Foundations of quantitative and real-time verification

—

Temporal logics

Nicolas Markey

December 2012-February 2013

Contents

1. Introduction to model checking

1.1 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[Wikc], 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 formal verification of reactive and embedded systems.

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;
  }
}
```

Deciding the termination of a Turing machine on any input is well-known to be undecidable. As a consequence, there is no general technique for detecting bugs in computer programmes, and we have to consider simpler families of models, or go for approximate techniques (in a wide sense), whose answer might never arrive or be wrong.

In this course, we focus on the former direction: our aim is to find exact, terminating algorithms on restricted classes of models (and possibly restricted classes of properties). The rest of this chapter is devoted to introducing the basics of model-checking techniques for finite-state systems. We refer to [CGP00, BBF$^+$01, BK08] for more details on model checking.

1.2 Simple classes of models

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

1.2.1 Transition systems

Definition 1. *A* transition system[1] over AP *is a* 2*-tuple* $\mathcal{A} = \langle S, T \rangle$ *where*

- S *is the set of* states*;*
- T *is the set of transitions, equipped with two mappings* $\mathsf{src}\colon T \to S$ *and* $\mathsf{tgt}\colon T \to S$ *indicating the source and target of each transition. Unless specified otherwise, in the sequel we may consider* T *as a subset of* $S \times S$*, with* $\mathsf{src}((s,s')) = s$ *and* $\mathsf{tgt}((s,s')) = s'$*;*

Most of the transition systems we consider in the sequel will be labelled with atomic propositions. Such labelled transition systems *are triples* $\mathcal{A} = \langle S, T, \ell \rangle$ *where* ℓ *labels states of* S *with subsets of AP.*

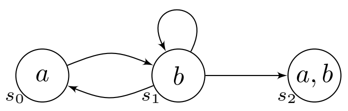
A transition system is said to be

- finite-state *if* S *is finite;*
- complete *if for all* $s \in S$*, there is a* $t \in T$ *s.t.* $\mathsf{src}(t) = s$*.*

Fig. 1 displays an example of a transition system, and its graphical representation.

Figure 1: Graphical representation of a transition system

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



Each single execution of a labelled transition system will be formalised as a word. We begin with formalising this and related notions (mostly to fix notations, as these are well-known formalisms).

Definition 2. *Let* Σ *be a finite non-empty alphabet. A* finite word *over* Σ *is a finite sequence* $w = (w_i)_{0 \le i \le n-1} \in \Sigma^n$*, for some* $n \in \mathbb{N}$*. We write* Σ^* *for the set of finite words over* Σ*.*

Let $w = (w_i)_{0 \le i \le n-1} \in \Sigma^*$*. The length* $|w|$ *if* w *is* n*. The* empty word ε *is the only word of length zero. Assuming* w *is not the empty word, we define* $\mathsf{first}(w) = w_0$ *and* $\mathsf{last}(w) = w_{|w|-1}$*. For any* $0 \le j \le k \le |w|-1$*, we define* $w_{[j,k]} = (w_i)_{j \le i \le k}$*. The prefixes of* ρ *are the subwords* $w_{[0,k]}$*, for all* $0 \le k \le n-1$*.*

Given two finite words w *and* w'*, their* concatenation *is the word* $v = w \cdot w'$ *of length* $|w| + |w'|$ *such that* $v_{[0,|w|-1]} = w$ *and* $v_{[|w|,|w|+|w'|-1]} = w'$*. Given two sets* L *and* L' *of finite words, we write* $L \cdot L' = \{w \cdot w' \mid w \in L \text{ and } w' \in L'\}$*. We inductively define* L^n *as* $L^0 = \{\varepsilon\}$ *and* $L^{i+1} = L^i \cdot L$*. Finally, we let* $L^* = \bigcup_{i \in \mathbb{N}} L^i$*.*

[1]Sometimes also called *automaton* or *graph* in the literature. In this course, *transition systems* may have infinite (possibly uncountable) state space.

Infinite words are defined similarly:

Definition 3. *An* infinite word *is an infinite sequence $w = (w_i)_{i\in\mathbb{N}}$. We write Σ^ω for the set of infinite words over Σ.*

For $0 \leq j \leq k < \infty$, the subword $w_{[i,j]}$ is defined as for finite words. For $0 \leq j < \infty$, we also define $w_{[j,\infty)}$ as the infinite word $v = (v_i)_{i\in\mathbb{N}}$ such that $v_i = w_{i+j}$ for all $i \in \mathbb{N}$. The concatenation of a finite word v with an infinite word w is the word u s.t. $u_{[0,|v|-1]} = v$ and $u_{[|v|,\infty)} = w$. For a set of finite words L and a set of infinite words L', we write $L \cdot L' = \{w \cdot w' \mid w \in L \text{ and } w' \in L'\}$. Given a set of finite words L, we write

$$L^\omega = \{w \in \Sigma^\omega \mid \forall i \geq 0.\ \exists j \geq i.\ w_{[0,j]} \in L^*\}.$$

The set of words finite words, equiped with concatenation, is a monoid. It will sometimes be convenient to use homomorphisms between this monoid $(\Sigma^*, \cdot)$ and other monoid $(M, +)$. Such homomorphisms are uniquely defined from their values on each single letter of Σ, since $f(w \cdot w') = f(w) + f(w')$. For example, defining $|\sigma| = 1$ (in $(\mathbb{N}, +)$) for each $\sigma \in \Sigma$, we get an homomorphism associating with each finite word its size. Given $\Sigma' \subseteq \Sigma$, the *projection on* Σ' is defined using the mapping $\mathsf{proj}\colon (\Sigma, \cdot) \to (\Sigma', \cdot)$ mapping letter of Σ' to themselves and letters of $\Sigma \setminus \Sigma'$ to ε. When infinite sums can be made to have a meaning in M, such mappings can be extended to infinite words in the natural way.

We can now define what *runs* of a labelled transition system are.

Definition 4. *Let $\mathcal{A} = \langle S, T, \ell\rangle$ be a labelled transition system over AP. The set of finite runs of $\mathcal{A}$ is the subset of T^* defined as follows:*

$$\mathsf{FinRuns}_{\mathcal{A}} = \{\rho = (t_i)_{0\leq i\leq n-1} \mid \mathsf{src}(t_{i+1}) = \mathsf{tgt}(t_i) \text{ for all } 0 \leq i \leq n-2\}.$$

The sequence of states *of $\rho^T = (t_i)_{0\leq i\leq n-1}$ is the finite word $\rho^S = (s_j)_{0\leq j\leq n}$ s.t. $s_i = \mathsf{src}(t_i)$ for all $0 \leq i \leq n-1$, and $s_n = \mathsf{tgt}(t_{n-1})$. When dealing with the labelling of the states, we will also use $\ell(\rho^S)$, seen as a finite word over the alphabet 2^{AP}. This word might also be denoted with ρ.*

The set of infinite runs of $\mathcal{A}$ is the subset of T^ω of infinite words having infinitely many prefixes in T^*. The sequences of states and labelling are defined as for finite words.

A run r is maximal *if it is infinite or if for any $t \in T$, $\mathsf{src}(t) \neq \mathsf{last}(r)$ (here seeing r as its sequence of states).*

For all $s \in S$, we write $\mathsf{MaxRuns}_{\mathcal{A}}(s)$ (or sometimes $\mathsf{MaxRuns}(s)$, when the underlying automaton is clear from the context) for the set of maximal *runs of $\mathcal{A}$ with first state s, and $\mathsf{FinRuns}_{\mathcal{A}}(s)$ (or $\mathsf{FinRuns}(s)$) 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.*

This view of the behaviours of a transition system as individual runs is complemented with another view of this behaviour as a tree of all the runs. We begin with formalizing the notion of trees:

Definition 5. *A Σ-labelled D-tree is a pair $\mathcal{T} = \langle T, \ell\rangle$ where $T \subseteq D^*$ is a prefix-closed set of words and $\ell\colon T \to \Sigma$ labels each element of T with a letter in Σ.*

The sequences of states of the all the runs starting from a given state form the *computation tree* of the labelled transition system: it is a 2^{AP}-labelled S-tree.

1.2.2 Symbolic representations of transition systems

Most of the systems we want to work with in practice will have a very large, but somewhat "regular", state space. This means that they might admit succinct representations.

One way of succinctly representing transition systems is to define them as the *synchronised product* of several subsystems. Subsystems in this setting are labelled transition systems in which transitions also carries a label from a finite set Σ, through a labelling function

$\lambda\colon T \to \Sigma$. In the synchonised product, when a transition labelled with $\sigma \in \Sigma$ is performed in one subsystem, the other subsystems also have to perform all available transitions labelled with σ.

Definition 6. *A* symbolic labelled transition system over AP *is a finite set of labelled transition systems $(\mathcal{A}_i)_{1\le i\le n}$ in which transitions also carry a label from a finite set Σ. More precisely, for any $1 \le i \le n$, we have $\mathcal{A}_i = \langle S_i, T_i, \ell_i, \lambda_i\rangle$. The semantics of a symbolic labelled transition system is defined as a labelled transition system $\mathcal{A} = \langle S, T, \ell\rangle$ over $AP \times \{1, ..., n\}$ obtained as follows:*

- $S = \prod_{1\le i\le n} S_i$*;*
- *there is a transition from $(s_1, ..., s_n)$ to $(s'_1, ..., s'_n)$ whenever there exists $\sigma \in \Sigma$ such that for all $1 \le i \le n$,*
  - *either there is a transition $t_i = (s_i, s'_i) \in T_i$ with $\lambda(t_i) = \sigma$,*
  - *or there is no such transition and $s'_i = s_i$.*
- *$\ell(s_1, ..., s_n)$ is the union over i of $\ell(s_i) \times \{i\}$.*

One can notice that the size of the resulting labelled transition system is exponential in the size of the input. This makes symbolic representations very useful in practice, but often has a negative effect on the complexity of the problems we will consider in the sequel.

1.2.3 Multi-agent systems

Besides the mere verification of computerized systems, a promising extension of model checking consists in considering *(controller) synthesis*: instead of just checking that a fully specified system is correct, we assume that the input system is not fully specified, and the question is whether it can be refined into a system satisfying a given property.

This setting is particularly interesting in the framework of *multi-agent systems*, where several subsystems are controlled by several agents. Such models can be conveniently dealt with through a parallel to game theory: the agents involved are named *players*, the refinements are their *strategies*, and the properties are their *winning conditions*.

We now define *concurrent game structures* (introduced in [AHK02]), which will fit our needs for modelling multi-agent systems:

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

- *$\langle S, T, \ell\rangle$ is a complete labelled transition system. In some cases, ℓ will not be used and might be omitted;*
- *$\mathbb{A}$ is a finite set of* agents *(equivalently called* players*);*
- *$\mathbb{M}$ is a finite, non-empty set of possible* actions *for the agents;*
- *$\mathsf{Ch}\colon S \times \mathbb{A} \to 2^{\mathbb{M}} \smallsetminus \{\varnothing\}$ indicates the set of allowed moves for a given agent in a given location;*
- *$\mathsf{Edg}\colon S \times \mathbb{M}^{\mathbb{A}} \to T$ returns the transition corresponding to the selected actions of the agents, with the requirement that $\mathsf{src}(\mathsf{Edg}(s, \langle m_A\rangle_{A\in\mathbb{A}})) = s$ for any $\langle m_A\rangle_{A\in\mathbb{A}}$.*

A concurrent game structure is said to be

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

Paths and related concepts in concurrent game structures are inherited through their underlying labelled transition systems.

A move vector *for $C \subseteq \mathbb{A}$ from $s \in S$ is a mapping $m_C \colon C \to \mathbb{M}$ s.t. for all $A \in C$, it holds $m_C(A) \in \mathsf{Ch}(s, A)$. Given such a move vector, we define*

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

A strategy *for $A \in \mathbb{A}$ is a mapping $\sigma_A \colon (\mathsf{FinRuns} \to \mathbb{M})$ such that for any $\rho \in \mathsf{FinRuns}$, it holds $\sigma_A(\rho) \in \mathsf{Ch}(\mathsf{last}(\rho), A)$. We write $\mathsf{Strat}_{\mathcal{A}}(A)$ for the set of strategies of A in $\mathcal{A}$, and $\mathsf{Strat}_{\mathcal{A}}$ for the set of all strategies of all agents (and omit the subscript when it is clear from the context). A strategy for a coalition $C \subseteq \mathbb{A}$ is a mapping $\sigma_C \colon A \in C \mapsto \sigma_A$, assigning a strategy to each agent of C.*

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

Let σ_C be a strategy for some coalition C. A run $\rho = s_0 \cdot t_1 \cdot s_1$ in $\mathcal{A}$ is compatible *with σ_C if for any $i < \mathsf{length}(\rho)$, it holds $t_{i+1} = \mathsf{Edg}(s_i, \langle m_A \rangle_{A \in \mathbb{A}})$ for some move vector $\langle m_A \rangle_{A \in \mathbb{A}}$ s.t. $m_B = \sigma_C(B)(s_0 \cdot t_1 \cdot s_1 \cdots t_i \cdot s_i)$.*

Given a state $s \in S$ and a strategy σ_C for some coalition C, we write $\mathsf{Out}_{\mathcal{A}}(s, \sigma_C)$ (or $\mathsf{Out}(s, \sigma_C)$ when the underlying concurrent game structure is clear from the context) for the set of maximal *runs of $\mathcal{A}$ with first state s that are compatible with σ_C, and $\mathsf{FinOut}_{\mathcal{A}}(s, \sigma_C)$ (or $\mathsf{FinOut}(s, \sigma_C)$) for the set of finite runs of $\mathcal{A}$ 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.*

Figure 2: A concurrent game structure

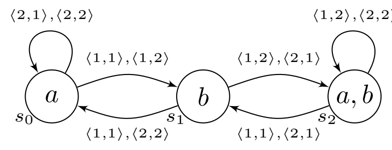


Fig. 2 displays an example of a concurrent game structure. It has two agents A and B, and two actions 1 and 2, always allowed to both agents. The Edg function is represented by decorating transitions with the set of actions they correspond to. For instance, $\mathsf{Edg}(s_0, \langle 1, 2\rangle)$ is 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. One easily sees in this example that from s_0, player A has a strategy that avoids visiting s_2; but this is not the case from s_1: any strategy of A has at least one outcome visiting s_2. These are the kind of questions that model checking will address on multi-agent systems.

1.3 Properties to be checked

Model checking aims at checking properties of models. Below we list various ways of expressing properties, and sketch some results in the various areas. The rest of this course will be mostly devoted to the study of temporal logics.

1.3.1 Reachability

The most basic property that we may want to check is the reachability of some state of the model:

Definition 8. *The* reachability problem *is defined as follows:*

Problem:	***Reachability***
Input:	*A transition system $\mathcal{A}$, and two states s and s';*
Question:	*Does there exist a finite run in $\mathcal{A}$ starting in s and ending in s'?*

Numerous algorithms exist for this problem, for all three kinds of models we defined. We first recall the following classical result for transition systems, which will serve as a basis for many of the subsequent results in this course.

Theorem 9. *The reachability problem in transition systems is decidable in deterministic polynomial time. It is* NLOGSPACE*-complete.*

Sketch of proof. We begin with a deterministic algorithm, running in polynomial time. It is based on the notion of predecessors:

Definition 10. *Let $\mathcal{A} = \langle S, T\rangle$ be a transition system, and $A \subseteq S$. The* set of predecessors *of A is the set*

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

It is easily checked that Pre is non-decreasing: if $A \subseteq A'$, then $\mathsf{Pre}(A) \subseteq \mathsf{Pre}(A')$. As a consequence, for any set A, the sequence $(C_n)_{n\in\mathbb{N}}$ defined by $C_0 = \varnothing$ and $C_{n+1} = A \cup \mathsf{Pre}(C_n)$ is non-decreasing and bounded (since S is finite), hence converges after finitely many iterations. A state s is in C_{k+1} iff there is a run of length at most k starting in s and ending in a state in A. In the end, a state s is in the limit C_∞ iff there is a run from s to some state in A. Hence, computing $\mathsf{Pre}(A)$ provides a solution to the reachability problem. It is easily seen that the fixpoint is reached after at most $|S|$ iterations, so that this (deterministic) algorithm runs in polynomial time.

The above computation can also be seen as the computation of the least fixpoint of the non-decreasing function from 2^S to 2^S which maps X to $A \cup \mathsf{Pre}(X)$: from Knaster-Tarski theorem, this function must have a least fixpoint, which is easily proved to be C_∞ (first, C_∞ clearly is a fixpoint, and aditionally any C_i is included in the least fixpoint, by induction). In the μ-calculus, the set of states from which A is reachable is usually written as $\mu T.(A \vee \mathbf{EX}\,T)$ (where μ is the operator representing the least fixpoint, and $\mathbf{EX}\,T$ means "there is a successor in T" (see Section 2.1), and precisely corresponds to the Pre operator).

In order to obtain a non-deterministic algorithm using logarithmic space, we first notice that if there is a run from s to some state in A, then there is one with length at most $|S|$. The algorithm thus just has to non-deterministically guess a sequence of states (starting from s) forming a run of A, and stop either when a state in A is reached, or when the length of the sequence exceeds $|S|$. Only the current state has to be written on the tape, as well as a counter that will be incremented at each step in order to count the number of steps. This uses only logarithmic space.

Finally, NLOGSPACE-hardness of the problem by encoding a non-deterministic Turing machine running with logarithmic space: the number of different tape contents is polynomial, and so that this Turing machine can be represented as an automaton whose states are the configurations (containing both the control state and the content of the tape), with transitions from any configuration to its possible successor configurations. The details (including the proof that the reduction can be achieved efficiently enough) are out of the scope of these notes. □

When considering symbolic transition systems, a similar algorithm can be used, but the state space to be considered is larger due to the state-space explosion. The deterministic algorithm above will run in exponential time. As we now prove, there is little hope for improvement:

Theorem 11. *The reachability problem in symbolic transition systems is decidable in deterministic exponential time. It is* PSPACE*-complete.*

Proof. The fixpoint computation can be run on the exponential state space of the input transition system, yielding a deterministic exponential-time algorithm. The non-deterministic algorithm which consists in guessing the witnessing run step-by-step can also be used, now requiring polynomial space (both for storing the current state and for limiting the size of the witnessing run).

We now prove hardness in PSPACE, by encoding a Turing machine equiped with a polynomial-size tape. We will have one transition system $\mathcal{A}_i$ per cell of the tape of the machine, whose states encode the content of that cell (together with the current state of the machine, in case the tape head is reading that cell); in other terms, the set of states of $\mathcal{A}_i$ is $\Sigma \times (Q \cup \{\varnothing\})$, and does not depend on i. Then, in each $\mathcal{A}_i$, for each transition $t = (q, a, q', b, d)$ of the Turing machine, there is a transition from (q, a) to b, labelled with $(i + d, t)$, and a transition from each state c to (q', c), labelled with (i, t). This way, any move in $\mathcal{A}_i$ from (q, a) to b also triggers the transition from c to (q', c) in s_{i+d}, which represents the left- or right neighbour of cell i.

One easily concludes the proof, by defining the initial state appropriately, and requiring that an accepting configuration is reachable. □

Finally, let us turn to reachability in multi-agent systems: now, the question is whether a given coalition has a strategy for reaching a given state, whatever the other agents do. One easily sees that this can be simplified to a two-player zero-sum[2] game, where the first player plays for the former coalition and aims at reaching the target state, while the second player controls the complement coalition and aims at avoiding the target state. This again can be solved using a fixpoint computation, as we now prove.

Theorem 12. *The reachability problem in multi-agent systems is decidable in deterministic polynomial time, and is actually* PTIME*-complete.*

Proof. We begin with defining the concept of *controllable predecessors* of a set of states A, containing those states from which some player can select a move all of whose resulting transitions lead to a state in A:

Definition 13. *Let $\mathcal{A} = \langle S, T, \mathbb{A}, \mathbb{M}, \mathit{Ch}, \mathit{Edg}\rangle$ be a concurrent game structure with $\mathbb{A} = \{1, 2\}$, and $A \subseteq S$. The* set of 1-controllable predecessors *of A is the set*

$$\mathit{CPre}_1(A) = \{s \in S \mid \exists m_1 \in \mathit{Ch}(s, 1).\ \forall m_2 \in \mathit{Ch}(s, 2).\ \mathit{tgt}(\mathit{Edg}(s, \langle m_1, m_2\rangle)) \in A\}.$$

Given a set A, computing $\mathsf{CPre}_1(A)$ can be done in polynomial time, by browsing the transition table Edg. The algorithm now follows the very same lines as the fixpoint algorithm for deciding reachability in plain transition systems: it iteratively computes the set of states from which A can be reached as the limit of the sequence defined by $C_0 = \varnothing$ and $C_{n+1} = A \cup \mathsf{CPre}_1(C_n)$. One easily sees that this algorithm is correct and runs in deterministic polynomial time.

Again, this can be seen as the least fixpoint of the function mapping a set C to $A \cup \mathsf{CPre}_1(C)$, which is non-decreasing. This can be written $\mu T.\ (A \vee \langle\!\langle 1 \rangle\!\rangle\, \mathbf{X}\, T)$ in the μ-calculus, where $\langle\!\langle 1 \rangle\!\rangle\, \mathbf{X}\, T$ means "Player 1 has a one-step strategy to reach T" (see Section 2.4).

We now prove PTIME-hardness by encoding a simplified version of the CIRCUIT-VALUE problem [Pap94]. Consider a boolean circuit (*i.e.*, a directed acyclic graph with nodes labelled with $\neg$ (having arity 1), $\vee$, $\wedge$ (both having arity at least 1), true and false (both having arity 0)). The value of such a circuit is evaluated in the obvious way. For instance, the circuits on Figure 3 evaluates to true.

From such a circuit, one can easily build another circuit contaning no negation (by creating a new subgraph in which disjunctions and conjunctions are exchanged, as well as true and false), and in which conjunctive and disjunctive nodes alternate along each branch (even if it means inserting trivial conjunctive of disjunctive nodes). Building this equivalent circuit from the initial one can be achieved using only logarithmic space (prove it!).

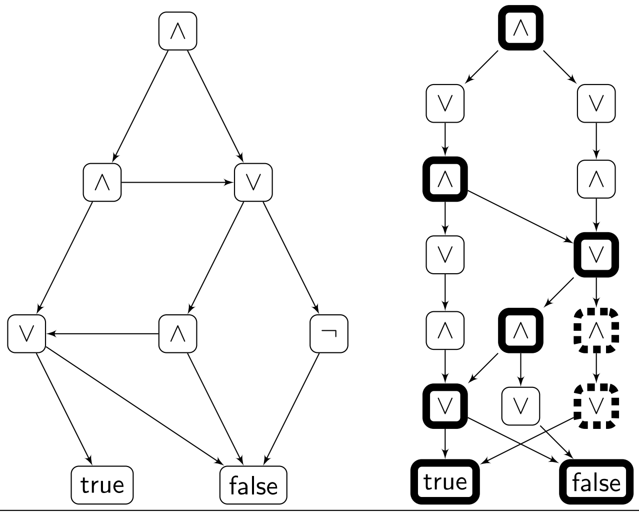
By seeing such a circuit as a (turn-based) game structure where $\vee$-states are controlled by Player 1 and $\wedge$-states are controlled by Player 2, one can show (inductively) that from any node, Player 1 has a strategy to reach true if, and only if, that node evaluates to true. □

1.3.2 Behavioural equivalences

Another natural family of properties of interest consist in comparing the behaviours of a system to the behaviours of another one. There can be numerous ways of comparing such behaviours, of which we only mention a few (and refer to [vG90] for a more exhaustive presentation).

[2]A game is *zero-sum* when it involves two players who have opposite objectives.

Figure 3: A boolean circuit and its simplified form



Definition 14. *Let $\mathcal{A}$ be a labelled transition system. Let s be a state of $\mathcal{A}$. The language generated by $\mathcal{A}$ from s is the set $\mathcal{L}_{\mathcal{A}}(s) = \{\ell(\rho) \mid \rho \in \mathsf{MaxRuns}_{\mathcal{A}}(s)\}$, where $\ell(\rho)$ is a word on 2^{AP} obtained by applying ℓ to the sequence of states of ρ.*

Two states s and s' are language-equivalent *whenever $\mathcal{L}_{\mathcal{A}}(s) = \mathcal{L}_{\mathcal{A}}(s')$.*

Definition 15. *Let $\mathcal{A}$ be a labelled transition system. A binary relation $R \subseteq S \times S$ on the set of states of $\mathcal{A}$ is a* bisimulation relation *if the following three properties are met:*

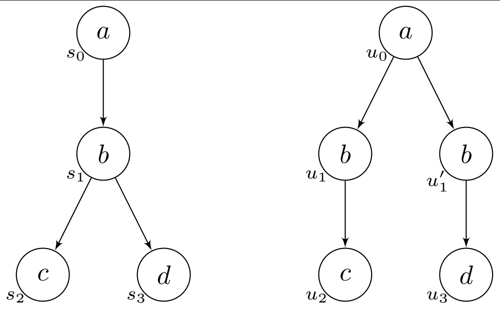
- *if $(s_1, s_2) \in R$, then $\ell(s_1) = \ell(s_2)$;*
- *if $(s, s') \in R$ and $(s, t) \in T$, then there exists a state $t' \in S$ s.t. $(s', t') \in T$ and $(t, t') \in R$;*
- *if $(s, s') \in R$ and $(s', t') \in T$, then there exists a state $t \in S$ s.t. $(s, t) \in T$ and $(t, t') \in R$.*

The relation R is a simulation *if only the first two requirements above hold.*

Two states s_1 and s_2 of a labelled transition system are (bi)similar *if there exists a (bi)simulation R such that $(s_1, s_2) \in R$.*

Basically, that s_1 and s_2 are bisimilar if any behaviour from s_1 can be mimicked from s_2 and conversely. As a classical example, consider the labelled transition system of Fig. 4: in this labelled transition system, states s_0 and u_0 are *not* bisimilar. Indeed, assume they are; then also s_1 and u_1 are. But there is no transition from u_1 corresponding to the transition (s_1, s_3). In fact, s_0 simulates u_0 (*i.e.*, there is a simulation relation containing (u_0, s_0)), as witnessed by the relation $\{(u_0, s_0), (u_1, s_1), (u_1', s_1), (u_2, s_2), (u_3, s_3)\}$. On the other hand, one easily sees that s_0 and u_0 are language-equivalent.

Figure 4: States s_0 and u_0 are not bisimilar, but are language-equivalent.



Theorem 16. *Language equivalence in labelled transition systems is decidable in deterministic exponential time, and is* PSPACE*-complete.*

Proof. This problem precisely corresponds to language equivalence in finite-state automata, for which these are classical results. The deterministic algorithm is obtained by checking emptiness of the intersection of the language of one automaton with the complement language of the second. Classical constructions can be used to build an automaton accepting this language, with an exponential blowup due to complementation. The PSPACE algorithm consists in non-deterministically building a witness word on-the-fly. □

Theorem 17. *Bisimilarity of two labelled transition systems is decidable in deterministic polynomial time, and is* PTIME*-complete.*

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

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

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 (greatest) fixpoint by Knaster-Tarski theorem. 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.

An alternative proof goes through games: given a labelled transition system $\mathcal{A}$, consider the turn-based game with state space $S \times S \times \{0, 1, 2\}$. The meaning of these sets of states is as follows:

- states in $S \times S \times \{0\}$ belong to Player 1. From some state $(s_1, s_2, 0)$, Player 1 will challenge Player 2, by selecting either s_1 or s_2, together with a transition (s, s') out of that selected state (hence $s = s_1$ or $s = s_2$). The resulting state is $(s', s_2, 2)$ or $(s_1, s', 1)$;
- states in $S \times S \times \{1, 2\}$ belong to Player 2: from there, the role of Player 2 is to answer the challenge proposed by Player 1, by matching the transition he selected. From $(s', s_2, 2)$, Player 2 has to select a transition from s_2, say (s_2, t'), and the game moves to $(s', t', 0)$; from $(s_1, s', 1)$, he has to select a transition from s_1, say (s_1, t'), and the game proceeds to $(t', s', 0)$.

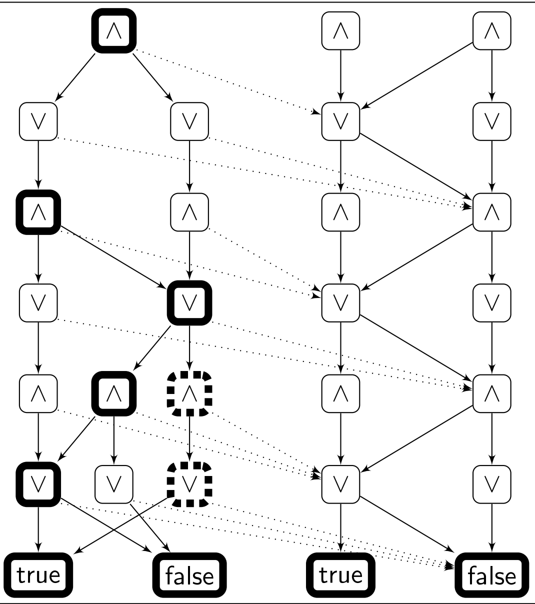
The aim of Player 2 is to avoid visiting states $(s, t, 0)$ such that $\ell(s) \neq \ell(t)$. Clearly enough, the states $(s, t, 0)$ from which Player 2 can achieve this objective are precisely the states for which s and t are bisimilar. This provides a polynomial-time algorithm for solving bisimilarity.

We now prove PTIME-hardness, again encoding the (simplified) CIRCUIT-VALUE problem (see page 10), as proposed in [BGS92]. Pick a Boolean circuit C, assuming that it has the normalized form discussed in the proof of Theorem 12.

As a first step, we build an auxiliary circuit C' as follows: C' has the same height k as C, and it has exactly two nodes at each level: we name t_i and f_i the two nodes at level i. Those nodes are of the same types as the nodes of C at the same level. At level 0, t_0 is the node true and f_0 is the node false. If i is an $\vee$-level, t_i has an edge to both t_{i-1} and f_{i-1}

and f_i has only one edge to f_{i-1}. Conversely, If i is an $\wedge$-level, t_i has an edge to t_{i-1} and f_i has edges to t_{i-1} and f_{i-1}. This way, one easily sees that for any i, t_i evaluates to true and f_i evaluates to false.

Figure 5: The boolean circuit C"



We now consider a new circuit C'' obtained as the "union" of C and C', with the following additional edges (dotted arrows on Fig. 5):

- from $\wedge$-nodes at level i of C to node t_{i-1} of C';
- from $\vee$-nodes at level i of C to node f_{i-1} of C'.

Clearly enough, this does not change the truth value of the nodes in C.

We now prove that the topmost node of C evaluates to true if, and only if, that node is bisimilar to node t_k of C', where k is the height of C. First, assume that the topmost node evaluates to true, and define relation R containing a pair (p, q) if, and only if, p and q are at the same level and have the same truth value in C''. We prove that R is a bisimulation, which will conclude this direction of the proof.

Pick a pair (p, q). If they both are at level 0, there is nothing to check as true and false have no outgoing transitions. If they both are at level 1, we consider four cases

- if nodes at level 1 are $\wedge$-nodes:
  - if nodes p and q both evaluate to true, then all their outgoing edges go to true, so that any edge from either p or q can be matched by an edge from the other node;
  - if they evaluate to false, then we prove that both p and q have edges to both states true and false: if they belong to C', this is by construction; if they belong to C, then they have at least on edge to false (as they evaluate to false), and they have been added an edge to true in the construction of C''.
- if nodes at level 1 are $\vee$-nodes: the arguments for both cases are symmetric.

Now, pick a pair (p, q) at some level $m > 1$. Considering again the same four cases, we can prove the required equivalence, hence showing that R is a bisimulation relation.

Now assume that the topmost node p_k of C is bisimilar to t_k of C'. Let R be any non-empty bisimulation relation (containing (p_k, t_k)). We inductively prove that for any two states p and q which have opposite truth value in C'', it cannot be the case that (p, q) belongs to R. This will prove our result.

First consider level 1. We assume that it contains $\vee$-nodes, with p evaluating to true and q to false. Then p must have an edge to true, while q cannot. Hence $(p, q) \notin R$. Now, consider some level $m > 1$, first assuming that both p and q are $\wedge$-nodes, with p evaluating to true and q to false. Then p can only have edges to nodes evaluating to true at level $m-1$. On the other hand, q has at leat an edge to a state that evaluates to false at level $m-1$. Pick that edge: then p has no matching edge, so that $(p, q) \notin R$. The case of $\vee$-nodes is symmetric. □

In the setting of multi-agent systems, the above relations are not really meaningful as they do not take the various agents into account. Developing this topic is beyond the scope of this course, and we just refer to *e.g.* [AHKV98] for a definition of *alternating bisimulation*, which is one way of extending bisimulations to this setting.

1.3.3 Temporal logics

