a strategy σ_C reaching t from q within at most n steps and duration d . The proof is by induction on n , and is left to the reader. \square

Definition 42.

- Problem:** *Exact reachability strategy*
Input: *A finite weighted game automaton \mathcal{G} , two states s and t , a coalition C , and an integer n ;*
Question: *Does there exist a strategy σ_C for coalition C s.t. $\text{dur}_t(s, \sigma) = n$?*

Theorem 43. *The problem exact reachability strategy is EXPTIME-complete.*

Proof. We use dynamical programming, and recursively build a table $T: S \times \{0, \dots, n\} \rightarrow \{\top, \perp\}$ such that

$$T(q, i) = \top \iff q \models \langle\langle A \rangle\rangle \mathbf{F}_{=i} s'. \quad (1)$$

When $i = 0$, we let $T(q, 0) = \top$ if, and only if, $q = s'$; this clearly fulfills equation (1) (since all durations are non-zero). Now, pick $i < n$, and assume all the $T(q, j)$ have been computed for $j \leq i$. Then

$$T(q, i + 1) = \top \iff \exists c \in \mathbf{M}(q, A). \forall \bar{c} \in \mathbf{M}(q, \bar{A}). \text{Edg}(q, c \cdot \bar{c}) = (q', t, q') \text{ with } T(q', i - t) = \top.$$

This computation can be done since all durations are non-zero. It is achieved by running through the transition table, and is thus in time linear in the size of Edg . It is clear that equation (1) is preserved by this construction, so that in the end, $q \models \phi$ iff $T(q, n) = \top$. This algorithm runs in time $O(n \times |\text{Edg}|)$. \square

Definition 44. *DATL is the extension of ATL with the constrained modality \mathbf{U}_I . $\text{DATL}_{\leq, \geq}$ is the logic where only inequality constraints are allowed.*

Theorem 45. *We consider the subclass of game automata with positive durations. Then model-checking DATL is EXPTIME-complete, and model-checking $\text{DATL}_{\leq, \geq}$ is PTIME-complete.*

Proof. We omit the proof, and refer to [LMO06] for more details. \square

Exercises

Exercise 9 \star Write the precise semantics of $\mathbf{F}_{[a,b]}$ and $\mathbf{G}_{[a,b]}$. Define a non-strict version of $\mathbf{U}_{[a,b]}$.

Exercise 10 $\star\star$ Prove that the DTL formula $\mathbf{F}_{=p} \phi$ can be expressed in $\text{DTL}_{\leq, \geq}$ over duration automata with integer durations.

Prove that the DCTL formula $\mathbf{EF}_{=p} \phi$ cannot be expressed in $\text{DCTL}_{\leq, \geq}$.

Exercise 11 \star Prove that any DCTL formula can be expressed using only modalities \mathbf{EU}_I and \mathbf{EG}_I .

Exercise 12 $\star\star\star\star$ Show that $\text{DCTL}_{\leq, \geq}$ can be checked in PTIME on weighted automata under the semi-continuous semantics.

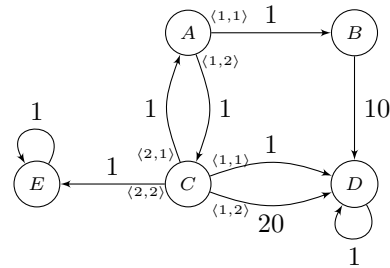
Exercise 13 \star The figure on the right represents a game automaton with durations involving two players (which we name 1 and 2). We assume that it is labelled with two atomic propositions P_1 and P_2 in the following way: P_1 holds everywhere except in D , and P_2 holds only in B and D .

Compute the best constant c for which state A satisfies $\langle\langle 1 \rangle\rangle P_1 \mathbf{U}_{\leq c} P_2$.

Similarly, what is the best value d for which A satisfies $\langle\langle 1 \rangle\rangle P_1 \mathbf{U}_{\geq c} P_2$?

Write an algorithm for $\text{DATL}_{\leq, \geq}$ model-checking.

Figure 5: A game automaton with durations

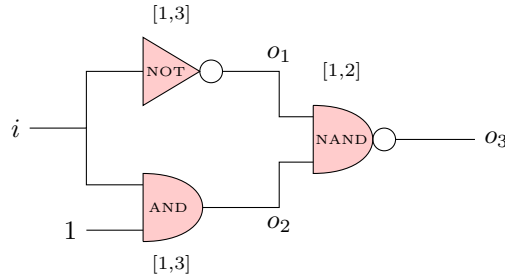


2. Timed automata

2.1 Discrete *vs* dense time

We illustrate the usefulness of dense time with an example of digital circuits [BS91]. Consider the circuit described on Fig. 6, where the intervals decorating each gate indicate the delay needed for this gate to

Figure 6: Example of a circuit



output the correct output value, after the stabilization of its input values. We start with $i = 0$ in a stable configuration, hence the output values are $[o_1 = 1, o_2 = 0, o_3 = 1]$ (in the sequel, we omit the name of the output variables, and write this output as the triple $[101]$).

When i turns to 1, one possible sequence of values (before stabilization) is

$$[101] \xrightarrow[1.2]{o_2} [111] \xrightarrow[2.5]{o_3} [110] \xrightarrow[2.8]{o_1} [010] \xrightarrow[4.5]{o_3} [011]$$

the last set of values corresponding to the stable configuration.

Now, consider the (more complex) circuit depicted on Fig. 7 (where the gate before output o_7 computes $\neg o_4 \wedge o_5 \wedge \neg o_6$). We will prove that considering this circuit under discrete-time does *not*

Figure 7: A circuit that is not 1-discretizable

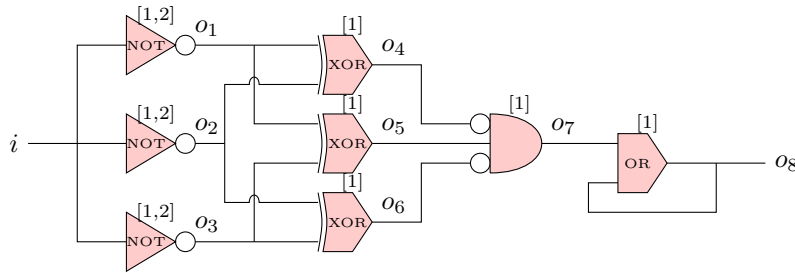


exhibit all possible behaviours. Again, we start with the input set to 0, with output values $[11100000]$ (notice that the last bit could also be 1). Now the input is turned to 1. One possible behaviour in dense time is the following:

$$[11100000] \xrightarrow[1]{o_1} [01100000] \xrightarrow[1.5]{o_2} [00100000] \xrightarrow[2]{o_3, o_5} [00001000] \xrightarrow[3]{o_5, o_7} [00000010] \xrightarrow[4]{o_7, o_8} [00000001]$$

It can be checked that this is a stable configuration.

On the other hand, we can easily list all possible behaviours in discrete time:

- if all three NOT-gates change at the same time, we end up with $[00000000]$, which is stable;

- if o_1 alone changes at date 1, we reach [01100000], then [00011000] and [00000000], which is stable. Similarly if o_3 alone changes at date 1;
- if o_1 and o_3 both change at date 1, we reach [01000000], then [00010100] and [00000000];
- if o_2 alone changes at date 1, we reach [10100000], then [00010100] and [00000000];
- finally, if o_1 and o_2 both change at date 1, we reach [00100000], then [00001100] and [00000000].

As a result, all discrete-time behaviours of the circuit of Fig. 7 reach configuration [00000000], with final output 0, while there are dense-time behaviours which eventually set the value of o_7 and o_8 to 1.

2.2 Timed automata

Let \mathfrak{C} is a finite set, whose elements will be called *clocks*. In the sequel, clocks have nonnegative real values (*i.e.*, the time domain \mathbb{T} is the set of nonnegative reals). We could equivalently choose the nonnegative rationals, which would preserve our results.

Definition 46. A clock valuation on \mathfrak{C} is a mapping $\nu: \mathfrak{C} \rightarrow \mathbb{T}$. We write $\mathbf{0}_{\mathfrak{C}}$ for the clock valuation assigning 0 to all clocks. We also define two operations on clock valuations:

- given a clock valuation ν and a value $d \in \mathbb{T}$, we write $\nu + d$ for the clock valuation w such that $w(c) = \nu(c) + d$ for all $c \in \mathfrak{C}$.
- given a valuation ν and a set $R \subseteq \mathfrak{C}$, we write $\nu[R \rightarrow 0]$ for the clock valuation w such that $w(c) = \nu(c) \cdot \mathbf{1}_R(c)$.

Definition 47. A clock constraint is a formula built on the following grammar:

$$\text{Constr}(\mathfrak{C}) \ni g ::= \top \mid c \sim n \mid g \wedge g$$

where c ranges over \mathfrak{C} , \sim ranges over $\{<, \leq, =, \geq, >\}$, and n over the naturals (or possibly nonnegative rationals). That a clock valuation ν satisfies a clock constraint g is defined inductively in the natural way: any valuation always satisfies \top , and

$$\nu \models c \sim n \quad \Leftrightarrow \quad \nu(c) \sim n \qquad \nu \models g_1 \wedge g_2 \quad \Leftrightarrow \quad \nu \models g_1 \text{ and } \nu \models g_2.$$

We write $\llbracket g \rrbracket_{\mathfrak{C}}$ for the set of valuations of \mathfrak{C} satisfying constraint g .

Definition 48. Let AP be a finite set of atomic propositions, and Σ be a finite alphabet. A timed automaton [AD94] over AP is a 5-tuple $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ where

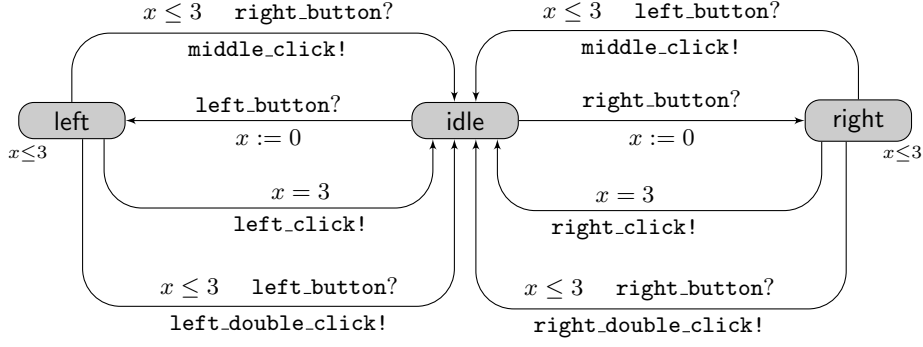
- S is a finite set of states (sometimes called locations);
- \mathfrak{C} is a finite set of clock variables;
- $T \subseteq S \times \text{Constr}(\mathfrak{C}) \times \Sigma \times 2^{\mathfrak{C}} \times S$ is the set of transitions;
- $\ell: S \rightarrow 2^{AP}$ labels states with atomic propositions;
- $\text{Inv}: S \rightarrow \text{Constr}(\mathfrak{C})$ assigns invariants to states.

Example. Before we give the formal semantics of timed automata, let us have a look at an (informal) example: a two-button mouse, as depicted on Fig. 8. In this example, *idle* is the “normal” state of the mouse, where no action is performed. When the user pushes one button (the left one, say), the mouse goes to the *left* state. It cannot yet say whether it will be a single click or a double one, hence no message is sent. If no other action is performed in the following three time units, it is a single click. If a left button is pressed a second time within three time units, then a double-click message is sent.

Definition 49. Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ be a timed automaton over AP and alphabet Σ . The semantics of \mathcal{A} , denoted with $\llbracket \mathcal{A} \rrbracket$, is the (infinite-state) automaton $\llbracket \mathcal{A} \rrbracket = \langle Q, R, l \rangle$ where

- $Q = \{(s, \nu) \in S \times \mathbb{T}^{\mathfrak{C}} \mid \nu \models \text{Inv}(s)\}$;
- R is the union of two sets of transitions:

Figure 8: Clicks and double-clicks of a mouse



- delay transitions: $((s, \nu), (s', \nu')) \in R_d$ iff (s, ν) and (s', ν') belong to Q , $s = s'$, and there exists $d \in \mathbb{R}_{\geq 0}$ s.t. $\nu' = \nu + d$;
- action transitions: $((s, \nu), (s', \nu')) \in R_a$ iff (s, ν) and (s', ν') belong to Q and there exists $t = (s, g, \sigma, r, s') \in T$ s.t. $\nu \models g$, $\nu' = \nu[r \rightarrow 0]$;

- $l((s, \nu)) = \ell(s)$ for all $(s, \nu) \in Q$.

A run of \mathcal{A} is a run of the underlying semantics.

Definition 50.

Problem: *Reachability in timed automata*

Input: A timed automaton \mathcal{A} , two states s and s' ;

Question: Is there a run in \mathcal{A} from $(s, \mathbf{0}_{\mathfrak{C}})$ to (s', ν) , for some valuation ν ?

The next section is devoted to the proof that this problem is decidable.

2.3 Region equivalence

Definition 51. Let \mathfrak{C} be a set of clocks, and \mathcal{G} be a set of clock constraints on \mathfrak{C} . Let \mathcal{R} be a partition of the set of clock valuation $\mathbb{T}^{\mathfrak{C}}$. This set \mathcal{R} is said to be compatible with \mathcal{G} if the following three conditions are met:

- for every $R, R' \in \mathcal{R}$, if there exists $v \in R$ and $t \in \mathbb{T}$ s.t. $v + t \in R'$, then for all $w \in R$, there exists $u \in \mathbb{T}$ s.t. $w + u \in R'$;
- for every $g \in \mathcal{G}$ and any $R \in \mathcal{R}$, either $R \cap \llbracket g \rrbracket_{\mathfrak{C}} = \emptyset$, or $R \subseteq \llbracket g \rrbracket_{\mathfrak{C}}$;
- for every $R, R' \in \mathcal{R}$ and every $Y \subseteq \mathfrak{C}$, letting $R[Y \rightarrow 0] = \{v[Y \rightarrow 0] \mid v \in R\}$, it holds either $R[Y \rightarrow 0] \cap R' = \emptyset$ or $R[Y \rightarrow 0] \subseteq R'$.

We write $\nu \equiv_{\mathcal{R}} \nu'$ when two valuations are equivalent for \mathcal{R} , meaning that they belong to the same set R of \mathcal{R} .

Proposition 52. Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ be a timed automaton, $\llbracket \mathcal{A} \rrbracket = \langle Q, R, l \rangle$ be its semantics, and \mathcal{G} be the set of clock constraints appearing either as guards on transitions or as invariants of states. Let \mathcal{R} be a partition of $\mathbb{T}^{\mathfrak{C}}$ compatible with \mathcal{G} . The relation on Q induced by \mathcal{R} , defined as

$$(s, \nu) \simeq_{\mathcal{R}} (s', \nu') \Leftrightarrow s = s' \text{ and } \nu \equiv_{\mathcal{R}} \nu'$$

is a bisimulation relation on $\llbracket \mathcal{A} \rrbracket$.

Proof. In this proof, \mathcal{R} is fixed, and given a valuation ν , we write $\llbracket \nu \rrbracket$ for the set $R \in \mathcal{R}$ s.t. $\nu \in R$.

Let (s, ν) and (s', ν') be two states of $\llbracket \mathcal{A} \rrbracket$. To prove that $\simeq_{\mathcal{R}}$ is a bisimulation relation, we have to prove that:

- if $(s, \nu) \simeq_{\mathcal{R}} (s', \nu')$, then $l((s, \nu)) = l((s', \nu'))$;

- if $(s, \nu) \simeq_{\mathcal{R}} (s', \nu')$ and $((s, \nu), (t, \mu)) \in R$, then there exists a state $(t', \mu') \in Q$ s.t. $((s', \nu'), (t', \mu')) \in R$ and $(t, \mu) \simeq_{\mathcal{R}} (t', \mu')$;
- if $(s, \nu) \simeq_{\mathcal{R}} (s', \nu')$ and $((s', \nu'), (t', \mu')) \in R$, then there exists a state $(t, \mu) \in Q$ s.t. $((s, \nu), (t, \mu)) \in R$ and $(t, \mu) \simeq_{\mathcal{R}} (t', \mu')$.

The first point is easy, by definition of the labelling function of $\llbracket \mathcal{A} \rrbracket$. The other two are symmetric, so we only prove one of them. Assume that $(s, \nu) \simeq_{\mathcal{R}} (s', \nu')$ and $((s, \nu), (t, \mu)) \in R$. We consider two cases:

- if $((s, \nu), (t, \mu))$ corresponds to a *delay transition*, then $s = t$ and $\mu = \nu + d$ for some $d \in \mathbb{T}$. Also, $\mu \models \text{Inv}(s)$. By definition of \mathcal{R} being compatible with \mathcal{G} , this entails that
 - for any $\rho \in \llbracket \nu \rrbracket$, there exists $d' \in \mathbb{T}$ s.t. $\rho + d' \in \llbracket \mu \rrbracket$,
 - $\llbracket \mu \rrbracket \subseteq \llbracket \text{Inv}(s) \rrbracket$.

Since $(s, \nu) \simeq_{\mathcal{R}} (s', \nu')$, we also have $s' = s$ and $\nu' \in \llbracket \nu \rrbracket$. Hence for some $d' \in \mathbb{T}$, it holds $\nu' + d' \in \llbracket \mu \rrbracket$. Letting $t' = s$ and $\mu' = \nu' + d'$, we have $((s', \nu'), (t', \mu')) \in R$ (because $\mu' \models \text{Inv}(s)$) and $(t, \mu) \simeq_{\mathcal{R}} (t', \mu')$.

- if $((s, \nu), (t, \mu))$ corresponds to an *action transition*, then there exists a transition $(s, g, \sigma, R, t) \in T$ such that $\nu \models g$ and $\mu = \nu[R \rightarrow 0] \models \text{Inv}(t)$. Since $(s, \nu) \simeq_{\mathcal{R}} (s', \nu')$, we have $s' = s$ and $\nu' \in \llbracket \nu \rrbracket$. Since \mathcal{R} is compatible with \mathcal{G} and $\nu \models g$, it holds $\llbracket \nu \rrbracket \subseteq \llbracket g \rrbracket$, hence $\nu' \models g$. Let $t' = t$, and $\mu' = \nu'[R \rightarrow 0]$. We have $\mu = \nu[R \rightarrow 0]$, meaning that $\llbracket \nu \rrbracket[R \rightarrow 0] \cap \llbracket \mu \rrbracket \neq \emptyset$. By definition of \mathcal{R} being compatible with \mathcal{G} , this entails that $\llbracket \nu \rrbracket[R \rightarrow 0] \subseteq \llbracket \mu \rrbracket$. Since $\nu' \in \llbracket \nu \rrbracket$, we deduce that $\mu' = \nu'[R \rightarrow 0] \in \llbracket \mu \rrbracket$. Hence $\mu' \models \text{Inv}(t)$, so that there exists a transition $((s', \nu'), (t', \mu')) \in R$ and $(t, \mu) \simeq_{\mathcal{R}} (t', \mu')$. \square

Remark. Notice that the bisimulation is time-abstract, meaning that it does not preserve the duration of delay transitions. This will be sufficient for most purposes.

Proposition 53. Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ be a timed automaton, with set of constraints \mathcal{G} . There exists a finite partition of $\mathbb{T}^{\mathfrak{C}}$ compatible with \mathcal{G} .

Proof. We define the partition by an equivalence relation on clock valuations: we assume that the clock constraints in \mathcal{A} only involve natural numbers, even if it means multiplying all constants with a positive integer. It is easy to prove that this preserves untimed properties (such as reachability) in the automaton.

For each $c \in \mathfrak{C}$, we let M_c be the maximal integer constant with which clock c is compared in \mathcal{A} . We write \mathbf{M} for $(M_c)_{c \in \mathfrak{C}}$.

Definition 54. Two valuations ν and ν' are said to be \mathbf{M} -equivalent, written $\nu \approx_{\mathbf{M}} \nu'$, if the following three conditions are fulfilled:

- for all $c \in \mathfrak{C}$, $\nu(c) > M_c$ iff $\nu'(c) > M_c$;
- for all $c \in \mathfrak{C}$ s.t. $\nu(c) \leq M_c$ (hence also $\nu'(c) \leq M_c$), it holds $\lfloor \nu(c) \rfloor = \lfloor \nu'(c) \rfloor$, and $\langle \nu(c) \rangle = 0$ iff $\langle \nu'(c) \rangle = 0$;
- for all $c, c' \in \mathfrak{C}$ with $\nu(c) \leq M_c$ and $\nu(c') \leq M_{c'}$ (hence also $\nu'(c) \leq M_c$ and $\nu'(c') \leq M_{c'}$), it holds $\langle \nu(c) \rangle \leq \langle \nu'(c) \rangle$ iff $\langle \nu'(c) \rangle \leq \langle \nu'(c') \rangle$.

This is obviously an equivalence relation. We prove that it defines a partition of the set of clock valuations that is compatible with the set $\mathcal{G}_{\mathfrak{C}}^{\mathbf{M}} = \{c \sim n_c \mid c \in \mathfrak{C}, \sim \in \{<, \leq, =, \geq, >\}, 0 \leq n_c \leq M_c\}$. As a consequence, it will also be compatible with the set of constraints that occur in \mathcal{A} .

We begin with the last two conditions:

- let $g = c \sim n$ in $\mathcal{G}_{\mathfrak{C}}^{\mathbf{M}}$, and ν be a valuation. Assume that $\llbracket \nu \rrbracket \cap \llbracket g \rrbracket \neq \emptyset$: there is a valuation $\mu \in \llbracket \nu \rrbracket$ s.t. $\mu(c) \sim n$. Pick any other valuation $\mu' \in \llbracket \nu \rrbracket$: if $\mu'(c) > M_c$, then also $\mu(c) > M_c$. Since $n \leq M_c$, it must be the case that $\sim \in \{>, \geq\}$, hence $\mu'(c) \sim n$.

Otherwise, we have $\mu'(c) \leq M_c$, and $\mu(c) \leq M_c$. Then $\lfloor \mu(c) \rfloor = \lfloor \mu'(c) \rfloor$, and $\langle \nu(c) \rangle = 0$ iff $\langle \nu'(c) \rangle = 0$. Thus either $\mu'(c)$ and $\mu(c)$ are the same integer, or both belong to the interval $(\lfloor \mu(c) \rfloor, \lfloor \mu(c) \rfloor + 1)$. The result follows.

- take two valuations ν and ν' , and a set $R \subseteq \mathfrak{C}$, and assume that $\llbracket \nu \rrbracket[R \rightarrow 0] \cap \llbracket \nu' \rrbracket \neq \emptyset$. Pick a valuation $\mu \in \llbracket \nu \rrbracket[R \rightarrow 0]$.

First if $R = \emptyset$, the result is trivial. Otherwise, by hypothesis, $\llbracket \nu \rrbracket$ contains a valuation ρ s.t. $\rho[R \rightarrow 0]$ is equivalent to ν' . We prove that $\mu[R \rightarrow 0]$ is also equivalent to ν' :

- for $r \in R$, it must be the case that $\nu'(r) = 0$, and also $\mu[R \rightarrow 0](r) = 0$, so that none of them is strictly more than M_r . For a clock $c \notin R$, $\mu[R \rightarrow 0](c) > M_c$ iff $\mu(c) > M_c$ iff $\rho(c) > M_c$ iff $\rho[R \rightarrow 0](c) > M_c$ iff $\nu'(c) > M_c$.
- the second property for clocks in R is obvious. For clocks not in R , $\lfloor \mu[R \rightarrow 0](c) \rfloor = \lfloor \mu(c) \rfloor = \lfloor \rho(c) \rfloor = \lfloor \rho[R \rightarrow 0](c) \rfloor = \lfloor \nu'(c) \rfloor$, and $\langle \mu[R \rightarrow 0](c) \rangle = 0$ iff $\langle \mu(c) \rangle = 0$ iff $\langle \rho(c) \rangle = 0$ iff $\langle \rho[R \rightarrow 0](c) \rangle = 0$ iff $\langle \nu'(c) \rangle = 0$.
- finally, for two clocks c and c' s.t. $\mu[R \rightarrow 0](c) \leq M_c$ and $\mu[R \rightarrow 0](c') \leq M_{c'}$ and $\langle \mu[R \rightarrow 0](c) \rangle \leq \langle \mu[R \rightarrow 0](c') \rangle$, we again have several cases:
 - * if c and c' are in R , then $\langle \rho[R \rightarrow 0](c) \rangle \leq \langle \rho[R \rightarrow 0](c') \rangle$, and similarly for ν' .
 - * if $c \in R$ and $c' \notin R$, then $\langle \rho[R \rightarrow 0](c) \rangle = 0$, hence $\langle \nu(c) \rangle = 0$, and the result follows. Similarly if $c' \in R$ and $c \notin R$.
 - * if $c \notin R$ and $c' \notin R$, then $\langle \mu(c) \rangle \leq \langle \mu(c') \rangle$, so that $\langle \rho(c) \rangle \leq \langle \rho(c') \rangle$, and $\langle \rho[R \rightarrow 0](c) \rangle \leq \langle \rho[R \rightarrow 0](c') \rangle$, and $\langle \nu'(c) \rangle \leq \langle \nu'(c') \rangle$.

We now prove the first condition, on time elapsing. Take two valuations ν and ν' , and write $\llbracket \nu \rrbracket$ and $\llbracket \nu' \rrbracket$ for the corresponding equivalence classes. Assume that for some $\mu \in \llbracket \nu \rrbracket$, there is a $t \in \mathbb{T}$ s.t. $\mu + t \in \llbracket \nu' \rrbracket$. Pick $\mu' \in \llbracket \nu \rrbracket$: we exhibit a $t' \in \mathbb{T}$ s.t. $\mu' + t' \in \llbracket \nu' \rrbracket$ by distinguishing between several cases:

- if $\nu(c) > M_c$ for all $c \in \mathfrak{C}$, then also $\mu(c) > M_c$ and $\mu'(c) > M_c$ for all $c \in \mathfrak{C}$. Moreover, for any $t \in \mathbb{T}$, it is also the case that $(\mu + t)(c) > M_c$ for all $c \in \mathfrak{C}$, so that $\llbracket \nu \rrbracket = \llbracket \nu' \rrbracket$. Then for any $t' \in \mathbb{T}$, $(\mu' + t')(c) > M_c$ for all $c \in \mathfrak{C}$, which entails $\mu' + t' \in \llbracket \nu' \rrbracket$.
- if for some $c \in \mathfrak{C}$, we have that $(\mu + t)(c)$ is an integer less than or equal to M_c , then we let $t' = (\mu + t)(c) - \mu'(c)$. First notice that $t' \in \mathbb{T}$: indeed, for all integer $\alpha \leq M_c$, $\mu(c) \leq \alpha$ iff $\mu'(c) \leq \alpha$; in particular for $\alpha = (\mu + t)(c)$, which implies that $\mu'(c) \leq (\mu + t)(c)$, so that $t' \geq 0$.

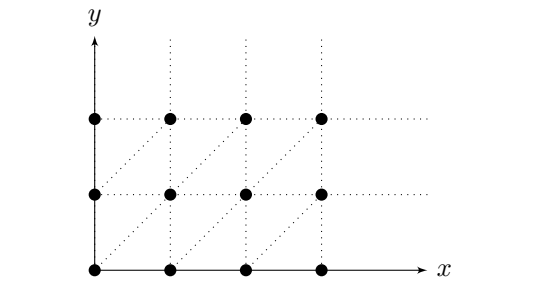
It remains to prove that $\mu' + t' \in \llbracket \nu' \rrbracket$, which we leave to the reader.

- finally, if for some $c \in \mathfrak{C}$, we have that $(\mu + t)(c) \leq M_c$, but no such clock is mapped to an integer value by $\mu + t$, then we distinguish two cases:
 - either for all $0 \leq u \leq t$, no valuation $\mu + u$ contains a clock c that is mapped to an integer less than or equal to M_c . This means that $\llbracket \mu + t \rrbracket = \llbracket \mu \rrbracket$, and taking $t' = 0$ is fine;
 - or for some $0 \leq u \leq t$, $\mu + u$ maps some clock c to an integer value less than or equal to M_c . Pick the largest such u . Then from the previous case, there exists u' such that $\llbracket \mu + u \rrbracket = \llbracket \mu' + u' \rrbracket$. We are now left with the case where we depart from a region $\llbracket \mu \rrbracket$ with $\mu(c)$ is an integer less than or equal to M_c and never visit such a region after a time elapse. This means that $t < \min\{1 - \mu(c) \mid c \in \mathfrak{C}\}$. From μ' , it suffices to apply a delay $t' < \min\{1 - \mu'(c) \mid c \in \mathfrak{C}\}$ to get to the same region. \square

The figure on the right is a schematic representation of \mathbf{M} -equivalence, for two clocks (for n clocks, it would be n -dimensional). It contains all the equivalence classes, namely:

- *punctual regions*;
- *1-dimensional bounded regions*, containing vertical or horizontal lines on the one hand, and diagonal lines on the other hand;
- *2-dimensional bounded regions*, which are triangular;

Figure 9: \mathbf{M} -equivalence for two clocks x and y , and maximal constants 3 and 2, resp.



- *unbounded regions*, which can be 1-dimensional or 2-dimensional.

Proposition 55. *Let $\mathbf{M} \in \mathbb{N}^{\mathfrak{C}}$. The number of equivalence classes of $\approx_{\mathbf{M}}$ is bounded by*

$$|\mathfrak{C}|! \cdot \prod_{c \in \mathfrak{C}} 4 \cdot (M_c + 1).$$

Proof. We provide another characterization of $\approx_{\mathbf{M}}$: with a valuation ν , we associate the following items:

- for each clock $c \in \mathfrak{C}$, the integral part $\lfloor \nu(c) \rfloor$ (or M_c if $\lfloor \nu(c) \rfloor > M_c$);
- for each clock $c \in \mathfrak{C}$, a bit telling whether $\nu(c)$ is an integer;
- the ordering of clocks according to their fractional parts;
- for each pair of consecutive clocks in that ordering, a bit telling whether their fractional parts are equal or not.

It is easily checked that any two valuations having the same characteristics on these four items are \mathbf{M} -equivalent. Moreover, the first item has $\prod_{c \in \mathfrak{C}} (M_c + 1)$ different values, the second and fourth have $2^{|\mathfrak{C}|}$, and the third one has $|\mathfrak{C}|!$. Hence the result. \square

Definition 56. *Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ be a timed automaton, with maximal constants \mathbf{M} . Let \mathcal{R} be a partition of the set of clock valuations that is compatible with the constraints in \mathcal{A} . The region automaton associated with \mathcal{A} and \mathcal{R} is the automaton $\mathcal{B} = \langle Q, R, l \rangle$ where*

- $Q = S \times \mathcal{R}$,
- $R \subseteq Q \times Q$ is such that $((s, r), (s', r')) \in R$ iff $((s, \nu), (s', \nu'))$ in $\llbracket \mathcal{A} \rrbracket$, with $\nu \in r$ and $\nu' \in r'$;
- $l(s, r) = \ell(s)$ for all $s \in S$ and $r \in \mathcal{R}$.

Proposition 57. *Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ be a timed automaton, and \mathcal{B} be its region automaton. Let (s, ν) be a state of $\llbracket \mathcal{A} \rrbracket$, and $(s, \llbracket \nu \rrbracket)$ be the corresponding state in \mathcal{B} . Then there exists a bisimulation (over the union of $\llbracket \mathcal{A} \rrbracket$ and \mathcal{B}) containing $((s, \nu), (s, \llbracket \nu \rrbracket))$.*

Proof. Consider the set

$$R = \{((t, \mu), (t, \llbracket \mu \rrbracket)) \mid \mu \models \text{Inv}(t)\}.$$

It is easily proved that this is a (time-abstract) bisimulation. \square

Theorem 58. *Reachability and repeated reachability in timed automata are PSPACE-complete, and can be solved in deterministic exponential time.*

Proof. The deterministic algorithm consists in building the region automaton, and check for (repeated) reachability in that automaton.

This algorithm can be adapted to run in PSPACE by working *on-the-fly*: the path for reaching the target state can be guessed step-by-step, without computing the whole region automaton. It requires polynomial space to store the current state, and to increment a counter for stopping the procedure in case the target state is not reached.

We now prove hardness in PSPACE for reachability (which will entail hardness for repeated reachability). Consider a non-deterministic linear-bounded Turing machine \mathcal{M} , and an input word w . Deciding whether \mathcal{M} accepts w is known to be PSPACE-complete. We reduce this problem to a reachability problem in a timed automaton.

We first fix our notations: we assume w.l.o.g. that \mathcal{M} works on a 2-letter alphabet $\{a, b\}$, with an extra blank symbol $\#$, and that its set of states is Q , and its set of transitions is R . We assume that it has one initial state q_i and one final q_f . Being linear-bounded for \mathcal{M} means that there is a linear function $S: \mathbb{N} \rightarrow \mathbb{N}$ s.t., on input w , \mathcal{M} uses at most $S(|w|)$ cells of the tape.

In our encoding, the content of each cell C_i of the tape is encoded by the value of a clock x_i when the extra clock t equals 0 and the timed automaton is in a special configuration (to be defined below): $x_i = 0$ will encode $\#$, $x_i \in (0, 2]$ will encode a , and $x_i > 2$ will encode b .

We now consider the timed automaton $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ s.t.

- $S = Q \times \{1, \dots, S(|w|)\} \times \{0, \dots, 2 \times S(|w|)\}$, where the last bit is 0 when the automaton is really in a state encoding a configuration of \mathcal{M} , and will have another value when it will have to “update” the values of the clocks (this will be necessary because our encoding is not invariant by time elapsing);
- $\mathfrak{C} = \{x_i \mid i \in \{1, \dots, S(|w|)\} \cup \{t\}\}$. Clock t is used as a tick: it will be reset when it reaches value 2;
- T is the set of transitions, to be defined below;
- ℓ is always empty (we will only use reachability properties of \mathcal{A});
- Inv is always true.

We now explain how we build the set of transitions: consider for instance a transition $(q, b, q', a, -1)$ (meaning that if in state q and reading an b , \mathcal{M} will write an a (over the b), move to the left (if possible), and go to state q'), and a position $i \in \{2, \dots, S(|w|)\}$. Then \mathcal{A} will have the following transitions:

- $((q, i, 0), (t = 1 \wedge x_i > 3), \{x_i\}, (q', i - 1, 1))$: the guard $t = 1 \wedge x_i > 3$ ensures that clock x_i encodes a b (because one time-unit earlier, when $t = 0$, it held $x_i > 2$); the fact that we reset x_i will enforce that next time we reset t (and go to a state really encoding a configuration of \mathcal{M}), we have $x_i \in (0, 2]$, thus encoding an a in cell C_i .
- then for each $j \neq i$, we have to take care of the values of x_j . We do this once when $t = 1$ (for a), and once when $t = 2$ (for $\#$). When $t = 1$, a 's are characterized by clock values 0 (in case the a has just be written on the tape) or in $(1, 3]$. In any case, resetting the corresponding clock will enforce that it is in $(0, 2]$ next time t is reset. Similarly, when $t = 2$, $\#$ are characterized by clock value 2. We reset clocks having this value, thus preserving our encoding. These sequences of updates are achieved using the sequences of states (q', i, j) where j ranges from 1 to $2 \times S(|w|)$ (but we don't modify clock x_i). When the last clock has been updated a second time, we reset t and go to state $(q', i, 0)$.

We leave it to the reader to prove that there is a weak bisimulation⁵ between the Turing machine and the automaton representing the semantics of timed automaton, and that reachability of a state $(q_f, i, 0)$ from $(q_i, 0, 0)$ in the timed automaton is equivalent to reachability of the final state from the initial state of the Turing machine. \square

Notice that this proof could be adapted to work with timed automata under a discrete-time semantics (but not for duration automata of Chapter 1, since we heavily use clocks here).

2.4 Language-theoretic properties of timed automata

Definition 59. Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ be a timed automaton over AP and alphabet Σ . The mixed-move semantics of \mathcal{A} , denoted with $\llbracket \mathcal{A} \rrbracket_m$, is the (infinite-state) automaton $\llbracket \mathcal{A} \rrbracket_m = \langle Q, R, l \rangle$ where

- $Q = \{(s, \nu) \in S \times \mathbb{T}^{\mathfrak{C}} \mid \nu \models \text{Inv}(s)\}$;
- R only contains mixed transitions: $((s, \nu), (d, \sigma), (s', \nu')) \in R$, with $d \in \mathbb{R}_{\geq 0}$ and $\sigma \in \Sigma$, when the classical semantics (as defined at Definition 49) contains a state $(s, \nu + d)$, a delay transition $((s, \nu), (s, \nu + d))$, and an action transition $((s, \nu + d), (s', \nu'))$ corresponding to σ ;
- $l((s, \nu)) = \ell(s)$ for all $(s, \nu) \in Q$.

Definition 60. Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ be a timed automaton, with maximal constants \mathbf{M} . Let \mathcal{R} be a partition of the set of clock valuations that is compatible with the constraints in \mathcal{A} . The mixed-move region automaton associated with \mathcal{A} and \mathcal{R} is the automaton $\mathcal{B} = \langle Q, R, l \rangle$ where

- $Q = S \times \mathcal{R}$,
- $R \subseteq Q \times \Sigma \times Q$ is such that $((s, r), \sigma, (s', r')) \in R$ iff $((s, \nu), (d, \sigma), (s', \nu'))$ in $\llbracket \mathcal{A} \rrbracket_m$, with $\nu \in r$ and $\nu' \in r'$;
- $l(s, r) = \ell(s)$ for all $s \in S$ and $r \in \mathcal{R}$.

⁵Weak meaning that we may have *silent* transitions, corresponding here to the sequence of transitions updating the values of the clocks.

We consider finite words generated by timed automata, which we extend with an initial and a final state to that purpose.

Definition 61. Let Σ be a finite alphabet. A timed word over Σ and \mathbb{T} is a (possibly infinite) sequence $(\sigma_i, t_i)_i$, where $\sigma_i \in \Sigma$ and $t_i \in \mathbb{T}$ for all i . We write $\mathfrak{W}(\Sigma, \mathbb{T})$ for the set of timed words over Σ and \mathbb{T} .

The untimed word of a timed word $w = (\sigma_i, t_i)_i$ is the word $\text{Untime}(w) = (\sigma_i)_i$. The untimed language of a language \mathcal{L} is the language $\text{Untime}(\mathcal{L}) = \{\text{untime}(w) \mid w \in \mathcal{L}\}$.

Definition 62. Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ be a timed automaton, with initial state s_i and final state s_f , and $\llbracket \mathcal{A} \rrbracket_m = \langle Q, R, l \rangle$ be its mixed-move semantics. The finite-word language of \mathcal{A} , which we denote with $\mathcal{L}_{\text{fw}}(\mathcal{A})$, is defined as

$$\mathcal{L}_{\text{fw}}(\mathcal{A}) = \{(\sigma_i, t_i)_i \in \mathfrak{W}(\Sigma, \mathbb{T}) \mid \exists (s_i, \nu_i)_{i \in [0, p]} \text{ s.t. } (s_0, \nu_0) = (s_i, \mathbf{0}_{\mathfrak{C}}) \text{ and } s_p = s_f \text{ and } ((s_j, \nu_j), (t_j, \sigma_j), (s_{j+1}, \nu_{j+1})) \in R \text{ for all } j\}.$$

Theorem 63. Let \mathcal{A} be a timed automaton, with initial state s_i and final state s_f . Then the finite-word language of the mixed-move region automaton (with initial state $(s_i, \mathbf{0}_{\mathfrak{C}})$ and final states $\{(s_f, r) \mid r \in \mathcal{R}\}$) is exactly the untimed language associated with $\mathcal{L}_{\text{fw}}(\mathcal{A})$.

Proof. Follows from the fact that the (mixed-move) region automaton is time-abstract bisimilar to the (mixed-move) semantics of the timed automaton: bisimulation implies trace-equivalence, provided that the initial and final states are in correspondence. This general fact is proved by induction on the length of the underlying runs. \square

Theorem 64. The class of finite-word languages accepted by timed automata are closed under (finite) union and intersection.

Proof. Closure under union is straightforward, by just taking the union of both timed automata.

Closure under intersection is proved by building the product of both timed automata: given two timed automata $\mathcal{A}_1 = \langle S_1, \mathfrak{C}_1, T_1, \ell_1, \text{Inv}_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, \mathfrak{C}_2, T_2, \ell_2, \text{Inv}_2 \rangle$, with initial and final states s_{i1}, s_{i2}, s_{f1} and s_{f2} , their product is the automaton $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ defined as follows:

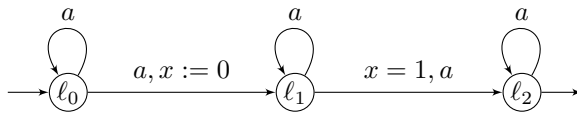
- $S = S_1 \times S_2$;
- $\mathfrak{C} = \mathfrak{C}_1 \cup \mathfrak{C}_2$ (assuming w.l.o.g. that \mathfrak{C}_1 and \mathfrak{C}_2 are disjoint);
- $((s_1, s_2), g, \sigma, R, (s'_1, s'_2)) \in T$ iff there exists transitions $(s_1, g_1, \sigma, R_1, s'_1) \in T_1$ and $(s_2, g_2, \sigma, R_2, s'_2) \in T_2$ s.t. $g = g_1 \wedge g_2$ and $R = R_1 \cup R_2$;
- $\text{Inv}((s_1, s_2)) = \text{Inv}(s_1) \wedge \text{Inv}(s_2)$ for all $(s_1, s_2) \in S_1 \times S_2$;
- the initial state is (s_{i1}, s_{i2}) and the final state is (s_{f1}, s_{f2}) .

It is straightforward to prove that a finite word is accepted by \mathcal{A}_1 and \mathcal{A}_2 iff it is accepted by \mathcal{A} . \square

Theorem 65. The class of languages accepted by a timed automaton is not closed under complement.

Proof. A classical witness of this fact is the language accepted by the automaton depicted on Figure 10. The proof is a bit tedious [AD94], though, and we provide another one [AM04], based on the automaton depicted on Figure 11.

Figure 10: A non-complementable timed automaton



The language accepted by the automaton on Figure 11 is

$$\{(\sigma_1, t_1) \cdots (a_k, t_k) \mid k \in \mathbb{N}, k > 0, \text{ and } \exists 1 \leq i \leq k. \sigma_i = a \text{ and } \forall i < j \leq k. \sum_{p=i+1}^j t_p \neq 1\}.$$

Figure 11: Another non-complementable timed automaton

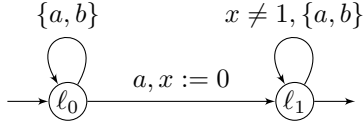
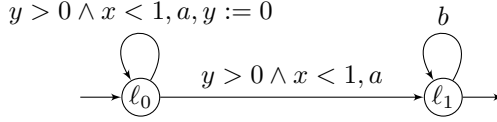


Figure 12: A timed automaton



Assume that some timed automaton \mathcal{B} accepts the complement of this language. The language of \mathcal{B} is the set of finite words in which any a if followed, one time unit later, by either an a or a b .

Now, consider the automaton \mathcal{C} depicted on Figure 12. The language of this automaton can be described as

$$\{(\sigma_i, t_i)_{i \in [0, k]} \mid \exists p. \forall j \leq p. (t_j > 0 \text{ and } \sigma_j = a) \text{ and } \sum_{i \leq p} t_i < 1 \text{ and } \forall p < j < k. \sigma_j = b\}.$$

The intersection of the languages accepted by \mathcal{B} and \mathcal{C} must then also be accepted by some timed automaton \mathcal{D} . The corresponding language contains exactly those words having a set of a 's in the first time unit, then a set of b 's, and each a is followed by a b one time unit later.

From Theorem 63, the corresponding untimed language is accepted by the region automaton of \mathcal{D} , which is a finite automaton. This is a contradiction, as it is well known that the untimed language, $\{a^n b^m \mid 1 \leq n \leq m\}$, cannot be accepted by a finite automaton. \square

Definition 66. A timed automaton is non-deterministic if it contains two transitions $(q, g_1, \sigma, R_1, q_1)$ and $(q, g_2, \sigma, R_2, q_2)$ s.t. $\llbracket g_1 \wedge g_2 \rrbracket \neq \emptyset$. It is deterministic otherwise.

Theorem 67. The class of deterministic timed automata is closed under complement.

Proof. Given a deterministic timed automaton, we can add a sink state, and from each state s , for each letter σ , for each valuation ν s.t. there is no σ -transition from (s, ν) , add a transition to the sink state. Obviously, this gives a deterministic timed automaton, whose complement language is recognized by the automaton obtained by swapping accepting and non-accepting states. Hence the result. \square

Corollary 68. Timed automata are not determinizable.

Theorem 69. Emptiness of the language of a timed automaton is decidable in PSPACE. Universality and inclusion are undecidable.

Proof. Emptiness corresponds to reachability of the accepting state, and can thus be checked in PSPACE.

We prove that universality (hence inclusion checking) is undecidable by reducing the halting problem of a two-counter machine \mathcal{M} [Min61] to the universality checking problem of a timed automaton.

We encode the executions of \mathcal{M} as a timed word $w = (\sigma_i, t_i)_i$. One configuration will be encoded as a 1-time-unit word, starting with letter q representing the current state of \mathcal{M} , followed by as many copies of letter c as the value of the first counter, followed by as many copies of letter d as the value of the second counter. The whole execution is encoded as the concatenation of the words encoding the successive configurations, with the following two conditions:

- in which we additionally require that no two events occur at the same time;
- between two consecutive configurations,
 - the evolution of the counters must correspond to the instruction of \mathcal{M} in the state of the first configuration;
 - in case the value of the first counter is unchanged, then each c in the first configuration must be followed by a c exactly one time unit later in the next one;

- in case it gets incremented, then each c in the first configuration has a corresponding c exactly one time unit later in the following configuration, and an extra c is added after those c in the second configuration;
- in case it gets decremented, the each c but the last one in the first configuration is followed by a c one time unit later in the following configuration;
- the same conditions apply for letter d and the second counter.

We now build a timed automaton \mathcal{A} which accepts all timed words that do *not* correspond to a correct encoding of a halting execution of \mathcal{M} , as described above. This timed automaton will be the conjunction of several timed automata, each responsible for checking that one of the conditions above is *not* fulfilled. Checking that the (untimed) word is not of the form $(Q \times \{c\}^* \times \{d\}^*)^*$, or that the final state of \mathcal{M} is not reached, is rather easy. Checking that two actions occur at the same time is also easy, as well as checking that there are two consecutive “state”-letters q and q' that are not 1-time-unit apart.

It remains to check that one of the instructions is not applied correctly. This can be either because the following state of \mathcal{M} is not the one indicated by the instruction, or because one of the counter is not modified correctly. The first condition is easily checked. We briefly explain the second one: several cases may correspond to a “failure”:

- either there is a c in some configuration (not the last one, in case of a decrementation) which is not followed by a c one time unit later;
- or there is a c in some configuration (not the last one in case of an incrementation) not preceded by a c one time unit earlier.