We now come to temporal logics, which will be the main topic developed in the rest of this course. Temporal logics are a family of *modal logics*, which extend propositional logics with modalities for expressing *e.g.* the possibility that something holds (alethic modalities), the obligation that something holds (deontic modalities), or the knowledge that something holds (epistemic modalities).

Temporal logics include modalities for expressing *e.g.* that "*it will be the case that something holds*". They were first studied by philosophers in the fifties [Pri57], and were introduced in computer science by Amir Pnueli in the late seventies for expressing properties of computer programs [Pnu77].

We do not define temporal logics in this intorduction, as they will be the main topic discussed in this course and, as such, deserve a better place at the beginning of the next chapter.

There are three main questions that we will address in this course:

- model checking is the main one: the principal aim of the course, and actually of research in the area of model checking, is the development of efficient algorithms for checking that some property, expressed in some temporal logic, holds true in a given model;
- satisfiability is an important problem: it aims at deciding the existence of a labelled transition system in which a given formula holds true;
- expressiveness is a bit transversal: there the question is not so much algorithmic, but it aims at comparing the expressive power of different logics. The algorithmic variant aims at building translations from one logic to the other one, if any.

1.4 Exercises

Exercice 1 ⋆ Prove that the procedure on Fig. 1 does not always terminate. Propose a patch for this procedure so that it always terminates (and computes the correct value).

Exercice 2 ⋆⋆⋆ Prove that the problem of deciding whether a Turing machine halts on any input is undecidable.

Exercice 3 ⋆⋆ Prove that bisimilarity implies language equivalence.

Exercice 4 ⋆⋆ Theorems 9, 11 and 12 characterise the exact complexity of reachability for all three kind of models we considered. Extend these results to safety properties (where the aim is to avoid reaching a set of states) and repeated-reachability properties (where the aim is to visit a set of states infinitely many times).

2. Classical temporal logics: CTL, LTL, and ATL

The first (and main) part of this chapter focuses on CTL and LTL, giving a unified presentation of CTL and LTL by introducing the full branching-time temporal logic with past-time modalities (PCTL*), of which CTL and LTL are fragments. Past-time modalities in the branching-time context will have a "linear" semantics, which means that they will only refer to the history of the path being considered. Most of those logics were formally defined in the early 1980's [Pnu77, QS82, CE82, SC85, CES86, KP95], and have been much studied since then.

In a second part, we introduce ATL and provide some results about this logic which can express advanced properties of multi-agent systems.

2.1 Syntax and semantics of PCTL*

Definition 18. *The* full branching-time temporal logic with past *is denoted by* PCTL*. *Formulas of* PCTL* *over AP are formulas built on the following grammar:*

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

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

Consider a maximal run ρ of a labelled transition system $\mathcal{A}$, and an integer j s.t. $j-1 <$ length(ρ). Write s_j for the j-th state of ρ. For any PCTL* *formula ϕ, that ϕ* holds at position j along ρ in $\mathcal{A}$*, denoted by $\mathcal{A},\rho \models \phi$, is defined inductively as follows*[3]*:*

$$\begin{array}{lcl}
\mathcal{A},\rho,j \models p & \Leftrightarrow & p \in \ell(s_j) \\
\mathcal{A},\rho,j \models \neg\phi_s & \Leftrightarrow & \mathcal{A},\rho,j \not\models \phi_s \\
\mathcal{A},\rho,j \models \phi_s \vee \phi'_s & \Leftrightarrow & \mathcal{A},\rho,j \models \phi_s \textit{ or } \mathcal{A},\rho,j \models \phi'_s \\
\mathcal{A},\rho,j \models \ \mathbf{E}\phi_p & \Leftrightarrow & \exists \rho' \in \mathit{MaxRuns}.\ \rho_{\leq j} = \rho'_{\leq j} \textit{ and } \mathcal{A},\rho',j \models \phi_p \\
\mathcal{A},\rho,j \models \ \mathbf{A}\phi_p & \Leftrightarrow & \forall \rho' \in \mathit{MaxRuns}(s_j).\ (\rho_{\leq j} = \rho'_{\leq j}) \Rightarrow \mathcal{A},\rho',j \models \phi_p \\
 & & \\
\mathcal{A},\rho,j \models \neg\phi_p & \Leftrightarrow & \mathcal{A},\rho,j \not\models \phi_p \\
\mathcal{A},\rho,j \models \phi_p \vee \phi'_p & \Leftrightarrow & \mathcal{A},\rho,j \models \phi_p \textit{ or } \mathcal{A},\rho,j \models \phi'_p \\
\mathcal{A},\rho,j \models \phi_p \,\mathbf{U}\, \phi'_p & \Leftrightarrow & \exists k \in [j+1, \mathit{length}(\rho))\ \textit{s.t.}\ \mathcal{A},\rho,k \models \phi'_p \textit{ and} \\
 & & \qquad \forall l \in [j+1, k-1].\ \mathcal{A},\rho,l \models \phi_p. \\
\mathcal{A},\rho,j \models \phi_p \,\mathbf{S}\, \phi'_p & \Leftrightarrow & \exists k \in [0, j-1].\ \mathcal{A},\rho,k \models \phi'_p \textit{ and} \\
 & & \qquad \forall l \in [k+1, j-1].\ \mathcal{A},\rho,l \models \phi_p.
\end{array}$$

[3]Notice that the semantics of the "until" modality **U** is *strict*, meaning that it does not take the first state into account. Other semantics can be defined, in which the present state is taken into account when evaluating "until" formulas.

The linear-time temporal logic with past*, denoted* PLTL*, is the fragment of* PCTL^* *defined by the following grammar:*

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

The computation-tree logic with past*, denoted* PCTL *is the fragment of* PCTL^* *defined by the following grammar:*

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

Their semantics follow from the semantics of PCTL^**. The corresponding logics without past, denoted* CTL^**,* CTL *and* LTL *respectively, are obtained by dropping modality* $\mathbf{S}$ *:*

$$\begin{aligned}\mathsf{LTL} \ni \phi_s &::= \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= p \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \,\mathbf{U}\, \phi_p \\[1ex] \mathsf{CTL} \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= \phi_s \,\mathbf{U}\, \phi_s \\[1ex] \mathsf{CTL}^* \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= \phi_s \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \,\mathbf{U}\, \phi_p.\end{aligned}$$

Definition 19. *We define the additional two modalities, which we will use very often:*

$$\begin{array}{lll}\mathbf{X}\,\phi \overset{def}{=} (\neg \mathit{true})\,\mathbf{U}\,\phi & \mathbf{F}\,\phi \overset{def}{=} \mathit{true}\,\mathbf{U}\,\phi & \mathbf{G}\,\phi \overset{def}{=} \neg\mathbf{F}\,\neg\phi \\ \mathbf{X}^{-1}\,\phi \overset{def}{=} (\neg \mathit{true})\,\mathbf{S}\,\phi & \mathbf{F}^{-1}\,\phi \overset{def}{=} \mathit{true}\,\mathbf{S}\,\phi & \mathbf{G}^{-1}\,\phi \overset{def}{=} \neg\mathbf{F}^{-1}\,\neg\phi\end{array}$$

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

2.2 Expressiveness issues in PCTL^*

In this section, we review a few expressiveness results about sublogics of PCTL^*. Before entering the technical developments, we first define the precise questions we are interested in.

Definition 20. *Two state-formulas* ϕ *and* ϕ' *in* PCTL^* *are* equivalent *if for any Kripke structure* $\mathcal{A}$*, any run* ρ *and any position* j *along* ρ*, it holds* $\mathcal{A}, \rho, j \models \phi$ *if, and only if,* $\mathcal{A}, \rho, j \models \phi'$*.*

It is important to notice that equivalence is defined w.r.t. a class of models in which formulas are evaluated (here, any position along paths of Kripke structures). A different kind of equivalence, namely *initial equivalence*, can be defined by requiring that $j = 0$. Initial equivalence is usually more relevant, unless we compare two logics having past-time modalities.

Definition 21. *A logic* $\mathcal{L}$ *is* at least as expressive *as a logic* $\mathcal{L}'$ *if any formula of* $\mathcal{L}'$ *has an equivalent formula in* $\mathcal{L}$*.*

Two logics $\mathcal{L}$ *and* $\mathcal{L}'$ *are* equally expressive *whenever they are at least as expressive as each other.*

In the case where two logics are equally expressive, one can be easier to use as the other. One way of measuring the handiness of a logic is in terms of their relative succinctness:

Definition 22. *Consider two equally-expressive logics* $\mathcal{L}$ *and* $\mathcal{L}'$*. Let* f *be an increasing function from* $\mathbb{N}$ *to* $\mathbb{N}$*. We say that* $\mathcal{L}$ can be f more succinct *than* $\mathcal{L}'$ *if there is an infinite sequence of formulas* $(\phi_i)_{i\in\mathbb{N}}$ *of* $\mathcal{L}$ *whose size tends to infinity and such that for any sequence* $(\psi_i)_{i\in\mathbb{N}}$ *of formulas of* $\mathcal{L}'$ *such that* $\phi_i \equiv \psi_i$ *for all* i*, it holds* $|\psi_i| \geq f(|\phi_i|)$*.*

In order to prove that some logic $\mathcal{L}$ is *not* as expressive as another logic $\mathcal{L}'$, we need to exhibit a formula ϕ in $\mathcal{L}'$ which has no equivalent formula in $\mathcal{L}$. Proving the latter property is generally non-trivial: it requires exhibiting that for any formula ψ in $\mathcal{L}$, a model $\mathcal{A}$ in which ϕ and ψ have different truth value. There are various ways of proving such a result, some of which are presented below. Since expressiveness is not the main topic of this course, proofs will most of the time be sketchy. It is an interesting exercise for the reader to write them in full details.

2.2.1 Expressiveness of fragments of CTL

Our first expressiveness results concern CTL. We begin with restricting the set of modalities used for defining CTL. This sometimes helps simplifying the proofs, replacing the binary modality **AU** with the unary **EG**.

Theorem 23. CTL *and the following fragments of* PCTL* *are equally expressive:*

$$\widetilde{\mathsf{CTL}} \ni \phi_s ::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_s \,\mathbf{U}\, \phi_s \mid \mathbf{A}\phi_s \,\mathbf{U}\, \phi_s$$
$$\widehat{\mathsf{CTL}} \ni \phi_s ::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_s \,\mathbf{U}\, \phi_s \mid \mathbf{EG}\, \phi_s$$

Proof. One can easily check that CTL and $\widetilde{\mathsf{CTL}}$ are syntactically equivalent. Then, using the following equivalence, any formula in $\widetilde{\mathsf{CTL}}$ can be translated into $\widehat{\mathsf{CTL}}$:

$$\mathbf{A}\psi \,\mathbf{U}\, \phi \equiv \neg\,\mathbf{EG}\,\neg\phi \wedge \neg\,\mathbf{E}(\neg\phi)\,\mathbf{U}\,(\neg\phi \wedge \neg\psi).$$

It remains to prove this equivalence. Assume that $\mathbf{A}\psi \,\mathbf{U}\, \phi$ fails to holds at some position j of a run ρ of $\mathcal{A}$. This means that there is a run ρ' from s_j (the j-th state of ρ) along which $\psi \,\mathbf{U}\, \phi$ fails to hold. There are two ways of falsifying an "until" formula:

- either ϕ never holds true along ρ',
- or ϕ is eventually satisfied, but ψ fails to hold at some point before the first position where ϕ holds.

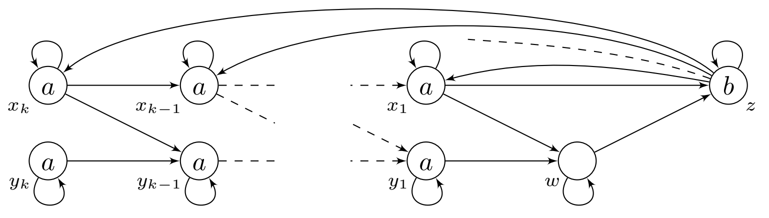
Hence $\neg(\mathbf{A}\phi \,\mathbf{U}\, \psi)$ is equivalent to the CTL* formula $\mathbf{E}(\mathbf{G}\,\neg\phi \vee (\neg\phi)\,\mathbf{U}\,(\neg\phi \wedge \neg\psi))$ from which our equivalence follows. ☐

Of course, it would be nice if we could even go further and drop one of **EG** or **EU**. But:

Theorem 24. **EU** *cannot be expressed using only* **EG** *and boolean operators, and vice versa.*

Proof. We prove the first claim, and reinforce the result by showing that **EU** cannot be expressed using **EF** and **AU** (this was shown in [Lar95], and is a negative counterpart to the fact that any CTL formula can be expressed using **EU** and **EG**), by considering the Kripke structure depicted on Fig. 6. We prove that no formula of size i built using **EF** and **AU** can distinguish between states x_j and y_j, for any $j \geq i$. On the other hand, one easily checks that **EU** can distinguish between x_j and y_j for any j, using *e.g.* $\mathbf{E}a \,\mathbf{U}\, b$.

Figure 6: States x_k and y_k cannot be distinguished by small **EG**-formulas



- the base case ($i = 1$) is trivial, as x_j and y_j carry the same atomic propositions.
- Assume the result holds for formulas of size at most $i-1 \geq 1$, and consider a formula ϕ of size i. If ϕ is a boolean combination of subformulas, those subformulas satisfy the equivalence, hence so does ϕ. Otherwise:
  - if $\phi = \mathbf{EF}\,\psi$: the sets of states reachable from x_j and from y_j are equal. The result for this case immediately follows.
  - if $\phi = \mathbf{A}\phi_1\,\mathbf{U}\,\phi_2$: assume $x_j \models \phi$. Because of the self-loop on x_j, it holds $x_j \models \phi_2$, and by induction, also $y_j \models \phi_2$. Consider a path from y_j which does not loop in y_j. Then this path goes to y_{j-1}. This path can be mimicked from x_j, so that it also has to satisfy ϕ. This proves one direction of the equivalence.

    Conversely, assume $y_j \models \phi$. Then $y_j \models \phi_2$, and so does x_j, which handles the path looping on x_j. By considering the path from y_j going to y_{j-1} and looping there, we get $y_{j-1} \models \phi_2$, which entails $x_{j-1} \models \phi_2$ by induction, which in turn entails that $x_j \models \phi$.

We now prove the converse, namely, that **EG** cannot be expressed using **EU**. The structure we use is adapted from that of Fig. 6 as follows:

- all states except w are labelled with a;
- there are no self-loops;
- there is a transition from any state in $\{z, x_1, \ldots, x_k\}$ to any other state (except the source state itself);
- there is a transition from y_j to y_l whenever $j > l$;
- there is a transition from any y_j to w.

We do not represent this structure as it has too many transitions. One can check that $x_j \models \mathbf{EG}\,a$, while $y_j \not\models \mathbf{EG}\,a$ as any maximal path from y_j will visit w. It remains to prove that no formula of size at most j built using only modality **EU** can distinguish between any two states in $\{z, x_1, \ldots, x_k, y_j, \ldots, y_k\}$. Again, the proof proceeds by induction:

- when $j = 1$, the result is straightforward.
- assume the result holds for formulas of size $j - 1 \geq 1$, and consider $\phi = \mathbf{E}\phi_1\,\mathbf{U}\,\phi_2$. If any state in $\{z, x_1, \ldots, x_k\}$ satisfies ϕ, then they all do as the same path can be taken from any of them. Now, if $z \models \phi$, then several cases ay occur, depending on which state witnesses ϕ_2:
  - if $w \models \phi_2$, then $y_j \models \phi$ since there is a direct path to w from y_j;
  - if some y_m witnesses ϕ_2 with $m < j$, then again this state is directly reachable from y_j;
  - if some y_m witnesses ϕ_2 with $m \geq j$, then using the induction hypothesis, also $y_{j-1} \models \phi_2$, and $y_l \models \phi$ for any $l \geq j$.

  Finally, if $y_l \models \phi$ for some $l \geq j$, then all states in $\{z, x_1, \ldots, x_k\}$ can mimick the same path, as well as the states in $\{z_l, \ldots z_k\}$. We have to prove the result for states in $\{y_j, \ldots, y_{l-1}\}$, again considering several cases depending on the witness for ϕ_2:
  - if the witness is $\{z, x_1, \ldots, x_k, y_l, \ldots, y_k\}$, then the witnessing path from y_l will visit w and then follow some suffix γ from there. From any state in $\{y_1, \ldots, y_k\}$, a path going directly to w and then following γ witnesses ϕ;
  - if the witness is $\{w, y_1, \ldots, y_{j-1}\}$, then there is a direct path to that state from $\{y_j, \ldots, y_{l-1}\}$, and the result follows. □

We conclude this section with a note on our semantics of modality "until": the semantics we use in these notes is that of *strict until*, where the present state cannot be used as a witness for the right-hand-side formula. A similar, non-strict modality can be defined as follows:

$$\mathcal{A},\rho,j \models \phi_p \,\hat{\mathbf{U}}\, \phi'_p \quad\Leftrightarrow\quad \exists k \in [j, \mathsf{length}(\rho)) \text{ s.t. } \mathcal{A},\rho,k \models \phi'_p \text{ and } \forall l \in [j, k-1].\ \mathcal{A},\rho,l \models \phi_p.$$

One can easily check that

$$\phi \,\mathbf{U}\, \psi \equiv \mathbf{X}\,(\phi \,\hat{\mathbf{U}}\, \psi) \qquad\qquad \phi \,\hat{\mathbf{U}}\, \psi \equiv \psi \vee (\phi \wedge \phi \,\mathbf{U}\, \psi).$$

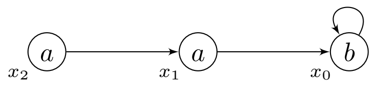
In other terms, the logic

$$\widehat{\mathsf{CTL}} \ni \phi_s ::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_s \,\hat{\mathbf{U}}\, \phi_s \mid \mathbf{EX}\,\phi_s \mid \mathbf{E}\hat{\mathbf{G}}\,\phi_s$$

(where $\mathbf{E}\hat{\mathbf{G}}$ is the relaxed version of **EG** also requiring the formula to hold at the present time) is equally expressive as CTL. It is not too difficult to prove that dropping **EX** would change expressiveness:

Theorem 25. **EX** *cannot be expressed using only* $\mathbf{E}\,\hat{\mathbf{U}}$ *and* $\mathbf{E}\hat{\mathbf{G}}$*.*

Figure 7: **EX** cannot be expressed with $\mathbf{E}\,\hat{\mathbf{U}}$ and $\mathbf{E}\hat{\mathbf{G}}$



Proof. Consider the three-state labelled transition system of Fig. 7. It is straightforward to prove that x_1 and x_2 satisfy the same formulas built with $\mathbf{E}\,\hat{\mathbf{U}}$ and $\mathbf{E}\hat{\mathbf{G}}$, while $\mathbf{EX}\,b$ can separate them. □

Symmetrically, $\mathbf{E}\,\hat{\mathbf{U}}$ cannot be expressed using only **EX** and $\mathbf{E}\hat{\mathbf{G}}$, since this would contradict Theorem 24.

2.2.2 CTL$^+$ and CTL

Now, let us look at extensions of CTL. One of them, CTL$^+$, allows to have boolean combinations of (simple) path formulas within a path quantifier:

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

This way, formula $\mathbf{E}(\mathbf{F}\,p \wedge \mathbf{F}\,q)$ is a CTL$^+$ formula, which syntactically does not belong to CTL. But clearly

$$\mathbf{E}(\mathbf{F}\,p \wedge \mathbf{F}\,q) \equiv \mathbf{EF}\,(p \wedge q) \vee \mathbf{EF}\,(p \wedge \mathbf{EF}\,q) \vee \mathbf{EF}\,(q \wedge \mathbf{EF}\,p), \tag{1}$$

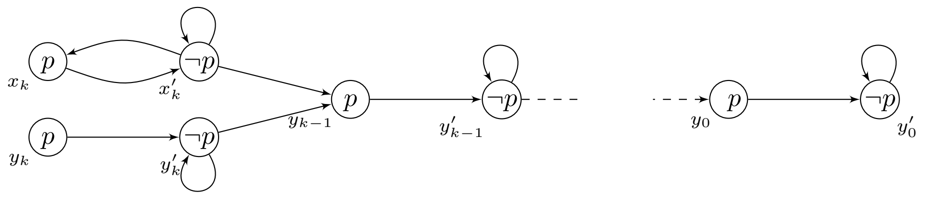
obtained by enumerating all possible orders of occurrence of p and q. It can be proved that:

Theorem 26. CTL$^+$ *and* CTL *are equally expressive, but* CTL$^+$ *can be exponentially more succinct.*

Proof. The translation follows the idea of Equation (1): it consists in considering all possible interleavings of the eventualities to be satisfied, and imposing all the required subformulas in the meantime.

Proving the succinctness gap is quite involved: the interested reader could look at [Wil99, AI03, Lan08] for detailed proofs of the result. Here we just mention the idea behind the proof in [Wil99]: First notice that formula $\mathbf{E}(\mathbf{F}\,p_1 \wedge \mathbf{F}\,p_2 \wedge \ldots \wedge Fp_n)$, which is our witness

Figure 8: States x_k and y_k cannot be distinguished by small CTL formulas



formula for the succinctness gap, has size $O(n)$. It can then be shown (see Section 2.3.5 where we prove this) that with any CTL formula ϕ of size l, we can associate an alternating tree automaton of size l accepting precisely those trees where ϕ holds. To conclude, it can be proved (this is the hard part) that any alternating tree automaton for our witness formula has size at least $2^{O(n)}$.

Similar ideas are used in the proof of Theorem 28, but in the simpler setting of automata on words. □

2.2.3 CTL and fairness

Fairness is a family of properties which, roughly, involve infinite occurrences of some events: for instance, "*if p occurs infinitely many times, then also q occurs infinitely many times*". That p occurs infinitely many times along an (infinite) run can be expressed as $\mathbf{G}\,\mathbf{F}\,p$. This construct unfortunately is not allowed in CTL, and we can prove that it does not have an equivalent formulation in CTL:

Theorem 27. *Formula $\mathbf{EG}\,\mathbf{F}\,p$ cannot be expressed in* CTL*.*

It follows that CTL is not as expressive as CTL* and LTL.

Proof. Consider the structure depicted on Fig. 8. One easily sees that $\mathbf{EG}\,\mathbf{F}\,p$ holds true from x_k and not from y_k. Then, by induction on the size of the formula, it can be proved that any CTL formula of size at most j has the same truth value on both x_l and y_l, for any $l \geq j$. This part is left to the reader as an exercice. □

2.2.4 LTL and PLTL

We conclude with a classical result concerning past-time modalities in linear-time temporal logics:

Theorem 28. LTL *and* PLTL *are equally expressive, but* PLTL *can be exponentially more succinct.*

Obviously, this result assumes *initial equivalence*, and would not hold for general equivalence. It also assumes discrete time: when considering continuous time (which we will do when considering temporal logics for timed models), formulas such as $\neg(\neg p\,\mathbf{S}\,\mathsf{true})$, which would be equivalent to false in discrete time, can be satisfied in continuous time: for instance, such a formula would be true at time one along a run where p holds precisely at dates $1 - 1/n$ for all $n > 0$. We leave it to the reader to formally prove that PLTL is strictly more expressive than LTL in this setting.

Proof. The proof relies on a translation to first-order logic (FO), which is the logic built on unary predicates, first-order quantification and an ordering relation. Over the integers equiped with their classical ordering relation, we can write for instance

$$\exists x.\ [p_b(x) \wedge \forall y.\ (y < x \Rightarrow p_a(y))].$$

This formula precisely expresses $a\,\mathbf{U}\,b$.

The fact that LTL and PLTL have the same expressive power was historically proven in two steps: first in [Kam68], PLTL is shown to be equivalent to FO. Since the semantics of

PLTL is (or can easily be) expressed in FO, only the converse inclusion is really involved. Then, in [GPSS80], the same is shown for LTL, which entails the expressiveness result. We omit the details of those proofs.

We now prove the succinctness result, using the following property

> Any two states that agree on atomic propositions p_1 to p_n also agree on proposition p_0. (2)

This formula can easily be expressed in LTL, e.g. by enumerating the set of valuations for p_1 to p_n:

$$\bigwedge_{a_0,a_1,\ldots,a_n\in\{\top,\bot\}} (\mathbf{F}\,(\bigwedge_{i\in[0,n]} p_i = a_i)) \Rightarrow (\mathbf{G}\,((\bigwedge_{i\in[1,n]} p_i = a_i) \Rightarrow p_0 = a_0)).$$

The set of words satisfying this property can thus be accepted by a Büchi automaton (see Section 2.3.2 for a proof of this result). Such an automaton cannot be too small:

Lemma 29 ([EVW02]). *Any Büchi automaton that accepts exactly the set of words satisfying Property (2) has at least 2^{2^n} states.*

Proof. Let $\{a_0, a_1, ..., a_{2^n-1}\}$ be the set of letters built on atomic propositions $\{p_1, ..., p_n\}$ (note that p_0 is not included here). There are 2^n such letters.

For each subset $K \subseteq \{0, 1, ..., 2^n - 1\}$, we define a set of letters $\{b_0, ..., b_{2^n-1}\}$ as follows:

$$b_i = \begin{cases} a_i & \text{if } i \notin K \\ a_i \cup \{p_0\} & \text{otherwise.} \end{cases}$$

Then, with each subset K, we associate a finite word $w_K = b_0\, b_1 \ldots b_{2^n-1}$. Clearly, whenever $K \neq K'$, then $w_K \neq w_{K'}$. Thus, this construction defines 2^{2^n} different finite words. It is clear also that for any K, the word ${w_K}^\omega$ satisfies Equation (2), while the word $w_{K'} \cdot {w_K}^\omega$ does not, as soon as $K \neq K'$.

Now, let $\mathcal{A}$ be an automaton accepting exactly the words satisfying Equation (2). Let S_K be the set of states that can be reached from an initial state of $\mathcal{A}$ after reading the finite word w_K. Clearly, this set is non-empty, since w_K is the prefix of an accepted word. Now, if $S_K \cap S_{K'} \neq \varnothing$ for two different sets K and K', then the word $w_{K'} \cdot {w_K}^\omega$ would be accepted by the automaton. Thus the sets S_K do not intersect, which entails that $\mathcal{A}$ has at least 2^{2^n} states. □

In the proof of Theorem 38, we will show how any PLTL formula of size k can be transformed into an alternating automaton having at most 2^{2k} states. As a consequence, any PLTL formula expressing Property (2) has size at least 2^{n-1}.

We now consider a slightly different property:

> Any state that agrees with the initial state on atomic propositions p_1 to p_n also agree with the initial state on proposition p_0. (3)

This formula can be captured by a polynomial-size PLTL formula:

$$\mathbf{G}\left[(\bigwedge_{i\in[1,n]} (p_i \iff \mathbf{F}^{-1}\,\mathbf{G}^{-1}\,p_i)) \Rightarrow (p_0 \iff \mathbf{F}^{-1}\,\mathbf{G}^{-1}\,p_0)\right].$$

It can thus also be expressed in LTL (again, possibly by enumerating the set of initial valuations). Let ϕ be an LTL formula expressing the property of Equation (3). Since ϕ is a pure-future formula, formula $\mathbf{G}\,\phi$ precisely expresses Property (2), and thus has size at least 2^{n-1}. Thus $|\phi| \geq 2^{n-1} - 1$.

2.3 Model-checking and satisfiability for PCTL^*

In this section, we study the complexities of the model-checking and satisfiability problems for (some of) the logics we defined. We begin with providing a formal definition of those problems:

Definition 30. *The model-checking problem is defined as follows:*

Problem:	PCTL^* ***model checking***
Input:	*An automaton $\mathcal{A}$, a run ρ, a position j along ρ, and a* PCTL^* *formula ϕ;*
Question:	*Is it the case that $\mathcal{A}, \rho, j \models \phi$?*

The satisfiability problem is defined as follows:

Problem:	PCTL^* ***satisfiability***
Input:	*A* PCTL^* *formula ϕ;*
Question:	*Does there exist an automaton $\mathcal{A}$, a run ρ, a position j along ρ, such that $\mathcal{A}, \rho, j \models \phi$?*

To begin with, the following lemma simplifies the above definitions, by showing that the value of a state-formula only depends on the prefix of the run. As a consequence, we will assume in the sequel that only the prefix $\rho_{\leq j}$ is given in the input.

Lemma 31. *Given two runs ρ and ρ' of an automaton $\mathcal{A}$, and a position j s.t. $\rho_{\leq j} = \rho'_{\leq j}$, it holds*

$$\forall \phi_s \in \mathsf{PCTL}^*. \quad \mathcal{A}, \rho, j \models \phi_s \Leftrightarrow \mathcal{A}, \rho', j \models \phi_s.$$

Proof. By induction on the structure of ϕ_s. □

Theorem 32 ([Pnu77, CE82, QS82, ES84b, Boz08])**.** *The model-checking and satisfiability problems are decidable for* PCTL^* *(hence for all its fragments). The complexity of those problems is given in Table 1.*

Table 1: Complexity of model checking and satisfiability for fragments of PCTL^*

	model checking	symbolic model checking	satisfiability
CTL	PTIME-c.	PSPACE-c.	EXPTIME-c.
LTL	PSPACE-c.	PSPACE-c.	PSPACE-c.
PLTL	PSPACE-c.	PSPACE-c.	PSPACE-c.
CTL^*	PSPACE-c.	PSPACE-c.	2-EXPTIME-c.
PCTL	PSPACE-h.	PSPACE-h.	EXPTIME-c.
PCTL^*	in 2-EXPTIME	in 2-EXPTIME	2-EXPTIME-c.

The rest of this section is devoted to the proofs of some of these results.

2.3.1 CTL model checking

Theorem 33. *The* CTL *model-checking problem is* PTIME*-complete.*

Using the fact that CTL^* has no past-time modalities, Lemma 31 can be reinforced as follows:

Lemma 34. *Given two runs ρ and ρ' of an automaton $\mathcal{A}$, and two positions j and j' s.t. $s_j = s'_{j'}$, it holds*

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

Proposition 35. *Let α, β be two subsets of the set of states of a finite-state automaton $\mathcal{A}$. We define the following two sequences:*

$$\begin{aligned} E_0 &= \varnothing & E_{i+1} &= \mathit{Pre}(\beta \cup (\alpha \cap E_i)) \\ A_0 &= S & A_{i+1} &= S \setminus \mathit{Pre}(S \setminus (\beta \cup (\alpha \cap A_i))) \end{aligned}$$

Then both sequences converge in finite time, and their limits (which we write E_∞ and A_∞) enjoy the following properties (where we assume that the states in α and β are labelled with corresponding atomic propositions α and β):

$$\begin{aligned} \mathcal{A}, \rho, j \models \mathbf{E}\alpha\,\mathbf{U}\,\beta \quad &\iff \quad s_j \in E_\infty \\ \mathcal{A}, \rho, j \models \mathbf{A}\alpha\,\mathbf{U}\,\beta \quad &\iff \quad s_j \in A_\infty \end{aligned}$$

Proof. We begin with proving that the limits exist. Remember that Pre (see Definition 10 on page 8) is non-decreasing; as a consequence, we inductively prove that both sequences $(E_i)_i$ and $(A_i)_i$ are non-decreasing:

- clearly, $E_0 \subseteq E_1$, since $\beta \subseteq \beta \cup (\alpha \cap E_0)$. Similarly, $A_0 \subseteq A_1$.
- Now, if $E_{i-1} \subseteq E_i$, then $\alpha \cap E_{i-1} \subseteq \alpha \cap E_i$, and $\beta \cup (\alpha \cap E_{i-1}) \subseteq \beta \cup (\alpha \cap E_i)$, and finally $E_i \subseteq E_{i+1}$. The arguments for showing $A_i \subseteq A_{i+1}$ are similar.

Since the state space is finite, those two non-decreasing sequences converge.

Pick a run ρ and a position j such that $\mathcal{A}, \rho, j \models \mathbf{E}\alpha\,\mathbf{U}\,\beta$. Following Lemma 34, we equivalently have $\mathcal{A}, \rho_{\geq j}, 0 \models \mathbf{E}\alpha\,\mathbf{U}\,\beta$, which means that there is a maximal run ρ' from s_j for which $\mathcal{A}, \rho', 0 \models \alpha\,\mathbf{U}\,\beta$. This in turn means that for some position k, it holds $s'_k \in \beta$, and for all $l \in [1, k-1]$, $s'_l \in \alpha$. We can prove by induction that $s'_m \in E_{k-1-m}$ for all $m \in [0, k-1]$: first, s'_{k-1} has a successor in β (namely s'_k); moreover, if $s'_m \in E_{k-1-m}$ for all $m \in [1, k-1]$, since also $s'_m \in \alpha$, we have that s'_{m-1} has a successor in $\alpha \cap E_{k-1-m}$ (namely s'_m), hence $s'_{m-1} \in E_{k-m}$.

Conversely, if $s_j \in E_\infty$, then $s_j \in E_m$ for some m. By induction on m, we can prove that there is a maximal path from s_j satisfying $\alpha\,\mathbf{U}\,\beta$: if $m = 0$, the result is trivial. Assume the result holds for some m, and assume $s_j \in E_{m+1}$. Then s_j has a successor in $\beta \cup (\alpha \cap E_m)$. If that successor is in β, the result follows. Otherwise, that successor is in $\alpha \cap E_m$, which gives rise to a maximal run from s_j satisfying $\alpha\,\mathbf{U}\,\beta$.

The proof of the second equivalence using similar arguments, and is left to the meticulous reader. □

Proof of Theorem 33. From Proposition 35, we obtain a model-checking algorithm for CTL (see Algorithm 2), consisting in inductively computing the sets of states satisfying the subformulas of the main formula being checked. It is not difficult to observe that the algorithm runs in time $O(|S| \times |\phi|)$.

It remains to prove that the problem is PTIME-hard. This is proven by expressing $\mathsf{CIRCUIT\text{-}VALUE}$ (see page 10) as a CTL model-checking problem.

The value of the circuit can be obtained by model checking the following CTL formula from the topmost vertex of the circuit, seen as a labelled transition system:

$$\mathbf{AX}\,(\,\mathbf{EX}\,(\,\mathbf{AX}\,(\,\mathbf{EX}\,(\,\mathbf{AX}\,(\,\mathbf{EX}\,\mathsf{true})))))$$

(so it turns out that even the fragment of CTL containing only $\mathbf{EX}$ has PTIME-complete model checking). □

Theorem 36. *The CTL symbolic-model-checking problem is PSPACE-complete.*

Proof. We begin with describing a (naive) algorithm requiring exponential time: it simply consists in explicitely building the transition system resulting from the synchronisation of the input systems, and applying the polynomial-time algorithm to that exponential-size system.

Algorithm 2: CTL model-checking algorithm

```
function CTLmc($\phi$) {
  // returns the set of states
  // satisfying $\phi$

  switch $\phi$ {
    case p:
      R := $\{s \in S \mid p \in \ell(s)\}$;
      return R;
    case $\neg\alpha$:
      R := S\CTLmc($\alpha$);
      return R;
    case $\alpha \vee \beta$:
      R := CTLmc($\alpha$) $\cup$ CTLmc($\beta$);
      return R;
    case $\mathbf{E}\alpha\,\mathbf{U}\,\beta$:
      A := CTLmc($\alpha$);
      B := CTLmc($\beta$);
      X := $\varnothing$;
      Y := Pre(B);
      while (X$\neq$Y) {
        X := Y;
        Y := Pre(B$\cup$(A$\cap$X));
      }
      R := Y;
      return R;
    case $\mathbf{A}\alpha\,\mathbf{U}\,\beta$:
      A := CTLmc($\alpha$);
      B := CTLmc($\beta$);
      X := $\varnothing$;
      Y := S\Pre(S\B);
      while (X$\neq$Y) {
        X := Y;
        Y := S\Pre(S\(B$\cup$(A$\cap$X)));
      }
      R := Y;
      return R;
  }
}
```

This algorithm uses exponential space because it has to write down the whole transition system, and to keep track of which formula holds true in each of the (exponentially many) states. We now describe a *space-efficient* implementation of the algorithm, where we will not store the intermediate results, at the expense of recomputing them when we need them. Also, we will not build the explicit transition system, but instead build a witnessing path on-the-fly when needed. This only requires that we are able to compute the successor of a state using polynomial space, which is the case in our situation.

Contrary to the classical CTL model-checking algorithm, the symbolic algorithm will not compute all states satisfying each subformulas (as this would consume exponential space), but instead compute the intermediary results as they are needed: the algorithm traverses the formula from the root to the leaves (*i.e.*, from the outermost modality down to atomic propositions), recursively launching computations for subformulas of the formula being checked. The witness path is guessed non-detemrinistically: we know that the witness can be chosen to have at most exponentially many states (*i.e.*, be acyclic in our exponential-size model), so a counter tells us to stop if no short witness is found. Algorithm 3 displays the algorithm. The algorithm is non-deterministic, but according to Savitch's theorem, it can be made deterministic. It uses only polynomial space, since it does not have to store the path it is building, but only the current state and the selected successor, as well as the length of the portion of the path that has already been built. This counter, which can be written within polynomial space, is used to stop the algorithm after exponentially many steps.

We now prove hardness in PSPACE, by encoding a Turing machine equiped with a polynomial-size tape. We will have one labelled transition system s per cell of the tape of the machine, whose states encode the content of that cell (together with the current state of the machine, in cade the tape head is reading that cell). For each transition $t = (q, a, q', b, d)$ of the Turing machine and each system s_i, there is a transition in the i-th system from (q, a) to b, labelled with $(i + d, t)$. Also, for each t, there is a transition (i, t) from each state c to (q', c). This way, any move in s_i from (q, a) to b also triggers the transition from c to (q', c) in s_{i+d}, which is represents the left- or right neighbour of cell i.

One easily concludes the proof, by defining the initial state appropriately, labelling accepting states with `accept`, and requiring that formula $\mathbf{EF}$ `acccept` holds true in the initial configuration. □

Algorithm 3: CTL symbolic-model-checking algorithm

```
function CTLsmc($\phi$, q, c, l) {
  // decides if q $\models \phi$
  // c=length of current path
  // l=repeated state (for $\mathbf{EG}$)