It is easy to build simple timed automata checking these conditions. The product of these automata is not universal iff there exists a correct encoding of a halting execution of \mathcal{M} , which is undecidable. \square

2.5 Timed temporal logics

2.5.1 Branching-time temporal logics

Theorem 70. *CTL model-checking on timed automata can be achieved in exponential time, and is PSPACE-complete.*

Proof. Since CTL is invariant under bisimulation, it suffices to check the value of the given formula on the region automaton. Doing this naively would require to store the intermediate results on the region automaton, hence involving exponential-size information to be stored. However, instead of storing the results of intermediate computations, it is possible to recompute them when they are needed. We omit the details, and refer to [HKV96] for more details. \square

Definition 71. *TCTL extends CTL as follows:*

$$\begin{aligned} TCTL \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \psi_s \mid \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= \phi_p \mathbf{U}_I \phi_p. \end{aligned}$$

where I is an interval with integral bounds (or $+\infty$). We define the semantics of the new “until” modality in the mixed-move semantics, along a run $\rho = ((s_i, \nu_i), (d_i, \sigma_i), (s'_i, \nu'_i))_i$ as follows

$$\begin{aligned} \mathcal{A}, \rho \models \phi_p \mathbf{U}_I \phi'_p &\Leftrightarrow \exists j \in [1, \text{length}(\rho)]. \mathcal{A}, \rho_{\geq j} \models \phi'_p \text{ and } \sum_{k=0}^j d_k \in I \text{ and} \\ &\forall i \in [1, j]. \mathcal{A}, \rho_{\geq i} \models \phi_p. \end{aligned}$$

Lemma 72. *Given two runs ρ and ρ' of an automaton \mathcal{A} with $\text{first}(\rho) = \text{first}(\rho')$, it holds*

$$\forall \phi_s \in TCTL. \quad \mathcal{A}, \rho \models \phi_s \Leftrightarrow \mathcal{A}, \rho' \models \phi_s.$$

Proof. By induction on the structure of ϕ_s . \square

Theorem 73 ([ACD93]). *TCTL model-checking on timed automata can be achieved in exponential time, and is PSPACE-complete.*

Proof. The first step in the proof consists in proving that two \mathbf{M} -equivalent configurations of a timed automaton satisfy the same set of TCTL state formulas. Assuming that it is the case for two subformulas ψ and ϕ , we prove that the result holds for $\mathbf{E}\phi \mathbf{U}_{\langle a, b \rangle} \psi$. To see this, we add an extra clock t to \mathcal{C} (which will be used to check the timing constraint $\langle a, b \rangle$), and consider \mathbf{M} -equivalence on the extended set of clocks, where we take b as maximal constant for t (or a if $b = +\infty$). Assume that there is a run from (s, ν) witnessing $\phi \mathbf{U}_{\langle a, b \rangle} \psi$. Equivalently, there is a run from (s, μ) , where μ extends ν with the value of t , with $\mu(t) = 0$. This run reaches a ψ -state with $\mu(t) \in \langle a, b \rangle$; Since $\mu \sim \mu'$, with μ' extending ν' with $\mu'(t) = 0$, this run can be simulated from (s, μ') . Since both runs visit the same regions, this second run reaches a ψ -state with $\mu'(t) \in \langle a, b \rangle$, which prove this intermediate result.

Our algorithm now is similar to the one for CTL, except that we have to consider this extra clock for each subformula to be checked. It can be implemented in a space-efficient way, requiring only polynomial space. \square

2.5.2 Linear-time temporal logics

Definition 74. *Let Σ be a finite alphabet. A timed word over Σ and \mathbb{T} is a (possibly infinite) sequence $(\sigma_i, t_i)_i$, where $\sigma_i \in \Sigma$ and $t_i \in \mathbb{T}$ for all i . We write $\mathfrak{W}(\Sigma, \mathbb{T})$ for the set of timed words over Σ and \mathbb{T} .*

A signal is a mapping $s: D \rightarrow \Sigma$, where D is a (possibly infinite) sub-interval of \mathbb{T} containing 0. We write $\mathfrak{S}(\Sigma, \mathbb{T})$ for the set of signals over Σ and \mathbb{T} .

Definition 75. *Let $\mathcal{A} = \langle S, \mathcal{C}, T, \ell, \text{Inv} \rangle$ be a timed automaton, $s \in S$, and $\llbracket \mathcal{A} \rrbracket_m = \langle Q, R, l \rangle$ be its mixed-move semantics. The discrete-trace semantics of \mathcal{A} from s , which we denote with $\llbracket \mathcal{A} \rrbracket_d(s)$, is defined as*

$$\llbracket \mathcal{A} \rrbracket_d(s) = \{ (\sigma_i, t_i)_i \in \mathfrak{W}(\Sigma, \mathbb{T}) \mid \text{there exists a maximal trace } (s_i, \nu_i)_i \\ \text{s.t. } (s_0, \nu_0) = (s, \mathbf{0}_{\mathcal{C}}) \text{ and } ((s_j, \nu_j), (t_j, \sigma_j), (s_{j+1}, \nu_{j+1})) \in R \text{ for all } j \}.$$

The continuous-trace semantics of \mathcal{A} , denoted with $\llbracket \mathcal{A} \rrbracket_c(s)$, is defined as

$$\llbracket \mathcal{A} \rrbracket_c(s) = \left\{ f \in \mathfrak{S}(2^{AP}, \mathbb{T}) \mid \text{there exists a maximal trace } ((s_j, \nu_j), (t_j, \sigma_j), (s'_j, \nu'_j))_j \text{ s.t.} \right. \\ \left. (s_0, \nu_0) = (s, \mathbf{0}_{\mathcal{C}}) \text{ and } f(t) = \ell(s_{i(t)}), \text{ where } i(t) \text{ is the least index } i \text{ for which } t \geq \sum_{j=1}^i t_j \right\}.$$

Definition 76. *MTL extends LTL as follows:*

$$\text{MTL } \exists \phi_s ::= \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p ::= p \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \mathbf{U}_I \phi_p.$$

Along a run $\rho = ((s_i, \nu_i), (\sigma_i, t_i), (s'_i, \nu'_i))$ (or, equivalently, along a timed word generated by \mathcal{A}), the semantics of the new “until” modality can be defined as follows⁶:

$$\mathcal{A}, \rho \models \phi_p \mathbf{U}_I \phi'_p \quad \Leftrightarrow \quad \exists j \in [1, \text{length}(\rho)]. \mathcal{A}, \rho_{\geq j} \models \phi'_p \text{ and } \sum_{k=0}^j d_k \in I \text{ and} \\ \forall i \in [1, j]. \mathcal{A}, \rho_{\geq i} \models \phi_p.$$

Along a signal $f: D \rightarrow 2^{AP}$, the semantics is defined as

$$\mathcal{A}, f \models \phi_p \mathbf{U}_I \phi'_p \quad \Leftrightarrow \quad \exists t \in (D \setminus \{0\}) \cap I. \mathcal{A}, f_{\geq t} \models \phi'_p \text{ and } \forall t' \in (0, t). \mathcal{A}, f_{\geq t'} \models \phi_p$$

where $f_{\geq t}(u) = f(t+u)$ for all $u \geq 0$ s.t. $t+u \in D$.

The semantics of LTL over timed words and signals is obtained by seeing LTL as a fragment of MTL where $I = (0, +\infty)$.

⁶Notice that, strictly speaking, in this version of the semantics of MTL, atomic propositions should be replaced by letters from Σ , since timed words are based on this alphabet.

Theorem 77. *LTL model-checking on timed automata is decidable in exponential time, and is PSPACE-complete.*

Proof. From the LTL formula, build the corresponding (co)Büchi automaton. Depending on the semantics, this automaton can be interpreted on timed words (synchronization on transitions) or on signals (synchronization on states). Then check for the existence of an accepting run in the product of this automaton with the original timed automaton, *via* the region automaton. This can be achieved *on-the-fly*, hence requiring only polynomial space. \square

Theorem 78 ([AH93]). *MTL model-checking on timed automata is undecidable.*

Proof. We first do the proof for the continuous-trace semantics. It is achieved by encoding the executions of a (deterministic) two-counter machines, in a way similar to the undecidability proof of Theorem 69: a configuration of the two-counter machine, containing the current state q and the values of both counters c_1 and c_2 , is encoded on a one-time-unit-long signal where an atomic proposition q holds continuously along that signal, and atomic propositions a_1 and a_2 hold punctually, with as many different occurrences as the values of the counters c_1 and c_2 . We also assume a special atomic proposition t to hold precisely at the beginning of each such signal. A sequence of such signals represents an execution of the two-counter machine if the transitions are applied correctly, which we will enforce by saying that all (but possibly the last) occurrences of c_1 and c_2 are followed, exactly one time unit later, with an occurrence of the same letter. This will enforce that the values of the counters are preserved, except possibly for the one that has to be updated.

We won't write the whole set of formulas required to ensure this encoding, but only some of them:

- t holds punctually at each integer date:

$$t \wedge \neg a_1 \wedge \neg a_2 \wedge \mathbf{G}(t \iff (\neg t) \mathbf{U}_{=1} (t \wedge \neg a_1 \wedge \neg a_2)) \quad (2)$$

- there is always exactly one state-letter at a time:

$$\mathbf{G} \left(\bigvee_{q \in Q} q \wedge \bigwedge_{q \neq q'} (\neg q \vee \neg q') \right) \quad (3)$$

- state-letters hold continuously, and can change only at integer dates:

$$\bigwedge_{q \in Q} \mathbf{G}(q \Rightarrow (q \wedge \neg t) \mathbf{U} t) \quad (4)$$

- a_1 and a_2 do not overlap:

$$\mathbf{G}(\neg a_1 \vee \neg a_2) \quad (5)$$

- occurrences and absences of a_1 (resp. a_2), except the last ones, are repeated one time unit later:

$$\mathbf{G}[(a_1 \wedge \neg(a_1 \mathbf{U}(\neg a_1 \wedge \neg a_1 \mathbf{U} t))) \Rightarrow \mathbf{F}_{=1} a_1] \wedge \mathbf{G}[(\neg a_1 \wedge \neg(\neg a_1 \mathbf{U} t)) \Rightarrow \mathbf{F}_{=1} \neg a_1] \quad (6)$$

The same formula can be written for a_2 ;

- transitions are applied correctly: assume that there is a transition of the form
 $q: \text{ if } c_1 > 0 \text{ then } c_1 := c_1 - 1; \text{ goto } q' \text{ else goto } q''$

This could be encoded as follows:

$$\mathbf{G} \left[(t \wedge q \wedge (\neg t \mathbf{U} a_1)) \Rightarrow (\mathbf{F}_{=1} q' \wedge \mathbf{F}_{<1} (\neg a_1 \wedge (\neg a_1 \wedge \neg t) \mathbf{U} [a_1 \wedge (a_1 \wedge \neg t) \mathbf{U} (\neg a_1 \wedge (\neg a_1 \wedge \neg t) \mathbf{U} t)]) \wedge \mathbf{F}_{=1} (\neg a_1 \wedge \neg a_1 \mathbf{U} t)) \wedge \mathbf{G}_{<1} (a_2 \iff \mathbf{F}_{=1} a_2) \right] \quad (7)$$

and

$$\mathbf{G} \left[(t \wedge q \wedge (\neg a_1 \mathbf{U} t)) \Rightarrow (\mathbf{F}_{=1} q'' \wedge \mathbf{G}_{<1} (a_1 \iff \mathbf{F}_{=1} a_1) \wedge \mathbf{G}_{<1} (a_2 \iff \mathbf{F}_{=1} a_2)) \right] \quad (8)$$

Other instructions can be handled similarly.

It then suffices to express that the halting state is reachable to fully encode the halting problem of the two-counter machine. This proves that the *satisfiability* of an MTL formula is undecidable. When applied to a universal timed automaton (such an automaton is easily constructed), this proves that model-checking MTL is also undecidable.

We adapt the same construction for proving undecidability in the discrete-trace semantics: instead of holding continuously, state-propositions will be required to hold at the beginning of their one-time-unit interval. It is possible to adapt the above formulas to this new setting, except one: we cannot enforce that the *absence* of a_1 is propagated, because we cannot evaluate formulas between two letters of the timed word. In other terms, we can guarantee that an a_1 is propagated, but we cannot prevent *new* occurrences of a_1 , without a corresponding a_1 one time unit earlier.

One way to cope with this problem is to have past-time modalities. The other way, without changing the logic, is a bit more tricky, and we just give a rough idea. First notice that it is easy to adapt the encoding above to handle three-counter machines. Now, given a two-counter machine, we add a third counter c_3 which gets incremented every other step. If there is an halting computation, then there is one where the maximal value of $c_1 + c_2 + c_3$ is bounded by some value M . On the contrary, if there is no halting computation, then for any bound M on $c_1 + c_2 + c_3$, any computation eventually reaches that bound (there can be no loop since c_3 only increases). The idea is to simulate the computation of the three-counter machine with different values of M , starting from 1.

More precisely, for a given value M , we initialize the simulation by labelling M positions on the first time unit with a special letter z . Any incrementation of c_1 or c_2 will have to replace one occurrence of z with a_1 or a_2 . When c_1 or c_2 is decreased, we replace one letter a_1 or a_2 with the letter a_3 encoding the value of counter c_3 , which gets incremented at the next instruction. Hence, in case there are no “insertion errors”, the number of z eventually reaches 0. In that case, we replace all occurrences of a_1 , a_2 and a_3 with z ’s, except one which is transformed into another special letter y , and restart the simulation with those z ’s as new marked positions. In case there are no z left, we replace all y ’s with z ’s, add one occurrence of z , and restart the process. Notice that there can be insertions of extra z ’s and y ’s all along the computation. Also notice that all these rules are “local”, and can be encoded using MTL formulas.

Now, assume that the three-counter machine does not halt. Then for any value M (*i.e.*, for any number of z at the beginning of the simulation phase), the simulation of the machine *without insertion error* eventually exhausts all z ’s and reaches a configuration containing only y ’s. Hence there is an infinite computation visiting configurations with only y ’s infinitely many times (which can be expressed in MTL).

We now prove the other direction: assume that there is a computation visiting infinitely many configurations having only y ’s, and that the three-counter machine halts, with a bound M_0 on the maximal value of the sum of the three counters. Since there are infinitely many configurations with only y ’s, and the number of y ’s and z ’s only increases, there must be one simulation containing initially at least M_0 marked positions. Moreover, between one simulation and the next one, the number of z ’s either increase, or decreases by 1 (in case of a simulation with no insertion errors). Since we visit infinitely many configurations with no z , there must be a simulation starting with at least M_0 marked positions, and decreasing the number of z by 1 (hence it is an exact simulation). But such a simulation must halt, since the (deterministic) three-counter machine halts when the sum of the three counter is bounded by M_0 . This contradicts the fact that we visit configurations with only y ’s infinitely many times. \square

Definition 79. *MITL is the fragment of MTL where the constraint intervals are not allowed to be singular.*

Theorem 80 ([AFH96]). *MITL model-checking on timed automata is decidable in doubly-exponential time, and is EXPSPACE-complete.*

Proof. The proof is very technical: it consists in transforming the formula into a timed automaton (which cannot be done in general for MTL ⁷). We refer to [AFH96] for a detailed proof. \square

2.6 Algorithmic issues

Definition 81. *Let \mathcal{C} be a finite set of clocks. A zone is the solution of a clock constraint in $\text{Constr}(\mathcal{C})$.*

⁷This fact directly follows from the fact that timed automata are not closed under negation. This also proves that MITL is strictly less expressive than MTL.

Notice that clock constraints only allow conjunction of simple constraints, so that zones represent convex sets.

Despite the fact that a region is a zone, checking reachability based on the region graph turns out to be computationally more expensive than the zone-based approach in practice. Zones have the advantage of being able to represent a union of regions, hence allowing to handle several regions at a time, which makes the algorithms more efficient (again, this is only a practical argument, and the theoretical complexity of checking reachability in the zone graph would be more expensive *in the worst case*).

Algorithm 2: Symbolic on-the-fly reachability in timed automata, checking whether l_f is reachable

```

1 PASSED = ∅, WAIT = { ⟨l0, D0⟩ }
2 while WAIT ≠ ∅ do
3   take ⟨l, D⟩ from WAIT
4   if (l = lf) then return "YES"
5   if D ⊈ D' for all ⟨l, D'⟩ ∈ PASSED then
6     add ⟨l, D⟩ to PASSED
7     for all ⟨l, D⟩ such that ⟨l, D⟩ → ⟨l', D'⟩ do
8       add ⟨l', D'⟩ to WAIT
9     end for
10  end if
11 end while
12 return "NO"

```

Algorithm 2 is an on-the-fly reachability algorithm, computing the set of reachable zones until the target state has been reached. Notice that it does not compute the whole zone-graph, but computes the set of successors of the zone being investigated. Notice also that the way we select the zone to be processed (line 3) is not explicitly mentioned, meaning that the algorithm can do depth-first search or breadth-first search (or random search). That the algorithm is correct is rather obvious. Termination is ensured by finiteness of the number of zones (given the set \mathbf{M} of maximal constants of the automaton).

The important question now is to efficiently handle zones (intersection, union, inclusion testing, ...). This is achieved using DBMs [Bel57, Dil90, LLPY97].

Definition 82. Let \mathcal{C} be a finite set of clocks. Let $\mathcal{C}_0 = \mathcal{C} \cup \{0\}$. A difference bound matrix (DBM) is a mapping $(\mathbb{N} \times \{<, \leq\} \cup \{\infty\})^{\mathcal{C}_0^2}$.

Proposition 83. Any zone Z built on \mathcal{C} can be represented as a DBM.

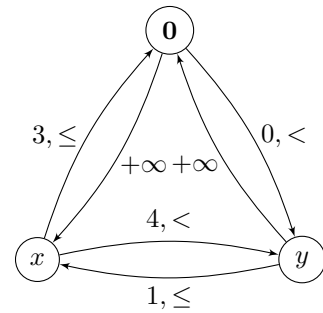
Proof. For each pair of clocks (x, y) , the constraints in Z involving x and y can always be written as $m_{x,y} \prec_m x - y \prec_M M_{x,y}$, with $m_{x,y}$ and $M_{x,y}$ in $\mathbb{N} \cup \{+\infty\}$ and \prec_m and \prec_M are in $\{<, \leq\}$. Then the $\langle x, y \rangle$ -cell of the matrix contains $(M_{x,y}, \prec_M)$, and the $\langle y, x \rangle$ -cell contains $(-m_{x,y}, \prec_m)$.

The case of constraints involving only one clock is handled by using the extra clock 0 , seen as a special clock whose value is always zero. \square

Example. The zone $(x \leq 3 \wedge y > 0 \wedge -1 \leq x - y < 4)$ is encoded by the following DBM (the graph representation on the right will be useful later):

$$\begin{pmatrix} 0, \leq & +\infty & 0, < \\ 3, \leq & 0, \leq & 4, < \\ +\infty & 1, \leq & 0, \leq \end{pmatrix}.$$

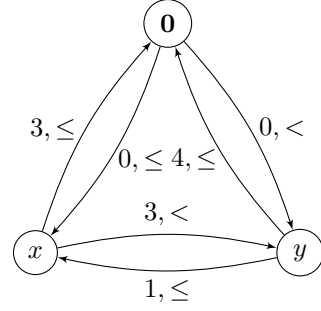
Notice that the values on the diagonal are not very relevant. We usually safely set them to $(0, \leq)$.



Proposition 84. DBMs admit a canonical form.

Proof. Several DBMs can represent the same zone: for instance, the zone of previous example is not tight, in that the constraints $x \leq 3$ and $y > 0$ imply $x - y < 3$, which is stronger than $x - y < 4$.

In order to build the tightest constraint, we represent DBMs as a weighted complete graph, whose nodes are the clocks in \mathcal{C}_0 and in which each transition (x, y) carries a weight $(n, <)$ if the DBM contains $(n, <)$ in cell (x, y) . Such a graph is depicted on the right of the above example (self-loops, labelled with $(0, \leq)$, have been omitted). It is then a matter of computing all-pairs shortest paths in order to get the tightest constraints. For instance, the edge from x to y can be shortened *via* node $\mathbf{0}$, yielding $(3, <)$. Similarly, the edge from $\mathbf{0}$ to x can be shortened into $(1, <)$, and the edge from $\mathbf{0}$ to y into $(4, \leq)$. This results in a *unique* tightest DBM equivalent to the initial one. Using the Floyd-Warshall algorithm, this can be achieved in time $O(|\mathcal{C}|^3)$.



Notice also that we require that clocks have nonnegative values. In the example above, the tightest constraints are $(0 \leq x \leq 3) \wedge (0 < y \leq 4) \wedge (-1 \leq x - y < 3)$. This is represented by the following canonical DBM:

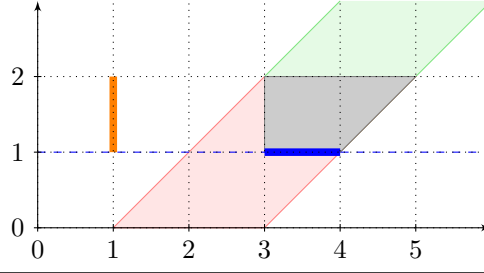
$$\begin{pmatrix} 0, \leq & 0, \leq & 0, < \\ 3, \leq & 0, \leq & 3, < \\ 4, \leq & 1, \leq & 0, \leq \end{pmatrix}.$$

□

We are now able to perform the following operations on DBMs:

- **[emptiness checking]** if the DBM is given in normal form, emptiness is equivalent to having an upper bound smaller than a lower bound (equivalently, this corresponds to having a negative self-loop on one of the nodes). This can be detected during all other operations, and can be encoded by setting the upper left cell of the matrix to $(-1, \leq)$.
- **[inclusion checking]** for two DBMs in normal form, inclusion of D into D' is equivalent to having all constraints in D stronger than (*i.e.*, less than or equal to) those of D' .
- **[intersection]** adding an extra conjunctive constraint $x_i - x_j < b$ to a DBM D in normal form can be achieved by first checking whether this constraint is not weaker than the current one (*i.e.*, if $(b, <)$ is strictly less than $D_{i,j}$). If not, we replace $D_{i,j}$ with $(b, <)$. This requires turning back the new DBM into normal form.
- **[future]** given a DBM, the aim is to compute the future on the corresponding zone. This is achieved by setting all the upper bounds on individual clocks (*i.e.*, cells $(x, \mathbf{0})$) to $+\infty$. It can be checked that this preserves normal form.
- **[past]** symmetrically, to get the past of a zone given as a DBM, it suffices to replace all lower bounds on single clocks (*i.e.*, cells $(\mathbf{0}, x)$) to $(0, \leq)$. This requires turning back the new DBM into normal form.
- **[reset]** we implement updates of the form $x := m$. From D in normal form, we replace $D(x, y)$ with $D(\mathbf{0}, y) + (m, \leq)$ and $D(y, x)$ with $D(y, \mathbf{0}) + (-m, \leq)$. This can be checked to preserve normal form.
- **[M -normalization]** it is not necessary to keep track of exact clock values when they are larger than their maximal constants. Given a maximal constant M in an automaton having no diagonal constraints, M -normalization consists in
 - replacing constraints $(m, <)$ that are weaker than (M, \leq) with $+\infty$. Intuitively, it means that the information that a clock is less than something too big does not give any information.
 - replacing constraints $(m, <)$ that are stronger than $(-M, <)$ with $(-M, \leq)$.
 - normalizing the resulting DBM.

Figure 13: Transformations on DBMs



Example. Consider the zone given by the DBM defined by $(3 \leq x \leq 5) \wedge (1 \leq y \leq 2) \wedge (x - y \leq 3)$. This corresponds to the grey zone depicted on Fig. 13, and to the following normal form DBM:

$$M = \begin{pmatrix} 0, \leq & -3, \leq & -1, \leq \\ 5, \leq & 0, \leq & 3, \leq \\ 2, \leq & -1, \leq & 0, \leq \end{pmatrix}$$

Intersection with the constraint $y \leq 1$ (i.e., $y - \mathbf{0} \leq 1$) is achieved as follows: this new constraint is stronger than the current one, which is $y - \mathbf{0} \leq 2$. Replacing this previous constraint with the new one gives the following DBM:

$$\begin{pmatrix} 0, \leq & -3, \leq & -1, \leq \\ 5, \leq & 0, \leq & 3, \leq \\ 1, \leq & -1, \leq & 0, \leq \end{pmatrix}$$

It can be checked that this DBM is not in normal form, as the constraint $x - \mathbf{0} \leq 5$ can be strengthened to $x - \mathbf{0} \leq 4$. Hence the normal form DBM is

$$\begin{pmatrix} 0, \leq & -3, \leq & -1, \leq \\ 4, \leq & 0, \leq & 3, \leq \\ 1, \leq & -1, \leq & 0, \leq \end{pmatrix}.$$

This corresponds to the solid blue zone on Fig. 13.

The future of the original DBM M is obtained by replacing all upper bounds (i.e., values in the first column of the DBM) with $+\infty$. We get

$$\begin{pmatrix} 0, \leq & -3, \leq & -1, \leq \\ +\infty & 0, \leq & 3, \leq \\ +\infty & -1, \leq & 0, \leq \end{pmatrix}$$

which is in normal form. The corresponding zone is depicted in green on Fig. 13 (and contains the initial zone M).

The past of M is obtained by replacing lower bounds of single-clock constraints with $(0, \leq)$. We get

$$\begin{pmatrix} 0, \leq & 0, \leq & 0, \leq \\ 5, \leq & 0, \leq & 3, \leq \\ 2, \leq & -1, \leq & 0, \leq \end{pmatrix}$$

Again, this is not in normal form, as the constraint $\mathbf{0} - x \leq 0$ can be strengthened to $\mathbf{0} - x \leq -1$. After normalization, we get

$$\begin{pmatrix} 0, \leq & -1, \leq & 0, \leq \\ 5, \leq & 0, \leq & 3, \leq \\ 2, \leq & -1, \leq & 0, \leq \end{pmatrix}.$$

The corresponding zone is depicted in red (and also contains the original zone).