  switch $\phi$ {
    case p:
      R := ($p \in \ell(q)$)
      return R;
    case $\neg\alpha$:
      R := not(CTLsmc($\alpha$,q,0,$\varnothing$));
      return R;
    case $\alpha \vee \beta$:
      R := CTLsmc($\alpha$,q,0,$\varnothing$);
      if (not R)
        R := CTLmc($\beta$,q,0,$\varnothing$));
      return R;
    case $\mathbf{E}\alpha\,\mathbf{U}\,\beta$:
      pick some successor q' of q;
      if CTLsmc($\beta$,q',0,$\varnothing$) {
        return true;
      } else {
        if (c<|S| &&
            CTLsmc($\alpha$,q',0,$\varnothing$)) {
          return CTLsmc($\phi$,q',c+1,$\varnothing$));
        } else {
          return false;
        }
      }
    case $\mathbf{EG}\,\alpha$:
      pick successor q' of q;
      if (q'==l) {
        return true;
      }
      if (c<|S| &&
          CTLsmc($\alpha$,q',0,$\varnothing$)) {
        if (l==$\varnothing$)
          pick l in {$\varnothing$,q'};
        return CTLsmc($\phi$,q',c+1,l')
      } else {
        return false;
      }
  }
}
```

2.3.2 PLTL and LTL model checking

Lemma 37. *Let $\phi = \mathbf{E}\psi$ be an existential* PLTL *formula, and assume that $\mathcal{A}, \rho, j \models \psi$. Then for any automaton $\mathcal{A}'$ s.t. $\rho \in \mathsf{MaxRuns}(\mathcal{A}')$, it holds $\mathcal{A}', \rho, j \models \psi$.*

Proof. No path quantifier may occur in ψ, so that all the subformulas will be evaluated (possibly at different positions) along the same path. The result can be formally proved by induction. □

As a consequence of this result, we might omit to mention $\mathcal{A}$ and simply write $\rho, j \models \psi$. In the sequel, we also omit to mention the path quantifier, always assuming existential quantification.

Theorem 38. *The* PLTL *and* LTL *model-checking problems are* PSPACE*-complete.*

Proof. The algorithm for LTL relies on a translation into (alternating) Büchi automata: alternation is used to encode disjunctive and conjunctive constructs in the formula (including temporal modalities such as "until", which requires that something is chacked locally and something else is checked in the next state), while the Büchi condition is used to enforce the occurrence of eventualities. Unfortunately, this algorithm cannot easily be adapted to handle PLTL (unless we use two-way automata). The interested reader can have a look at Section 2.3.5, where we use alternating automata (over trees) to solve CTL satisfiability. The ideas for handling LTL with alternating automata (over words) are similar.

Here we present another algorithm, using non-deterministic Büchi automata: as we have no "conjunctive" transitions in such automata, we will encode conjunctions within states (which results in an exponential blowup in the number of states). The main result is stated as follows:

Proposition 39. *Given a* PLTL *formula $\mathbf{E}\phi$, one can build a non-deterministic Büchi automaton $\mathcal{B}_\phi$ of size $2^{O(|\phi|)}$ such that $\mathcal{B}_\phi$ accepts a word w if, and only if, $w, 0 \models \phi$.*

The construction of the automaton requires the following definitions:

Definition 40. *Let $\phi \in \mathsf{PLTL}$. The* set of subformulas *of ϕ, denoted $\mathit{Subf}(\phi)$, is the smallest subset of PLTL containing ϕ and such that*

- *if a formula of the form $\neg\psi$ is in $\mathit{Subf}(\phi)$, then also $\psi \in \mathit{Subf}(\phi)$;*
- *if a formula of the form $\psi_1 \vee \psi_2$, $\psi_1 \mathbf{U} \psi_2$ or $\psi_1 \mathbf{S} \psi_2$ is in $\mathit{Subf}(\phi)$, then also ψ_1 and ψ_2 are in $\mathit{Subf}(\phi)$.*

The extended set of subformulas *of ϕ, written $\mathit{Subf}^*(\phi)$, is obtained from $\mathit{Subf}(\phi)$ by adding*

- *formulas $\mathbf{X}\,\psi_2$, $\mathbf{X}\,\psi_1$ and $\mathbf{X}\,(\psi_1 \mathbf{U} \psi_2)$ for each formula of the form $\psi_1 \mathbf{U} \psi_2$;*
- *formula $\mathbf{X}^{-1}\,\psi_2$, $\mathbf{X}^{-1}\,\psi_1$ and $\mathbf{X}^{-1}\,(\psi_1 \mathbf{S} \psi_2)$ for each formula of the form $\psi_1 \mathbf{S} \psi_2$.*

Finally, the Fischer-Ladner closure *of ϕ is the set*

$$\mathit{FL}(\phi) = \{\psi, \neg\psi \mid \psi \in \mathit{Subf}^*(\phi)\}.$$

The reader will easily prove the following result:

Lemma 41. *A formula $\phi \in \mathsf{PLTL}$ has at most $|\phi|$ subformulas. Its Fischer-Ladner closure has at most $4|\phi|$ formulas.*

Given a formula ϕ, we now build a corresponding Büchi automaton, which we then prove accepts exactly the words corresponding to maximal paths satisfying ϕ.

We begin with building an automaton $\mathcal{B}_\phi = \langle S, \mathsf{Init}, \delta, F\rangle$, reading words over the alphabet $\Sigma = 2^{\mathrm{AP}}$, with a *generalised* Büchi acceptance condition. It is obtained as follows:

- the set of states S is the set of maximal consistent subsets of $\mathsf{FL}(\phi)$. A subset of $\mathsf{FL}(\phi)$ is *consistent* if
  - it does not contain a formula and its negation;
  - it contains $\psi_1 \vee \psi_2$ if, and only if, it also contains ψ_1 or ψ_2;
  - it contains $\psi_1 \wedge \psi_2$ if, and only if, it also contains ψ_1 and ψ_2;
  - it contains $\psi_1 \mathbf{U} \psi_2$ if, and only if, it also contains either $\mathbf{X}\,\psi_2$, or both $\mathbf{X}\,\psi_1$ and $\mathbf{X}\,(\psi_1 \mathbf{U} \psi_2)$;
  - it contains $\psi_1 \mathbf{S} \psi_2$ if, and only if, it also contains either $\mathbf{X}^{-1}\,\psi_2$, or both $\mathbf{X}^{-1}\,\psi_1$ and $\mathbf{X}^{-1}\,(\psi_1 \mathbf{S} \psi_2)$.

  It is *maximal* if it cannot be consistently extended. Notice that a maximal consistent subset of $\mathsf{Subf}(\phi)$ contains exactly one of ψ and $\neg\psi$, for any $\psi \in \mathsf{Subf}^*(\phi)$. As a consequence of this and of Lemma 41, the size of S is at most $2^{4|\phi|}$.

  The idea is that the formulas contained in a state are the formulas that hold true (provided that we follow an accepting run from that state).

- $\mathsf{Init} \subseteq S$ is the set of initial states: it contains those states that contain ϕ and contain no "since" subformulas.

- there is a transition from state s to state s' labelled with $\sigma \subseteq \mathrm{AP}$ iff the following conditions hold:
  - $\sigma = s \cap \mathrm{AP}$; notice that if $p \in \mathrm{AP}$ is not in s, then $\neg p$ is in s, so it must be the case that $p \notin \sigma$;
  - for each $\mathbf{X}\,\psi \in s$, it must be the case that $\psi \in s'$;
  - conversely, for each $\mathbf{X}^{-1}\,\psi \in s'$, it must be the case that $\psi \in s$.

- the acceptance condition is as follows: for each "until" subformula $\psi = \alpha \mathbf{U} \beta$ of ϕ, we define F_ψ as the set of states containing $\neg(\alpha \mathbf{U} \beta)$ or β. A run of automaton $\mathcal{B}_\phi$ is accepting if, and only if, for each "until" subformula ψ, the run visits F_ψ infinitely often.

As we can see, the generalised acceptance condition is a conjunction of Büchi conditions. Notice that this can be translated into a plain Büchi automaton as follows: we first number the "until" subformulas from 1 to k, and define the set of states S' as $S \times \{0, k\}$. If there is a transition from s to s' in $\mathcal{B}_\phi$, then

- if s' contains $\neg\psi_p$, where ψ_p is the p-th "until" subformula, then there is a transition from (s, p) to $(s', p+1 \mod k)$;
- otherwise, there is a transition from (s, p) to (s', p).

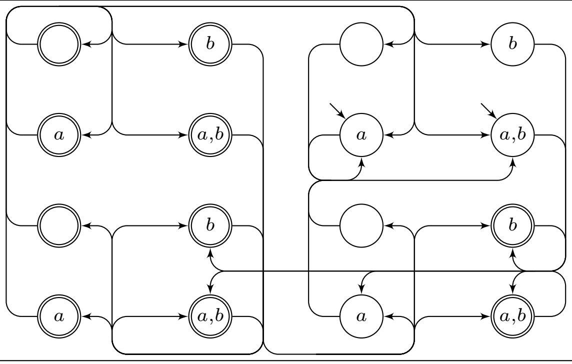
Finally, the Büchi acceptance condition is to visit $S \times \{0\}$ infinitely often. One easily sees that this new automaton accepts the same language as $\mathcal{B}_\phi$ does.

We now prove that $\mathcal{B}_\phi$ accepts the expected language. First, pick a path ρ (which we will equivalently see as a word over 2^{AP} in the sequel) such that $\rho, 0 \models \phi$. Label each letter of this word with the set of formulas of $\mathsf{FL}(\phi)$ that hold true at that position. Then each state is labelled with a maximal consistent subset of $\mathsf{FL}(\phi)$. The resulting sequence of states is easily seen to be a run of the Büchi automaton: in particular, it begins in an initial state, and any state labelled with $\psi_1 \mathbf{U} \psi_2$ is also labelled with $\mathbf{X}\psi_2 \vee (\mathbf{X}\psi_1 \wedge \mathbf{X}(\psi_1 \mathbf{U} \psi_2))$ (and similarly for "since" subformulas). Moreover, any "until" formula that holds true at some position is eventually fulfilled, so that the generalized Büchi condition is satisfied and the run is accepting.

Conversely, pick an accepting run $\pi = (s_i)_i$ of $\mathcal{B}_\phi$, and let $w = (w_i)_i$ be the word obtained by letting $w_i = s_i \cap \mathrm{AP}$ for all i. One easily proves, by induction on the formula, that for any i and any $\psi \in s_i$, it holds $w, i \models \psi$. Seeing w as a path ρ in some automaton $\mathcal{A}$, it follows that $w, 0 \models \phi$.

Example. *Figure 9 displays the Büchi automaton corresponding to formula $a \mathbf{U} (b \wedge \mathbf{X}^{-1} b)$. the closure of this formula contains* 12 *subformulas, thus the automaton could contain up to* 2^6, *i.e.,* 64 *states. Still, consistency requirements lower this value to* 16. *For the sake of clarity, labels are not shown on transitions, since they correspond, by construction, to the set of atomic propositions labeling the source state.*

Figure 9: The Büchi automaton corresponding to formula $a \mathbf{U} (b \wedge \mathbf{X}^{-1} b)$.



This provides us with an algorithm for PLTL model checking: given an automaton $\mathcal{A}$ and a formula $\mathbf{E}\phi \in \mathsf{PLTL}$, we built the automaton $\mathcal{B}_\phi$, and consider the *product automaton* $\mathcal{A} \times \mathcal{B}_\phi$. This product automaton is built as follows: its states are pairs (q, s) where q is a state of $\mathcal{A}$ and s is a state of $\mathcal{B}_\phi$. Valid states are those states for which $\ell(q) = s \cap \mathrm{AP}$. Then for any path ρ in $\mathcal{A}$, it holds $\mathcal{A}, \rho, 0 \models \mathbf{E}\phi$ if, and only if, the product automaton has an accepting run.

Since the automaton $\mathcal{B}_\phi$ has exponential size, the product $\mathcal{A} \times \mathcal{B}_\phi$ also has exponentially many states. However, deciding whether this automaton has an accepting run does not require that we build the automaton explicitly: it suffices to non-deterministically guess the path step-by-step, which can be achieved using only polynomial space. This is a generic exercice about Büchi automata, which we leave as an exercice for the reader. It must be noticed that, in fact, when the formula is fixed (and thus is not taken into account when measuring the complexity of the problem), the non-deterministic algorithm only requires logarithmic space.

It remains to prove hardness in PSPACE. Before we prove this result, we begin with explaining how LTL can be used to count from 0 to $2^n - 1$, using n boolean variables. More preceisely, we write an LTL formula over $\text{AP} = \{b_i \mid 0 \leq i \leq n-1\}$ that holds true only along the path $(c_0 c_1 ... c_{2^n-1})^\omega$, where $c_i \in 2^{\text{AP}}$ is the binary representation of integer i with bits b_0 to b_{n-1} (for instance, $c_{23} = \{b_4, b_2, b_1, b_0\}$).

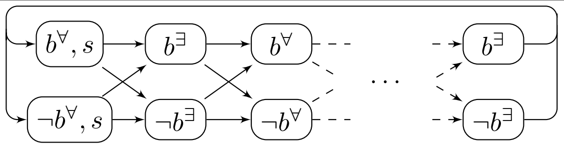
We now write the LTL formula enforcing such a pattern, as a conjunction of several subformulas:

- we begin with $c_0 = 0$:
$$\phi_{\text{zero}} = \bigwedge_{0 \leq i \leq n-1} \neg b_i$$
- bit b_0 alternates between 0 and 1 between any two consecutive states:
$$\mathbf{G}\,(b_0 \iff \mathbf{X}\,\neg b_0)$$
- all other bits are modified from one state to the next one if, and only if, all less significant bits have value 1:
$$\bigwedge_{1 \leq i \leq n-1} \mathbf{G}\Big[\big(\bigwedge_{0 \leq j \leq i-1} b_j\big) \iff (b_i \iff \mathbf{X}\,\neg b_i)\Big]$$

The reader will check that the conjunction of these three formulas achieves the expected result. Notice that we could do the same with one single atomic proposition b, by replacing b_j with the value of b at position j (which is tested using $\mathbf{X}^j b$).

Using similar ideas, we can now encode any instance of QBF into an instance of the LTL model-checking problem: let $(\forall x_{2i}.\ \exists x_{2i+1})_{0 \leq i \leq n-1} \phi(x_0, ..., x_{2n-1})$ be an instance of QBF. We let $\text{AP} = \{b, s\}$ be the set of atomic propositions, with b being used at position k (modulo $2n$) to represent the value of variable x_k, and s being used as a separator between consecutive valuations of $(x_k)_{0 \leq k \leq 2n-1}$. We consider the automaton depicted at Figure 10: between any two visits to an s-state, one valuation of the variables $(x_k)_{0 \leq k \leq 2n-1}$ is generated.

Figure 10: Encoding QBF as an LTL model-checking problem



Using LTL, we impose that the sequence of valuations witnesses the truth value of the QBF instance (*i.e.*, for universal quantifications, both values of the corresponding variable are listed), and that formula ϕ is true under all those valuations.

- we have to impose that each universally-quantified variable appears both positively and negatively. Notice that it sufffices to have $x_0 \cdot x_2 \cdot x_4 \cdots x_{2n-2}$, seen as the binary encoding of some integer (with x_{2n-2} as the least significant bit), take all values from 0 to 2^{n-1}, which we can do as above.
- we have to make sure that as long as no universally-quantified variables have switched from one valuation to the next one, then the preceding variables should keep the same value:
$$\mathbf{G}\Big[s \Rightarrow \big[(b \iff \mathbf{X}^{2n} b)\,\mathbf{U}\,(b^\forall \iff \mathbf{X}^{2n} \neg b^\forall)\big]\Big]$$

- finally, we check that ϕ holds: this is easily achieved by noticing that the value of variable x_k is encodes by the value of b in the k-th cell after s:

$$\mathbf{G}\Big[s \Rightarrow \phi(x_k \leftarrow \mathbf{X}^k b)\Big].$$

Write ϕ_{QBF} for the conjunction of the formulas above (including those enforcing that $x_0 \cdot x_2 \cdot x_4 \cdots x_{2n-2}$ encodes a binary counter from 0 to $2^n - 1$). Then the QBF instance is positive if, and only if, formula $\mathbf{E}\phi$ holds true in the lower left state of the automaton of Figure 10. □

Theorem 42. *The* PLTL *and* LTL *symbolic-model-checking problems are* PSPACE*-complete.*

Proof. The above algorithm can be run when the structure is given symbolically, provided that the successor relation in the resulting exponential-space transition system can be computed in polynomial space: it suffices to build the Büchi automaton and to run it in parallel with the global transition system. □

2.3.3 CTL* model checking

Theorem 43. *The* CTL* *model-checking problem is* PSPACE*-complete.*

Proof. Hardness in PSPACE is a direct corollary of the hardness result for LTL. The CTL* model-checking algorithm turns out to also be an easy adaptation of the algorithm for LTL: following Lemma 34, we can make use of a labelling algorithm, by iteratively computing the sets of states satisfying each state-subformulas of the CTL* formula to be checked. This relies on substitutivity: if two CTL* state formulas ϕ and ψ are equivalent for a fixed class of models, and if $\zeta[p]$ is a CTL* formula involving a special atomic proposition p, then $\zeta[\phi]$ and $\zeta[\psi]$ are equivalent for the same class of models (where $\zeta[\phi]$ is obtained from $\zeta[p]$ by replacing each occurrence of p with ϕ).

Now, we proceed by induction: assume ϕ is the formula to be checked, and that it has k path quantifiers. If $k = 0$, then the formula is propositional, and the algorithm is straightforward. When $k = 1$, ϕ is a boolean combination of propositional formulas and of one LTL formula ψ. For each state, deciding whether ψ holds in that state can be achieved using polynomial space. Storing this information itself only requires polynomial space (and thus is affordable). Labelling those states with a fresh atomic proposition p and replacing ψ with p in ϕ, we get a formula which is equivalent on the class made of our single model. We end up with a CTL* formula having one less path quantifier, which we know how to handle (by induction). When $k > 1$, the algorithm follows the same ideas, after selecting an LTL subformula of ϕ (*i.e.*, a formula starting with one of the *deepest* path quantifiers). The global algorithm then uses polynomial space, as required. □

Applying similar ideas in a space-efficient manner, we can prove:

Theorem 44. *The* CTL* *symbolic-model-checking problem is* PSPACE*-complete.*

We omit the tedious details of the proof of this result.

2.3.4 PLTL and LTL satisfiability

Theorem 45. *The* PLTL *and* LTL *satisfiability problems are* PSPACE*-complete.*

Proof. The proof uses the same ideas as for model checking. First, given a PLTL formula, we can build the Büchi automaton $\mathcal{B}_\phi$ for ϕ. Any word accepted by this automaton witnesses the existence of a path satisfying ϕ, so that we obtain a polynomial-space algorithm.

PSPACE-hardness is proved by slightly adapting the hardness proof for the model-checking problem: more precisely, we encode the behaviours of the automaton of Figure 10 as an LTL formula: these behaviours can be characterized by

- the presence of s initially and exactly every $2n$ positions, which is written as

$$\big[s \wedge \bigwedge_{1\leq i\leq 2n-1} \mathbf{X}^{i}\neg s\big] \;\wedge\; \mathbf{G}\,\big[s \iff \mathbf{X}^{2n}s\big]$$

- the alternation of $\exists$ and $\forall$:

$$\cdot^{\forall} \wedge \mathbf{G}\,(\cdot^{\forall} \iff \mathbf{X}\,\cdot^{\exists}).$$

□

2.3.5 CTL satisfiability

CTL satisfiability is the most technical of the proofs in this section. It proceeds in three parts: first we prove that satisfiability can be witnessed by an infinite tree having finite branching. We then prove that this problem (satisfiability by an infinite tree having finite branching) is decidable, and finally that when this is the case, then the formula is also satisfiable in a finite Kripke structure.

Theorem 46. *The* CTL *satisfiability problem is* EXPTIME*-complete.*

Proof. Tree automata accept trees in pretty much the same way as word automata accept words: when reading a tree, the automaton forks computations in each successor nodes of the current node, with the aim of satisfying the acceptance condition along each branch. For dealing with CTL satisfiability, the tree that we consider is the unfolding of the Kripke structure that should satisfy the formula. We restrict to trees (seen as infinite-state Kripke structures) thanks to the following lemma:

Lemma 47. *Let $\phi \in$ CTL*, $\mathcal{A}$ be a (possibly infinite-state) Kripke structure, and q be a state of $\mathcal{A}$. Then $\mathcal{A}, q \models \phi$ if, and only if, the root of the execution tree of $\mathcal{A}$ from q also satisfies ϕ.*

Proof. This is easily proved by induction on the structure of ϕ. □

The algorithm for CTL satisfiability checking will use similar ideas as for PLTL: we build an automaton and check the emptiness of its language. One main difference between both approaches is that CTL is, in some sense, bi-dimensional: we also have to take care about the width of the trees we consider. We directly prove it for CTL*:

Proposition 48 ([ES84a])**.** *Let $\phi \in$ CTL*. Then ϕ is satisfiable if, and only if, it is satisfiable in an infinite tree $\mathcal{T}$ with branching $O(|\phi|)$.*

Proof. The proof is in two steps: we begin with transforming ϕ into an *equisatisfiable* formula ψ which has a simpler form, only involving subformulas of the form $\mathbf{E}\zeta$, $\mathbf{A}\zeta$ and $\mathbf{AG}\ \mathbf{E}\zeta$ where ζ is a path formula involving no path quantifier (which we can see as an LTL formula). The second step proves the result for those simplified formulas.

The basic idea of the first step is to replace each state subformula $\mathbf{E}\xi$ with an extra atomic proposition x, which is made "equivalent" to the state subformula by imposing the extra conjunct $\mathbf{AG}\,(x \Leftrightarrow \mathbf{E}\xi)$. Applying this inductively, we end up with the original formula being in the expected form, and a conjunction of formulas of the above form, with ξ being a path formula without path quantifier. Clearly enough, the original formula is satisfiable if, and only if, the resulting formula is.

Finally, $\mathbf{AG}\,(x \Leftrightarrow \mathbf{E}\xi)$ is equivalent to $\mathbf{AG}\,(x \Rightarrow \mathbf{E}\xi) \wedge \mathbf{AG}\,(\mathbf{E}\xi \Rightarrow x)$, which in turn is equivalent to $\mathbf{AG}\ \mathbf{E}(x \Rightarrow \xi) \wedge \mathbf{AG}\ \mathbf{E}(\xi \Rightarrow x)$. This concludes the first step.

Now, let $\mathcal{A}$ and q_0 such that $\mathcal{A}, q_0 \models \psi$. Assume that ψ contains n state subformulas of the form $\mathbf{E}\xi$. We inductively build a tree $\mathcal{T}$ with branching degree at most $n+1$ and such that ψ holds true at its root. The tree $\mathcal{T} = \langle T, m\rangle$ has $T = \{0, ..., n\}^*$, and its labelling is defined as follows:

- $m(\varepsilon) = \ell(q_0)$: the root corresponds to q_0, so it has the same labelling. We also label ε with q_0, to keep track of the state it corresponds to;

- pick a node z corresponding to some state q; for each subformula of the form $\mathbf{E}\xi_i$ that holds true at q in $\mathcal{A}$, consider a maximal path $\pi = (q_j)_j$ such that $\mathcal{A}, \pi, 0 \models \xi$, and label $z \cdot i$ with $\ell(q_1)$ and q_1, and $z \cdot i \cdot 0^k$ with $\ell(q_{k+1})$ and q_{k+1} (as long as $k < \mathsf{length}\pi$). For subformulas $\mathbf{E}\xi_i$ that do not hold true at q in $\mathcal{A}$, we (randomly) pick a path from q, and plug it in the tree in the same way along the branch $z \cdot i \cdot 0^*$.

This way, any branch of $\mathcal{T}$ corresponds to a run of $calA$ from q. Hence any formula of the form $\mathbf{A}\xi$ that holds true at q_0 in $\mathcal{A}$ also holds true at the root of $\mathcal{T}$. Also, for each node z of $\mathcal{T}$ corresponding to some state q, and each subformula $\mathbf{E}\xi$, if $calA, q \models \mathbf{E}\xi$ then also $\mathcal{T}, z \models \mathbf{E}\xi$. It follows that any formula of the form $\mathbf{E}\xi$ and $\mathbf{AG}\ \mathbf{E}\xi$ that holds true at q_0 in $\mathcal{A}$ also holds true at the root of $\mathcal{T}$, which concludes the proof. $\square$

One we have fixed a maximal branching degree of the trees to be considered, we are ready to build the alternating tree automaton corresponding to a given CTL formula.

Proposition 49. *Given a* CTL *formula* ϕ*, we can build an alternating tree automaton* $\mathcal{A}_\phi$ *of size linear in* $|\phi|$ *such that* $\mathcal{A}_\phi$ *accepts a tree* $\mathcal{T}$ *if, and only if, the root of* $\mathcal{T}$ *satisfies* ϕ.

Proof. When running on its input tree, an (alternating) tree automaton launches one (or several) computations in each subtree of the node being read. This will allow us to check CTL properties in trees. Usually, the transitions of the tree automaton might depend on the branching degree of the node being read. We will not use this in our construction, as CTL only imposes conditions one all the successors, or on one of the successors.

Before we can build the automaton, we need the following two definitions:

Definition 50. *Let* $\phi \in \mathsf{CTL}$. *The* set of subformulas *of* ϕ*, denoted with* $\mathsf{Subf}(\phi)$*, is the smallest subset of* CTL *containing* ϕ *and such that*

- *if a formula of the form* $\neg\psi$ *is in* $\mathsf{Subf}(\phi)$*, then also* $\psi \in \mathsf{Subf}(\phi)$*;*
- *if a formula of the form* $\psi_1 \vee \psi_2$, $\mathbf{E}\psi_1 \mathbf{U} \psi_2$ *or* $\mathbf{A}\psi_1 \mathbf{U} \psi_2$ *is in* $\mathsf{Subf}(\phi)$*, then also* ψ_1 *and* ψ_2 *are in* $\mathsf{Subf}(\phi)$.

Definition 51. *The set of* positive boolean formulas *built on AP is defined by the following grammar:*

$$\mathsf{PBF}(AP) \ni \phi ::= \top \mid \bot \mid p \mid \phi \vee \phi \mid \phi \wedge \phi$$

We now turn to the construction of the automaton. Let ϕ be a CTL formula. We assume w.l.o.g. that negations only appear in front of atomic propositions of temporal modalities. We then build the alternating tree automaton $\mathcal{A}_\phi = \langle Q, q_0, \Sigma, \delta, F\rangle$ as follows:

- $Q = \mathsf{Subf}(\phi)$ is the set of subformulas of ϕ (including true and false);
- q_0 is the state ϕ;
- $\Sigma = 2^{\mathrm{AP}}$;
- $\delta\colon Q \times \Sigma \to \mathsf{PBF}(\{\Box, \Diamond\} \times Q)$ is defined below;
- $F \subseteq Q$ is a Büchi acceptance condition, requiring that some state *not* of the form $\mathbf{EU}$ or $\mathbf{AU}$ appears infinitely often along each branch.

Now, the transition relation is defined in a straightforward way:

$$\delta(\mathsf{true}, \sigma) = \top \qquad \delta(\phi_1 \vee \psi_2, \sigma) = \delta(\phi_1, \sigma) \vee \delta(\phi_2, \sigma)$$

$$\delta(\mathsf{false}, \sigma) = \bot \qquad \delta(\phi_1 \wedge \psi_2, \sigma) = \delta(\phi_1, \sigma) \wedge \delta(\phi_2, \sigma)$$

$$\delta(p, \sigma) = \begin{cases} \top & \text{if } p \in \sigma \\ \bot & \text{if } p \notin \sigma \end{cases} \qquad \delta(\neg p, \sigma) = \begin{cases} \top & \text{if } p \notin \sigma \\ \bot & \text{if } p \in \sigma \end{cases}$$

$$\begin{aligned}
\delta(\mathbf{E}\phi_1 \mathbf{U} \phi_2, \sigma) &= \Diamond \phi_2 \vee (\Diamond \phi_1 \wedge \Diamond (\mathbf{E}\phi_1 \mathbf{U} \phi_2)) \\
\delta(\mathbf{A}\phi_1 \mathbf{U} \phi_2, \sigma) &= \Box \phi_2 \vee (\Box \phi_1 \wedge \Box (\mathbf{E}\phi_1 \mathbf{U} \phi_2)) \\
\delta(\neg(\mathbf{A}\phi_1 \mathbf{U} \phi_2), \sigma) &= \Diamond \phi_2 \wedge (\Diamond \phi_1 \vee \Diamond (\neg(\mathbf{E}\phi_1 \mathbf{U} \phi_2))) \\
\delta(\neg(\mathbf{E}\phi_1 \mathbf{U} \phi_2), \sigma) &= \Box \phi_2 \wedge (\Box \phi_1 \vee \Box (\neg(\mathbf{E}\phi_1 \mathbf{U} \phi_2)))
\end{aligned}$$

It remains to prove, inductively on ψ, that a tree is accepted by $\mathcal{A}_\phi$ with initial state ψ if, and only if, ψ holds true at its root. This is quite straightforward, except possibly for temporal modalities. We handle the case of **EU**, and leave the other cases to the reader.

Assume $\psi = \mathbf{E}\psi_1 \mathbf{U} \psi_2$ holds true at the root of some tree $\mathcal{T} = \langle T, l\rangle$. Then there is a node n in the tree where ψ_2 holds, and ψ_1 holds at all intermediary nodes. The root of the subtree rooted at n satisfies ψ_2, and so is accepted by $\mathcal{A}_\phi$ when starting from ψ_2. Similarly, all intermediary nodes satisfy ψ_1, and the corresponding subtrees are thus satisfied by $\mathcal{A}_\phi$ if starting from ψ_1. Plugging the corresponding accepting runs together, we get an accepting run of $\mathcal{A}_\phi$ from ψ on $\mathcal{T}$.

Conversely, if there exists an accepting run, then any node visited by state ζ along that accepting run satisfies ζ. We leave this part of the proof to the reader. □

Emptiness of alternating tree automata can then be checked in deterministic exponential time, *e.g.* by seeing this problem as a game. The formal proof of this statement is far beyond the scope of this course, and we better refer to [EJ99, Kir02] for more details.

This provides a way to decide satisfiability of a CTL formula in a possibly infinite model. However, it can be shown (see *e.g.* [Kir02]) that when an alternating tree automaton accepts some tree, then it accepts a *regular* one, *i.e.*, one obtained as the unfolding of a finite-state transition system. This proves that CTL satisfiability can be decided in EXPTIME.

EXPTIME-hardness can be proven by encoding the emptiness problem for linear-bounded alternating Turing machines. We leave this as an exercice for the courageous reader.

2.4 ATL: a temporal logic for multi-agent systems

In this section, we define and study the *alternating-time temporal logic* ATL, which extends CTL with quantification on *strategies* of the agents, rather than just on paths.

Definition 52. *The* full alternating-time temporal logic *is denoted by* ATL*. *Formulas of* ATL* *over atomic propositions AP and agents $\mathbb{A}$ are formulas built on the following grammar:*

$$\begin{aligned} \mathsf{ATL}^* \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \langle\!\langle A\rangle\!\rangle\, \phi_p \mid [\![A]\!]\, \phi_p \\ \phi_p &::= \phi_s \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \,\mathbf{U}\, \phi_p \mid \phi_p \,\mathbf{S}\, \phi_p. \end{aligned}$$

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

ATL* *can be seen as an extension of* CTL* *(as will be explained below). The semantics of* ATL* *operators is the same as for* CTL**, with the additional two rules:*

$$\begin{aligned} \mathcal{A}, \rho, j \models \langle\!\langle A\rangle\!\rangle\, \phi_p \quad &\Leftrightarrow \quad \exists \sigma_A \in \mathit{Strat}_{\mathcal{A}}(A).\ \forall \rho' \in \mathit{Out}(s_j, \sigma_A).\ \mathcal{A}, \rho', 0 \models \phi_p \\ \mathcal{A}, \rho, j \models [\![A]\!]\, \phi_p \quad &\Leftrightarrow \quad \forall \sigma_A \in \mathit{Strat}_{\mathcal{A}}(A).\ \exists \rho' \in \mathit{Out}(s_j, \sigma_A).\ \mathcal{A}, \rho', 0 \models \phi_p \end{aligned}$$

While the existential quantification over strategy is quite intuitive, the dual quantifier is a bit less: formula $[\![A]\!]\,\phi$ means that for any strategy of A, at least one outcome satisfies ϕ.

Notice also that the existential quantification over all players $\langle\!\langle \mathbb{A}\rangle\!\rangle$ corresponds to existential path quantification of CTL*, while the empty quantification $\langle\!\langle \varnothing\rangle\!\rangle$ corresponds to the universal one. Hence ATL* is at least as expressive as CTL*.

As a natural fragment of ATL*, ATL is the counterpart of CTL, where only simple path formulas are allowed:

$$\begin{aligned} \mathsf{ATL} \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \langle\!\langle A\rangle\!\rangle\, \phi_p \mid [\![A]\!]\, \phi_p \\ \phi_p &::= \phi_s \,\mathbf{U}\, \phi_s \end{aligned}$$

2.4.1 Expressiveness issues

Several expressiveness results can be derived from the inclusion of CTL in ATL: for instance, if $\langle\!\langle A\rangle\!\rangle\, \mathbf{G}\, \mathbf{F}\, p$ could be expressed as a formula $\phi_{\{A\}}$ in ATL, then in the one-player setting, $\phi_{\mathbb{A}}$ would be equivalent to $\mathbf{EG}\, \mathbf{F}\, p$, and would only involve $\langle\!\langle \varnothing\rangle\!\rangle$ and $\langle\!\langle \mathbb{A}\rangle\!\rangle$, which correspond to $\mathbf{A}$ and $\mathbf{E}$.

Using such ideas, negative expressiveness results in CTL can be lifted to ATL. The converse is not necessarily true. First notice that we have the following duality:

$$\langle\!\langle A\rangle\!\rangle\, \mathbf{G}\, \phi \equiv \neg\, [\![A]\!]\, \mathbf{F}\, \phi.$$

Quite naturally, one would define

$$\widetilde{\mathsf{ATL}} \ni \phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \langle\!\langle A\rangle\!\rangle\, \phi\, \mathbf{U}\, \phi \mid \langle\!\langle A\rangle\!\rangle\, \mathbf{G}\, \phi$$

and claim that $\widetilde{\mathsf{ATL}}$ and ATL are equally expressive, as is the case for CTL. This is what was silently assumed in many papers about ATL, where the logic $\widetilde{\mathsf{ATL}}$ is actually considered (in particular in [AHK97, AHK02]).

However, consider the following equivalence:

$$\mathbf{E}\neg(a\, \mathbf{U}\, b) \equiv \mathbf{E}\big[\mathbf{G}\,(\neg b) \vee (\neg b)\, \mathbf{U}\,(\neg a \wedge \neg b)\big] \equiv \mathbf{EG}\,(\neg b) \vee \mathbf{E}(\neg b)\, \mathbf{U}\,(\neg a \wedge \neg b).$$

When lifted to ATL, we get

$$\langle\!\langle A\rangle\!\rangle\, \neg(a\, \mathbf{U}\, b) \equiv \langle\!\langle A\rangle\!\rangle\, \big[\mathbf{G}\,(\neg b) \vee (\neg b)\, \mathbf{U}\,(\neg a \wedge \neg b)\big] \overset{?}{\equiv} \langle\!\langle A\rangle\!\rangle\, \mathbf{G}\,(\neg b) \vee \langle\!\langle A\rangle\!\rangle\, (\neg b)\, \mathbf{U}\,(\neg a \wedge \neg b).$$

The second equivalence turns out to be wrong: only one implications holds. Indeed, if A has a strategy for enforcing $\mathbf{G}\,(\neg b)$, then this very strategy enforces $\neg(a\, \mathbf{U}\, b)$. Similarly if A has a strategy to enforce $(\neg b)\, \mathbf{U}\,(\neg a \wedge \neg b)$. But the converse does not holds: if A has a strategy to enforce $\neg(a\, \mathbf{U}\, b)$, then that strategy enforces $\mathbf{G}\,(\neg b) \vee (\neg b)\, \mathbf{U}\,(\neg a \wedge \neg b)$. But it might be the case that it does not enforce only one of the disjunct along all its outcomes. This can be formalised:

Theorem 53. ATL *is strictly more expressive than* $\widetilde{\mathsf{ATL}}$.

The details of the proof of this result are a bit tedious, and we better refer to [LMO07] for the full proof.

To conclude on this note, let us focus on the subclass of *turn-based games*, where in each state, all the moves of all but one player are equivalent. Such games have the important property of being *determined* (for reasonnable objectives) [Mar75]: more precisely, if a coalition A has no strategy to reach a given objective, then the complement coalition $\mathbb{A} \setminus A$ has a strategy to enforce the opposite objective. In other terms, turn-based games have the following equivalence

$$\langle\!\langle A\rangle\!\rangle\, \neg(a\, \mathbf{U}\, b) \equiv \langle\!\langle \mathbb{A} \setminus A\rangle\!\rangle\, a\, \mathbf{U}\, b$$

so that ATL and $\widetilde{\mathsf{ATL}}$ are equivalent on this subclass.

2.4.2 Model checking and satisfiability

As we saw in Section 1.3, reachability in games can be handled in a way very similar to how it is handled in transition systems. We show here how this can be generalised, entailing the following result:

Theorem 54. ATL *model checking is* PTIME*-complete.*

Proof. As for CTL, we begin with a crucial lemma, which will allow us to use a *labelling algorithm* similar to the CTL model-checking algorithm:

Lemma 55. *Given two runs ρ and ρ' of an automaton $\mathcal{A}$, and two positions j and j' s.t. $s_j = s'_{j'}$, it holds*

$$\forall \phi_s \in \mathsf{ATL}^*. \quad \mathcal{A}, \rho, j \models \phi_s \Leftrightarrow \mathcal{A}, \rho', j' \models \phi_s.$$

This lemma is easily proved by induction on the formula. A consequence of this result is that we can compute, for each ATL subformula of the formula to be checked, the exact set of states where this formula holds. Following the ideas developped for solving reachability games, we rely on the operator CPre, and on the usual fixpoint equations. The following result is the counterpart of Prop. 35:

Proposition 56. *Let α, β be two subsets of the set of states of a finite-state concurrent game $\mathcal{G}$, and A be a player in $\mathcal{G}$. We define the following two sequences:*

$$\begin{aligned} E_0 &= \varnothing & E_{i+1} &= \mathit{CPre}_A(\beta \cup (\alpha \cap E_i)) \\ A_0 &= S & A_{i+1} &= S \setminus \mathit{CPre}_A(S \setminus (\beta \cup (\alpha \cap A_i))) \end{aligned}$$

Then both sequences converge in finite time, and their limits (which we write E_∞ and A_∞) enjoy the following properties (where we assume that the states in α and β are labelled with corresponding atomic propositions α and β):

$$\begin{aligned} \mathcal{A}, \rho, j \models \langle\!\langle A \rangle\!\rangle\, \alpha \,\mathbf{U}\, \beta &\iff s_j \in E_\infty \\ \mathcal{A}, \rho, j \models [\![A]\!]\, \alpha \,\mathbf{U}\, \beta &\iff s_j \in A_\infty \end{aligned}$$

Proof. That both sequences converge can be proved in the same way as for CTL.

Pick a run ρ and a position j such that $\mathcal{A}, \rho, j \models \langle\!\langle A \rangle\!\rangle\, \alpha \,\mathbf{U}\, \beta$. Following Lemma 55, we equivalently have $\mathcal{A}, \rho_{\geq j}, 0 \models \langle\!\langle A \rangle\!\rangle\, \alpha \,\mathbf{U}\, \beta$, which means that A has a strategy whose outcomes from s_j all satisfy $\alpha \,\mathbf{U}\, \beta$. Write k for the maximal number of states visited before witnessing β. This exists by König's lemma.

We prove by induction on k that s_j is in E_∞. If $k = 1$, then β is reached in one move from s_j, whatever the other players do. In other terms, $s_j \in \mathsf{CPre}_A(\beta)$, where we abusively write β for the set of states satisfying β. Hence $s_j \in E_1 \subseteq E_\infty$. We now prove the result when $k = k' + 1$, assuming it holds for k'. Consider the sets of states reached by one move of the strategy played by A. In each of these states, either β holds, or α holds and A has a strategy enforcing $\alpha \,\mathbf{U}\, \beta$ within at most k' states. Hence s_j is in $\mathsf{CPre}_A(\beta \cup (\alpha \cap E_\infty))$, wich precisely is E_∞.

Conversely, we prove by induction that from a state in E_k, there is a strategy for A to enforce $\alpha \,\mathbf{U}\, \beta$ and reach β within k steps. This is clear for E_1, and is easily extended to any E_k.

We leave the other proof to the reader as an exercice, where the aim is to show that from any state in E_k, A has a strategy for which at least one outcome will satisfy $\alpha \,\mathbf{U}\, \beta$ and reach β within k steps. ☐

Remark. *It can be remarked that this algorithm can be made* constructive, *i.e., it can effectively generate the witnessing strategies. Indeed, when a state is proven to belong to $\mathit{CPre}_A(X)$, it can be associated with a corresponding action for A which enforces that X will be reahced at the next step. It also follows that* memoryless strategies *are sufficient for objectives in* ATL *(reacability, safety, ...)*

Remark. *The algorithm runs in polynomial time if we consider that the transition table* Edg *is given explicitly. In that case, this table has size $O(|S| \times |\mathbb{M}|^{|\mathbb{A}|})$. Such a representation is not usable in practice.*

A symbolic setting *can also be defined for concurrent games: not only the state space would be given as the product of several components, but also the transition table has to be given symbolically, in order to avoid the above blow-up. We leave it to the reader to define this setting and prove that* ATL *symbolic model-checking is* EXPTIME*-complete.*

Theorem 57. ATL* *model checking is* 2-EXPTIME*-complete.*

Proof. The algorithm again consists in labelling states with the subformulas they satisfy, so that we are left with the problem of deciding whether some player A has a strategy to satisfy some LTL condition. We prove the more general result that given a winning condition as a non-deterministic Büchi automaton, deciding whether player A has a winning strategy is decidable in exponential time.

The algorithm consists in first building a deterministic automaton equivalent to the Büchi automaton given as input, and then play on the "product" of this automaton with the game structure. The deterministic automaton will have a Rabin winning condition, which is then also the winning condition in the product. While the deterministic automaton has exponentially many states, is only has polynomially many Rabin pairs. Solving games with n states and k Rabin pairs can be achieved in time $O((nk)^{3k})$ [EJ99], which matches the announced complexity.

Hardness in 2-EXPTIME can be proved by a direct encoding of the LTL realisability problem [PR89]. □

2.5 Exercises

Exercice 5 ⋆⋆ 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 finitely many times;
- 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) (with no constraints on odd positions).

Exercice 6 ⋆ Prove that model-checking the fragment of CTL with only **EF** is PTIME-complete.

Exercice 7 ⋆⋆⋆ Define a symbolic representation for concurrent games, and prove that ATL model checking is EXPTIME-complete in that framework.

3. Weighted temporal logics

In this section we extend temporal logics with quantitative constraints. Syntactically, such extended temporal logics could be used in any quantitative setting, be it discrete or continuous. Here we focus on a discrete semantics, in two different settings: weighted labelled transition systems on the one hand, and discrete-time timed automata on the other hand.

3.1 Quantitative temporal logics on weighted labelled transition systems

3.1.1 Weighted labelled transition systems

Weighted labelled transition systems simply are labelled transition systems whose transitions are decorated with sets of integers, which can be seen as the *price to pay* to take the transition.

Definition 58. *Write $\mathcal{I}(\mathbb{N})$ for the set of non-empty intervals of $\mathbb{N}$. A* weighted labelled transition system *is a 4-tuple $\mathcal{A} = \langle S, T, \ell, w\rangle$ where $\langle S, T, \ell\rangle$ is a labelled transition system and $w\colon T \to \mathcal{I}(\mathbb{N})$ associate each transition with an interval.*

Semantically, it is more convenient to consider weighted labelled transition systems in which each transition carries only one integer weight (instead of an interval). With a weighted labelled transition system $\mathcal{A} = \langle S, T, \ell, w\rangle$, we associate a *tightly weighted* labelled transition system $\mathcal{A}' = \langle S, T', \ell, w'\rangle$ where $T' = \{(t, n) \mid t \in T \text{ and } n \in w(t)\}$, with $\mathsf{src}((t, n)) = \mathsf{src}(t)$ and $\mathsf{tgt}((t, n)) = \mathsf{tgt}(t)$ and $w'((t, n)) = n$. This transition system might have infinitely many transitions, but each transition carries only one integer weight.

Runs of a weighted labelled transition system are runs of the underlying tightly weighted labelled transition system. The weight of a (finite or infinite) run is defined through the homomorphism mapping each transition to its weight. The weight of a run is then an integer. We (abusively) write $w(\rho)$ for the weight of a run ρ.

Notice that weights are *nonnegative*, so that the accumulated weight along a run is nondecreasing. Allowing for negative weights makes the setting richer and more complex, and the natural questions do not only concern the final accumulated weight, but also the intermediary values (it can be required that they remain nonnegative, for instance [BFL+08]).

On another note, we will not use the term *weighted automata* as an equivalent for weighted labelled transition systems: *weighted automata* are kind of a registered trademark for weighted transition systems whose weights are taken from a semi-group, for which general results are expected, independently of the underlying semi-group). Our weighted labelled transition systems are only a special kind of weighted automata, and not all our results would extend to any weighted automata.

The most basic problem on duration automata concerns *optimal reachability*: given two states s and s', what is the weight of the shortest path (here "shortest" means "with least weight") going from s to s'? The associated decision problem is as follows:

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 59. *The shortest-path problem is solvable in* PTIME.

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 weights", as it detects cycles). This algorithm iteratively computes the weight of the shortest run of length i from each state of $\mathcal{A}$ to s, starting with $i = 0$. When no such run exists, the weight is $+\infty$.

- initially, only s can reach itself with a path of length zero. Thus $d_0(s) = 0$ and $d_0(q) = +\infty$ for any $q \neq s$.
- for each i ranging from 0 to n with $n = |S|$, apply the following procedure: first set $d_{i+1}(q)$ to $d_i(q)$ for each state q. Then, for each edge $(q, w, q') \in \delta$, if $d_{i+1}(q) > d_i(q') + w$, then $d_{i+1}(q)$ is set to $d_i(q') + w$.
- finally, if $d_n(q) > d_{n+1}(q)$, then $d(q)$ is set to $-\infty$. Then, for each state q' that can reach 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 $O(|S| \cdot |\delta|)$).

Algorithm 4: Bellman-Ford single-destination shortest-path algorithm