Finally, resetting x to 1 can be achieved in two ways. The naive way is to impose $1 \leq x - \mathbf{0} \leq 1$ and removing all constraints on x . This gives:

$$\begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ 1, \leq & 0, \leq & +\infty \\ 2, \leq & +\infty & 0, \leq \end{pmatrix}$$

which obviously is not in normal form. Normalizing this DBM yields:

$$\begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ 1, \leq & 0, \leq & 0, \leq \\ 2, \leq & 1, \leq & 0, \leq \end{pmatrix}.$$

The more clever algorithm proposed above consists in modifying all the entries in the row and column corresponding to x . This gives the following DBM:

$$\begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ 1, \leq & 0, \leq & 0, \leq \\ 2, \leq & 1, \leq & 0, \leq \end{pmatrix}$$

which is directly in normal form (and hopefully corresponds to the previous result). This zone is depicted in orange.

2.7 Timed games

2.7.1 Control-oriented timed games

Definition 85 ([MPS95]). A 2-player timed game is a 6-tuple $\mathcal{G} = \langle S, \mathfrak{C}, T_1, T_2, \ell, \text{Inv} \rangle$ s.t. $\langle S, \mathfrak{C}, T_1 \cup T_2, \ell, \text{Inv} \rangle$ is a timed automaton.

Transitions in T_1 belong to Player 1, while those in T_2 belong to Player 2 (they are said to be uncontrollable). A run in a timed game is a run in the underlying automaton. Given a finite run ρ , we write $\text{last}(\rho)$ for the configuration reached after the last transition of ρ .

Definition 86. A strategy for Player 1 is a mapping σ_1 associating with each finite run ρ :

- either a transition $t \in T_1$ that is fireable from $\text{last}(\rho)$;
- or the special symbol λ (meaning “wait”).

Definition 87. Let \mathcal{G} be a timed game, s be a state of \mathcal{G} , and σ_1 be a strategy for Player 1. The set of compatible runs with σ_1 from s is defined inductively as follows:

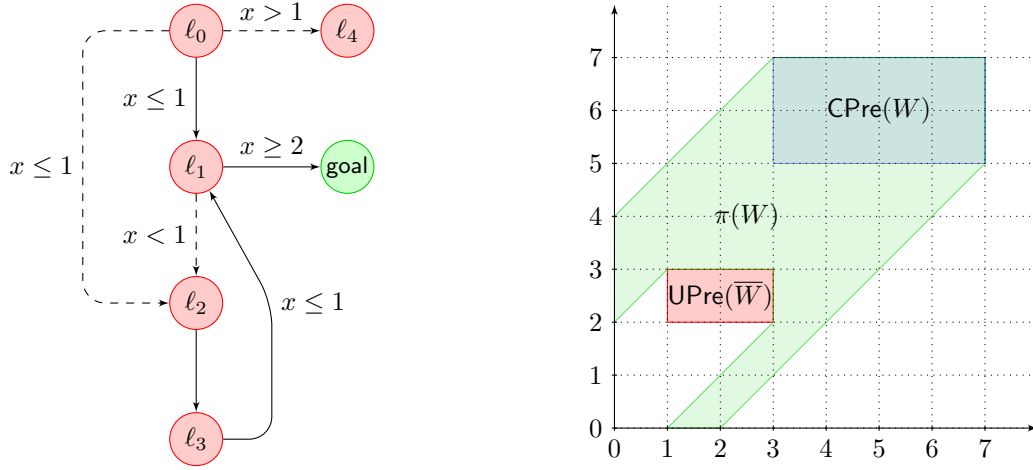
- $(s, \mathbf{0}_{\mathfrak{C}})$ is a compatible run (assuming that $\mathbf{0}_{\mathfrak{C}} \models \text{Inv}(s)$);
- if ρ is a compatible run ending in $\text{last}(\rho) = (q, \nu)$, and there is a transition $t = (q, g, \sigma, R, q') \in T_1 \cup T_2$ s.t. $\nu \models g$ and $\nu[R \rightarrow 0] \models \text{Inv}(q')$, then $\rho \cdot ((q, \nu), (q', \nu[R \rightarrow 0]))$ (or $((q, \nu), (q', \nu[R \rightarrow 0]))$ if ρ is a single state) is a compatible run iff
 - either $t \in T_2$;
 - or $\sigma_1(\rho) = t \in T_1$.
- if ρ is a compatible run ending in $\text{last}(\rho) = (q, \nu)$, and $d \in \mathbb{T}$ is s.t. $\nu \models \text{Inv}(q)$ and $\nu + d \models \text{Inv}(q)$, and $\sigma_1(\rho \cdot ((q, \nu), (q, \nu + d))) = \lambda$ for all $0 \leq d' < d$, then $\rho \cdot ((q, \nu), (q, \nu + d))$ is a compatible run.
- an infinite run ρ is compatible with σ_1 iff all its finite prefixes are.

Example. Figure 14 displays a timed game: solid transitions are controlled by Player 1, while dashed transitions belong to Player 2. A possible (memoryless) strategy for Player 1 is the following:

- in ℓ_0 : if $x < 0.5$, then wait; if $x = 0.5$, then go to ℓ_1 ; otherwise, wait;
- in ℓ_1 : if $x \leq 2$, then wait; if $x = 2$, then go to goal; otherwise, wait;
- in ℓ_2 : go to ℓ_3 ;
- in ℓ_3 : go to ℓ_1 .

Possible outcomes include the run going from ℓ_0 to ℓ_1 at date 0.5, then to goal at date 2, but also the run going from ℓ_0 to ℓ_2 at date 0.3 and then looping, without delays, between ℓ_2 , ℓ_3 and ℓ_1 . The latter example shows that our strategy is not winning for the objective of reaching the goal state.

Figure 14: Example of a timed game, and computation of $\pi(W)$



Definition 88.

Problem: *Reachability strategy in timed games*

Input: *A timed game \mathcal{G} , and two states s and s' ;*

Question: *Is there a strategy for Player 1 that ensures reaching s' from s*

Definition 89. Let $\mathcal{G} = \langle S, \mathcal{C}, T_1, T_2, \ell, \text{Inv} \rangle$ be a timed game. Given two sets of configurations W and W' of \mathcal{G} , we define

$$\begin{aligned} \text{CPre}(W) &= \{(s, \nu) \in S \times \mathbb{T}^{\mathcal{C}} \mid \exists t = (s, g, \sigma, R, s') \in T_1. \\ &\quad \nu \models g \wedge \text{Inv}(s) \wedge \nu[R \rightarrow 0] \models \text{Inv}(g') \wedge (s, \nu[R \rightarrow 0]) \in W\} \end{aligned}$$

$$\begin{aligned} \text{UPre}(W) &= \{(s, \nu) \in S \times \mathbb{T}^{\mathcal{C}} \mid \exists t = (s, g, \sigma, R, s') \in T_2. \\ &\quad \nu \models g \wedge \text{Inv}(s) \wedge \nu[R \rightarrow 0] \models \text{Inv}(g') \wedge (s, \nu[R \rightarrow 0]) \in W\} \end{aligned}$$

$$\text{Pre}_t(W, W') = \{(s, \nu) \in S \times \mathbb{T}^{\mathcal{C}} \mid \exists d \in \mathbb{T}. (s, \nu + d) \in W \wedge \forall 0 \leq d' \leq d. (s, \nu + d') \notin W'\}.$$

Lemma 90. Let $\mathcal{G} = \langle S, \mathcal{C}, T_1, T_2, \ell, \text{Inv} \rangle$ be a timed game, and $\mathbf{M} = (M_c)_{c \in \mathcal{C}}$ be the maximal constants for all clocks in \mathcal{C} . If W and W' are unions of regions for \mathbf{M} , then so are $\text{CPre}(W)$, $\text{UPre}(W)$ and $\text{Pre}_t(W, W')$.

Proof. Take $\nu \in \text{CPre}(W)$, and $\nu' \in \llbracket \nu \rrbracket$. Then obviously ν' satisfies the constraints for being in $\text{CPre}(W)$. Similarly for the other two operators. \square

Definition 91. We define $\pi(W) = W \cup \text{Pre}_t(\text{CPre}(W), \text{UPre}(\overline{W}))$.

Proposition 92. Let $\mathcal{G} = \langle S, \mathcal{C}, T_1, T_2, \ell, \text{Inv} \rangle$ be a timed game, and W be a set of winning configurations. Then for any $w \in \pi(W)$, there is a winning strategy, that forces the game to reach W after one transition.

Proof. First, if $w \in \text{CPre}(W)$, then the result follows by playing the transition witnessing the fact that w belongs to $\text{CPre}(W)$.

Now, if $w \in \pi(W)$, a winning strategy consists in waiting until the game reaches $\text{CPre}(W)$, and then play the witnessing transition as above. Since we do not visit $\text{UPre}(\overline{W})$ during the time we wait, Player 2 cannot take the game to \overline{W} in the meantime, hence the strategy is winning. \square

Proposition 93. Let $\mathcal{G} = \langle S, \mathcal{C}, T_1, T_2, \ell, \text{Inv} \rangle$ be a timed game, and Goal be a union of regions. Then the sequence $(\pi^i(\text{Goal}))_i$ converges in a finite number of steps, and contains exactly the set of configurations from which Player 1 has a winning strategy to reach Goal .

Proof. Obviously, π is non-decreasing. By Lemma 90, $\pi^i(\text{Goal})$ is a union of regions. Since there are finitely many regions, the sequence converges. By induction, it is easily proved that a state is in some $\pi^i(\text{Goal})$ iff there is a strategy leading to Goal within at most i steps, which proves the result. \square

Theorem 94. *Reachability strategy in timed games is decidable in deterministic exponential time, and is EXPTIME-complete.*

Proof. The exponential-time algorithm is obtained by building the region automaton and effectively building the sequence of Proposition 93. Hardness in EXPTIME can be proved by encoding an alternating linear-bounded Turing machine, in the same way as we encoded a linear-bounded Turing machine as a reachability problem for plain timed automata. We leave the details to the reader. \square

2.7.2 Game-theoretic approach to timed games

We now present an alternative semantics for timed games [dAFH⁺03], which is more *symmetric* and really involves the second player.

Definition 95. *A 2-player timed game is a 6-tuple $\mathcal{G} = \langle S, \mathfrak{C}, T_1, T_2, \ell, \text{Inv} \rangle$ s.t. $\langle S, \mathfrak{C}, T_1 \cup T_2, \ell, \text{Inv} \rangle$ is a timed automaton, and with the following constraint: the alphabet Σ contains a special letter λ , and for each state s , there is a transition $(s, \text{true}, \lambda, \emptyset, s)$ in T_1 and in T_2 .*

A run of a timed game is a run in the underlying timed automaton with the *mixed-move semantics*.

Definition 96. *A move for Player k from a configuration (s, ν) is a pair (d, a) where d is a nonnegative real s.t. $\nu + d \models \text{Inv}(s)$, and a is*

- either a transition $t \in T_1$ that is firable from $(s, \nu + d)$;
- or the special symbol λ (meaning “wait”).

A strategy for Player k is a mapping associating a move to each finite run.

Definition 97. *Let (s, ν) be a configuration of a timed game \mathcal{G} . Let (d_1, a_1) and (d_2, a_2) the moves of Players 1 and 2, as given by their strategy. Depending on the relative values of d_1 and d_2 , these moves give rise to the following successor state:*

- if $d_1 < d_2$, then there is one successor, obtained by applying transition a_1 from $(s, \nu + d_1)$;
- if $d_2 < d_1$, then there is one successor, obtained by applying transition a_2 from $(s, \nu + d_2)$;
- if $d_1 = d_2$, then both successors defined above are possible successors (which will introduce non-determinism).

A run $\rho = (s_i, \nu_i)_i$ is compatible with a strategy μ_j of Player j if for all i , there exists a move (d, a) for Player $3 - j$ s.t. (s_{i+1}, ν_{i+1}) is a successor state of (s_i, ν_i) for moves (d, a) and $\mu_j(\rho_{\leq i})$.

In this setting, we require that the set of winning runs for a player contains only time-diverging infinite runs. Forcing finite-time infinite runs is never winning.

Definition 98. *Let μ_i be a strategy for Player i , and ρ be a compatible outcome. Player i is blamed at step j along ρ if the j -th transition along ρ corresponds to the transition indicated by $\mu_i(\rho_{\leq j})$. Then strategy μ_i is winning for Player i if, along all the outcomes,*

- either the outcome is time-diverging, and the objective of Player i is met;
- or the outcome is time-converging, and Player i has received only finitely many blames (meaning the Player $3 - i$ has received infinitely many, hence is responsible for making time converge).

Example. *Consider the games depicted on Fig. 15 (where the silent self-loops have been omitted for the sake of readability). The game on the left, where the aim of Player 1 is to reach state q , is an example of a game where none of the player has a winning strategy: whatever the delay chosen by Player 1, Player 2 can choose a smaller delay and fire the self-loop. This constantly prevent Player 1 from applying her transition, while preserving time divergence.*

The game on the right is a bit more tricky: the situation seems intuitively similar to the previous one, except that divergence of time is not guaranteed: if Player 1 plays her transition with a converging sequence of delays, either she will be allowed to play her move at some point (because Player 2 will play a larger delay), or Player 2 will always play delays less than or equal to those of Player 1, hence preventing time divergence. If Player 1 can never apply her move, she will never be blamed, and the corresponding outcome will be winning. Otherwise, the outcome will be winning (as the goal state is reached), and it suffices for Player 1 to play a time-diverging strategy, from that point on, to win.

Figure 15: Examples of timed games



Definition 99. The extended semantics of a timed game $\mathcal{G} = \langle S, \mathfrak{C}, T_1, T_2, \ell, \text{Inv} \rangle$ in this setting is the concurrent game $\mathcal{S} = \langle Q, R, l, \{A_1, A_2\}, \mathbb{M}, \text{Ch}, \text{Edg} \rangle$ such that:

- $Q = (S \times \mathbb{R}_{\geq 0}^{\mathfrak{C} \cup \{z\}}) \times \{\top, \perp\} \times \{\top, \perp\}$;
- R contains all transitions of the form $((s, \nu), z, b), ((s', \nu'), z', b')$ as soon as there is a transition $((s, \nu), (d, \sigma), (s', \nu'))$ in the mixed-move semantics of the underlying automaton;
- l labels states according to the labelling of \mathcal{G} ;
- $\mathbb{M} = \mathbb{R}_{\geq 0} \times (T_1 \cup T_2 \cup \{\lambda\})$;
- $\text{Ch}: Q \times \{A_1, A_2\} \rightarrow 2^{\mathbb{M}} \setminus \emptyset$ indicates the set of possible pairs (d, a) are allowed in a given state (with the special move $(0, \lambda)$ being always allowed). Two states $((s, \nu), z, b)$ and $((s, \nu), z', b')$ representing the same location and valuation must have the same set of choices;
- $\text{Edg}: Q \times \mathbb{M}^2 \rightarrow 2^Q$ is the (non-deterministic) transition function, associating the successors of a state under a give pair of moves:

$$\text{Edg}(((s, \nu), z, b), (d_1, a_1), (d_2, a_2)) = \begin{cases} \{((s'_1, \nu'_1), z'_1, \top)\} & \text{if } d_1 < d_2 \\ \{((s'_2, \nu'_2), z'_2, \perp)\} & \text{if } d_2 < d_1 \\ \{((s'_1, \nu'_1), z'_1, \top), ((s'_2, \nu'_2), z'_2, \top)\} & \text{if } d_1 = d_2 \end{cases}$$

where (s'_i, ν'_i) are the successors obtained by applying the mixed-move transition (d_i, a_i) , and z'_i is a \top if $\lfloor \nu(z) \rfloor < \lfloor \nu_i(z) \rfloor$, and \perp otherwise.

In other terms, the state spaces keeps track of time divergence (which holds iff states in $S \times \{\top\} \times \{\top, \perp\}$ are visited infinitely often), and of blames (Player 1 will lose along an outcome where time converges and states in $S \times \{\top, \perp\} \times \{\top\}$ are visited infinitely often).

Definition 100. Let $X \subseteq Q$. We write $X = X_{\top} \cup X_{\perp}$, where $X_{\top} \subseteq S \times \{\top, \perp\} \times \{\top\}$ and $X_{\perp} \subseteq S \times \{\top, \perp\} \times \{\perp\}$.

A state $q = ((s, \nu), z, b)$ is in the set of controllable predecessors $\text{CPre}_1(X)$ of X iff

$$\begin{aligned} \exists (d_1, a_1) \in \text{Ch}(q, A_1). \\ \left[\forall (d_2, a_2) \in \text{Ch}(q, A_2). (d_2 \leq d_1 \Rightarrow (\delta((s, \nu), (d_2, a_2)), \text{tick}(\nu(z), d_2)) \in X_{\perp}) \wedge \right. \\ \left. ((\delta((s, \nu), (d_1, a_1)), \text{tick}(\nu(z), d_2)) \in X_{\top} \vee \forall (d_2, a_2) \in \text{Ch}(q, A_2). d_2 < d_1) \right] \end{aligned}$$

where $\text{tick}(\nu(z), d_2)$ is \top iff $\lfloor \nu(z) + d_2 \rfloor > \lfloor \nu(z) \rfloor$.

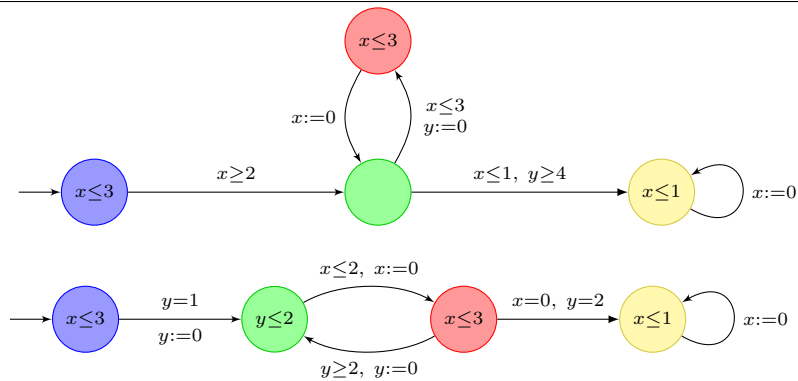
The important, but not so surprising, property is that CPre preserves unions of regions:

Lemma 101. Let $X = \bigcup_i S_i \times \{z_i\} \times \{b_i\}$ be a subset of Q , where $S_i \subseteq S \times \mathbb{R}_{\geq 0}^{\mathfrak{C} \cup \{z\}}$ are regions. Then $\text{CPre}(X)$ is a union of regions.

We leave this proof to the interested reader. As a corollary, the existence of a winning strategy for Player 1 is decidable (for reasonable winning conditions), by reasoning in terms of regions. For instance, reachability (associated with the fairness condition stating that Player 1 must receive only finitely many blames along time-convergent outcomes) can be checked in polynomial time (in the size of the underlying game graph).

Theorem 102. The existence of a winning strategy for reachability is EXPTIME-complete.

Figure 16: Examples of timed automata



Exercises

Exercise 14 ★ Is the right-most state reachable in the timed automata of Fig. 16?

Exercise 15 ★ Build a timed automaton representing the behaviour of a “soft-touch lamp”: such a lamp turns on when it is touched, and becomes brighter if it is touched a second time within 3 seconds. When it is on, touching it (even twice within a one-second interval) turns it off.

Build a timed automaton (as a product of small automata) representing a 3-floor lift. It will be made of the cabin, one door at each floor, one button at each floor (or two directional buttons), 3 buttons in the cabin, and a controller.

Exercise 16 ★★★ Diagonal constraints are constraints of the form $x - y \sim c$. Prove that the M -equivalence defined at Definition 54 is not compatible with those constraints, but that it can be refined in order to be. Prove that reachability remains PSPACE-complete for timed automata extended with diagonal constraints.

Exercise 17 ★★ As an alternative proof of decidability of timed automata with diagonal constraints, prove that any timed automaton with diagonal constraints can be transformed into a bisimilar timed automaton without diagonal constraints.

Exercise 18 ★★★ Prove that reachability is undecidable in timed automata extended with *additive guards*, *i.e.*, guards of the form $x + y \sim c$.

Exercise 19 ★★★ Prove that reachability is decidable in 2-clock timed automata extended with *additive guards*, *i.e.*, guards of the form $x + y \sim c$.

Exercise 20 ★★★ Prove that reachability is undecidable in timed automata extended with *irrational constants*, *i.e.*, guards of the form $x \sim c$ with $c \in \mathbb{R}$.

Exercise 21 ★★ Prove that reachability is undecidable in timed automata extended with *stop-watches*, *i.e.*, clocks that can be stopped in some states.

Exercise 22 ★★ We restrict to timed automata involving only one clock. Prove that reachability can be checked in polynomial time in this class of timed automata. What if we consider timed automata with two clocks?

Exercise 23 ★ We write $\mathbf{F}_{=1} \phi$ for $\top \mathbf{U}_{[1,1]} \phi$. Is it the case that $\mathbf{F}_{=1} \mathbf{F}_{=1} \phi$ is equivalent to $\mathbf{F}_{=2} \phi$? Explain why.

Exercise 24 ★★★ Is it possible to express that, along a timed word (resp. a signal), there is a two-time-unit-long period which contains an a , a b and a c , in that order?

Exercise 25 ★ Applying the algorithm, check whether there a winning strategy in the timed game of Fig. 14.

3. Hybrid automata

3.1 Hybrid automata: quantities beyond time

We only give an informal definition of hybrid automata, which we then specialize in the next subsections.

Given a finite set of real-valued variables \mathcal{V} and a set $S \subseteq \mathbb{R}$, we let $F(\mathcal{V}, S)$ be the set of expressions with free variables in \mathcal{V} and with values in S . For instance, if $\mathcal{V} = \{x, y\}$ and $S = \{0, 1\}$, then the expression “ $x = 1 \vee y > x$ ” belongs to $F(\mathcal{V}, \{0, 1\})$, while “ $x + y^2$ ” belongs to $F(\mathcal{V}, \mathbb{R})$. We write \mathcal{B} for the set of intervals of \mathbb{R} .

Definition 103. A hybrid automaton is a 9-tuple $\mathcal{H} = \langle S, T, \ell, \mathcal{V}, \text{Inv}, \text{Act}, \text{Pre}, \text{Post}, \text{Upd} \rangle$ where

- $\langle S, T, \ell \rangle$ is a finite-state automaton. Notice that we don't impose that $T \subseteq S \times S$: instead, an element $r \in T$ has a source state $\text{src}(r)$ and target state $\text{trg}(r)$, but several transitions may share the same sources and targets.
- $\mathcal{V} = \{x_1, \dots, x_n\}$ is a finite set of variables. We write n for the number of variables in \mathcal{V} ;
- $\text{Inv}: S \rightarrow F(\mathcal{V}, \{0, 1\})$ indicates the invariant in all states;
- $\text{Act}: S \rightarrow F(\mathcal{V} \cup \dot{\mathcal{V}}, \mathbb{B})$ maps each state to an expression involving the variables and their first derivative. This indicates the activity or flow condition governing the evolution of the variables;
- $\text{Pre}: T \rightarrow F(\mathcal{V}, \{0, 1\})$ and $\text{Post}: T \rightarrow F(\mathcal{V}, \{0, 1\})$ indicate the pre- and post-conditions to be fulfilled on each transition;
- $\text{Upd}: T \rightarrow 2^{\mathcal{V}}$ indicates which variables must be updated upon firing a given transition.

Since we have been a bit informal in the nature of the **Act** function, we won't give the formal semantics of hybrid automata here. Roughly, in a state, the variables must follow some differential equations. Though this is not the only source of undecidability, this makes the reachability problem very hard, as differential equations make irrational numbers come into play.

In the rest of this chapter, we present two important classes with decidable reachability, and show how slight extensions become undecidable.