```
function shortest(G, q) {
  // compute shortest distance to q
  // from any state in G
  for all s∈S {
    if (s=q)
      v(s):=0
    else
      v(s):=+∞;
  }

  repeat $|G|$ times {
    v':=v;
    for all t∈T {
      if (v(src(t))>v'(tgt(t))+w(t))
        v(src(t)):=v'(tgt(t))+w(t);
    }
  }
  for all s in S {
    d(s):=v(s)
    d'(s):=-∞
    if (v(s) < v'(s)) {
      d(s):=-∞
    }
  }
  while (d'≠d) {
    d':=d;
    for all t∈T {
      if d(tgt(t))=-∞ {
        d(src(t)):=-∞
      }
    }
  }

  return(d);
```

By induction, we can prove the following invariants:

Lemma 60. *For each i between 0 and $n+1$ and each $q \in S$,*

- *if $d_i(q)$ is finite, it corresponds to the weight of some path in $\mathcal{A}$ from q to s;*
- *if there is a path from q to s of length at most i, then $d_i(q)$ is less than or equal to the weight of the shortest path from q to s 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. Simlarly, there is a path from q to s 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 q' to s of length $d_i(q')$. Prepending the transition (q, w, q') to this path yields a path from q to s of weight $d_{i+1}(q)$, as required.

For the second claim, let π be one of the shortest paths from q to s of length at most $i+1$, assuming that such a path exists. Let (q, w, q') be its first transition, and consider the suffix $\pi_{>1}$, and its first state q'. It must be the case that $\pi_{>1}$ is one of the shortest path from q' to s of length at most i, since otherwise we could find a path from q to s 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 q' to s 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. □

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

Proposition 61. *For each state q,*

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

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 q to s with weight $(d_{n+1}(q))$ strictly less than the shortest paths from q to s 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 q and s can be made arbitrarily small, by repeating this negative cycle; similarly for any state tat can reach such a state q.

Conversely, assume that $d(q)$ is finite (*i.e.*, q can reach s but $d(q)$ has not been set to $-\infty$), and that q cannot reach s *via* a negative cycle. From the first statement of the lemma above, there is a path from q to s 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 q to s 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 q to s.

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

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

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

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

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

3.1.2 Weighted temporal logics

Quantitative temporal logics are extensions of classical temporal logics in which temporal modalities are decorated with constraints that the accumulated weight has to satisfy for the formula to hold. We only consider pure-future temporal logics here, but past-time modalities could of course be defined similarly.

Definition 62. *The* full weighted branching-time temporal logic *is denoted by* WCTL^**. Formulas of* WCTL^* *over* AP *are formulas built on the following grammar:*

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

where p *ranges over* AP *and* I *ranges over* $\mathcal{I}(\mathbb{N})$*.*

Consider a maximal run ρ *of a labelled transition system* $\mathcal{A}$*, an integer* j *s.t.* $j-1 < \textsf{length}(\rho)$*, and write* s_j *for the* j*-th state of* ρ*. For any* WCTL^* *formula* ϕ*, that* ϕ holds at position j along ρ in $\mathcal{A}$*, denoted by* $\mathcal{A}, \rho \models \phi$*, is defined in the same way as for* PCTL^**, except for the new "until" modality:*

$$\begin{aligned}\mathcal{A}, \rho, j \models \phi_p \,\mathbf{U}_I\, \phi'_p \quad\Leftrightarrow\quad \exists k \in [j+1, \textsf{length}(\rho)) \text{ s.t. } & \mathcal{A}, \rho, k \models \phi'_p \text{ and} \\ & w(\rho_{[j,k]}) \in I \text{ and} \\ & \forall l \in [j+1, k-1].\ \mathcal{A}, \rho, l \models \phi_p.\end{aligned}$$

The weighted linear-time temporal logic*, denoted* WLTL*, is the fragment of* WCTL^* *defined by the following grammar:*

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

The weighted computation-tree logic*, denoted* WCTL*, is the fragment of* WCTL^* *defined by the following grammar:*

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

Their semantics follow from the semantics of WCTL^**.*

We also define the fragments $\mathsf{WCTL}_\sim$ *and* $\mathsf{WLTL}_\sim$ *by restricting their syntax to only allow decorating intervals to include zero or be unbounded. In that case, modality* $\mathbf{U}_{[0,c]}$ *will be written* $\mathbf{U}_{\leq c}$ *and modality* $\mathbf{U}_{[c,+\infty)}$ *will be written* $\mathbf{U}_{\geq c}$*. Fragments* $\mathsf{WCTL}_\approx$ *and* $\mathsf{WLTL}_\approx$ *extend* $\mathsf{WCTL}_\sim$ *and* $\mathsf{WLTL}_\sim$ *with the addition of punctual modalities* $\mathbf{U}_{[c,c]}$*, denoted with* $\mathbf{U}_{=c}$*.*

Before we study expressiveness questions, let us first see what the "constrained until" modality can express. The meaning of $\phi \,\mathbf{U}_I\, \psi$ is rather clear: it holds true along any path ρ that contains a prefix ρ' whose weight is in I, ending in a ψ-state, and all of whose non-extremal states satisfy ϕ. In particular, $\mathbf{F}_{\leq 5}\, p$ means that p will hold true at some point with total accumulated weight at most 5. Dually, $\mathbf{G}_I\, \phi$ holds true if ϕ holds at any point whose accumulated weight is in I.

3.1.3 Expressiveness of WCTL and WLTL

As in the classical setting, the linear-time and branching-time frameworks have uncomparable expressiveness. We will not exhaustively compare all fragments in each framework, but just list a few results. We begin with the branching-time setting.

Proposition 63. *Over weighted labelled transition systems,* WCTL *and* $\mathsf{WCTL}_\approx$ *are equally expressive, and they are strictly more expressive than* $\mathsf{WCTL}_\sim$*.*

Proof. $\mathsf{WCTL}_{\approx}$ is a syntactic fragment of WCTL, so we just have to translate WCTL into $\mathsf{WCTL}_{\approx}$. The naive way—consisting in replacing $\mathbf{U}_I$ with a conjunction of $\mathbf{U}_{=c}$ for all $c \in I$—is not correct. For instance, one can easily prove that the following two formulas are not equivalent:

$$\mathbf{A}\phi\,\mathbf{U}_{[1,2]}\,\psi \not\equiv \mathbf{A}\phi\,\mathbf{U}_{=1}\,\psi \vee \mathbf{A}\phi\,\mathbf{U}_{=2}\,\psi.$$

This would be correct for "exists-until" modality. Notice that it involves a blow-up in the size of the formula, since ϕ and ψ are duplicated.

A more refined—but still wrong—approach consists in replacing each subformulas $\mathbf{E}\phi\,\mathbf{U}_{[m,n]}\,\psi$ and $\mathbf{A}\phi\,\mathbf{U}_{[m,n]}\,\psi$ respectively with

$$\mathbf{E}\phi\,\mathbf{U}_{=m}\,(\,\mathbf{E}\phi\,\mathbf{U}_{\leq n-m}\,\psi) \qquad\qquad \mathbf{A}\phi\,\mathbf{U}_{=m}\,(\,\mathbf{A}\phi\,\mathbf{U}_{\leq n-m}\,\psi).$$

The problem here is that in our semantics, transitions (and their weights) are atomic: in order to fulfill $\phi\,\mathbf{U}_{=m}\,\zeta$, there must be a prefix of weight exactly m, which is not required for satisfying $\phi\,\mathbf{U}_{[m,n]}\,\psi$. The translation above is correct for $\{0,1\}$-weighted labelled transition systems.

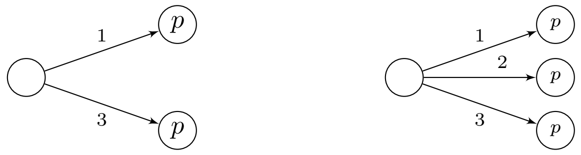
Finally, a correct translation can be obtained by combining both approaches. We only present it on a simple example and leave it to the sleepless reader: consider formula $\mathbf{EG}_{[3,5]}\,\phi$, requiring that ϕ holds true at any point where the accumulated weight is between 3 and 5. We enumerate all possible values of the weight of the transition taken at the time when the interval $[3,5]$ is entered: hence we have

$$\begin{aligned}\mathbf{EG}_{[3,5]}\,\phi \equiv{}& \mathbf{EF}_{=0}\Big[\mathbf{EX}_{=3}\,(\phi\wedge\mathbf{EG}_{\leq 2}\,\phi)\vee\mathbf{EX}_{=4}\,(\phi\wedge\mathbf{EG}_{\leq 1}\,\phi)\vee\mathbf{EX}_{=5}\,(\phi\wedge\mathbf{EG}_{\leq 0}\,\phi)\Big]\vee\\ &\mathbf{EF}_{=1}\Big[\mathbf{EX}_{=2}\,(\phi\wedge\mathbf{EG}_{\leq 2}\,\phi)\vee\mathbf{EX}_{=3}\,(\phi\wedge\mathbf{EG}_{\leq 1}\,\phi)\vee\mathbf{EX}_{=4}\,(\phi\wedge\mathbf{EG}_{\leq 0}\,\phi)\Big]\vee\\ &\mathbf{EF}_{=2}\Big[\mathbf{EX}_{=1}\,(\phi\wedge\mathbf{EG}_{\leq 2}\,\phi)\vee\mathbf{EX}_{=2}\,(\phi\wedge\mathbf{EG}_{\leq 1}\,\phi)\vee\mathbf{EX}_{=3}\,(\phi\wedge\mathbf{EG}_{\leq 0}\,\phi)\Big]\end{aligned}$$

A dual construction can be used to express $\mathbf{AF}_I\,\phi$ (which will be a conjunction of $\mathbf{AG}$-formulas). With some more work, $\mathbf{A}\,\mathbf{U}_I$ can then be derived.

What is easier to show is that $\mathsf{WCTL}_{\sim}$ is not as expressive as $\mathsf{WCTL}_{\approx}$ and WCTL: consider the weighted labelled transition systems depicted on Figure 11 (assuming self-loops on p-states). Obviously, they satisfy the same $\mathsf{WCTL}_{\sim}$ formulas, while only the second one satisfies $\mathbf{EX}_{=2}\,p$. ☐

Figure 11: $\mathsf{WCTL}_{\sim}$ cannot distinguish between these weighted labelled transition systems



The situation in the linear-time setting is easier: the above naive translation is correct here (because path quantifiers do not interfere), so that WLTL and $\mathsf{WLTL}_{\approx}$ are equally expressive. Expressing equality constraints using only inequalities can also be achieved with linear-time: again, the naive approach consisting in writing $\mathbf{F}_{=3}\,p \equiv \mathbf{F}_{\leq 3}\,p \wedge \mathbf{F}_{\geq 3}\,p$ fails, as two different witnesses could be used on the right-hand side. For this idea to work, we have to use the $\mathbf{X}$ modality (with $\mathbf{X}_{\sim c}\,q \equiv \bot\,\mathbf{U}_{\sim c}\,q$), which univocally refers to the very next transition. Then

$$\mathbf{F}_{=1}\,p \equiv \mathbf{F}_{\leq 0}\,(\mathbf{X}_{\leq 1}\,\mathbf{F}_{\leq 0}\,p \wedge \mathbf{X}_{\geq 1}\,\mathbf{F}_{\leq 0}\,p).$$

Then

$$\mathbf{F}_{=2}\,p \equiv \mathbf{F}_{\leq 0}\,(\mathbf{X}_{\leq 2}\,\mathbf{F}_{\leq 0}\,p \wedge \mathbf{X}_{\geq 2}\,\mathbf{F}_{\leq 0}\,p) \vee \mathbf{F}_{=1}\,(\mathbf{F}_{=1}\,p).$$

This generalises to the "until" and "release" modalities, and to any integer or interval constraint. In the end:

Proposition 64. *Over weighted labelled transition systems,* WLTL, $\mathsf{WLTL}_{\approx}$ *and* $\mathsf{WLTL}_{\sim}$ *are equally expressive.*

3.1.4 Model checking WCTL and WLTL

Theorem 65. *Model checking* WCTL$_\sim$ *is* PTIME*-complete.*

Proof. Since WCTL$_\sim$ contains CTL, its model-checking problem is PTIME-hard. We prove memberhip in PTIME: the algorithm is quite similar in spirit to that of CTL, in that it consists in labelling states with the subformulas they satisfy. As a consequence, we just define procedures to decide whether a state satisfies a simple formula, of the form $\mathbf{E}\,\mathbf{U}_{\leq c}$, $\mathbf{E}\,\mathbf{U}_{\geq c}$, $\mathbf{A}\,\mathbf{U}_{\leq c}$ and $\mathbf{A}\,\mathbf{U}_{\geq c}$.

- $\mathbf{E}p\,\mathbf{U}_{\leq c}\,q$: we begin with keeping only those states in which $\mathbf{E}p\,\mathbf{U}\,q$ holds. We have to prove that the shortest path to reach q in the resulting graph has length at most c. Since we are only interested in shortest path, we also restrict each transition of our graph to only carry its minimal possible weight. Effectively computing the weight of the shortest path to some q state can be achieved in time $O(|S|\cdot|T|)$, using Bellman-Ford algorithm (see Algorithm 4).

- $\mathbf{E}p\,\mathbf{U}_{\geq c}\,q$: there are two ways to satisfy this kind of formula: either through a direct path, or via a cycle with positive accumulated weight (in order to have the accumulated weight go above c). The former case can be handled by computing the weight of the longest path reaching q in at most n steps (which might include cyclic paths, but we don't care); this can be achieved by adapting the Bellman-Ford algorithm, running the loop exactly n times and aiming at computing the longest path instead of the shortest one.

  The latter case is handled by first detecting strongly-connected components visiting only p-states and taking at least one non-zero transition. Using Tarjan's algorithm, strongly-connected components can be computed in time $O(|T|+|S|)$ (see Algorithm 5). Keeping only those components that visit only p-states and contain at least one non-zero-weight transition is then easy. After labelling with s each state

Algorithm 5: Computing SCCs

```
function SCC(G,v) {
  // compute SCCs of graph G
  // that are reachable from state v
  count := 0;
  nb_scc := 0;
  list := $\varnothing$;
  tree := {v};
  scc := $\varnothing$;
  visit(v);

  return scc;
}

function visit(s) {
  push(s, list);
  number(s) := count;
  low_reach(s) := count;
  count++;
  for all (s$\rightarrow$r in T) {
    if (r $\notin$ tree) {
      tree := tree $\cup$ {r};
      visit(r);
      low_reach(s) := min(low_reach(s),
                          low_reach(r));
    } else if (r $\in$ list) {
      low_reach(s) := min(low_reach(s),
                          number(r));
    }
  }
  if (low_reach(s)=number(s)) {
    nb_scc++;
    q := pop(list);
    while (q $\neq$ s) {
      scc[nb_scc] := scc[nb_scc] $\cup$ {q};
      q := pop(list);
    }
  }
}
```

  that belongs to such a strongly connected component, it suffices to find those states satisfying $\mathbf{E}\big[p\,\mathbf{U}\,(s\wedge\mathbf{E}p\,\mathbf{U}\,q)\big]$.

- $\mathbf{A}p\,\mathbf{U}_{\leq c}\,q$: we first notice the following equivalence:

$$\mathbf{A}p\,\mathbf{U}_{\leq c}\,q \equiv \mathbf{AF}_{\leq c}\,q \wedge \mathbf{A}p\,\mathbf{U}\,q.$$

This way, it jst remains to find those states in which $\mathbf{AF}_{\leq c}\, q$ holds. Now, there are two ways for a path to not satisfy $\mathbf{F}_{\leq c}\, q$: either the accumulated weight never reaches value c and never visits any q state, or it exceeds c at some points but does not visit a q-state in the meantime. Then

$$\neg\, \mathbf{AF}_{\leq c}\, q \equiv \mathbf{E}(\neg q)\, \mathbf{U}_{\geq c+1} \top \;\vee\; \mathbf{EG}\,(\neg q).$$

- $\mathbf{A}p\, \mathbf{U}_{\geq c}\, q$: when $c = 0$, this is equivalent to $\mathbf{A}p\, \mathbf{U}\, q$. Otherwise, one can check that it holds
$$\mathbf{A}p\, \mathbf{U}_{\geq c}\, q \equiv \mathbf{AG}_{\leq c-1}\,(p \wedge \mathbf{A}p\, \mathbf{U}_{\geq 1}\, q).$$
Modality $\mathbf{AG}_{\leq c-1}$ is dual to $\mathbf{EF}_{\leq c-1}$, which we know how to handle. Finally, $\mathbf{A}p\, \mathbf{U}_{\geq 1}\, q$ can be handled by checking $\mathbf{A}p\, \mathbf{U}\,(\texttt{positive\_weight} \wedge (q \vee p \wedge \mathbf{A}p\, \mathbf{U}\, q))$ where `positive_weight` labels states reached after a positive weight (which might require duplicating some states).

All four procedures run in polynomial time. They have to be applied at most a polynomial number of times, so that the whole algorithm is in PTIME. □

Theorem 66. *Model checking* WCTL$_{\approx}$ *and* WCTL *are* Δ_2^{P}*-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* $\mathsf{PTIME}^{\mathsf{NP}}$*, which is precisely the class* Δ_2^{P} *of the above Theorem. Obviously, a problem solvable in* NP *is also solvable in* $\mathsf{PTIME}^{\mathsf{NP}}$*: it suffices to ask the oracle. Similarly, a problem solvable in* coNP *is also solvable in* $\mathsf{PTIME}^{\mathsf{NP}}$*: 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.*

Proof. We begin with proving that deciding whether a state satisfies $\mathbf{E}p\, \mathbf{U}_{=c}\, q$ is NP-complete. Notice that because c is encoded in binary, the length of a witness path might have size exponential. Simply guessing a witness path would not run in polynomial time. Instead, what the algorithm guesses is the *Parikh image* of a witness path, *i.e.*, the number $n(t)$ of times each transition t is taken. Each transition being taken at most exponentially many times, the guessed integers have polynomial size, and can be guessed in polynomial time. We then have to check that

- the numbers guessed indeed correspond to a path, *i.e.*, that all states (or all states but two) are entered and left the same number of times. In case this is not the case for two of the states, then one of those two exceptions has to be left once more than the number of times it is entered, and symmetrically for the other exception. The first exception state is named s_0, the second one is named s_f;

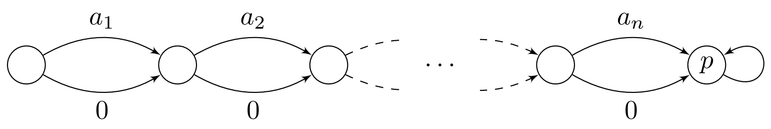
- the subgraph induced by the transitions that are assigned a positive number is connected. At this point, we have checked that the numbers we have guessed correspond to a path;
- state s_f is labelled with q, and all other states, except possibly s_0, are labelled with p;
- write $l(t)$ and $u(t)$ for the lower- and upper bound of each transition t (possibly with $u = (t) + \infty$). Then we have to check that

$$\sum_{t \in T} n(t) \cdot l(t) \leq c \leq \sum_{t \in T} n(t) \cdot u(t).$$

Clearly enough, this algorithm accepts if, and only if, there exists a witness path.

NP-hardness is proven by encoding the SUBSET-SUM problem, which can be expressed as follows: given a set of positive integers $(a_i)_{1 \leq i \leq n}$ and a target value b, does there exists $J \subseteq [1, n]$ such that $\sum_{j \in J} a_j = b$? One easily observes that this is equivalent to the initial state of the weighted labelled transition system of Fig. 12 satisfying $\mathbf{EF}_{=b}\, p$.

Figure 12: Encoding SUBSET-SUM as a $\mathsf{WCTL}_{\approx}$ model-checking problem



We now describe a procedure to decide whether $\mathbf{A}p\,\mathbf{U}_{=c}\,q$ holds at some state. This is achieved by first applying the following equivalence:

$$\mathbf{A}p\,\mathbf{U}_{=c}\,q \equiv \mathbf{A}p\,\mathbf{U}_{\geq c}\,q \wedge \neg\,\mathbf{EG}_{=c}\,\neg q.$$

Then we are left with checking formula of the form $\mathbf{EG}_{=c}\,p$. The NP algorithm will work in the same way as for $\mathbf{E}p\,\mathbf{U}_{=c}\,q$, but will distinguish several cases:

- either it will find an infinite path along which the accumulated weight never exceeds c;
- or it will first find a prefix until the very transition that makes the accumulated weight reach or exceed c. Then either there is a transition that makes the accumulated weight exceed c, in which case the formula is witnessed; or the next transition makes the weight exactly c, and then we have to check that we can find a path along which p holds for ever or until a positive-weight transition is fired.

Finally, notice that handling more general modalities $\mathbf{E}\,\mathbf{U}_I$ and $\mathbf{A}\,\mathbf{U}_I$ for any interval I, can be achieved using similar ideas.

Now consider a Turing machine which could apply these NP algorithms in constant time. Then checking a WCTL ou $\mathsf{WCTL}_{\approx}$ formula can be achieved in deterministic polynomial time, since all other modalities can be solved in polynomial time. This provides us with an algorithm in Δ_2^{P}.

It remains to prove Δ_2^{P}-hardness. To this aim, we consider the SNSAT problem [LMS02]: an instance of SNSAT is as follows:

$$\mathcal{I} = \left[\begin{array}{rl} x_1 := & \exists Z_1.\ \phi_1(Z_1) \\ x_2 := & \exists Z_2.\ \phi_2(Z_2, x_1) \\ x_3 := & \exists Z_3.\ \phi_3(Z_3, x_2, x_1) \\ & \cdots \\ x_n := & \exists Z_n.\ \phi_n(Z_n, x_{n-1}, ..., x_2, x_1) \end{array}\right]$$

In other terms, SNSAT is a sequence of SAT queries, each query depending on the results of all previous queries. Solving an SNSAT instance can clearly be achieved in Δ_2^{P}, and SNSAT can be proved Δ_2^{P}-complete.

Write $X = \{x_1, x_2, ..., x_n\}$, and $Z = Z_1 \cup Z_2 \cup ... \cup Z_n$. We also assume (w.l.o.g.) that all propositional formulas ϕ_k are conjunctions of disjunctive clauses with three disjuncts. If ϕ_k contains a clause $C_{k,l} = \alpha_{k,l,1} \vee \alpha_{k,l,2} \vee \alpha_{k,l,3}$, we let $\overline{C_{k,l}}$ be the clause $\neg x_k \vee C_{k,l}$, and define $\mathcal{C} = \{C_1, ..., C_r\}$ for the set of all resulting clauses. An instance $\mathcal{I}$ of SNSAT naturally defines a unique valuation $v_\mathcal{I}$ of X, satisfying $v_\mathcal{I}(x_k) = \top$ iff $\phi_k(Z_k, v_\mathcal{I}(x_{k-1}), ..., v_\mathcal{I}(x_1))$ is satisfiable.

Now, a valuation w of $X \cup Z$ is said

- **safe** if for all k, $w(x_k)$ implies $\phi_k(w(Z_k), w(x_{k-1}), ..., w(x_1))$;
- **correct** if for all k, $w(x_k)$ equals $\phi_k(w(Z_k), w(x_{k-1}), ..., w(x_1))$;
- **admissible** if it is correct and coincides with $v_\mathcal{I}$ on X.

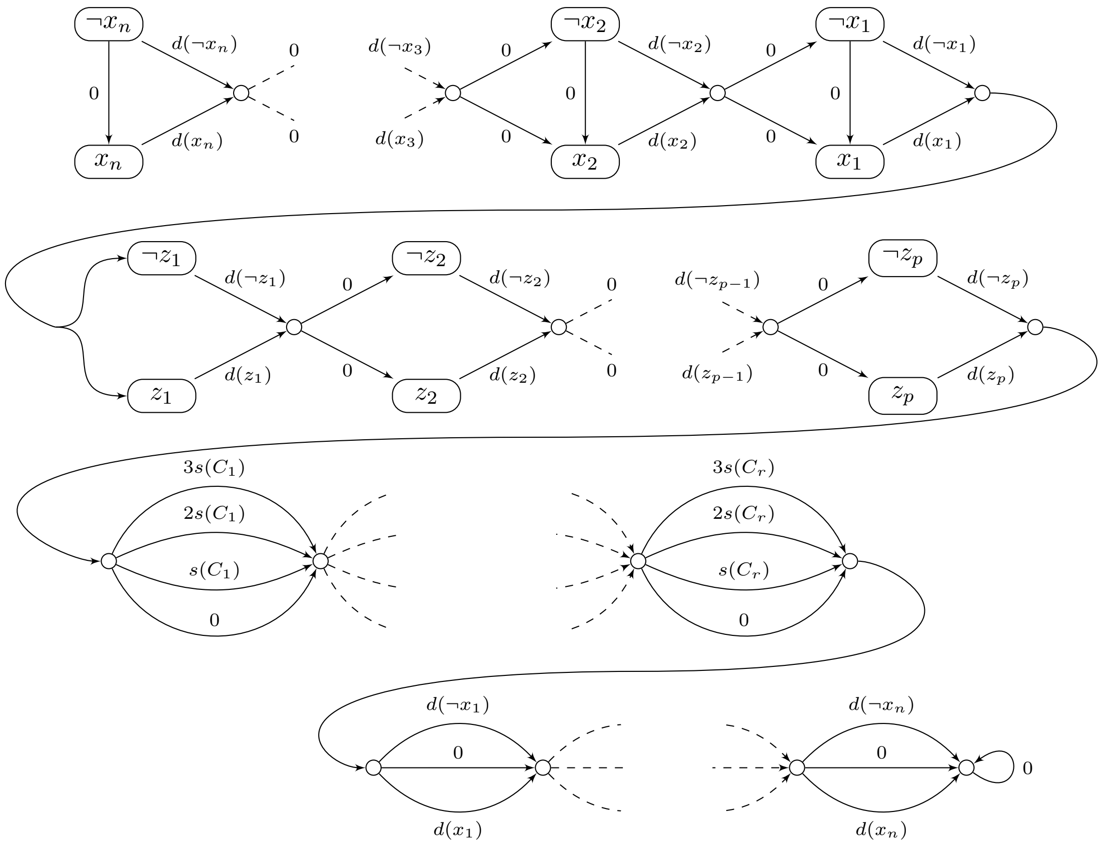
Clearly, being admissible requires being correct, which in turn requires being safe. Notice in particular that a valuation can be correct but not admissible: it can be the case for instance that $\phi_k(w(Z_k), w(x_{k-1}), ..., w(x_1))$ evaluates to false while $\phi_k(Z_k, w(x_{k-1}), ..., w(x_1))$ is satisfiable.

We show that checking whether a given valuation is admissible is Δ_2^P-complete. To this aim, fix some $K \in \mathbb{N}$, whose value will be made explicit later. Let

$$s(x_i) = K \qquad s(z_i) = K^{n+i} \qquad s(C_i) = K_{n+p+i}$$

where $n = |X|$ and $p = |Z|$. Consider a mapping $\mathcal{M} \colon X \cup Z \cup \mathcal{C} \to \mathbb{N}$ (which can be seen as a multiset of $X \cup Z \cup \mathcal{C}$). Its weight is defined as $s(\mathcal{M}) = \sum_{v \in X \cup Z \cup \mathcal{C}} s(v) \cdot \mathcal{M}(v)$. Notice that if $\mathcal{M}(v) < K$ and $\mathcal{M}'(v) < K$ for all v, then $s(\mathcal{M}) = s(\mathcal{M}')$ iff $\mathcal{M} = \mathcal{M}'$.

Figure 13: Reducing SNSAT to WCTL model checking



Now, consider the weighted labelled transition system of Fig. 13. In this figure, for a literal α involving variable x, we let $d(\alpha) = s(x) + \sum\{s(C) \mid C \in \mathcal{C},\ \alpha \Rightarrow C\}$.

A path in the first two rows of the figure defines a valuation (provided that it does not take the vertical transitions, which we will enforce), while the other two rows contain "filling nodes", which will be used to reach a given accumulated weight. In the first two rows, a path collects the weights of the the variables it visits, plus the weights of the clauses that are entailed by each literal (thus each clause may be counted up to four times).

Write $K' = \sum_{v \in X \cup Z} s(v) + 4 \times \sum_{c \in \mathcal{C}} s(C)$, and pick a path π of weight K' (if any). Following our earlier remark, if K is large enough, path π must visit exactly one of x_k and $\neg x_k$, for all k. Since $s(C)$ might be accumulated up to eleven times along a path (three times in the third row, four times in the first two rows, and four times in the last row), it suffices to take $K > 11$. Moreover π also has to accumulate $4s(C)$ for all $C \in \mathcal{C}$, so that it must visit some literal implying C at least once. In other terms, any path of length K' corresponds to a valuation that satisfy all the clauses.

Now, we define the following formulas:

$$\zeta_0 = \top$$
$$\zeta_k = \mathbf{E}\left[P_x \Rightarrow \mathbf{EX}\left(P_{\neg x} \wedge \neg\zeta_{k-1}\right] \mathbf{U}_{=K'} \top$$

where P_x and $P_{\neg x}$ label all states x_i and $\neg x_i$, respectively. We prove the following:

Lemma 67. *For any integers k and $1 \leq r \leq n$,*

- *if $k \geq 2r - 1$, then ($v_{\mathcal{I}}(x_r) = \top$ if, and only if, state x_r satisfies ζ_k);*
- *if $k \geq 2r$, then ($v_{\mathcal{I}}(x_r) = \bot$ if, and only if, state $\neg x_r$ satisfies ζ_k).*

This lemma is proved inductively on k. It is trivial when $k = 0$. Let $k > 0$, assuming that the result holds true for $k - 1$. Let r satisfying the constraint of the lemma. Let w be an admissible valuation, and π_w be a path satisfying the following requirements:

- it starts in x_r or $\neg x_r$;
- it only visits literal that are true under w;
- it has total weight K'.

Because of the second condition, the part of π_w in the first two rows is uniquely defined. Moreover, if the path visits x_l for $l \leq r$, then all the clauses originating from ϕ_l are satisfied, so that their weight appears at least once in the weight of π; on the other hand, if the path visits $\neg x_l$, then all the clauses originating from ϕ_l also appears at least once, because they contains $\neg x_l$ as a disjunct. By carefully selecting the path in the filling nodes, it is then possible to find a path π_w as above.

Now, we claim that π_w witnesses the fact that $x_r \models \zeta_k$ (or $\neg x_r \models \zeta_k$): this requires any state of the form $\neg x_l$ visited by π_w to satisfy $\mathbf{EX}\left(P_x \wedge \neg\zeta_{k-1}\right)$, which equivalently means that if x_l is false under $v_{\mathcal{I}}$, then state x_l has to satisfy $\neg\zeta_{k-1}$; this follows from the induction hypothesis.

Conversely, assume that $k \geq 2r - 1$ and state x_r satisfies ζ_k. Pick a witness path π, of total weight K'. As explained above, this path corresponds to a valuation w that satisfies all the clauses. It remains to prove that this valuation coincides with $v_{\mathcal{I}}$ on $\{x_1, ..., x_r\}$, by induction on i:

- if $w(x_i) = \top$ (meaning that π visits x_i), then since w satisfies all clauses in $\mathcal{C}$, it must be the case that all clauses of ϕ_i are true under w. It follows that $\phi_i(Z_i, w(x_{i-1}), ..., w(x_1))$ is satisfiable. When $i = 1$, this reads $\phi_1(Z_1)$ is satisfiable, so that $v_{\mathcal{I}}(x_1) = \top$. Assuming that the result holds up to $i - 1$, the fact that $\phi_i(Z_i, w(x_{i-1}), ..., w(x_1))$ entails that $\phi_i(Z_i, v_{\mathcal{I}}(x_{i-1}), ..., v_{\mathcal{I}}(x_1))$ is satisfiable, so that $v_{\mathcal{I}}(x_i) = \top$.
- if $w(x_i) = \bot$ (then the path visits $\neg x_i$; in particular, $i < r$), then in x_i, formula $\neg\zeta_{k-1}$ holds true. Since $i < r$, then $k - 1 \geq 2i - 1$, and the induction hypothesis (on k) implies $v_{\mathcal{I}}(x_i) = \bot$.

Finally, the case where $k \geq 2r$ and state $\neg x_r$ satisfies ζ_k is handled similarly. Only the last point changes: it can be the case that $i = r$. But then again $k-1 \geq 2i-1$, and the induction hypothesis (on k) again yields $v_{\mathcal{I}}(x_i) = \bot$. □

Remark. *When weights are in $\{0,1\}$, the situation (with equality) is easier. We explain the algorithm for modality $\mathbf{E}\,\mathbf{U}_{=k}$: assume the formula to be checked is $\mathbf{E}p\,\mathbf{U}_{=k}\,q$, with $k > 0$, and that we already know which states satisfy subformulas p and q. We define the following $|S|^2$-matrices T_p and $T_{p,q}$ as follows:*

- $T_p = \{(s, s') \mid s \models \mathbf{E}p\,\mathbf{U}_{=1}\,(s' \wedge p)\}$*;*
- $T_{p,q} = \{(s, s') \mid s \models \mathbf{E}p\,\mathbf{U}_{=1}\,(s' \wedge q)\}$*;*

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

- *when $k = 1$: this is by definition of $T_{p,q}$;*
- *if the result holds up to k, we pick a state for which the corresponding line in $T_p^k \cdot T_{p,q}$ contains a positive number. This matrix is the result of the product of T_p and $T_p^{k-1} \cdot T_{p,q}$. 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. $T_p(s, s')$ and $[T_p^{k-1} \cdot T_{p,q}](s', s'')$ are positive. This entails the result.*

The other modalities can be handled in a similar way.

We now turn to linear time. There again, equality contraints will make a difference:

Theorem 68. *Model checking* $\mathsf{WLTL}_{\sim}$ *on weighted labelled transition systems is* PSPACE-*complete.*

Proof. Hardness follows from that of LTL model checking. We delay this proof until Section 3.2, where we deal with the discrete-time semantics of timed automata. Weighted labelled transition systems can easily be encoded as timed automata under discrete semantics, for which we will develop a model-checking algorithm for WLTL. □

Theorem 69. *Model checking* $\mathsf{WLTL}_{\approx}$ *and* WLTL *on weighted labelled transition systems is* EXPSPACE-*complete.*

Proof. Again, we only prove the hardness part, and delay the algorithm to Section 3.2. The hardness proof consists in encoding the halting problem of an exponential-space Turing machine. Each configuration of the Turing machine is encoded through 2^n consecutive states, each state representing the content of one cell of the tape of the Turing machine (and possibly the position of the tape head and the current state of the machine). Each transition has weight 1. The ensuring the correct application of a transition of the machine from one configuration to the next one can easily be achieved by using modality $\mathbf{F}_{=2^n}$, which has size $O(n)$ because of the binary encoding of integer constants. The details are straightforward, and left to the reader. □

3.1.5 Semi-continuous semantics of weighted labelled transition systems

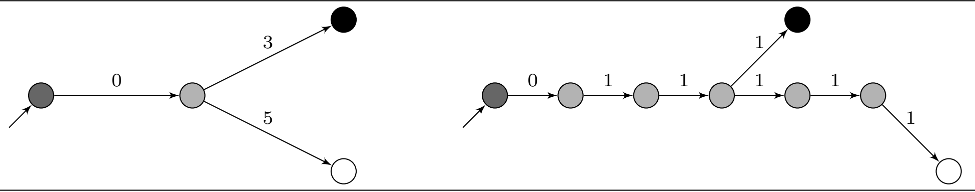
Our semantics for weighted automata uses atomic steps for the transitions. In order to get closer to the semantics of timed automata, we could define a *semi-continuous* semantics, where a step of weight n would correspond to n successive steps fo weight 1: in other terms, a weighted automaton $\mathcal{A}$ 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 14 explains the intuition behing the semi-continuous semantics on a small example. It is easily noticed that the semi-continuous is really different from the discrete one, even when no quantitative constrinat is used: for instance, formula* $\mathbf{EF}\,(◐ \wedge \mathbf{EF}\,○ \wedge \neg\,\mathbf{EF}\,●)$ *is true in the semi-continuous semantics on our example, while it does not hold under the discrete semantics.*

Figure 14: The continuous semantics for weighted automata



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

Theorem 70. *The* WCTL *model-checking problem on weighted labelled transition systems with the semi-continuous semantics is* PSPACE*-complete.*

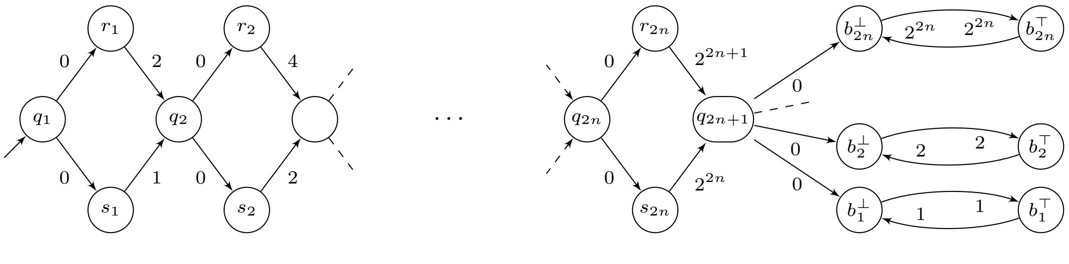
Proof. Membership in PSPACE is proved by adapting the CTL labelling algorithm: we inductively label the states of the (original) weighted automaton with the subformulas it satisfies. Checking that $\mathbf{E}p\,\mathbf{U}_{[a,b]}\,q$ holds in a state can be achieved by guessing a (possibly exponential) witness step by step, whose states can be stored in polynomial space.

PSPACE-hardness is proved by encoding the QBF problem:

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

This problem is well-known to be PSPACE-complete. The reduction is achieved as follows: we consider the weighted automaton depicted on Figure 15. Notice that this automaton only depends on the number of variables in the QBF instance, and not on the boolean formula itself: this formula will be transformed into a WCTL formula to be checked on the automaton.

Figure 15: Reduction from QBF



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

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

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

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

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

Indeed, from s, it takes weight i to reach q_{2n+1}. We then have to loop between $b_i^\top$ and $b_i^\perp$ for sepnding the remaining weight of $2^{2n+1} - 1 - i$. If the j-th bit of $2^{2n+1} - 1 - i$ is 1, then this will leave us in state $b_i^\top$, and if that bit is 0, we will end up in $b_i^\perp$.

Replacing each occurrence of x_i with $\mathbf{EF}_{=2^{2n+1}-1}\, b_i^\top$ 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. □

When forbiding equality constraints, we can recover tractability:

Theorem 71. *The* $\mathsf{WCTL}_\sim$ *model-checking problem on weighted labelled transition systems with the semi-continuous semantics is* PTIME*-complete.*

Proof. Assume $\mathcal{A} = \langle Q, R, l, w\rangle$, and let $\mathcal{T}$ be the (possibly infinite) weighted automaton $\mathfrak{C}(\mathcal{A})$ representing its semi-continuous semantics. We design an algorithm for computing, for each state $q \in Q$ and each subformula ψ, the values of i for which $\mathcal{T}, (q, i) \models \psi$. Given a state q and a subformula ψ of the formula ϕ under study, we define $\mathsf{Sat}[q, \psi]$ as the (possibly infinite) sequence of integer intervals $S_j = [\alpha_j, \beta_j)$ such that:

- $\mathcal{T}, (q, i) \models \psi$ **iff** $i \in \bigcup_j S_j$;
- for any $S_j = [\alpha_j, \beta_j)$, we have
  - $[\alpha_j, \beta_j) \subseteq [0, \delta_{\max}(q))$, where $\delta_{\max}(q)$ is the maximal constant labelling transitions exiting q;
  - $\alpha_j < \beta_j$, and
  - $\beta_j < \alpha_{j+1}$ if $\mathsf{Sat}[q, \psi]$ contains at least $j + 1$ items.

For any q and ψ, this clearly defines a unique set $\mathsf{Sat}[q, \psi]$. Its number of intervals in $\mathsf{Sat}[q, \psi]$ is the *size* of $\mathsf{Sat}[q, \psi]$ (denoted by $|\mathsf{Sat}[q, \psi]|$). In the sequel, we write $\mathsf{Sat}[q, \psi]$ for the union $\bigcup_j S_j$.

We define procedures for inductively computing $\mathsf{Sat}[q, \xi]$ for all subformulas ξ of a given formula $\phi \in \mathsf{WCTL}_\sim$ and for all states $q \in Q$. Along with these procedures, we show that

- $|\mathsf{Sat}[q, \xi]|$ is finite and bounded by $|\xi| \cdot |R(q)|$, where $R(q)$ is the set of transitions out of q in $\mathcal{A}$, and
- $\mathsf{Sat}[q, \xi]$ (for all states q and a given formula ξ) can be computed in time $O(|\xi|^2 \cdot |R|^3)$.

This will globally ensure that the whole algorithm runs in time $O(|\phi|^3 \cdot |R|^3)$.

Before going further, we introduce some new notations: For a given integer interval $\rho = [l,u)$, we write $\rho - 1$ for the interval $[\max(0,l-1),\infty)$ if $u=\infty$, and $[\max(0,l-1),u-1)$ otherwise. We also define $\overleftarrow{\rho}$ as the interval ρ itself if it equals the singleton $[0,1)$, and as $\rho - 1$ otherwise.

We now describe our procedures and prove the above statements. The cases of atomic propositions and Boolean connectives are straightforward and clearly satisfy the requirements w.r.t. the size of $\mathsf{Sat}[q,\psi]$. We now consider the remaining cases:

- Case $\xi = \mathbf{E}\zeta\,\mathbf{U}_{\leq c}\,\theta$: For each state (q,i) in $\mathcal{T}$, we compute the duration of the shortest path (if any) witnessing the property $\mathbf{E}\zeta\,\mathbf{U}\,\theta$, and compare it to c.

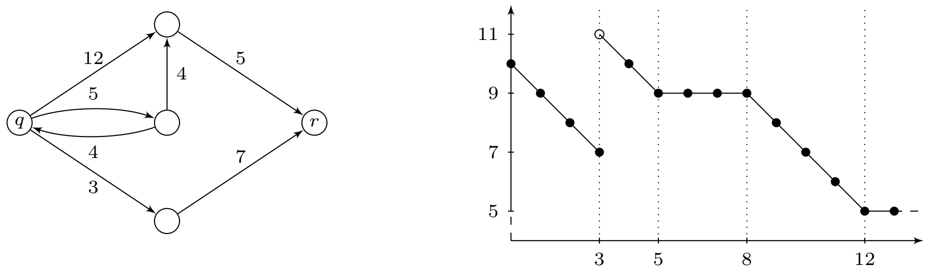
  For each state q in Q, assuming $\mathsf{Sat}[q,\zeta]$ and $\mathsf{Sat}[q,\theta]$ have already been computed, we first refine these intervals by computing the smallest list of intervals $L(q) = \bigcup_{j=1\dots l}[a_j,b_j)$ s.t.:

  1. For any j, $a_j < b_j$, and $b_j \leq a_{j+1}$ if $j+1 \leq l$;
  2. For any i, we have $i \in L(q) \Leftrightarrow i \in \mathsf{Sat}[q,\zeta] \cup \mathsf{Sat}[q,\theta]$, and each interval $[a_j,b_j)$ is either included in one of $\mathsf{Sat}[q,\theta]$'s intervals, or disjoint with $\mathsf{Sat}[q,\theta]$.
  3. Intervals in $L(q)$ are *homogeneous* w.r.t. action transitions: for any transition $(q,\rho,q') \in R$, for any j, either $[a_j,b_j) \subseteq \overleftarrow{\rho}$ or $[a_j,b_j) \cap \overleftarrow{\rho} = \varnothing$.
  4. The special interval $[0,1)$ is handled separately: If $0 \in \mathsf{Sat}[q,\zeta] \cup \mathsf{Sat}[q,\theta]$, then it is the first interval in $L(q)$.

  Building $L(q)$ is easy from $\mathsf{Sat}[q,\zeta]$ and $\mathsf{Sat}[q,\theta]$: Computing the special union of condition 2 yields at most $|\mathsf{Sat}[q,\zeta]| + 2|\mathsf{Sat}[q,\theta]|$ intervals. Then, by condition 3, any transition (q,ρ,q') might split one of these intervals into two or three smaller ones, *i.e.*, add two intervals. Last, condition 4 possibly adds another one. Thus $|L(q)| \leq |\mathsf{Sat}[q,\zeta]| + 2|\mathsf{Sat}[q,\theta]| + 2|R(q)| + 1$.

  Let $\delta^{\min}_{q,i}$ be the duration of the shortest paths satisfying ζ and leading to some θ-state. Clearly $(q,i) \models \xi$ iff $\delta^{\min}_{q,i} \leq c$. Let $[a,b)$ be an interval in $L(q)$. Since any point in $[a,b)$ may fire the same set of action transitions, the function $i \mapsto \delta^{\min}_{q,i}$ is non-increasing over $[a,b)$: any execution starting by an action transition (leading to some $(q',0)$) enabled from (q,i) is also enabled from $(q,i+1)$ if $i,i+1 \in [a,b)$. Figure 16 describes an example of such duration function.

Figure 16: A duration function for a weighted automaton



We have the following important properties:

- assume that $\delta^{\min}_{q,a}$ is known for every left-end point a of the intervals in $L(q)$, then it is possible to deduce easily $\delta^{\min}_{q,i}$ for any $i \in L(q)$. Indeed, for $[a,b)$ in $L(q)$, we have:
  - either there is an interval in $L(q)$ of the form $[b,b')$. Then for any position $i \in [a,b)$, a shortest path leading to θ may start either by an action transition—and then $\delta^{\min}_{q,i} = \delta^{\min}_{q,a}$—or by letting time elapse until the interval $[b,b')$—and then $\delta^{\min}_{q,i} = \delta^{\min}_{q,b} - b - i$.

    - or there is no interval $[b, b')$ in $L(q)$. Then for any $i \in [a, b)$, we have $\delta^{\min}_{q,i} = \delta^{\min}_{q,a}$, since in that case, the shortest path necessarily begins with an action transition.
  - a shortest path from some (q, a) with $[a, b) \in L(q)$ starts by an action transition or by a delay transition of at least $b - a$ time units: it is never pertinent to wait before performing an enabled action transition when considering shortest paths. Time elapsing only occurs when it is necessary to reach the next interval of $L(q)$.

Therefore it is sufficient to compute the duration of shortest paths from the left-end point of any interval of $L(q)$, and we can consider a jump-semantics point of view restricted to left-end points: The intermediary states (inside the intervals) are not relevant for this. Consider the DTG $\mathcal{G} = (V_\mathcal{G}, \to_\mathcal{G}, \ell_\mathcal{G})$ as follows:

  - $V_\mathcal{G} = \{(q, [a, b)) \mid [a, b) \in L(q)\}$;
  - $\ell_\mathcal{G} : V_\mathcal{G} \to \{\theta, \zeta \wedge \neg\theta\}$ labels each state (q, ρ) depending on whether $\rho \subseteq \mathsf{Sat}[q, \theta]$;
  - Transitions $\to_\mathcal{G}$ are computed as follows:
    - Consider $(q, \rho, q') \in R$ s.t. $[0, 1) \in L(q')$. We have: $(q, [a, b)) \xrightarrow{1}_\mathcal{G} (q', [0, 1))$ whenever $[a, b) \in L(q)$ and $a + 1 \in \rho$. Moreover we have $(q, [0, 1)) \xrightarrow{0}_\mathcal{G} (q', [0, 1))$ whenever $[0, 1) \in L(q)$ and $0 \in \rho$.
    - If $[a, b), [b, b') \in L(q)$, then we have $(q, [a, b)) \xrightarrow{b-a}_\mathcal{G} (q, [b, b'))$.

Then we have: $|\mathcal{G}| = |V_\mathcal{G}| + |{\to_\mathcal{G}}| \leq \sum_{q \in Q} |L(q)| + \sum_{q \in Q} |L(q)| \cdot (|R| + 1)$. Now we can adapt the PTIME procedure for $\mathsf{WCTL}_\sim$ (Theorem 65) to get the duration of shortest paths leading to θ for any state $(q, [a, b))$ of $\mathcal{G}$, and it corresponds precisely to $\delta^{\min}_{q,a}$. This is achieved in time $O(|V_\mathcal{G}| \cdot | \to_\mathcal{G} |)$.

Now it remains to compute $\mathsf{Sat}[q, \mathbf{E}\zeta \, \mathbf{U}_{\leq c} \, \theta]$ from $\delta^{\min}_{q,a}$ and c. If $\delta^{\min}_{q,a} \leq c$, we have $[a, b) \subseteq \mathsf{Sat}[q, \xi]$. Otherwise if, for some b', $[b, b') \in L(q)$ and $\delta^{\min}_{q,b} \leq c$, then $[b - (c - \delta^{\min}_{q,b}), b) \subseteq \mathsf{Sat}[q, \xi]$. Then we merge the intervals in $\mathsf{Sat}[q, \xi]$ in order to fulfill its requirements.

The size of $\mathsf{Sat}[q, \xi]$ can be bounded by $|\mathsf{Sat}[q, \theta]| + |\mathsf{Sat}[q, \zeta]| + |R(q)|$. Indeed, $\mathsf{Sat}[q, \mathbf{E}\zeta \, \mathbf{U} \, \theta]$ contains at most $|\mathsf{Sat}[q, \theta]| + |\mathsf{Sat}[q, \zeta]|$ intervals. Now, as explained above, we may have to split these intervals depending on the length of the shortest path. Two cases may arise:

  - the splitting occurs while the length of the shortest path is decreasing (and thus becomes smaller than c). This case occurs when we are waiting for a transition to be enabled, *i.e.*, it is bound to a constraint $x \geq i$. Thus one transition contains at most one such constraint, and thus may give rise to at most one such splitting;
  - the splitting occurs at a point where the shortest path is increasing, *i.e.*, the shortest path is longer than c after that splitting. This may only happen when a transition becomes disabled, that is, it is bound to a constraint $x \leq i$. Here again, one transition may give rise to at most one such splitting.

Thus one transition (q, ρ, q') may at most add one interval in $\mathsf{Sat}[q, \xi]$. Finally, we get $|\mathsf{Sat}[q, \xi]| \leq |\mathsf{Sat}[q, \theta]| + |\mathsf{Sat}[q, \zeta]| + |R(q)|$.

- Case $\xi = \mathbf{E}\zeta \, \mathbf{U}_{\geq c} \, \theta$: We assume $c > 0$—the case $c = 0$ corresponds to the standard CTL modality. We use similar techniques as in the previous case. Now in $L(q)$ we distinguish the sub-intervals satisfying $\zeta \wedge \neg\theta$, $\zeta \wedge \theta$ or $\neg\zeta \wedge \theta$. Moreover we replace every interval $[a, b)$ labeled by $\neg\zeta \wedge \theta$ with $[a, a+1)$ because only point a may witness ξ. We have $|L(q)| \leq 2 \cdot (|\mathsf{Sat}[q, \zeta]| + |\mathsf{Sat}[q, \theta]| + |R(q)|)$. We also build a DTG $\mathcal{G} = (V_\mathcal{G}, \to_\mathcal{G}, \ell_\mathcal{G})$ with $V_\mathcal{G} = \{(q, [a, b)) \mid [a, b) \in L(q)\}$ and $\ell_\mathcal{G} : V_\mathcal{G} \to \{\zeta \wedge \theta, \zeta \wedge \neg\theta, \neg\zeta \wedge \theta\}$. But now we look for maximal durations $\delta^{\max}_{q,a}$ to reach θ and we distinguish finite intervals and unbounded intervals:

- For finite intervals in $L(q)$, we only consider the right-end points because as soon as a long path goes through the interval $[a, b)$ with $b < \infty$, it goes through the point $b-1$. And we have $\delta^{\max}_{q,i} = \delta^{\max}_{q,b-1} + b - 1 - i$ for any $i \in [a, b)$.