3.2 Rectangular hybrid automata

Definition 104 ([HKPV98]). A rectangular automaton is a 9-tuple $\mathcal{H} = \langle S, T, \ell, \mathcal{V}, \text{Inv}, \text{Act}, \text{Pre}, \text{Post}, \text{Upd} \rangle$ where

- $\langle S, T, \ell \rangle$ is a finite-state automaton. Notice that we don't impose that $T \subseteq S \times S$: instead, an element $r \in T$ has a source state $\text{src}(r)$ and target state $\text{trg}(r)$, but several transitions may share the same sources and targets.
- $\mathcal{V} = \{x_1, \dots, x_n\}$ is a finite set of variables. We write n for the number of variables in \mathcal{V} ;
- $\text{Inv}: S \rightarrow \mathcal{B}^{\mathcal{V}}$ indicates the invariant in all states;
- $\text{Act}: S \rightarrow \mathcal{B}^{\mathcal{V}}$ indicates the evolution of variables in each state;
- $\text{Pre}: T \rightarrow \mathcal{B}^{\mathcal{V}}$ and $\text{Post}: T \rightarrow \mathcal{B}^{\mathcal{V}}$ indicate the pre- and post-conditions to be fulfilled on each transition;
- $\text{Upd}: T \rightarrow 2^{\mathcal{V}}$ indicates which variables must be updated upon firing a given transition.

A rectangular automaton is initialized if for all edge $r \in T$, for all $x \in \mathcal{V}$, it holds $\text{Act}(\text{src}(r))(x) = \text{Act}(\text{trg}(r))(x)$ unless $x \in \text{Upd}(r)$.

Definition 105. Let $\mathcal{H} = \langle S, T, \ell, \mathcal{V}, \text{Inv}, \text{Act}, \text{Pre}, \text{Post}, \text{Upd} \rangle$ be a rectangular hybrid automaton. The semantics of \mathcal{H} is the infinite-state automaton $\llbracket \mathcal{H} \rrbracket = \langle Q, R, l \rangle$ defined as follows:

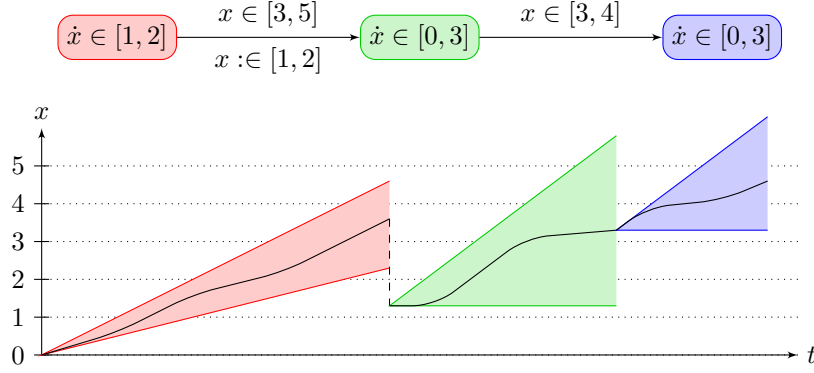
- $Q = \{(s, \nu) \in S \times \mathbb{R}^{\mathcal{V} \cup \{t\}} \mid \nu \models \text{Inv}(s)\}$, where t is an extra clock representing time;
- $R \subseteq Q \times Q$ is such that $((s, \nu), (s', \nu')) \in R$ iff one of the following two conditions hold:
 - **[continuous step]** $s = s'$ and either $\nu = \nu'$, or $\nu'(t) > \nu(t)$ and for all $x \in \mathcal{V}$, the derivative of x against t (which we write \dot{x} , lies within $\text{Act}(s)(x)$. Equivalently,

$$\frac{\nu'(x) - \nu(x)}{\nu'(t) - \nu(t)} \in \text{Act}(s)(x).$$

- **[discrete step]** there is a transition $r \in T$ s.t. $\text{src}(r) = s$, $\text{trg}(r) = s'$, $\nu \models \text{Pre}(r)$, $\nu' \models \text{Post}(r)$, $\nu(t) = \nu'(t)$ and for all $x \in \mathcal{V}$, $\nu(x) = \nu'(x)$ unless $x \in \text{Upd}(r)$.

Example. Fig. 17 depicts an example of a rectangular automaton, together with one possible behaviour. Colored triangles indicate the possible behaviours of variable x in the corresponding state.

Figure 17: Example of an initialized rectangular automaton



Remark. Obviously, (initialized) rectangular automata generalize timed automata.

3.2.1 Decidability of initialized rectangular automata

Definition 106.

Problem: *Reachability in hybrid automata*

Input: *A hybrid automaton \mathcal{H} , two states s and s' ;*

Question: *Is there a run in \mathcal{H} from $(s, \mathbf{0}_{\mathcal{V}})$ to (s', ν) , for some valuation ν ?*

Theorem 107. *Reachability in initialized rectangular hybrid automata is PSPACE-complete.*

Proof. W.l.o.g., we assume the following properties:

- $\text{Pre}((s, s')) \subseteq \text{Inv}(s)$,
- $\text{Post}((s, s')) \subseteq \text{Inv}(s')$,
- $\text{Pre}((s, s')) = \text{Post}((s, s'))$ whenever $i \notin \text{Upd}((s, s'))$.

We first consider the case of *compact* rectangular automata, in which Pre , Post and Act have compact values (hence they are closed finite intervals), and Inv always returns true. This special case already contains the important arguments.

As a first step, we transform \mathcal{H} into an intermediate automaton \mathcal{M} , called a *multi-rate automaton*, which is a rectangular automaton whose activities are singular (*i.e.*, the slope has a fixed value in each

We now prove correctness of this transformation. This will be expressed using the following transformation χ , mapping a configuration of \mathcal{M} to a set of configuration of \mathcal{H} :

$$\chi(s, \nu) = \{s\} \times \prod_{x \in \mathcal{V}} [\nu(x^-), \nu(x^+)].$$

We write U for the set of states $\{(s, \nu) \mid \forall x \in \mathcal{V}. \nu(x^-) \leq \nu(x^+)\}$. This is exactly the set of states whose image by χ is non-empty.

The following two lemmas are easily proved:

Lemma 108. *Let $(s, \nu) \in U$ and $t \in \mathbb{R}_{\geq 0}$. Then*

$$\text{Post}^t(\chi(s, \nu)) = \chi(\text{Post}^t(s, \nu))$$

where $\text{Post}^t(q)$ represents the set of successors of configuration q after a continuous transition of duration t .

Lemma 109. *Let $(s, \nu) \in U$ and $r \in T$. Then*

$$\text{Post}^r(\chi(s, \nu)) = \chi(\text{Post}^{\{(r, k) \in R\}}(s, \nu))$$

where $\text{Post}^r(q)$ represents the set of successors of configuration q by transition r .

As a consequence, the set of reachable states in \mathcal{H} is precisely the image by χ of the set of reachable states in \mathcal{M} .

In the case where the original automaton does not contain only compact intervals, it suffices to modify the corresponding intervals in the transformation above in the natural way.

We now transform our initialized multi-rate automaton \mathcal{M} into an *initialized stopwatch automaton* \mathcal{S} , which is an initialized multi-rate automaton all of whose rates are in $\{0, 1\}$. The idea is just to scale the variables by the correct factor: for each location $s \in S$ of \mathcal{M} , we let $m_i = \text{Act}'(s)(x_i)$ if non-zero, and $m_i = 1$ otherwise. We then define the mapping $\alpha_s: \mathbb{R}^{2n} \rightarrow \mathbb{R}^{2n}$ by $\alpha_s(x_1, \dots, x_{2n}) = (x_1/m_1, \dots, x_{2n}/m_{2n})$. The stopwatch automaton is then obtained from \mathcal{M} by applying α_s to $\text{Act}'(s)$ and $\text{Pre}((s, s'))$, and $\alpha_{s'}$ to $\text{Post}((s, s'))$.

It is quite clear that we get an initialized stopwatch automaton, and that it enjoys the following property:

Lemma 110. *Given a configuration (s, ν) of \mathcal{M} , we define $\alpha(s, \nu) = (s, \alpha_s(\nu))$, and extend this definition to sets of states in the obvious way. Then for any set of states Z of \mathcal{M} , $\alpha(\text{Post}_{\mathcal{M}}^*(Z)) = \text{Post}_{\mathcal{S}}^*(\alpha(Z))$.*

As a consequence, the set of reachable states of \mathcal{M} is the inverse image by α of the set of reachable states in \mathcal{S} . It then remains to prove that we can decide reachability of a location in an initialized stopwatch automaton:

Proposition 111. *Reachability for an initialized stopwatch automaton \mathcal{S} is PSPACE-complete.*

Proof. We transform \mathcal{S} into a plain timed automaton \mathcal{A} . Let K be the set of all finite endpoints of intervals appearing in \mathcal{S} . Then the state space of \mathcal{A} is $S \times (K \cup \{\perp\})^{\mathcal{V}'}$, where \mathcal{V}' is the set of variables of \mathcal{S} . Intuitively, \perp will indicate that the corresponding variable has slope 1, while any other value c indicates that the corresponding variable is stopped (slope 0) and the last time this variable was assigned a value, it was precisely c .

It remains to encode the pre- and post-conditions of \mathcal{S} into guards and updates of \mathcal{A} (notice that we use rational updates rather than just resets, but this is known not to change the nature of timed automata).

Then, when a stopwatch is stopped in \mathcal{S} , its value is stored in the control state of \mathcal{A} . The corresponding clock in \mathcal{A} continues to run, but its value is irrelevant. When this stopwatch is restarted in \mathcal{S} , it is updated to a new value. So is the corresponding clock in \mathcal{A} .

Finally, since reachability in timed automata is decidable in PSPACE, and in space linear in the number of states of the automaton, we get a polynomial-space algorithm. Hardness in PSPACE follows from that of timed automata. \square

3.2.2 Undecidability of rectangular automata

Theorem 112. *Reachability is undecidable in stopwatch automata.*

Proof. This result is weaker than the next result, but its proof is much simpler: we encode each counter using two stopwatches: $c_i = x_i - y_i$. Incrementing or decrementing a counter is achieved by stopping one or the other stopwatch. The rest of the encoding is straightforward (using diagonal constraints). \square

Definition 113. *A rectangular automaton is simple if*

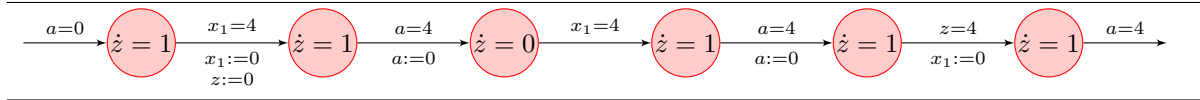
- only one variable is not a clock (i.e., has derivative different from 1 in some state),
- *Inv*, *Act*, *Pre* and *Post* have compact values,
- $\text{Post}(e)(x) = [0, 0]$ if $x \in \text{Upd}(e)$, and $\text{Post}(e)(x) = \text{Pre}(e)(x)$ otherwise.

Theorem 114. *Reachability is undecidable in simple 2-slope multi-rate automata.*

Proof. The result holds for any two different rationals serving as the slopes of the 2-slope variable, but we only prove it for the special case where the slopes are 0 and 1, and 1 and 2.

We begin with the case of stopwatches. We encode the halting problem for two-counter machines. Counters c_1 and c_2 are encoded by clocks x_1 and x_2 , with $x_i = 2^{1-c_i}$. We now have to be able to double and halve the value of a clock, preserving the value of the other clock at the same time. Doubling the value of a clock is achieved by the automaton depicted on Fig. 19. Notice that, in this figure, we omit

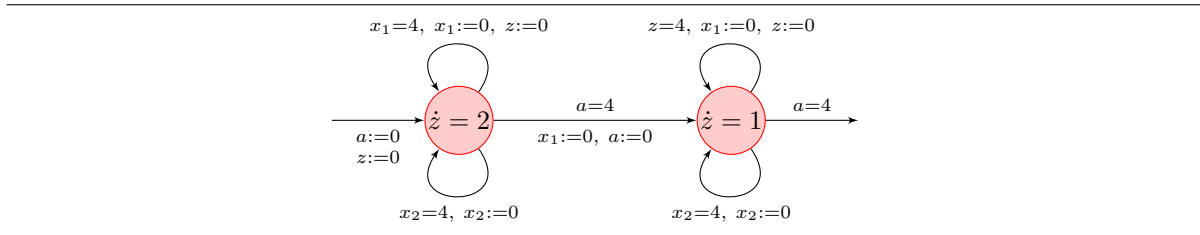
Figure 19: Automaton for doubling the value of a clock



self-loop on each state resetting clock c_2 when it reaches 4, as well as invariants in each state, imposing each variable to be bounded by 4. Then it can be checked that the value of c_1 is doubled between the entry and exit of this automaton, while the value of c_2 is preserved. Halving the clock is achieved in a similar way.

The encoding in the case of slopes 1 and 2 follows similar ideas. The module for doubling the value of c_1 is depicted on Fig. 20. \square

Figure 20: Automaton for doubling the value of a clock



3.3 Polygonal hybrid systems

Definition 115. *Let d be a positive integer, and S be a set in \mathbb{R}^d .*

- The affine hull of S is the set of affine combinations of points in S . We denote it by $\text{Aff}(S)$.
- The closure of S is the set of points $o \in \mathbb{R}^d$ s.t. for all $\varepsilon > 0$, there is a point $s \in S$ at distance at most ε from o . We write $\text{Cl}(S)$ for the closure of S .
- The interior of S is the set $I \subseteq S$ such that, for all $i \in I$, there exists $\varepsilon > 0$ s.t. any point $s \in \mathbb{R}^d$ located at distance at most ε from i is in S . We write $\text{Int}(S)$ for the interior of S .

- The relative interior of S is the set $I \subseteq S$ such that, for all $i \in I$, there exists $\varepsilon > 0$ s.t. any point $s \in \text{Aff}(S)$ located at distance at most ε from i is in S . We write $\text{RInt}(S)$ for the relative interior of S .
- The boundary of S is the difference between its closure and its interior.
- The relative boundary of S is the difference between its closure and its relative interior.

Definition 116. Let d be a positive integer.

- A closed half space of \mathbb{R}^d is a set of points $x \in \mathbb{R}^d$ satisfying $a \cdot x + b \geq 0$, for some $a \in \mathbb{R}^d$ and $b \in \mathbb{R}$ (where $a \cdot x$ is the standard dot product).
- A closed convex polyhedral set of \mathbb{R}^d is the intersection of finitely many closed half spaces of \mathbb{R}^d .
- A convex polyhedral partition of \mathbb{R}^d is a finite set $(P_s)_{s \in S}$ of closed convex polyhedral sets of \mathbb{R}^d such that
 - $\bigcup_{s \in S} P_s = \mathbb{R}^d$,
 - $\text{Int}(P_s) \neq \emptyset$ for all $s \in S$,
 - $\text{Int}(P_s \cap P_r) = \emptyset$ for all $r \neq s \in S$.

Definition 117. A d -dimensional polygonal differential inclusion system (PDIS_d for short) is a pair $\mathcal{H} = \langle P, F \rangle$ where

- $P = (P_s)_{s \in S}$ is a convex polyhedral partition of \mathbb{R}^d ;
- $F: S \rightarrow \mathbb{R}^d$ maps each element of the partition to a convex subset of \mathbb{R}^d .

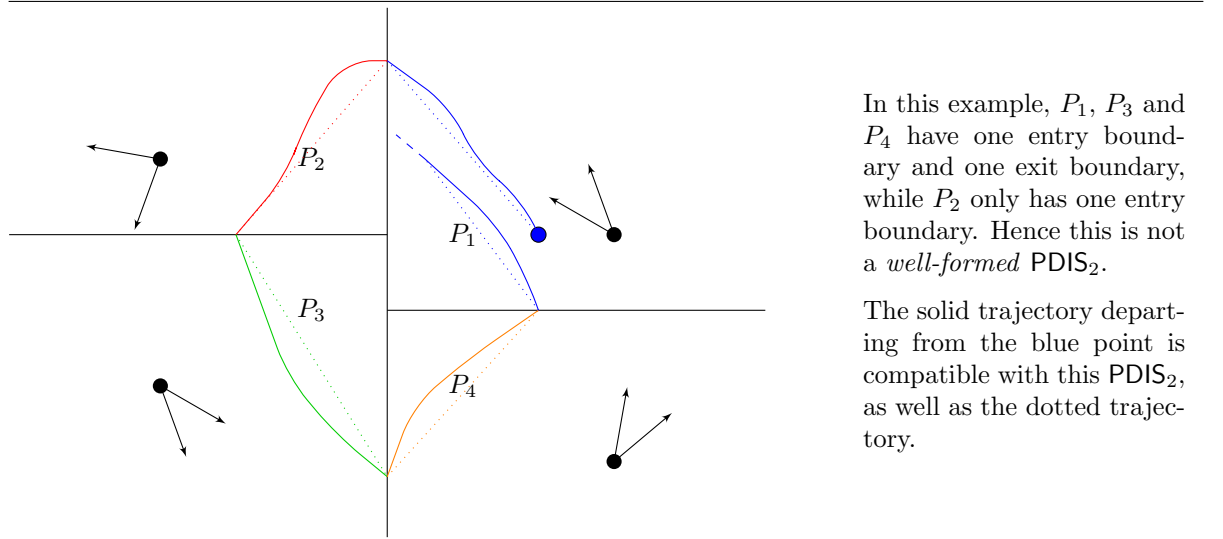
Each set in the partition is called a region.

Definition 118. Let \mathcal{H} be a PDIS_d . A boundary e of one of the regions P_s is said to be an entry of P_s if, for all $x \in \text{RInt}(e)$ and all $c \in F(s)$, there is a positive ε s.t. $x + \varepsilon \cdot c$ belongs to P_s . It is an exit of P_s if the same condition holds for a negative ε . We write $\text{In}(P_s)$ and $\text{Out}(P_s)$ for the set of entry and exit boundaries of P_s .

Definition 119. A PDIS_d is well-formed any boundary of any region in the partition is either an entry or an exit.

In the sequel, we restrict to the class of *well-formed* PDIS_d .

Figure 21: Example of a PDIS_2



Definition 120. A trajectory segment is a continuous, almost-everywhere differentiable function $\chi: T \rightarrow \mathbb{R}^d$, for some $T \subseteq \mathbb{R}$. A trajectory is a trajectory segment with $T = \mathbb{R}$.

Definition 121. A trajectory segment χ is compatible with a $\text{PDIS}_d \mathcal{H}$ if, for all $t \in T$,

- if $\chi(t) \in \text{Int}(P_s)$, then $\dot{\chi}(t)$ is defined and belongs to $F(s)$;
- if $\chi(t)$ belongs to a boundary $e \in \text{In}(P_s)$, then the right derivative of χ must be defined in t , and it must belong to $F(s)$.

Definition 122.

Problem: *Reachability in a PDIS_d*

Input: *A $\text{PDIS}_d \mathcal{H}$, and two points x_0 and x_1 ;*

Question: *In there a compatible trajectory from x_0 to x_1 ?*

3.3.1 Decidability of planar polygonal hybrid systems

Theorem 123. *Reachability in a PDIS_2 is decidable.*

Proof. The proof is long and technical, and we only review the main arguments. The interested reader will find a detailed proof in [ASY07].

An *edge* e is a maximal set of points of \mathbb{R}^d such that, for any boundary b of any region, $e \subseteq b$ or $\text{RInt}(e \cap b) = \emptyset$. The *signature* of a trajectory is the sequence of edges it visits. The *trace* of a trajectory is the sequence of the intersections with the edges it visits.

Lemma 124. *For every trajectory segment χ , there is a trajectory segment χ' with the same initial and final points, same signature and trace, and such that the derivative is constant between any two consecutively visited edges.*

The new trajectory is obtained from χ by replacing the trajectory segment between two consecutive intersections with the edges by the straight line between these two points. It is easily seen that each new segment is compatible with \mathcal{H} , by convexity of the regions and of the values of F .

A *self-crossing* of a trajectory χ is a pair (t, t') , with $t \neq t'$, for which $\chi(t) = \chi(t')$.

Lemma 125. *For any trajectory segment χ , there is a trajectory segment χ' with the same initial and final points, and having no self-crossing.*

Notice that this may change the signature and trace of the trajectory segment.

Lemma 126. *Let $\sigma = e_1 \cdots e_p$ be a signature of a non-self-crossing trajectory. Then σ can be written as $r_1 \cdot s_1^{k_1} \cdots r_n \cdot s_n^{k_n} \cdot r_{n+1}$, where all r_i are simple paths, and all s_i are simple cycles.*

Such a representation can be achieved for any finite word, by a greedy algorithm performing backward. The *type* of a signature is the sequence $(r_1, s_1, r_2, \dots, r_{n+1})$.

Lemma 127. *Let σ be a signature corresponding to a trajectory segment χ , and $(r_1, s_1, r_2, \dots, r_{n+1})$ be its type. Then*

- for any $1 \leq i \neq j \leq n+1$, r_i and r_j are disjoint;
- for any $1 \leq i \neq j \leq n$, s_i and s_j are different.

This indicates that we only have to consider finitely many traces for checking reachability.

Let $e \in \text{In}(P_s)$ and $e' \in \text{Out}(P_s)$. The successor in e' of a point x of e for direction $c \in F(s)$ is the unique point $x' \in e'$ s.t. $x' = x + t \cdot c$ for some $t > 0$. Notice that uniqueness follows from the fact that the PDIS_2 under study are well-formed. We write $\text{Succ}_{e,e'}^c(x)$ for this point x' . Then $\text{Succ}_{e,e'}(x)$ is the set of all successors in e' of x , for all possible directions in $F(s)$. This set can be computed by first computing all successors of x under all directions, and then intersecting with e' . These successor functions can then be composed together, in order to compute the successors along a given trace. The important point is that iteration of cycles always yield a finite union of intervals, which ensures termination. \square

3.3.2 Undecidability of 3-dimensional polygonal hybrid systems

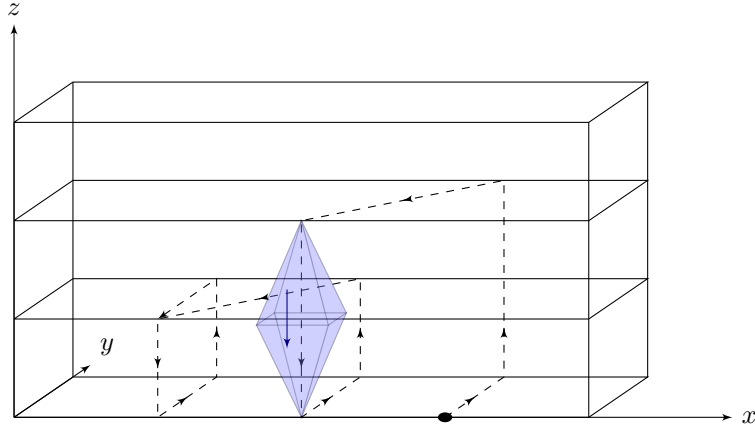
Theorem 128. *Reachability in a PDIS₃ is undecidable.*

Proof. We first explain how to encode deterministic finite-state automata: a deterministic automaton with n states is encoded through a PDIS₃ having the following regions and slopes:

- a region F , defined by $(z = 0) \wedge (y \leq 1)$ and vector $(0, 1, 0)$;
- if $\delta(q_i) = q_j$, a region $U_{i,j}$ defined by $(x = i) \wedge (y = 1) \wedge (z \leq j)$, and vector $(0, 0, 1)$;
- still if $\delta(q_i) = q_j$, a region $B_{i,j}$ defined by $(z = j) \wedge (x + (j - i)y = j) \wedge (y \geq 0)$ and vector $(j - i, -1, 0)$;
- a region D defined by $(z \geq 0) \wedge (y = 0)$ and vector $(0, 0, -1)$.

Notice that this definition does not satisfy the requirement of having non-empty interior. However, since each region has only one direction, it suffices to replace these 2-dimensional regions with cones around the corresponding trajectories (see the blue region on Fig. 22), in order to preserve well-formedness. It can be checked that this PDIS₃ encodes the behaviour of an automaton with transitions from state q_3 to q_2 , from q_2 to q_1 and with a self-loop on q_1 .

Figure 22: Simulating a finite-state automaton by a PDIS₃



We now extend this construction to also handle one counter. The counter is encoded in the fractional part of variable x , with $\langle x \rangle = 2^{-(c+1)}$ where c is the value of the counter. This fractional part will be updated on the plane $z = 0$ by the two basic PDIS₂ depicted on Fig. 23. It is easily seen that, starting

Figure 23: Incrementing and decrementing a counter



from $(x_0, 0)$, the trajectory reaches $(x_0/2, 1)$ in the incrementing block and $(2x_0, 1)$ in the decrementing block, which precisely encodes incrementation and decrementation of c . The zero-test is achieved as follows: we first perform decrementation, hence possibly reaching a point where the fractional part of x comes close to 1, when $y = 1$. If this is the case, it indicates that the value of c was 0; we will reset the value of x to $1/2$, and switch to the new state. This is achieved by having different regions depending on the value of $\langle x \rangle$, whether it is larger than $2/3$ (hence it tends to 1) or not (hence less than or equal to $1/2$). Fig. 24 depicts a 1-counter machine with two instructions:

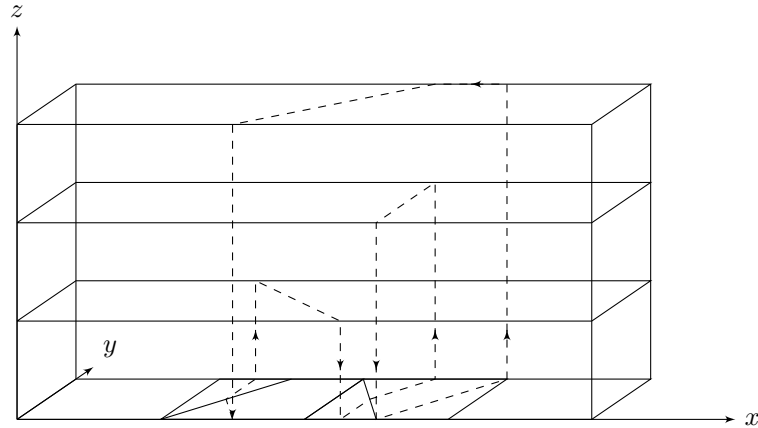
```

q1: c++; goto q2
q2: if c>0 then c--; goto q2 else goto q1

```

Notice that on the “ground floor” of this PDIS_3 , we have the incrementing gadget (with $x \in [1, 2)$, corresponding to state q_1) and the decrementing gadget (with $x \in [2, 3)$, corresponding to state q_2). One easily gets convinced of the correctness of the encoding, via the following correspondence: point $(x, 0, 0)$ in the PDIS_3 corresponds to state $q_{\lfloor x \rfloor}$ and counter value $-\log_2(\langle x \rangle)$.

Figure 24: Simulating a 1-counter automaton by a PDIS_3



Using similar ideas, we can encode two-counter machines with PDIS_4 , which proves undecidability of reachability for these hybrid systems. With some more work, it is possible to prove that some restricted (but still undecidable) class of 2-counter machines can be encoded into a PDIS_3 , proving undecidability also in three dimensions. \square

4. Timed automata with observers

4.1 Timed automata with linear observers

Definition 129 ([ALP01, BFH⁺01]). A timed automaton with linear observers (*TALO* for short, but sometimes also called priced timed automata or weighted timed automata) is a 7-tuple $\mathcal{W} = \langle S, \mathfrak{C}, T, \ell, \text{Inv}, \text{Var}, \text{rate}, \text{upd} \rangle$ where

- $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \text{Inv} \rangle$ is a timed automaton (referred to as the underlying timed automaton in the sequel);
- Var is a finite set of variables (called observers, prices or weights);
- $\text{rate}: S \rightarrow \mathbb{R}^{\text{Var}}$ indicates the derivative of the observer variables in each state.
- $\text{upd}: T \rightarrow \mathbb{R}^{\text{Var}}$ associates with each transition the values to be added to each observer upon firing this transition;

Definition 130. Let \mathcal{W} be a timed automaton with linear observers, and $\langle Q, R, l \rangle$ be the semantics of the underlying timed automaton, as given in Definition 49. The semantics of \mathcal{W} is the infinite-state weighted automaton $\langle V, E, \lambda \rangle$ where

- $V = \{(s, \nu, \mu) \in S \times \mathbb{T}^c \times \mathbb{R}^{\text{Var}} \mid (s, \nu) \in Q\}$;
- $E \subseteq V \times \mathbb{R}^{\text{Var}} \times V$ is the union of two sets of transitions:
 - delay transitions: $((s, \nu, \mu), c, (s', \nu', \mu')) \in E_d$ iff (s, ν) and (s', ν') belong to Q , $s = s'$, and there exists $d \in \mathbb{R}_{\geq 0}$ s.t.
 - * $\nu' = \nu + d$;
 - * $c = \text{rate}(s) \times d$;
 - * $\mu' = \mu + c$.
 - action transitions: $((s, \nu, \mu), c, (s', \nu', \mu')) \in E_a$ iff (s, ν) and (s', ν') belong to Q and there exists $t = (s, g, \sigma, r, s') \in T$ s.t. $\nu \models g$, $\nu' = \nu[r \rightarrow 0]$, $c = \text{upd}(t)$ and $\mu' = \mu + c$.
- λ labels a each state (s, ν, μ) following $\ell(s)$.

Remark. It must be noted that the existence of a transition never depends on μ in the semantics above: contrary to hybrid systems, and as their name indicates, observers are just there to observe the evolution of some quantities along the executions of the underlying timed automaton, without modifying the behaviour of the automaton.

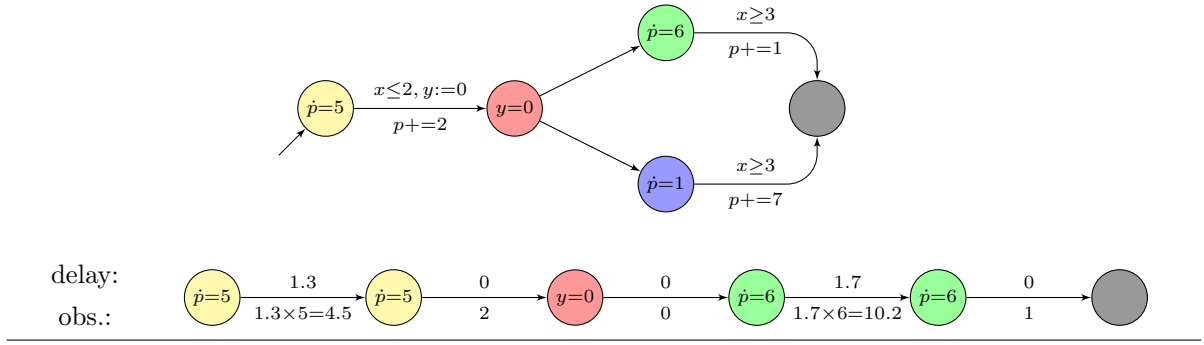
Example. Fig. 25 depicts a timed automaton with a single linear observer (where there is only one observer p , and where we omitted to indicate weights when they equal zero).

An example of a run in this automaton is the following: delay for 1.3 time units in the first state, then go to the middle state and immediately to the topmost state (where $\dot{p} = 6$). Delay for another 1.7 time units there, and go to the final state. This run is depicted on Fig. 25, and the increase in the value of the observer variable is 17.7.

It is easy to compute the optimal run (i.e., increasing the value of the observer by the least possible value) for reaching the last location in this example: obviously, there is no reason for waiting in the blue and green states once x has reached 3. There remains two parameters: the delay in the first state, and the choice in the second state. In the end, the optimal run is characterized by

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

Figure 25: Example of a timed automaton with linear observers



which can be rewritten as

$$\inf_{0 \leq t \leq 2} \min\{21 - t, 12 + 4t\}.$$

This equals 12, which corresponds to elapsing all three time units in the state with observer rate 1.

4.2 Optimal reachability

In the following sections until Section 4.5, we restrict to timed automata with linear observers in which upd and rate have nonnegative values (hence the values of the observers are non-decreasing).

Definition 131.

Problem: *Optimal reachability in timed automata with linear observers*

Input: *A timed automaton \mathcal{W} with one linear observer, two states s and s' , and $m \in \mathbb{Q}$;*

Question: *Is there a run from $(s, \mathbf{0}_{\mathbf{c}}, 0)$ to (s', ν', μ') in \mathcal{W} along which the observer variable increases by at most m ?*

Theorem 132. *Optimal reachability is decidable in timed automata with linear observer. It is PSPACE-complete.*

Proof. The proof is based on (a refinement of) the region graph: the path will be guessed on-the-fly. The problem, however, is that the region abstraction is too coarse to manipulate observers: not all points in the same region have the same properties w.r.t. optimal reachability: in particular, we must have information about whether we are entering the region, and can delay in it, or we are about to exit it. In order to have this information, we refine the region abstraction into the so-called *corner-point abstraction*:

Definition 133. *A corner-point is a pair (r, ν) where r is a region and ν is a corner of r , i.e., a valuation in the topological closure of r having integer clock values.*

Definition 134. *The corner-point abstraction of a TALO is the weighted graph $\langle Q, R, l \rangle$ where*

- $Q = \{(s, r, \nu) \mid (r, \nu) \text{ is a corner point}\}$;
- $R: Q \times \mathbb{R}^{\mathcal{V}} \times Q$ has three kinds of transitions:
 - delay transitions within a region: they are of the form $((s, r, \nu), c, (s, r, \nu'))$ where $\nu' = \nu + 1$ and $c = \text{rate}(s)$;
 - delay transition from one region to the next one: they are of the form $((s, r, \nu), \{0\}^{\mathcal{V}}, (s, r', \nu))$ where r' is the immediate successor or r ;
 - action transitions: they are of the form $((s, r, \nu), c, (s', r', \nu'))$, such that there is a transition $t = (s, g, \sigma, r, s')$ in the TALO, with $\text{upd}(t) = c$, $r \subseteq g$ and $\nu' = \nu[r \rightarrow 0]$.
- the labelling function follows that of the TALO.

Lemma 135. Let f be a function defined on a compact convex set $A \subset \mathbb{R}^n$ of the form

$$f(x_1, \dots, x_n) = \sum_{i=1}^n c_i \cdot x_i + c.$$

Then the minimum of f on A is reached on the border of A .

If A is a zone (defined by a conjunction of constraints $x_i \sim c$ and $x_i - x_j \sim c$ with $c \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$) then the minimum of f is obtained in a point of $\text{Cl}(A)$ with integer coordinates.

Proof. The proof is by induction on n . Both results are obvious when $n = 1$. Assume that the result holds for some n ; we prove that it propagates to $n + 1$ variables. Let α be a value for x_1 where the minimum of f is obtained. Consider $g_\alpha(x_2, \dots, x_n) = f(\alpha, x_2, \dots, x_n)$, defined on the projection A' of $A \cap \{x_1 = \alpha\}$ on the last $n - 1$ coordinates. From the induction hypothesis, g_α reaches its minimum on the border of A' . It remains to prove that if (x_2, \dots, x_n) is in the border of A' , then $(\alpha, x_2, \dots, x_n)$ is in the border of A . That (x_2, \dots, x_n) is in the border of A' means that for all $\varepsilon > 0$, there is a point (y_2, \dots, y_n) not in A' and with $\|(x_2, \dots, x_n) - (y_2, \dots, y_n)\|_\infty < \varepsilon$. Now, $(\alpha, y_2, \dots, y_n) \notin A$, since A' is the projection of $A \cap \{x_1 = \alpha\}$ on the last $n - 1$ coordinates. Moreover the distance between $(\alpha, x_2, \dots, x_n)$ and $(\alpha, y_2, \dots, y_n)$ is less than ε . Hence $(\alpha, x_2, \dots, x_n)$ is in the border of A .

We now prove that the second result also propagates. From the previous result, the minimum of f is reached on the border of Z , hence at a point where $x_i - x_j = c$ for some indices i and j and some integer c (or possibly a point where $x_i = c$ for some i and c). Replacing x_i with $x_j + c$ (or c) in f yields a function with $n - 1$ variables, whose minimum is reached at an integer point. This minimum corresponds to the minimum of f , which is then also reached at an integer point. \square

Lemma 136. Assume that there is a run from $(s, \mathbf{0}_{\mathfrak{C}}, 0)$ to (s', ν', μ') in \mathcal{W} along which the observer variable increases by at most m . Then there is a run in the corner-point abstraction of \mathcal{W} from $(s, \mathbf{0}, 0)$ to (s', r', γ') , where $\nu' \in r'$, with total weight at most m .

Conversely, for all $\varepsilon > 0$, if there is a run from $(s, \mathbf{0}, 0)$ to (s', r', γ') with weight m , then there is exists $\nu' \in r'$ s.t. there is a run from $(s, \mathbf{0}_{\mathfrak{C}}, 0)$ to (s', ν', μ') s.t. $\mu'(p) \leq m + \varepsilon$.

Proof. Consider a run from (s, ν, μ) to (s', ν', μ') in \mathcal{W} (where we assume w.l.o.g. that delay- and action transitions alternate). This runs corresponds to a sequence of delays (d_1, \dots, d_n) , together with the corresponding sequence of transitions $(v_i)_{1 \leq i \leq n-1}$ with $v_i = (s_{2i}, g_i, \sigma_i, R_i, s_{2i+1})$ for each i . Write $(s_i, \nu_i, \mu_i)_{1 \leq i \leq 2n}$ for the sequence of states being visited along this run, $t_j = \sum_{1 \leq k \leq j} d_k$, and $(r_i)_i$ for the sequence of regions s.t. $\nu_i \in r_i$. Then in a state $(s_{2j}, \nu_{2j}, \mu_{2j})$ (i.e., after a delay transition), the value of a clock x is the sum of the delays between the latest preceding reset and the current state:

$$\nu_{2j}(x) = t_j - t_{k+1} \quad \text{where } k \text{ is the largest index less than } j \text{ with } x \in R_k \text{ if any, and } 0 \text{ otherwise.}$$

For this sequence of transitions, any sequence $(t'_i)_i$ satisfying the constraints that $\nu_{2j} \models g_j$ and $\nu_{2j} \in r_{2j}$ corresponds to a run in \mathcal{W} . This defines an n -dimensional zone Z . Moreover, the increase in the value of the observer is

$$f(t'_1, \dots, t'_n) = \sum_{1 \leq i \leq n} \text{rate}(s_{2i-1}) \cdot t'_i + \sum_{1 \leq i \leq n-1} \text{upd}(v_i).$$

From Lemma 135, there is a point α on the border of Z s.t. $f(\alpha) \leq m$. From α , we can build a new sequence of valuations

$$\gamma_{2j}(x) = \alpha_j - \alpha_{k+1} \quad \text{where } k \text{ is the largest index less than } j \text{ with } x \in R_k \text{ if any, and } 0 \text{ otherwise.}$$

Since α is on the border of Z , it follows that $\gamma_{2j} \models \overline{g_j}$, where $\overline{g_j}$ is the closure of g_j , obtained by replacing strict inequalities with non-strict ones, and that $\gamma_{2j} \in \text{Cl}r_{2j}$. Moreover, α has integer values, thus also γ_{2j} has integer values.

From this, we can build a path in the corner-point abstraction, visiting the states $(s_{2j}, r_{2j}, \gamma_{2j})$. The weight of this run can be checked to be exactly $f(\alpha)$, hence the result.

Conversely, fix $\varepsilon' > 0$, and consider a finite path of weight m , visiting the states $(s_i, r_i, \gamma_i)_i$ in the corner-point abstraction. Given a valuation ν , we define its *distance to a corner* as follows:

$$d(\nu) = \max \left\{ \left\{ \min\{\nu(x) - p \mid p \in \mathbb{N}\}, \min\{\nu(x) - \nu(y) - p \mid p \in \mathbb{N}\} \mid x, y \in \mathfrak{C} \right\} \right\}.$$

Now, if there is a transition $((s, r, \alpha), (s', r', \alpha'))$ in the corner-point abstraction, then from a state (s, ν) with $\nu \in r$, close to α and with $d(\nu) < \varepsilon$, there is a transition in \mathcal{W} to a point (s', ν') with $\nu' \in r'$ close to α' and with $d(\nu') < \varepsilon'$. This can easily be proved for all three kinds of transitions in the corner-point abstraction. The difference between the cost of the transition of the corner-point abstraction and the increase in the observer value along the transition in \mathcal{W} is at most $2\varepsilon' \cdot M$ where M is the maximal value of rate on \mathcal{W} . Finally, since the weights are nonnegative, we can find a simple run from $((s, \mathbf{0}, 0)$ to (s', r', γ') , whose length is bounded by the number K of corner-point regions. From the above argument, this run can be mimicked by a run in \mathcal{W} visiting points that remain at distance at most $\varepsilon' = \varepsilon/(2MK)$. In the end, we get a run in \mathcal{W} along which the value of the observer variable is increased by at most $m + \varepsilon$. \square

The algorithm then simply consists in building a witnessing path on-the-fly in the corner-point abstraction. This yields a polynomial-space algorithm. Hardness in PSPACE follows from the PSPACE-hardness of reachability in timed automata. \square

4.3 Quantitative model-checking on priced-timed automata

Definition 137. *WCTL extends CTL as follows:*

$$\begin{aligned} \text{WCTL} \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= \phi_p \mathbf{U}_{v \in I} \phi_p. \end{aligned}$$

where v is an observer variable, I is an interval with integral bounds (or $+\infty$). Thus, syntactically, WCTL is the same logic as TCTL. We define the semantics of the weighted “until” modality along a run $\rho = ((s_i, \nu_i, \mu_i), (d_i, \sigma_i), (s'_i, \nu'_i, \mu'_i))_i$ as follows

$$\begin{aligned} \mathcal{A}, \rho \models \phi_p \mathbf{U}_{v \in I} \phi'_p \quad \Leftrightarrow \quad & \exists j \in [1, \text{length}(\rho)]. \mathcal{A}, \rho_{\geq j} \models \phi'_p \text{ and } \mu'_j(v) - \mu_0(v) \in I \text{ and} \\ & \forall i \in [1, j]. \mathcal{A}, \rho_{\geq i} \models \phi_p. \end{aligned}$$

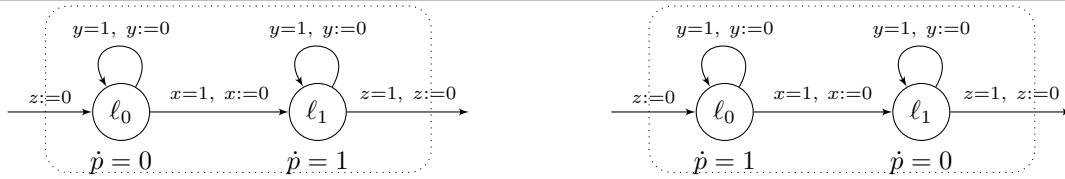
Notice that we allow constraints on different observer variables (hence in particular on time), but always *one at a time*.

Theorem 138. *WCTL model-checking on TALO is undecidable.*

Proof. The proof consists in reducing the reachability problem for two-counter machines.

We begin with defining simple *modules* which realize intermediate operations on the values of the observer. As a first step, we build modules for adding the value of a clock to a cost variable: this is achieved using modules $\text{Add}^+(x, \{z\})$ and $\text{Add}^-(x, \{z\})$, displayed on Fig. 26.

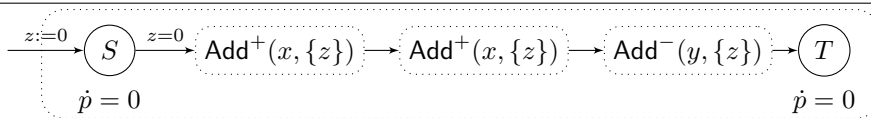
Figure 26: Automata $\text{Add}^+(x, \{z\})$ and $\text{Add}^-(x, \{z\})$



Those automata clearly satisfy the following Lemma:

Lemma 139. *If a run enters location l_0 of $\text{Add}^+(x, \{z\})$ (resp. $\text{Add}^-(x, \{z\})$) with $x = \alpha_0 \in [0, 1]$, $y = \beta_0 \in [0, 1]$ and $\mu(p) = \gamma_0$, it then leaves location l_1 with the same values for x and y , and with $\mu(p) = \gamma_0 + \alpha_0$ (resp. $\mu(p) = \gamma_0 + 1 - \alpha_0$).*

Figure 27: Automaton $\text{Test}(y = 2x, \{z\})$

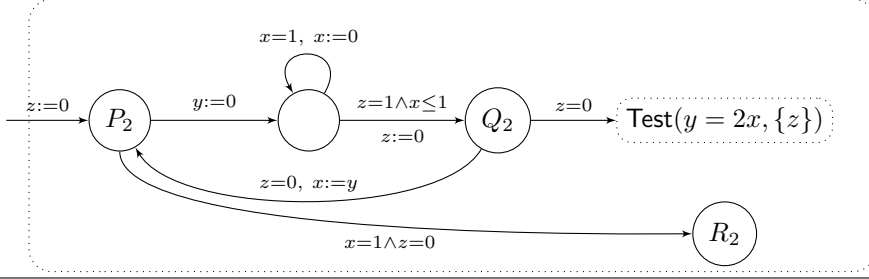


Module $\text{Test}(y = 2x, \{z\})$ is the (deterministic) automaton displayed on Fig. 27. It sets the cost to $2x + 1 - y$. Let $\varphi_1 = S \wedge \mathbf{EF}_{p \leq 1} T \wedge \mathbf{EF}_{p \geq 1} T$. The following Lemma clearly holds:

Lemma 140. *Formula φ_1 holds in S along module $\text{Test}(y = 2x, \{z\})$ with $x = \alpha_0 \in [0, 1]$ and $y = \beta_0 \in [0, 1]$ if, and only if, $\beta_0 = 2\alpha_0$.*

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

Figure 28: Automaton $\text{Power}_2(x, \{y, z\})$



Module $\text{Power}_2(x, \{y, z\})$ is displayed on Fig. 28. Note that it requires two auxiliary clocks. Note also that it uses an update “ $x := y$ ”, instead of classical resets. This is for the sake of simplicity, as the module could be adapted (by duplicating the periodic part and changing the roles of the clocks, involving no extra clock) in order to only have standard resets.

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

Lemma 141. *Formula φ_2 holds in P_2 in module $\text{Power}_2(x, \{y, z\})$ with $x = \alpha_0 \in (0, 1]$ if, and only if, there exists a non-negative integer d s.t. $\alpha_0 = 1/2^d$.*

It is easy to adapt this construction in order to build a module $\text{Power}_3(x, \{y, z\})$ and a formula φ_3 that check if x is of the form $1/3^d$, for some integer d .

We now tackle the main part of the reduction. Let \mathcal{M} be such a two-counter machine. We build a weighted timed automaton $\mathcal{W}_{\mathcal{M}}$ (with three clocks and one stopwatch observer) and a WCTL-formula Φ such that given q_0 , a well-chosen state of $\mathcal{W}_{\mathcal{M}}$, we have that \mathcal{M} halts if, and only if, $q_0 \models \Phi$. The two counters c_1 and c_2 will be encoded alternately by three clocks x, y and z . The value of c_1 is encoded by $x_1 = 1/2^{c_1}$ (with $x_1 \in \{x, y, z\}$) whereas the value of c_2 is encoded by $x_2 = 1/3^{c_2}$ (with $x_2 \in \{x, y, z\}$). To each instruction will be associated six modules, one for each injective function $\{x_1, x_2\} \rightarrow \{x, y, z\}$.

Consider the following instruction of the two-counter machine:

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

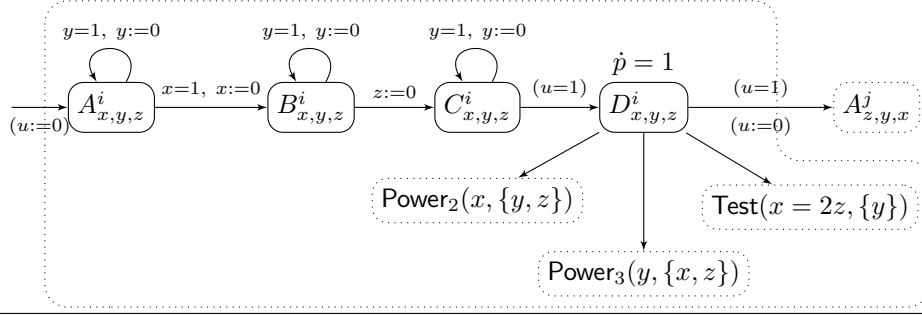
We assume that the initial value of c_1 is stored in clock x , whereas that of c_2 is stored in y . To p_i , we associate the automaton $\text{Aut}_{1,+}^i(x, y, z)$ as in Fig. 29. In that figure (and in all the other ones), observer rates which are omitted are equal to zero. The subscript $1, +$ is a reminder that instruction p_i deals with counter stored in the first clock (x here) and is an incrementation (we might omit it when it is not necessary), the tuple (x, y, z) indicates which clocks encode counters c_1 and c_2 : here, c_1 is initially stored in x and c_2 is initially stored in y . At the end of this module, the new values of c_1 and c_2 are stored in z and y , resp.; that’s why we swap x and z when leaving the module (transition from $D_{x,y,z}^i$ to $A_{z,y,x}^j$). Finally note that constraints on clock u (which as in parentheses) are present for the sake of clarity, and will be removed thanks to Lemma 142, so that the whole reduction can be achieved with only three clocks.

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

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

The last point will be ensured through a global WCTL-formula stating that the value of the observer is unchanged in location $D_{x,y,z}^i$. The second point is obtained by a module $\text{Test}(x = 2z, \{y\})$, together with a WCTL-formula ϕ_1 . Finally, according to Lemma 142 below, the first point is enforced by checking that the values of x and y when entering $D_{x,y,z}^i$ are $1/2^n$ and $1/3^m$ for some integers n and m . Those conditions are ensured by modules Power_2 and Power_3 and the associated formulas ϕ_2 and ϕ_3 .

Figure 29: Incrementing a counter



Lemma 142. *If a run enters location $A^i_{x,y,z}$ with $x = 1/2^{c_1}$, $y = 1/3^{c_2}$ and enters location $D^i_{x,y,z}$ t time units later with the value of x of the form $1/2^n$ for some n , and the value of y of the form $1/3^m$ for some m , then $t = 1$, $n = c_1$ and $m = c_2$.*

This lemma can easily be proved using elementary arithmetical manipulations. It plays a crucial role in our reduction: it explains how comparing clocks to powers of $1/2$ and $1/3$ gives a way to measure exactly 1 t.u., and thus why we encode the counters as powers of $1/2$ and $1/3$. Note that 2 and 3 could be replaced by any two relatively prime numbers.

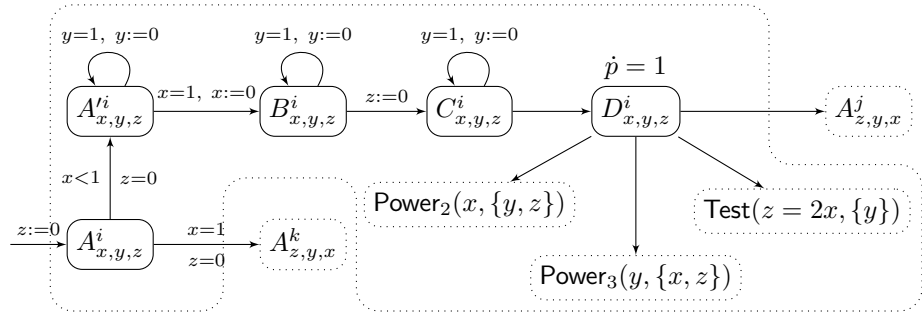
Similar ideas can be used for designing an automaton $\text{Aut}_{2,+}^i(x, y, z)$ that increments the second counter (*i.e.* ends up with $z = y/3$, while x returns to its original value), involving module $\text{Test}(x = 3z, \{y\})$.

We now treat instruction:

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

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

Figure 30: Incrementing a counter



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

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

As previously, we can prove that these three conditions express correctness of the construction. Lemma 142 clearly also holds for $\text{Aut}_{1,-}^i(x, y, z)$. Automaton $\text{Aut}_{2,-}^i(x, y, z)$ is built in the same way.

It then suffices to plug the different modules into each other following the initial two-counter machine. In order to conclude the construction, we label each $D^i_{x,y,z}$ state with an atomic proposition D , and consider the formula

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

This in particular enforces that no time is spent in the D -states along the “main” run reaching Halt.

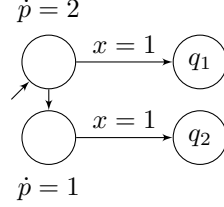
Lemma 143. $\mathcal{W}_{\mathcal{M}, q_0} \models \Phi$ iff the two-counter machine \mathcal{M} has a halting computation. \square

Remark. Notice that this reduction only uses three clocks and one (stopwatch) observer variable.

Theorem 144. The WCTL model-checking problem is decidable (in PSPACE) for nonnegative one-clock weighted timed automata.

Proof. The idea of the proof is to refine the regions. We begin with an example showing that this is necessary. Consider the 1-clock weighted timed automaton depicted on Fig. 31. Then formula $\mathbf{EF}_{\leq 1} q_1$ only

Figure 31: A one-clock weighted timed automaton



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

On the other hand, we have the following proposition:

Lemma 145. Let Φ be a WCTL formula and let \mathcal{W} be a one-clock weighted timed automaton. Then there exist finitely many constants $0 = a_0 < a_1 < \dots < a_n < a_{n+1} = +\infty$ s.t. for every location q of \mathcal{W} , for every $0 \leq i \leq n$, the truth of Φ is uniform over $\{(q, x) \mid a_i < x < a_{i+1}\}$. Moreover,

- $\{a_0, \dots, a_n\}$ contains all the constants appearing in clock constraints of \mathcal{W} ;
- the constants are integral multiples of $1/C^{|\Phi|}$ where $|\Phi|$ is the constrained temporal height of Φ , i.e. the maximal number of nested constrained modalities in Φ , and C is the lcm of all positive observer rates labeling a location of \mathcal{W} ;
- a_n equals the largest constant M appearing in the guards of \mathcal{W} ;
- $n \leq M \cdot C^{|\Phi|} + 1$.

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

Definition 146. WMTL extends LTL as follows:

$$\begin{aligned} \text{WMTL} \ni \phi_s &::= \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= p \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \mathbf{U}_{v \in I} \phi_p. \end{aligned}$$

The semantics of the new “until” modality (along weighted timed words⁸) is the same as for WCTL.

Theorem 147. WMTL model checking on TALO is undecidable.

Proof. We only sketch the reduction of the halting problem for a two-counter machine. The unique clock of the automaton will store both values of the counters. If the first (resp. second) counter has value c_1 (resp. c_2), then the value of the clock will be $2^{-c_1} 3^{-c_2}$.

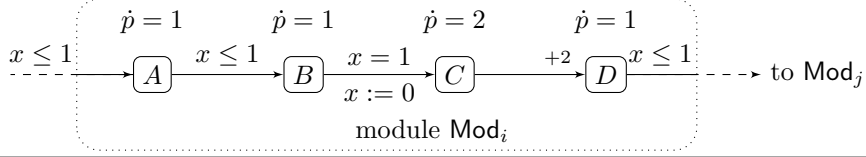
The module depicted on Fig. 32 encodes incrementation of the first counter (i.e., it divides the value of the clock by 2 between entrance and exit).

The following lemma is then easy to prove:

Lemma 148. Assume that there is a run ρ entering module Mod_i with $x = x_0 \leq 1$, exiting with $x = x_1$, and such that no time elapses in A and D and the value of the observer between A and D increases by 3. Then $x_1 = x_0/2$.

⁸We could of course also define the semantics over weighted signals.

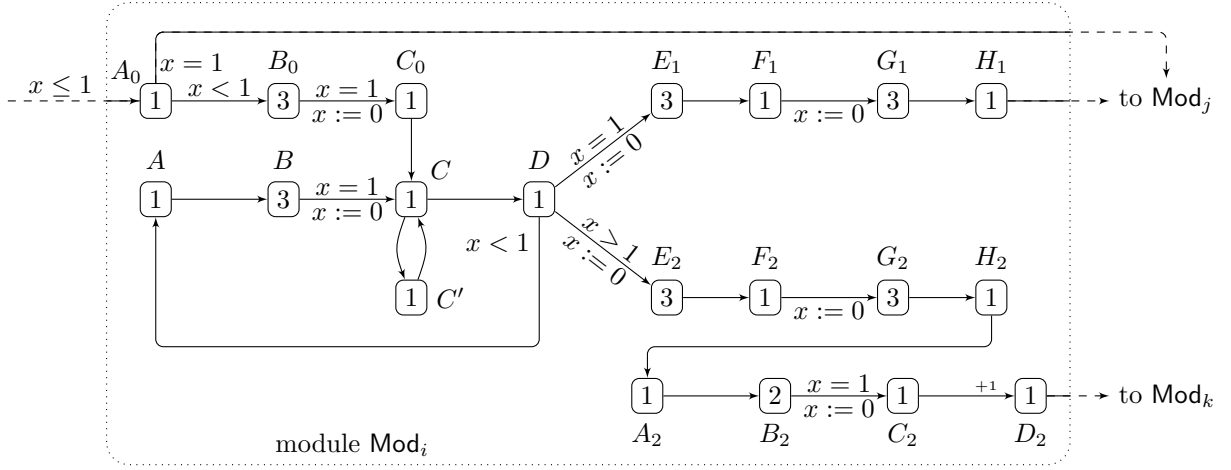
Figure 32: Module for incrementing c_1



A similar result can be obtained for a module incrementing c_2 : it simply suffices to replace the observer rate in C by 3 instead of 2.

The simulation of decrementation for counter c_1 is much more involved than the previous instruction. Indeed, we first have to check whether the value of x when entering the module is of the form 3^{-c_2} (i.e., whether $c_1 = 0$). This is achieved, roughly, by multiplying the value of x by 3 until it reaches (or exceeds) 1. Depending on the result, this module will then branch to module Mod_j or decrement counter c_1 and go to module Mod_k . The difficult point is that clock x must be re-set to its original value between the first and the second part. We consider the module Mod_i depicted on Figure 33.

Figure 33: Module testing/decrementing c_1 (the observer rates are indicated in the locations)



The following two lemmas express the correctness of the construction. We leave their proofs to the reader.

Lemma 149. *Assume there exists a run ρ entering module Mod_i with $x = x_0 \leq 1$, exiting to module Mod_j with $x = x_1$, and such that*

- no time elapses in A_0, C_0, D, A, C', F_1 and H_1 ;
- any visit to C_0 or C' is eventually followed (strictly) by a visit to C' or F_1 ;
- the observer variable increases by exactly 3 along each part of ρ between A or A_0 and the next visit in D , between C_0 or C' and the next visit in C' or F_1 , and between the last visit to D and H_1 .

Then $x_1 = x_0$ and there exists $n \in \mathbb{N}$ s.t. $x_0 = 3^{-n}$.

Lemma 150. *Assume there exists a run ρ entering module Mod_i with $x = x_0 \leq 1$, exiting to module Mod_k with $x = x_1$, and such that*

- no time elapses in $A_0, C_0, D, A, C', F_2, H_2, A_2$ and D_2 ;
- any visit to C_0 or C' is eventually followed (strictly) by a visit to C' or F_2 ;
- the observer variable increases by exactly 3 along each part of ρ between A or A_0 and the next visit in D , between C_0 or C' and the next visit in C' or F_2 , between the last visit to D and H_2 , and between H_2 and D_2 .

Then $x_1 = 2x_0$ and for every $n \in \mathbb{N}$, $x_0 \neq 3^{-n}$.

Similar modules and lemmas can be obtained for counter c_2 . Moreover, the conditions of these two lemmas can be expressed in WMTL, as well as reachability of the halting state. This concludes the reduction, proving that WMTL model checking is undecidable. \square

Remark. We used only one clock and one observer variable, which has three slopes. It can be proved that if we restrict to one clock and one stopwatch variable, the problem becomes decidable.

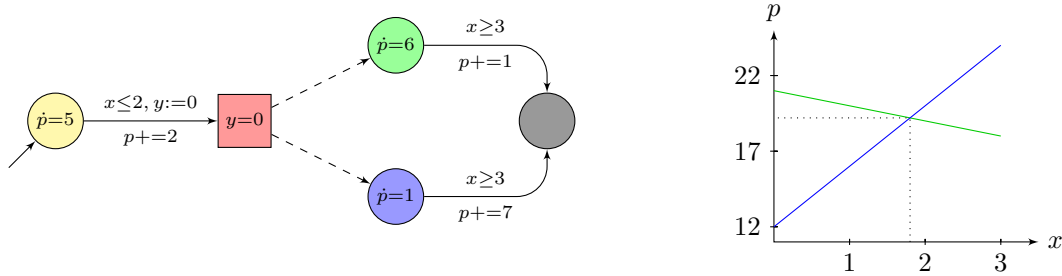
4.4 Timed game automata with linear observers

Definition 151. A timed game automaton with linear observers (TGALO for short, but sometimes also called priced timed game or weighted timed game) is a 8-tuple $\mathcal{G} = \langle S, \mathfrak{C}, T_1, T_2, \ell, \text{Inv}, \text{Var}, \text{rate}, \text{upd} \rangle$ where

- $\mathcal{A} = \langle S, \mathfrak{C}, T_1 \cup T_2, \ell, \text{Inv} \rangle$ is a timed automaton (referred to as the underlying timed automaton in the sequel);
- Var is a finite set of variables (called observers, prices or weights);
- $\text{rate}: S \rightarrow \mathbb{R}^{\text{Var}}$ indicates the derivative of the observer variables in each state.
- $\text{upd}: T \rightarrow \mathbb{R}^{\text{Var}}$ associates with each transition the values to be added to each observer upon firing this transition;

Example. Fig. 34 depicts a timed game automaton with a single linear observer (where there is only one observer p , and where we omitted to indicate weights when they equal zero). The notion of a run is the same as for TALO.

Figure 34: Example of a timed game automaton with linear observers (dashed transitions are uncontrollable)



Computing the optimal strategy in this simple example can again be achieved easily: the player only has to find the optimal delay to elapse in the first location, assuming that the opponent will select the worst branch in the second state. Hence

$$\inf_{0 \leq t \leq 2} \max\{21 - t, 12 + 4t\}$$

which equals 19.2 and is reached when $t = 1.8$. As an immediate consequence, optimal (or nearly-optimal) strategies cannot be based on regions.

The notion of strategies and compatible outcomes is the same as for classical timed games (Definition 87). The values of observer variables then just decorate the compatible runs, as for TALO.

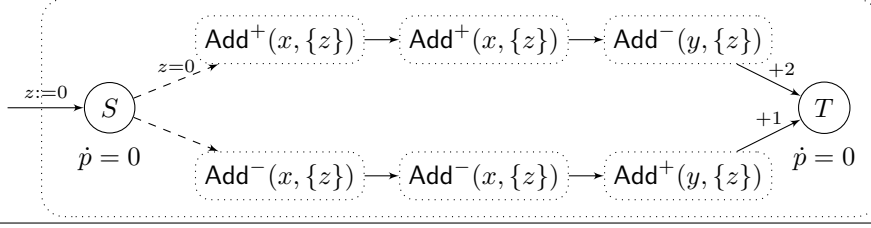
Definition 152.

Problem: *Optimal reachability in timed game automata with linear observers*

Input: *A timed game automaton \mathcal{G} with one linear observer, two states s and s' , and $m \in \mathbb{Q}$;*

Question: *Is there a strategy σ such that any compatible run from $(s, \mathbf{0}_{\mathfrak{C}}, 0)$ eventually reaches s , and such that along any compatible run, the observer variable increases by at most m before reaching s ?*

Figure 35: Automaton $\text{Test}(y = 2x, \{z\})$



Theorem 153. *Optimal reachability is undecidable in T GALO.*

Proof. The proof is similar to the proof that WCTL model-checking is undecidable. The changes are the following:

- we cannot use temporal logic formulas to enforce that the observer increases by 3 along a Test module. Hence this module is modified as depicted on Fig. 35. Apart from state S , this module is deterministic: the upper branch adds $3 + 2x - y$ to the observer, while the lower branch adds $3 - 2x + y$. Hence it is possible for Player 1 to reach T from S while increasing the observer by 3 if and only if $y = 2x$ when entering S . The modification is similar for checking $y = 3x$.
- in all the modules, the transitions leading to Test modules and Check modules are made uncontrollable. This allows the opponent player to check the corresponding conditions.
- finally, the transition leading to the halting state is assign a discrete increase of 3.

In this game, there is a strategy for reaching a terminal state (either Halt or the last state of a Test or Power module) with accumulated value 3 iff the two-counter machine halts. \square

Remark. *Notice that this game is turn-based, and uses only three clocks and one stopwatch observer.*

Theorem 154. *Optimal reachability is decidable in one-clock turn-based T GALO.*

We consider a single-observer T GALO, and a target state s . We assume that s has no outgoing edge, so that all the runs reaching s stop there. We begin with some definitions:

Definition 155. *Let \mathcal{G} be a single-observer T GALO, and s be a state.*

- for a finite run ρ in the T GALO starting in (s_0, ν_0, μ_0) and ending in (s_n, ν_n, μ_n) , we let

$$\text{obs}(\rho) = \mu_n(p) - \mu_0(p).$$

- for a configuration (s_0, ν_0) and a strategy σ for Player 1, we consider the set of runs from (s_0, ν_0, μ_0) that are compatible with σ . If there is an infinite run (hence not reaching s), the cost of this strategy from (s_0, ν_0) is set to $+\infty$. Otherwise,

$$\text{obs}((s_0, \nu_0), \sigma) = \sup\{\text{obs}(\rho) \mid \rho \text{ finite run from } (s_0, \nu_0, \mu_0) \text{ compatible with } \sigma_1\}.$$

- if there exists a winning strategy σ from (s_0, ν_0) for reaching s , then

$$\text{opt-obs}(s_0, \nu_0) = \inf\{\text{obs}((s_0, \nu_0), \sigma) \mid \sigma \text{ winning strategy from } (s_0, \nu_0) \text{ for reaching } s\}.$$

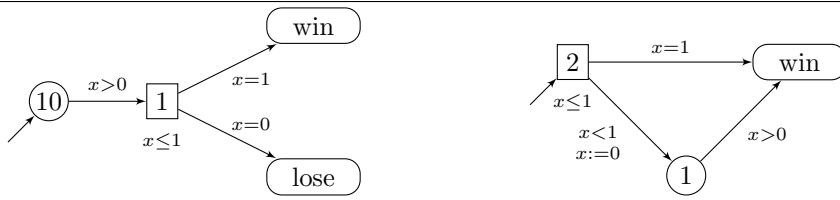
The above theorem is proved via the following result:

Theorem 156. *Let \mathcal{G} be a one-clock turn-based weighted timed game automaton, with one special goal state s_g . Then for each $s \in S$, the function $x \mapsto \text{opt-obs}_{\mathcal{G}}(q, x)$ (assigning to each initial value of the clock the optimal cost of reaching the goal state) is piecewise affine. Moreover, it can be computed exactly, and for any $\varepsilon > 0$, we can compute a memoryless strategy σ achieving cost $\text{opt-obs}(q, x) + \varepsilon$.*

Example. *Before turning to the proof, we first depict (see Fig. 36) examples where the optimal strategy does not exist or is not memoryless. Still, on both cases, there is an almost-optimal memoryless strategy.*

In the first game, the optimal observer value is 1, but it is obviously not achievable. In the second game, the optimal observer value is 2, and there is an optimal winning strategy. But this strategy must remember the date at which the transition to the lower state has been taken in order to achieve cost less than 2.

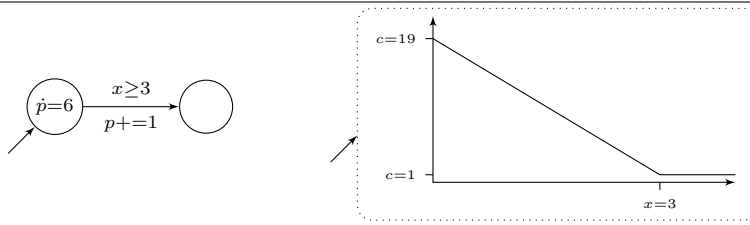
Figure 36: No optimal memoryless strategy



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

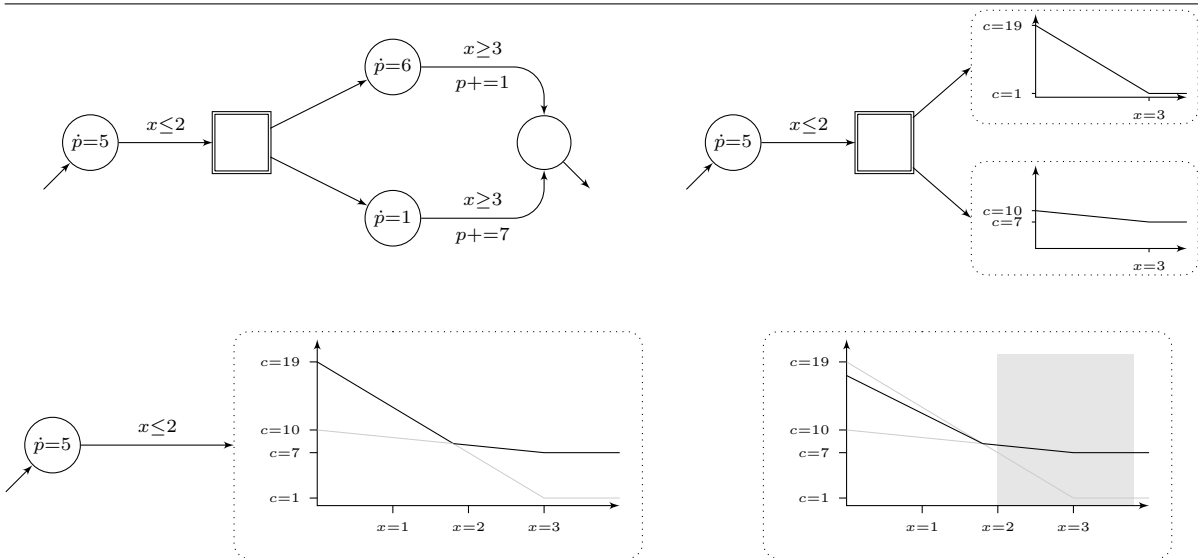
- an *urgent* state is a state where time cannot elapse. This can be modelled by adding an extra clock which is reset upon entering an urgent state, which must also be decorated with an invariant requiring that this clock must be 0.
- an *outside observer function* is a function assigning to each value of the clock the optimal observer value for winning when starting with that value of the clock. This can be used to “simplify” the game, as depicted on Fig. 37: in that figure, the game on the left is “equivalent” to the game on the right, which is reduced to the function representing the optimal observer value for winning when entering that state as a function of the value of clock x .

Figure 37: Outside observer functions



Example. We revisit our previous example of Fig. 34. This example has two clocks, but clock y can be dropped after making the second state urgent. We solve this problem by recursively computing the outside observer functions corresponding to each state, as depicted on Fig. 38. In the end, we obtain the optimal observer value for winning from the initial state for any initial value of clock x . We (hopefully) obtain the same values as computed before: the optimal observer value for winning is 17.2, and the optimal strategy consists in switching when $x = 1.8$.

Figure 38: Solving the example of Fig. 34



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

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

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

Lemma 157. *Let $\mathcal{G} = \langle S_1, S_2, S_0, X, Inv, \delta, r_s, r_t \rangle$ be a turn-based one-clock weighted timed game automaton without reset, with global invariant “ $x \leq 1$ ”, and whose underlying graph is strongly connected (or contains a single state). Let $U \subseteq S_2$ be the set of urgent locations (which are required to be uncontrollable). Let f map some of the states of \mathcal{G} to outside observer functions (those states are assumed to have no outgoing edge). Then*

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

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

- if all locations are urgent, and if there are at least two states, then Player 2 can avoid the play to reach the final state. If there is only one state, then time cannot elapse and computing the optimal observer function is straightforward;
- if there is one controllable location, then the result is easily obtained by considering $m: x \mapsto \min_{s \rightarrow s'} f(s')(x)$. The optimal observer function is then

$$\text{opt-obs}(s, x) = \inf_{x \leq x' \leq 1} r_s(s) \cdot (x' - x) + m(x').$$

An almost-optimal strategy is easily derived, which is composed of finitely many different moves (see Fig. 38 for an example);

- for the induction step, we consider one of the non-urgent locations s_{\min} having least observer rate $\text{rate}(s_{\min})$.
 - if it is controllable, then we can prove that there is no need to delay twice in s_{\min} : since it has the least rate in \mathcal{G} , it is more profitable to elapse in s_{\min} as long as possible. We can then consider the game \mathcal{G}' made of two copies of each location (except s_{\min}). We enter the second copy after visiting s_{\min} . Then both games are equivalent w.r.t. the optimal observer value, and from an almost-optimal strategy in \mathcal{G}' , we can obtain an almost-optimal one in \mathcal{G} preserving the nice properties (memorylessness, piecewise constant) of the original one.
 - if it is uncontrollable, we make it urgent, but give Player 2 an extra transition going to an outside observer function corresponding to the optimal observer value for winning in s_{\min} , which is computed iteratively (from $x = 1$). \square

4.5 Energy constraints

In this section, we lift the restriction that observer rates must be nonnegative, and allow any integer (or rational) values for observer variations.

Definition 158. Let L and U be two integers. Let \mathcal{W} be a one-observer TALO, c be an integer, and ρ be a (finite or infinite) run starting from a state (s, ν, μ) with $\mu(p) = c$. The run ρ is said to be

- L -feasible if for all state (s', ν', μ') along ρ , it holds $L \leq \mu'(p)$;
- LU -feasible if for all state (s', ν', μ') along ρ , it holds $L \leq \mu'(p) \leq U$.

Definition 159.

Problem: *Reachability under energy constraints*

Input: A TALO \mathcal{W} , two bounds L and U , two states s and s' , and an initial credit c ;

Question: Is there a run from s to s' in \mathcal{W} starting with observer value c and L - (or LU -)feasible?

Theorem 160. In the untimed case, reachability under energy constraints is in PTIME for L -feasibility, and in PSPACE (and NP-hard) for LU -feasibility.

Proof. The PTIME algorithm is a small variation of the Bellman-Ford algorithm: we inductively compute, for each state, the largest possible value of the observer variable with which this state can be reached from s in i steps. Initially, this value is $-\infty$ for all states except s , for which it is c (unless $c < L$, in which case we immediately reject). Then we look, for each edge, if it improves the current observer value of its target state, while keeping this value above the lower bound L . The basic procedure runs this for $|S|$ rounds.

The main algorithm then performs the following computations:

- it first detects states that are one a positive loop (taking the lower bound into account). As in the Bellman-Ford algorithm, this can be achieved by performing twice the above procedure. Those states in which the value is improved are assigned value $+\infty$.
- if s' is reachable from a state marked $+\infty$, or if s' has been assigned a value different from $-\infty$ in the previous step, then we accept.

The algorithm for LU -feasibility simply considers the automaton obtained by enriching each state with the current value of the observer variable. As usual, we won't build the whole automaton but guess a run in this automaton, hence using only polynomial space. Hardness in NP is easily obtained by encoding the SUBSET-SUM problem (as in the proof of Theorem 33). \square

Theorem 161. Reachability under energy constraints for L -feasibility in one-clock TALO without discrete observer updates is decidable in PTIME.

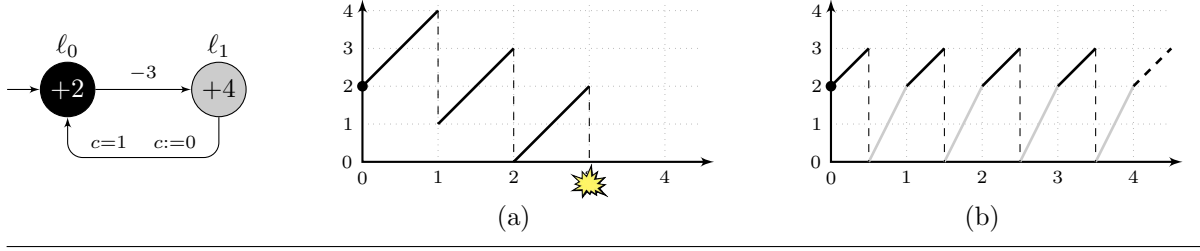
Proof. The result is proved by considering the corner-point abstraction of the TALO \mathcal{W} . The main idea is that if some time is to be spent in some state, then it is always better to spend as much time as possible in this state. Since there are no observer updates on transitions, the only constraints for firing a transition depend on the clock, hence are defined in terms of regions. See below for an example where this does not hold with discrete observer updates.

- Let γ be an L -feasible run in \mathcal{W} from $(s, 0, c)$ to s' . Then there exists an L -feasible run γ' from $(q_0, \langle\{0\}, 0\rangle, c)$ to s' in the corner-point abstraction \mathcal{C} .
- Let γ' be an L -feasible run in \mathcal{C} from $(s, \langle\{0\}, 0\rangle, c)$ to s' . Then, for any $\varepsilon > 0$, there exists an L -feasible run γ from $(s, 0, c + \varepsilon)$ to s' in \mathcal{W} .

The first part of the proof is achieved by considering slices of the run between two resets of the clock. In each slice, we can transfer (almost) all delays to the state where the observer rate is maximal among the visited states. This obviously increases the final observer value, and yields an L -feasible run. This run can be projected in the corner-point abstraction, which provides us with an L -feasible run in \mathcal{C} from s to s' . The converse is obtained by mimicking, up to ε , the run of \mathcal{C} in \mathcal{W} .

Now, the corner-point abstraction has size polynomial since we only have one clock. It suffices to apply the algorithm from the previous proof in \mathcal{C} in order to solve our initial problem. \square

Figure 39: One-clock priced timed automaton with discrete observer updates. Infeasibility of region-defined lower-bound schedule (a) and optimal lower-bound schedule (b).



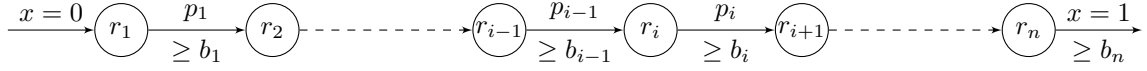
Example. In the presence of discrete observer updates, the corner-point-abstraction argument fails, as depicted in Fig. 39.

Theorem 162. Reachability under energy constraints for L -feasibility in one-clock TALO is decidable.

Proof. The proof is a bit long and tedious, we only give the important ingredients. The main ingredient is optimization along a one-time-unit path. A TALO is a unit path if it has the following shape:

- states are numbered from 1 to n , with i having exactly one successor $i + 1$ for all $i < n$;
- the first state is entered with $x = 0$, and the last one is exited with $x = 1$, and all the intermediary transitions have no guard and empty reset;
- states have observer rates r_i , and transitions have discrete observer value p_i ;
- we extend the model with an extra feature: all edges are decorated with an *observer lower-bound constraint*, of the form $p \geq b_i$. This constraint must be satisfied *before* firing the transition (*i.e.*, before adding the discrete observer value p_i) for the transition to be allowed.

The general shape of a unit path is as follows:



For such a path π , our aim is to compute the function f_π which maps an initial value of the observer to the maximal possible value of the observer at the end of the path.

Normal form for unit paths. Our first task is to simplify the unit path. If, for some reason to be explained later, we know that no time elapses in location with rate r_i along the above path, then we can drop state r_i and merge the surrounding transitions as follows:



It is easily proved that the optimal final observer value along the first path, restricted to runs where no time elapses in r_i , equals the optimal final observer value along the second path.

A unit path is said to be in *normal form* (for linear observers) if one of the following three conditions holds:

- $n = 1$ (*trivial normal form*);
- all rates are positive, and $r_i < r_{i+1}$ for $1 \leq i \leq n - 1$, and for every $1 \leq i \leq n - 1$, it holds that $b_{i-1} + p_{i-1} < b_i$ (*positive normal form*);
- all rates are negative, and $r_i > r_{i+1}$ for $1 \leq i \leq n - 1$, and for every $2 \leq i \leq n$, it holds that $b_{i-1} + p_{i-1} > b_i$ (*negative normal form*).

Any unit path can be turned into normal form as follows:

Case $\max\{r_i \mid i = 1, \dots, n\} = 0$: In this case, a L -feasible run (if any) which maximizes observer value will delay in one of the locations with rate 0, hence all other locations can be removed from the path (and the corresponding edges contracted). As a matter of fact, one only needs to keep *one* of the locations with zero rate; all others can be removed as well. Hence one arrives at the *trivial normal form*:

Lemma 163. *For any annotated path π such that $\max\{r_i \mid i = 1, \dots, n\} = 0$, an annotated path $\tilde{\pi}$ in trivial normal form can be constructed in polynomial time with $f_\pi = f_{\tilde{\pi}}$.*

Case $\max\{r_i \mid i = 1, \dots, n\} > 0$: In this case, if $r_j < r_{j-1}$ and $0 \leq r_{j-1}$ for some j , then it is possible and more profitable to spend no time in r_j , and transfer the corresponding delay to r_{j-1} . Similarly, if $r_j > r_{j-1}$ and $r_{j-1} \leq 0$ for some j , then it is possible and more profitable to transfer time from r_{j-1} to r_j . Applying these simplifications yields a path along which observer rates are positive and increasing. Finally, if $b_{i-1} + p_{i-1} \geq b_i$ along such a path, then it is possible and more profitable to spend no time in the state with rate r_i , as the condition on the observer value is immediately satisfied and the rate r_{i+1} is larger than r_i . This proves the following:

Lemma 164. *For any annotated path π such that $\max\{r_i \mid i = 1, \dots, n\} > 0$, an annotated path $\tilde{\pi}$ in positive or trivial normal form can be constructed in polynomial time with $f_\pi = f_{\tilde{\pi}}$.*

Case $\max\{r_i \mid i = 1, \dots, n\} < 0$: this is handled similarly, and yields:

Lemma 165. *For any annotated path π such that $\max\{r_i \mid i = 1, \dots, n\} < 0$, an annotated path $\bar{\pi}$ in negative (or trivial) normal form can be constructed in polynomial time with $f_\pi = f_{\bar{\pi}}$.*

Energy function We now turn to the computation of the function mapping initial to final observer value along a unit path in normal form for linear observers. For the trivial normal form this is easy, as there is only one possible run along π . For the positive normal form we detail the computations below, and the negative normal form can be handled in an analogous manner.

We define the n -tuple $t^{\text{opt}} = (t_i^{\text{opt}})_{1 \leq i \leq n}$ by

$$t_i^{\text{opt}} = \frac{b_i - (b_{i-1} + p_{i-1})}{r_i}$$

for $i < n$, and

$$t_n^{\text{opt}} = \frac{\max\{0, b_n - (b_{n-1} + p_{n-1})\}}{r_n}$$

Since the rates are all positive and $b_i > b_{i-1} + p_{i-1}$ for all $1 \leq i \leq n-1$, these values are well-defined and positive. An important equality to notice is the following:

$$b_{n-1} + p_{n-1} + r_n \cdot t_n^{\text{opt}} = \max(b_{n-1} + p_{n-1}, b_n)$$

We prove in the sequel that those delays represent the “optimal” delays one should wait in each location, and correspond to the policy where each transition is fired as soon as the observer value satisfies the lower-bound constraint ($\geq b_i$ for the transition leaving ℓ_i).

As it may be the case that the optimal delays collected in t^{opt} do not sum up to 1 (which is the total time to be spent along π), we define another tuple t^* containing the delays which (as we shall show) have to be spent on an optimal run.

- In case $\sum_{i=1}^n t_i^{\text{opt}} > 1$, we have to cut down on the time we delay in the locations. The more profitable locations are the ones with higher rates at the end of the path, hence this is where we shall spend the delays: Letting ι_π be the largest index for which $\sum_{i=\iota_\pi}^n t_i^{\text{opt}} > 1$ (so that $\sum_{i=\iota_\pi+1}^n t_i^{\text{opt}} \leq 1$), we set

$$t_i^* = \begin{cases} 0 & \text{for } i < \iota_\pi \\ 1 - \sum_{i=\iota_\pi+1}^n t_i^{\text{opt}} & \text{for } i = \iota_\pi \\ t_i^{\text{opt}} & \text{for } i > \iota_\pi \end{cases}$$

- In case $\sum_{i=1}^n t_i^{\text{opt}} \leq 1$, we may have to spend some extra time in one of the locations. The most profitable location for this delay is the last, hence we define $t_i^* = t_i^{\text{opt}}$ for $1 \leq i \leq n-1$, and $t_n^* = 1 - \sum_{i=1}^{n-1} t_i^{\text{opt}}$. We also let $\iota_\pi = 0$ in this case.

Before we prove that those delays are indeed optimal, we first compute the initial observer value needed to traverse the whole path under this policy.

- in the first case ($\iota_\pi \geq 1$), the minimal initial observer value is

$$w_{\iota_\pi}^* = b_{\iota_\pi-1} - \sum_{k=0}^{\iota_\pi-2} p_k + (t_{\iota_\pi}^{\text{opt}} - t_{\iota_\pi}^*) \cdot r_{\iota_\pi},$$

and the final accumulated cost is $\omega_{\iota_\pi}^* = \max(b_n, b_{n-1} + p_{n-1}) + p_n$;

- if $\iota_\pi = 0$, the minimal initial observer value is $w_0^* = b_0$, and the final accumulated cost is $\omega_0^* = \max(b_n, b_{n-1} + p_{n-1}) + p_n + (t_n^* - t_n^{\text{opt}}) \cdot r_n$. These values actually equal w_1^* and ω_1^* defined below.

We generalize the previous construction by letting, for $\iota_\pi + 1 \leq i \leq n$:

$$w_i^* = b_{i-1} - \sum_{k=0}^{i-2} p_k$$

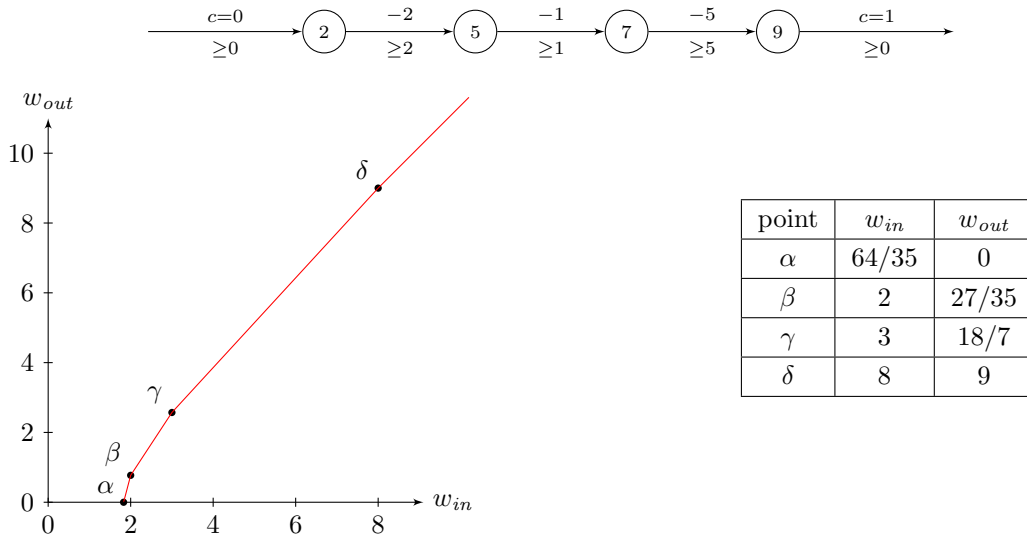
$$\omega_i^* = \max(b_n, b_{n-1} + p_{n-1}) + p_n + ((t_n^* - t_n^{\text{opt}}) + \sum_{j<i} t_j^*) \cdot r_n$$

We claim that w_i^* is the minimal initial observer value for which it is possible to spend no delay in locations ℓ_0 to ℓ_{i-1} along a feasible run, and ω_i^* is the corresponding optimal observer value at the end of the run. This can be expressed as follows:

Proposition 166. *The function f_π is a piecewise affine function defined on the interval $[w_{\iota_\pi}^*, +\infty)$, visiting points (w_i^*, ω_i^*) , for all $\iota_\pi \leq i \leq n$, with constant slope $\dot{f}_\pi(x) \geq 1$ between two consecutive such points, and with slope $\dot{f}_\pi(x) = 1$ after (w_n^*, ω_n^*) .*

Example. *We consider the unit path depicted on Fig. 40, which is already in normal form, and compute, using the above algorithm, its energy function.*

Figure 40: Function f_π for a unit path



For instance, if we enter the path with initial observer value 2, the optimal policy is to spend no time in the location with rate 2 (as we can leave it directly), then spend $1/5$ time units in the next location (so that we have value 1 and can fire the outgoing transition), then spend $5/7$ time units with rate 7, and the remaining $3/35$ time units in the location with rate 9, ending with final observer value $27/35$ (point β).

It remains to compute optimal paths for general TALO. This is achieved by first transforming the automaton in such a way that the clock is bounded by 1, and having only three types of transitions:

- non-resetting transitions with guard true,
- resetting transitions with guards $x = 0$ or $x = 1$.

This can be obtained by adding new states with rate 0. The main idea is then to compute optimal delays along parts of runs between two resets, and composing the resulting functions. We omit these technical details. \square

Remark. Notice that from the above, we deduce the existence of a positive granularity for 1-clock timed automata with energy constraints: for each such automaton, there is an integer κ such that the corner-point abstraction, refined to regions of duration $1/\kappa$, is a correct finite-state abstraction for computing optimal schedules in timed automata with observers under energy constraints.

Remark. Quite surprisingly, the same kind of results can be obtained for exponential observers, i.e., observers for which a “rate” r_i means $\dot{p} = r_i \cdot p$. In this setting, p is an exponential function, of the form $p = p_0 \cdot \exp(r_i \cdot (t - t_0))$. Under some restrictions, the above notion of normal form can be adapted to exponential observers, and we can compute optimal delays along unit paths, and then in timed automata with exponential observers. We refer to [BFLM10] for more details.

Exercice 26 ★★★ Prove that optimal reachability in three-clock timed automata with linear observers under lower-and-upper-bound energy constraints is undecidable.

Exercice 27 ★★★ Prove that optimal reachability in one-clock timed games with linear observers under lower-and-upper-bound energy constraints is undecidable.

Conclusions and research directions

Quick summary of the course

Simply-timed automata. Simply timed automata are finite-state automata whose edges carry non-negative integer *durations* (or *weight, cost*). We have introduced the corresponding *quantitative* temporal logics. The model-checking problem remains (to some extent) finite-state, and we could easily obtain model-checking algorithms (both for automata and games) with reasonable complexity (as a general rule, equality-constraints involve an exponential blow-up, while non-punctual constraints comes “for free”).

Timed automata. Timed automata extend finite-state automata with real-valued *clocks*. The state-space is then non-countable. The main tool for handling these automata is the *region abstraction*, which gathers together valuations from which the behaviour of the automaton is the same (up to small changes in the delays). There are finitely (exponentially many) regions, which provides us with a finite-state abstraction for timed automata and yields decidability (in exponential time and polynomial space) of reachability and branching-time model-checking. Quantitative linear-time temporal logics are undecidable in general, with the important exception of MITL, where punctual constraints are banned.

Hybrid automata. Hybrid automata extend timed automata with variables having richer dynamics. This directly leads to undecidability, with two notable exceptions: initialized rectangular automata and 2-dimensional polygonal systems.

Timed automata with observers. Timed automata with observer extend timed automata with hybrid variables *which do not modify the behaviour of the automaton*. This makes the optimal-reachability problem decidable. Model-checking is undecidable. Energy constraints are another semantics for timed automata with observers, in which hybrid variables do modify the behaviour of the automaton, in a weak sense. Again, partial decidability results can be obtained, but many problems remain open in that setting.

Possible research directions

Several research directions are open in this area:

- implementability of timed automata: the mathematical semantics of timed automata prevents faithful implementation of such automata on physical hardware: hardware have digital clocks and positive reaction delays, and correctness properties on the models could be lost at the implementation step. Several approaches have been proposed to circumvent this problem, but the problem of finding an *implementable* semantics for timed automata remains, for a large part, open.
- timed automata with observers under robustness: the above problem can be combined with timed automata (and even hybrid systems): not only timing informations cannot be assumed to be infinitely precise, but also the rates of observer variables, or the values of hybrid variables in general, are only known up to some precision. Undecidability proofs often rely on exact encodings with infinite precision.
- probabilistic timed systems: probabilities are an important aspect of computer science, and of verification in particular. Mixing time and probabilities is a very active research direction. In particular, probabilistic strategies in timed games (where probabilities would also concern delays) is a challenging problem.

References

- [ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.
- [AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.
- [AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, January 1996.
- [AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, May 1993.
- [AH94] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–203, January 1994.
- [AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, September 2002.
- [ALP01] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovani-Vincentelli, editors, *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer-Verlag, March 2001.
- [AM04] Rajeev Alur and Parthasarathy Madhusudan. Decision problems for timed automata: A survey. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems — Revised Lectures of the International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM-RT'04)*, volume 3185 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, September 2004.
- [ASY07] Eugene Asarin, Gerardo Schneider, and Sergio Yovine. Algorithmic analysis of polygonal hybrid systems, part I: Reachability. *Theoretical Computer Science*, 379(1-2):231–265, June 2007.
- [BBF⁺01] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, Philippe Schnoebelen, and Pierre McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-Verlag, 2001.
- [Bel57] Richard Bellman. *Dynamic Programming*. Princeton University, 1957.
- [Bel58] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.
- [BFH⁺01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Gulstrand Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovani-Vincentelli, editors, *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer-Verlag, March 2001.

- [BFLM10] Patricia Bouyer, Uli Fahrenberg, Kim Gulstrand Larsen, and Nicolas Markey. Timed automata with observers under energy constraints. In Karl Henrik Johansson and Wang Yi, editors, *Proceedings of the 13th International Workshop on Hybrid Systems: Computation and Control (HSCC'10)*, pages 61–70. ACM Press, April 2010.
- [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model-Checking*. MIT Press, May 2008.
- [BLMR06] Patricia Bouyer, Kim Gulstrand Larsen, Nicolas Markey, and Jacob Illum Rasmussen. Almost optimal strategies in pre-clock priced timed automata. In S. Arun-Kumar and Naveen Garg, editors, *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer-Verlag, December 2006.
- [BS91] Janusz A. Brzozowski and Carl-Johan H. Seger. Advances in asynchronous circuit theory part II: Bounded inertial delay models, MOS circuits, design techniques. *EATCS Bulletin*, 43:199–263, February 1991.
- [CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter C. Kozen, editor, *Proceedings of the 3rd Workshop on Logics of Programs (LOP'81)*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1982.
- [CGP00] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2000.
- [dAFH⁺03] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In Roberto Amadio and Denis Lugiez, editors, *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer-Verlag, August-September 2003.
- [Dil90] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Joseph Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer-Verlag, 1990.
- [ES84] E. Allen Emerson and A. Prasad Sistla. Deciding full branching time logic: A triple exponential decision procedure for ctl^* . In Edmund M. Clarke and Dexter C. Kozen, editors, *Proceedings of the 4th Workshop on Logics of Programs (LOP'83)*, volume 164 of *Lecture Notes in Computer Science*, pages 176–192. Springer-Verlag, 1984.
- [GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.
- [HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What is decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, August 1998.
- [HKV96] Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 514–529. Springer-Verlag, August 1996.
- [KVV00] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model-checking. *Journal of the ACM*, 47(2):312–360, March 2000.
- [LLPY97] Kim Gulstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: compact data structure and state-space reduction. In *Proceedings of the 18th Symposium on Real-Time Systems (RTSS'97)*, pages 14–24. IEEE Comp. Soc. Press, December 1997.

- [LMO06] François Laroussinie, Nicolas Markey, and Ghassan Oreiby. Model-checking timed ATL for durational concurrent game structures. In Eugene Asarin and Patricia Bouyer, editors, *Proceedings of the 4th International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 245–259. Springer-Verlag, September 2006.
- [LMS01] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking CTL⁺ and FCTL is hard. In Furio Honsell and Marino Miculan, editors, *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'01)*, volume 2030 of *Lecture Notes in Computer Science*, pages 318–331. Springer-Verlag, April 2001.
- [Min61] Marvin L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, November 1961.
- [MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In Ernst W. Mayr and Claude Puech, editors, *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer-Verlag, March 1995.
- [Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Comp. Soc. Press, October–November 1977.
- [QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Proceedings of the 5th International Symposium on Programming (SOP'82)*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, April 1982.
- [Saw03] Zdeněk Sawa. Equivalence checking of non-flat systems is exptime-hard. In Roberto Amadio and Denis Lugiez, editors, *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 237–250. Springer-Verlag, August–September 2003.
- [VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Annual Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Comp. Soc. Press, June 1986.
- [Wika] Wikipedia. Ariane 5 flight 501.
- [Wikb] Wikipedia. Mars climate orbiter.
- [Wikc] Wikipedia. Pentium FDIV bug.
- [Wikd] Wikipedia. Therac-25.