- For unbounded interval $[a, \infty)$ in $L(q)$, we have $\delta^{\max}_{q,i} = \delta^{\max}_{q,j}$ for any $i, j \in [a, \infty)$—and then $(q, i) \models \xi$ iff $(q, j) \models \xi$—therefore we can restrict ourselves to look for the truth value of ξ at a.

We then define the transitions of $\mathcal{G}$ in order to represent these right-end points of finite intervals and the left-end point of unbounded intervals; the aim is to use the algorithm defined for the jump semantics to compute the maximal durations. We define $\rightarrow_{\mathcal{G}}$ as follows:

- Consider $(q, \rho, q') \in R$ s.t. $[0, 1) \in L(q')$. We have: $(q, [a, b)) \xrightarrow{1}_{\mathcal{G}} (q', [0, 1))$ whenever $[a, b) \in L(q)$ and $[a, b) \subseteq \rho$. Moreover we have $(q, [0, 1)) \xrightarrow{0}_{\mathcal{G}} (q', [0, 1))$ whenever $[0, 1) \in L(q)$ and $\rho = [0, 0]$.
- For any $[a, b), [a', b')$ in $L(q)$ s.t. $b = a'$, we have $(q, [a, b)) \xrightarrow{b'-b}_{\mathcal{G}} (q, [b, b'))$ (resp. $(q, [a, b)) \xrightarrow{1}_{\mathcal{G}} (q, [b, \infty)))$ if $b' < \infty$ (resp. $b' = \infty$).
- If $[a, \infty) \in L(q)$, we have $(q, [a, \infty) \xrightarrow{1}_{\mathcal{G}} (q, [a, \infty))$.

A state $(q, [a, b))$ with $b < \infty$ of $\mathcal{G}$ stands for the state $(q, b - 1)$ in WA while a state $(q, [a, \infty))$ in $\mathcal{G}$ stands for (q, a) in $\mathcal{A}$. The third kind of transition is used to represent time elapsing in unbounded intervals.

Note that a transition $(q, [a, b)) \xrightarrow{1} (q, [b, b'))$ in $\mathcal{G}$ with $b' < \infty$ represents the path $(q, b-1) \xrightarrow{1} (q, b) \xrightarrow{1} \ldots (q, b'-1)$ in $\mathcal{T}$. Then the labeling of intermediary states is given by the target node (contrary to the case where nodes correspond to the left-end points), but this does not matter for $\mathbf{E}\zeta\,\mathbf{U}\,\theta$ modality because these intermediary states exist iff $b' > b + 1$ and this entails $(q, [b, b')) \subseteq \mathsf{Sat}[q, \zeta]$.

Again, we then use the algorithm for model-checking $\mathsf{WCTL}_{\sim}$ in the classical semantics (Theorem 65): we assume that it returns maximal durations for states of $\mathcal{G}$, and ∞ (resp. $-\infty$) when paths until θ can be made arbitrary long (resp. there is no path reaching θ). The algorithm runs in time $O(|V_{\mathcal{G}}| \cdot | \rightarrow_{\mathcal{G}} |)$.

It remains to merge contiguous intervals in order to get $\mathsf{Sat}[q, \xi]$. As in the previous case, we end up with at most $|\mathsf{Sat}[q, \theta]| + |\mathsf{Sat}[q, \zeta]| + |R(q)|$ intervals.

- Case $\xi = \mathbf{A}\zeta\,\mathbf{U}_{\leq c}\,\theta$: we reduce to the previous cases using equivalence

$$\mathbf{A}\zeta\,\mathbf{U}_{\leq c}\,\theta \equiv \mathbf{AF}_{\leq c}\,\theta \wedge \neg\,\mathbf{E}(\neg\theta)\,\mathbf{U}\,(\neg\zeta \wedge \neg\theta)$$

  and

$$\mathbf{AF}_{\leq c}\,\theta \equiv \neg\mathsf{E}(\neg\theta)\,\mathbf{U}_{>c}\,\top \wedge \neg\,\mathbf{EG}\,(\neg\theta)$$

- Case $\xi = \mathbf{A}\zeta\,\mathbf{U}_{\geq c}\,\theta$: we reduce to the previous cases using equivalence

$$\mathbf{A}\zeta\,\mathbf{U}_{\geq c}\,\theta \equiv \mathbf{AG}_{<c}\,(\zeta \wedge \mathbf{A}\zeta\,\mathbf{U}_{>0}\,\theta)$$

  and $\mathbf{AG}_{<c}\,\zeta \equiv \neg\,\mathbf{EF}_{<c}\,\neg\zeta$ and

$$\mathbf{A}\zeta\,\mathbf{U}_{>0}\,\theta \equiv \mathbf{AG}_{\leq 0}\,(\zeta \wedge \mathbf{AX}\,(\,\mathbf{A}\zeta\,\mathbf{U}\,\theta)) \wedge \mathbf{AF}_{\geq 1}\,\top.$$

  Here formula $\mathbf{AF}_{\geq 1}\,\top$ means that there is no run of null duration (we may assume that $c \geq 1$, since otherwise $\xi \equiv \mathbf{A}\zeta\,\mathbf{U}\,\theta$), and is equivalent to $\neg\,\mathbf{EF}_{\leq 0}\,P^{0}_{\top}$.

Now we can show that the algorithm preserves the fact that $|\mathsf{Sat}[q, \xi]|$ is bounded by $|\xi| \cdot |R(q)|$. This entails that the weighted automaton $\mathcal{G}$ built for $\mathbf{E}\zeta\,\mathbf{U}_{\sim c}\,\theta$ is such that $|V_{\mathcal{G}}|$ is in $O(|\xi| \cdot |R|)$ and $| \rightarrow_G |$ is in $O(|\xi| \cdot |R|^2)$; thus the procedures run in time $O(|\xi|^2 \cdot |R|^3)$. □

3.1.6 Model checking WATL

In the same way as weighted labelled transition systems extend labelled transition systems with a weight function on transitions, weighted concurrent game structures add weight on the transitions of their underlying transition system. Notice that here the weight on each transition is a single integer, so that each move vector selects one transition and one weight. Formally:

Definition 72. *An* weighted concurrent game structure *is an* 8*-tuple* $\mathcal{A} = \langle S, T, \ell, w, \mathbb{A}, \mathbb{M}, \mathit{Ch}, \mathit{Edg}\rangle$ *where* $\langle S, T, \ell, \mathbb{A}, \mathbb{M}, \mathit{Ch}, \mathit{Edg}\rangle$ *is a concurrent game structure and* $w\colon T \to \mathbb{N}^*$ *assigns a positive*[4] *weight to each transition.*

The definitions of strategies and outcomes are unchanged. Each run in a weighted concurrent game structure can now be assigned a weight, though the homomorphism extending w.

The extension of ATL^* to the weighted setting follows the same pattern as for WCTL^*:

Definition 73. *The* full weighted alternating-time temporal logic *is denoted by* WATL^*. *Formulas of* WATL^* *over* AP *are built on the following grammar:*

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

where p *ranges over* AP *and* I *ranges over* $\mathcal{I}(\mathbb{N})$. *The* weighted alternating-time logic, *denoted* WATL, *is the fragment of* WATL^* *defined by the following grammar:*

$$\begin{aligned}\mathsf{WATL} \ni \phi_s &::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \langle\!\langle A\rangle\!\rangle\, \phi_p \mid [\![A]\!]\, \phi_p \\ \phi_p &::= \phi_s \,\mathbf{U}_I\, \phi_s.\end{aligned}$$

We also define the fragments $\mathsf{WATL}_\sim$ *and* $\mathsf{WATL}_\approx$, *in the same way as for* WCTL. *The semantics of these logics follows from the semantics of* ATL *and of* WCTL^*.

For the usual reason (that $\langle\!\langle A\rangle\!\rangle\,(\phi \vee \psi)$ is not equivalent to $\langle\!\langle A\rangle\!\rangle\,\phi \vee \langle\!\langle A\rangle\!\rangle\,\psi$), the proof that $\mathsf{QCTL}_\approx$ and QCTL are equally expressive does not carry over to $\mathsf{WATL}_\approx$ and WATL. That $\mathsf{WATL}_\sim$ is strictly less expressive than $\mathsf{WCTL}_\approx$ is still true.

We now turn to model checking, first proving the following result:

Theorem 74. *Model checking* $\mathsf{WATL}_\sim$ *is* PTIME*-complete.*

Proof. We begin with $\mathsf{WATL}_\sim$, proposing polynomial-time algorithm for handling each modality. The global algorithm will label states with the subformulas they satisfy, as for CTL.

Lemma 75. *Let* $\mathcal{A}$ *be a weighted game structure, and* $\phi = \langle\!\langle A\rangle\!\rangle\, q\,\mathbf{U}_{\leq c}\, q$ *be a* WATL *formula. Then we can compute in time* $O(|S|\cdot|\mathit{Edg}|)$ *the set of locations of* $\mathcal{A}$ *where* ϕ *holds.*

To prove this result, we define the extra modality $\mathbf{U}^{\leq i}_{\leq c}$, with the following semantics:

$$\begin{aligned}\rho \models p\,\mathbf{U}^{\leq i}_{\leq c}\, q \quad &\iff \quad \exists j.\ 0 < j \leq i,\ s_j \models q,\ w(\rho_{\leq j}) \leq c, \\ &\qquad\qquad \text{and } \forall k.\ 0 < k < j \Rightarrow s_k \models p\end{aligned}$$

This modality requires that the right-hand side formula be satisfied within at most i steps (while still bounding the total weight). It is clear that, for any $n \in \mathbb{N}$,

$$s \models \langle\!\langle A\rangle\!\rangle\, p\,\mathbf{U}_{\leq c}\, q \quad\iff\quad s \models \langle\!\langle A\rangle\!\rangle\, p\,\mathbf{U}^{\leq |S|}_{\leq c}\, q.$$

The implication from right to left is easy. Conversely, if all the outcomes of a strategy satisfy $p\,\mathbf{U}_{\leq c}\, q$, it is possible to make the strategy simpler, so that each state is visited at most once along each outcome (while still reaching a q-state within weight n). This is achieved

[4]The restriction to positive weights is a technical requirement for proving correctness of our algorithms.

by considering the (finite) tree of all the outcomes of that strategy σ. Assume some state r is visited several times along some branch ρ. Consider the shortest and longest prefixes ρ_0 and ρ_1 of ρ ending in r, and define $\sigma'(\rho_0) = \sigma(\rho_1)$. One easily proves that the execution tree of σ' pruned at the first q-state is strictly smaller than that of σ, while still enforcing $p \, \mathbf{U}_{\leq c} \, q$. Applied iteratively, this proves the existence of a "simple" strategy.

We now define functions $v_i(q)$, for $i \in \mathbb{N}$ and $s \in S$, by the following recursive rules. We fist let $q(s) = 0$ when $q \in \ell(s)$ and $q(s) = +\infty$ otherwise, and $p(s) = 0$ when $p \in \ell(s)$ and $p(s) = +\infty$ otherwise. Then we inductively define

$$v_1(s) = \min_{m \in \mathbb{M}(s,A)} \max_{\bar{m} \in \mathbb{M}(s,\overline{A})} \Big(w(\mathsf{Edg}(s, m \cdot \bar{m})) + q(\mathsf{tgt}(\mathsf{Edg}(s, m \cdot \bar{m}))\Big)$$
$$v_{i+1}(s) = \min_{m \in \mathbb{M}(s,A)} \max_{\bar{m} \in \mathbb{M}(s,\overline{A})} \Big(w(\mathsf{Edg}(s, m \cdot \bar{m})) + \min_{s' = \mathsf{tgt}(\mathsf{Edg}(s, m \cdot \bar{m}))} [q(s'), p(s') + v_i(s')]\Big)$$

Our proof now amounts to showing the following result: for any $i \in \mathbb{N}^*$, for any $n \in \mathbb{N}$ and any $s \in S$, we have:

$$n \geq v_i(s) \qquad \Longleftrightarrow \qquad s \models \langle\!\langle A \rangle\!\rangle \, p \, \mathbf{U}^{\leq i}_{\leq n} \, q.$$

The proof is by induction on i:

- when $i = 1$: pick $n \geq v_1(s)$ (hence assuming $v_0(s)$ is finite). Then there is a move $m \in \mathbb{M}(s, A)$ such that, for all move $\bar{m} \in \mathbb{M}(s, \overline{A})$, n is larger than $w(\mathsf{Edg}(s, m \cdot \bar{m}))$, and moreover all states reached in one step from s when A plays m are labelled with q. This precisely proves that $s \models \langle\!\langle A \rangle\!\rangle \, p \, \mathbf{U}^{\leq 1}_{\leq n} \, q$.

  Symmetrically, if $s \models \langle\!\langle A \rangle\!\rangle \, p \, \mathbf{U}^{\leq 1}_{\leq n} \, q$, then there is a move for A which enforces an immediate visit to a q-state via transitions having weight less than or equal to n. This precisely witnesses the fact that $v_1(s) \leq n$.

- the inductive step is proven similarly: assume the result holds for some i: pick $n \geq v_i(s)$, and a move m for A such that for any $\bar{m}$ for $\overline{A}$, n is larger than $w(t) + p(\mathsf{tgt}(t)) + v_i(\mathsf{tgt}(t))$ or larger than $w(t) + q(\mathsf{tgt}(t))$, where $t = \mathsf{Edg}(s, m \cdot \bar{m})$. Then for all move $\bar{m}$, either the resulting state is labelled with q and the weight of the corresponding transition is at most n, or the resulting state is labelled with p, and satisfies $\langle\!\langle A \rangle\!\rangle \, p \, \mathbf{U}^{\leq i-1}_{\leq n-w(t)} \, q$, by induction hypothesis. In both cases, we conclude that $s \models \langle\!\langle A \rangle\!\rangle \, p \, \mathbf{U}^{\leq i}_{\leq n} \, q$.

  Symmetrically, if $s \models \langle\!\langle A \rangle\!\rangle \, p \, \mathbf{U}^{\leq i}_{\leq n} \, q$, then the first move of a witnessing strategy witnesses the fact that $n \geq v_i(s)$, following the same lines as above: depending on the move played by $\overline{A}$, writing t for the resulting transition, either a q-state is reached after t, which entails $n \geq w(t)$, or a p-state is reached where $\langle\!\langle A \rangle\!\rangle \, p \, \mathbf{U}^{\leq i-1}_{\leq n-w(t)} \, q$ is reached, which by induction hypothesis means that $n - w(t) \geq v_{i-1}(\mathsf{tgt}(t))$, and entails our result.

It then suffices to compute $v_{|S|}(s)$, for each $s \in S$, to deduce the set of locations where ϕ holds. This algorithm thus runs in time $O(|S| \cdot |\mathsf{Edg}|)$.

The release modality is handled in the same way: we define modality $\mathbf{R}^{\leq i}_{\leq c}$ as the dual of $\mathbf{U}^{\leq i}_{\leq c}$: it has the following semantics:

$$\rho \models \mathbf{R}^{\leq i}_{\leq c} \, q \qquad \Longleftrightarrow \qquad \forall j.\ 0 < j \leq i \Rightarrow \big[w(\rho_{\leq j}) \leq c \Rightarrow (s_j \models q \text{ or } \exists 0 < k < j.\ s_k \models p)\big]$$

Then we have the following equivalence:

$$s \models \langle\!\langle A \rangle\!\rangle \, p \, \mathbf{R}_{\leq c} \, q \qquad \Longleftrightarrow \qquad s \models \langle\!\langle A \rangle\!\rangle \, p \, \mathbf{R}^{\leq |S|}_{\leq c} \, q.$$

This is proven with arguments similar to the case of the "until" modality. The easy implication is from left to right. Conversely, assume that $s \models \langle\!\langle A \rangle\!\rangle \, p \, \mathbf{R}^{\leq |S|}_{\leq c} \, q$, and consider the finite tree of all outcomes of this strategy, pruned after $|S|$ transitions. Those outcomes along which some p-state is visited already satisfy $p\,\mathbf{R}_{\leq c}\,q$, and can be left unchanged. The other outcomes only visit q-states, and there is at least one state that is visited twice along those outcomes. They can then be made infinite, while always satisfying q-states, hence satisfying $p\,\mathbf{R}_{\leq c}\,q$.

Computing the set of states where $\langle\!\langle A\rangle\!\rangle\, p\,\mathbf{R}^{\leq|S|}_{\leq c}\,q$ holds can then be achieved in a similar way as for $\mathbf{U}^{\leq i}_{\leq c}$. This time, we let $q'(s) = +\infty$ when $q \in \ell(s)$ and $q'(s) = 0$ otherwise, and $p'(s) = +\infty$ when $p \in \ell(s)$ and $p'(s) = 0$ otherwise, and inductively define

$$\begin{aligned}
v'_1(s) &= \max_{m\in\mathbb{M}(s,A)}\ \min_{\bar{m}\in\mathbb{M}(s,\overline{A})} \Big(w(\mathsf{Edg}(s, m\cdot\bar{m})) + q'(\mathsf{tgt}(\mathsf{Edg}(s, m\cdot\bar{m})))\Big)\\
v'_{i+1}(s) &= \max_{m\in\mathbb{M}(s,A)}\ \min_{\bar{m}\in\mathbb{M}(s,\overline{A})} \Big(w(\mathsf{Edg}(s, m\cdot\bar{m})) + \min_{s'=\mathsf{tgt}(\mathsf{Edg}(s,m\cdot\bar{m}))} [q'(s'), p'(s') + v'_i(s')]\Big).
\end{aligned}$$

then for any $i \in \mathbb{N}^*$, for any $n \in \mathbb{N}$ and any $s \in S$, we have:

$$n < v'_i(s) \qquad\Longleftrightarrow\qquad s \models \langle\!\langle A\rangle\!\rangle\, p\,\mathbf{R}^{\leq i}_{\leq n}\,q.$$

- when $i = 1$: First assume $v'_1(s) = +\infty$ (hence any n satisfies the condition on the left). This means that for some strategy of A, all the resulting states satisfy q, which entails that $p\,\mathbf{R}^{\leq 1}_{\leq n}\,q$ for any n, as required. If $v'_1(s)$ is finite, then there is a move for A for any move of $\overline{A}$, either a q-state is reached, or n is strictly less than the weight of the resulting transition. This again enforces $p\,\mathbf{R}^{\leq 1}_{\leq n}\,q$. Conversely, when $s \models \langle\!\langle A\rangle\!\rangle\, p\,\mathbf{R}^{\leq 1}_{\leq n}\,q$, then there is a strategy for A to either go to a q-state, or to take a transition with weight strictly more than n, which entails the result.
- the inductive step is proven similarly: pick some $n < v'_i(s)$, and a move m for A such that, for any move $\bar{m}$ of $\overline{A}$, writing t for the resulting transition, n is strictly less than both $w(t) + q'(\mathsf{tgt}(t))$ and $w(t) + p'(\mathsf{tgt}(t)) + v'_{i-1}(\mathsf{tgt}(t))$. In case $\mathsf{tgt}(t)$ satisfies $p \wedge q$, that outcome satisfies $p\,\mathbf{R}^{\leq i}_{\leq n}\,q$. In case $\mathsf{tgt}(t)$ satisfies $\neg p \wedge q$, then $n - w(\mathsf{tgt}(t)) < v'_{i-1}(\mathsf{tgt}(t))$, and the induction hypothesis entails that $\mathsf{tgt}(t) \models \langle\!\langle A\rangle\!\rangle\, p\,\mathbf{R}^{\leq i-1}_{\leq n-w(t)}\,q$, so that we again have our result. Finally, in case it satisfies $\neg q$, then $n < w(t)$, and the outcome again satisfies $p\,\mathbf{R}^{\leq i}_{\leq n}\,q$. The converse direction is proven similarly.

We now handle modalities $\langle\!\langle A\rangle\!\rangle\, p\,\mathbf{U}_{\geq c}\,q$ and $\langle\!\langle A\rangle\!\rangle\, p\,\mathbf{R}_{\geq c}\,q$, using again similar techniques. We first define one more modality $p\,\mathbf{U}^{\geq i}\,q$, requiring that the eventuality is fulfilled after at least i steps:

$$\rho \models p\,\mathbf{U}^{\geq i}\,q \qquad\Longleftrightarrow\qquad \exists j.\ j \geq i,\ s_j \models q,\ \text{and } \forall k.\ 0 < k < j \Rightarrow s_k \models p.$$

Then we have

$$s \models \langle\!\langle A\rangle\!\rangle\, p\,\mathbf{U}_{\geq n}\,q \iff s \models \langle\!\langle A\rangle\!\rangle\,\big[(p\,\mathbf{U}_{\geq n}\,q) \vee (p\,\mathbf{U}^{\geq|S|+1}\,q)\big].$$

This equivalence relies on the fact that all durations are strictly positive: if some outcome of the strategy satisfies $p\,\mathbf{U}^{\geq|S|+1}\,q$, then one location is visited twice along that outcome, and it is possible to adapt the strategy so that it is visited n times (thus with total duration larger than n) before visiting q.

We now define our sequence of values as follows: first, we let $p''(s) = 0$ when $p \in \ell(s)$, and $p''(s) = -\infty$ otherwise. Similarly, $q''(s) = 0$ when $q \in \ell(s)$, and $q''(s) = -\infty$ otherwise. We then define

$$\begin{aligned}
v''_0(s) &= 0 \qquad \text{if } s \models \langle\!\langle A\rangle\!\rangle\, p\,\mathbf{U}\,q\\
v''_0(s) &= -\infty \quad \text{otherwise}\\
v''_{i+1}(s) &= \max_{m\in\mathbb{M}(s,A)}\ \min_{\bar{m}\in\mathbb{M}(s,\overline{A})} \Big(w(\mathsf{Edg}(s, m\cdot\bar{m})) + \max_{s'=\mathsf{tgt}(\mathsf{Edg}(s,m\cdot\bar{m}))} [q''(s'), p''(s') + v''_i(s')]\Big).
\end{aligned}$$

The proof then relies on the following equivalence: for any $i \in \mathbb{N}$, for any $n \in \mathbb{N}$ and any $s \in S$, we have:

$$n \leq v'_i(s) \qquad \Longleftrightarrow \qquad s \models \langle\!\langle A \rangle\!\rangle\, (p\, \mathbf{U}_{\geq n}\, q \vee p\, \mathbf{U}^{\geq i+1}\, q).$$

We leave the reader prove this equivalence, following similar arguments as above.

Finally, $p\, \mathbf{R}_{\geq c}\, q$ can be handled similarly, by first defining modality $\mathbf{R}^{\geq i}$. We then show that

$$s \models \langle\!\langle A \rangle\!\rangle\, p\, \mathbf{R}_{\geq c}\, q \qquad \Longleftrightarrow \qquad \langle\!\langle A \rangle\!\rangle\, (p\, \mathbf{R}_{\geq c}\, q \wedge p\, \mathbf{R}^{\geq |S|+1}\, q).$$

To conclude, it suffices to compute the following sequence:

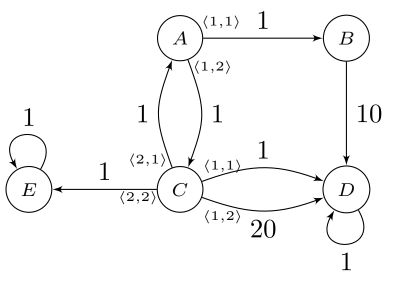
$$\begin{aligned} v'''_0(s) &= -\infty \quad \text{if } s \models \langle\!\langle A \rangle\!\rangle\, p\, \mathbf{R}\, q \\ v'''_0(s) &= 0 \qquad \text{otherwise} \\ v'''_{i+1}(s) &= \min_{m \in \mathbb{M}(s,A)}\ \max_{\bar{m} \in \mathbb{M}(s,\overline{A})} \Big(w(\mathsf{Edg}(s, m \cdot \bar{m})) + \max_{s' = \mathsf{tgt}(\mathsf{Edg}(s, m \cdot \bar{m}))} [q''(s'), p''(s') + v''_i(s')] \Big). \end{aligned}$$

The reader will show that for any $i \in \mathbb{N}$, for any $n \in \mathbb{N}$ and any $s \in S$, we have:

$$n > v'_i(s) \qquad \Longleftrightarrow \qquad s \models \langle\!\langle A \rangle\!\rangle\, (p\, \mathbf{R}_{\geq n}\, q \wedge p\, \mathbf{R}^{\geq i+1}\, q).$$

To conclude, we have proposed polynomial-time algorithms to handle all four modalities of $\mathsf{WCTL}_{\sim}$. The PTIME algorithm follows. PTIME-hardness already holds for plain ATL. $\square$

Figure 17: The algorithm for $\mathbf{U}_{\leq n}$.



$v_i(q)$	1	2	3
A (p, $\neg q$)	$+\infty$	21	21
B (p, q)	10	10	10
C (p, $\neg q$)	20	20	20
D ($\neg p$, q)	1	1	1
E (p, $\neg q$)	$+\infty$	$+\infty$	$+\infty$

Example. *Consider the example depicted on Figure 17. On that weighted game, the duration is the integer written in the middle of each transition. The tuples that are written close to the source location indicates the choices of the agents for firing that transition (for instance, $\langle 2, 1\rangle$ means that player a_1 chooses move 2 and player a_2 chooses move 1). They are omitted when each agent has a single choice.*

The valuations of atomic propositions are given in the table on the right of the figure. This table shows the computation of $v_i(s)$, for each location. This computation converges in three steps. For instance, that $v_3(A) = 21$ indicates that $A \models \langle\!\langle a_1 \rangle\!\rangle\, P_1\, \mathbf{U}_{\leq 21}\, P_2$ holds, but $A \not\models \langle\!\langle a_1 \rangle\!\rangle\, P_1\, \mathbf{U}_{\leq 20}\, P_2$.

Theorem 76. *Model-checking* $\mathsf{WATL}_{\approx}$ *is* $\mathsf{EXPTIME}$*-complete.*

Proof. We explain the algorithm for $\langle\!\langle A \rangle\!\rangle\, p\, \mathbf{U}_{=c}\, q$: it consists in filling in a boolean table T of size $|S| \times c$ (which is exponential-size since c is encoded in binary). The table is computed as follows: we first consider the case where the "until" modality is fulfilled in one step:

$$\begin{aligned} T(s,i) = \top \qquad \Longleftrightarrow \qquad & \exists m \in \mathbb{M}(s,A).\ \forall \bar{m} \in \mathbb{M}(s,\overline{A}). \\ & \qquad w(\mathsf{Edg}(s, m \cdot \bar{m})) = i\ \wedge\ q \in \ell(\mathsf{tgt}(\mathsf{Edg}(s, m \cdot \bar{m}))). \end{aligned}$$

We then dynamically fill in the table, setting

$$T(s,i)=\top \iff \exists m\in \mathbb{M}(s,A).\ \forall \bar{m}\in \mathbb{M}(s,\overline{A}).\\ T(\mathsf{tgt}(\mathsf{Edg}(s,m\cdot\bar{m})), i-w(\mathsf{Edg}(s,m\cdot\bar{m})))=\top.$$

This computation can be achieved because all durations are assumed to be positive, so that there is no cyclic dependency. Its correctness is clear.

The "release" modality is handled similarly. In the end, the computation runs in exponential time.

We now prove EXPTIME-hardness, by encoding the countdown-game problem: a countdown game is played on a weighted graph; a configuration of the game is a pair (v, C) where v is a state of the graph and C is an integer. At each round, Player A selects one weight d, and Player B answers with a new state v' for which (v, d, v') is a transition of the weighted graph (player A has to ensure that such a v' exists). The play continues as long as $C > 0$. The new configuration of the countdown game is $(v', C - d)$. A configuration (v, C) is winning for Player A if $C = 0$. It is losing if $C < 0$.

Deciding the winner of a countdown game is EXPTIME-complete [JLS07]. One can easily encode such a game as a $\mathsf{WATL}_{\approx}$ model-checking problem: the structure of transition system underlying the game under checking is the weighted graph. In each state, each weight labelling an outgoing edge is a valid move of Player A. Player B then has as many moves as needed to select one of the transition carrying the weight selected by A. The formula to be checked then writes $\langle\langle A\rangle\rangle\, \mathbf{F}_{=C_0}\, \top$. □

3.2 Quantitative temporal logics and timed automata under discrete-time semantics

3.2.1 Discrete-time semantics for timed automata

In this section, we begin with (re)introducing timed automata, but with a discrete-time semantics: all clocks still evolve at the same speed and can be reset along transitions, but they are restricted to only take integer values. Before we formally define timed automata, we need some preliminary definitions.

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 77. *A* clock valuation *on $\mathfrak{C}$ is a mapping $\nu\colon \mathfrak{C}\to\mathbb{T}$. 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 v and a value $d \in \mathbb{T}$, we write $v + t$ for the clock valuation w such that $w(c) = v(c) + t$ for all $c \in \mathfrak{C}$.*
- *given a valuation v and a set $R \subseteq \mathfrak{C}$, we write $v[R \to 0]$ for the clock valuation w such that $w(c) = v(c) \cdot \mathbb{1}_R(c)$.*

Definition 78. *A* clock constraint *is a formula built on the following grammar:*

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

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

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

We write $[\![g]\!]_{\mathfrak{C}}$ for the set of valuations of $\mathfrak{C}$ satisfying constraint g.

Definition 79. *Let AP be a finite set of atomic propositions. A* timed automaton *[AD94] over AP is a 5-tuple $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \mathsf{Inv}\rangle$ where*

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

The discrete-time semantics of timed automata is defined as an (countably-)infinite-state labelled transition system as follows:

Definition 80. *Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \mathsf{Inv}\rangle$ be a timed automaton over AP. The* semantics *of $\mathcal{A}$, denoted with $[\![\mathcal{A}]\!]$, is the (infinite-state) weighted labelled transition system $[\![\mathcal{A}]\!] = \langle Q, R, l\rangle$ where*

- *$Q = \{(s, \nu) \in S \times \mathbb{N}^{\mathfrak{C}} \mid \nu \models \mathsf{Inv}(s)\}$;*
- *R is the union of two sets of transitions:*
  - delay transitions*: $((s, \nu), d, (s', \nu')) \in R$ iff (s, ν) and (s', ν') belong to Q, $s = s'$, and $d \in \mathbb{N}_{>0}$ and $\nu' = \nu + d$;*
  - action transitions*: $((s, \nu), 0, (s', \nu')) \in R$ iff (s, ν) and (s', ν') belong to Q and there exists $t = (s, g, r, s') \in T$ s.t. $\nu \models g$, $\nu' = \nu[r \to 0]$;*
- *$l((s, \nu)) = \ell(s)$ for all $(s, \nu) \in Q$.*

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

Remark. *Notice that our semantics is a kind of mix between atomic and "semi-continuous" delays: transitions are taken in an atomic way, but when there is a transition from (s, ν) to (s, ν') of duration d, then it is also possible to take a sequence of d transitions of duration 1 from (s, ν) to (s, ν').*

One way of forcing atomic delays is by only considering one type of transitions: these transitions are obtained by merging a delay transition with an action transition (hence requiring that there is only one delay transition between any two action transitions—notice that we have to allow delays of duration zero here). We will not consider this semantics here.

We could also go in the other direction, and consider only unitary delays from some clock valuation ν to $\nu + 1$. Again, we leave the interested reader develop this semantics.

3.2.2 Timed temporal logics

We now define TCTL^* on these models. This will very closely resemble WCTL^*:

Definition 81. *The* full timed branching-time temporal logic *is denoted by TCTL^*. Formulas of TCTL^* over AP are formulas built on the following grammar:*

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

where p ranges over AP and I ranges over $\mathcal{I}(\mathbb{N})$.

Semantically, given a state $s \in S$ and a clock valuation ν, that (s, ν) satisfies a TCTL^* state formula ϕ is equivalent to having state (s, ν) satisfy ϕ, seen as a WCTL^* formula, in the weighted automata $[\![\mathcal{A}]\!]$.

The fragments TCTL and TLTL (often named MTL, for Metric Temporal Logic*, in the context of timed automata), correspond to WCTL and WLTL, respectively. Their semantics follow from that of TCTL^*.*

In the real-time setting, another extension of temporal logics with quantitative constraints has been considered: it consists in having "formula clocks", in the same way as there are clocks in the automata. Formula clocks can be reset during the evaluation of the formula, and they can be compared to integer values. Formally:

Definition 82. *The logic* TCTL_c^* *is defined by the following grammar:*

$$\begin{aligned}\mathsf{TCTL}_c^* \ni \phi_s &::= p \mid c \sim n \mid c.\phi_s \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= \phi_s \mid c.\phi_p \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \,\mathbf{U}\, \phi_p\end{aligned}$$

where c *ranges over a finite set of variables,* n *ranges over* $\mathbb{N}$*, and* $\sim \in \{<, \leq, =, \geq, >\}$*. The fragments* TCTL_c *and* TLTL_c *(most often called* TPTL*, for* Timed Propositional Temporal Logic*, in the timed setting) are defined as follows:*

$$\begin{aligned}\mathsf{TCTL}_c \ni \phi_s &::= p \mid c \sim n \mid c.\phi_s \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= \phi_s \,\mathbf{U}\, \phi_s\end{aligned}$$

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

Finally, we also define the following other fragment, which we name TBTL_c*:*

$$\begin{aligned}\mathsf{TBTL}_c \ni \phi_s &::= p \mid c \sim n \mid c.\phi_s \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \mathbf{E}\phi_p \mid \mathbf{A}\phi_p \\ \phi_p &::= \phi_s \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \,\mathbf{U}\, \phi_p\end{aligned}$$

Notice that, compared to TCTL_c^**, it only forbids clock resets within path formulas. In other terms, clock resets are bound to path quantifiers, and path formulas are in* LTL *(extended with clock constraints).*

The evaluation of a TCTL_c^* *formula involves a valuation of the clock formulas, which is updated during the evaluation. We redefine the semantics of all the operators, as this clock valuation changes the formalism. Let* $\mathcal{A}$ *be a weighted labelled transition system,* ρ *be a maximal path in* $\mathcal{A}$ *(writing* $(s_i)_{i\geq 0}$ *for the sequence of states, and* w_i *for the duration of the transition from* s_i *to* s_{i+1}*),* ϕ *be a formula in* TCTL_c^* *and* ζ *be an (integer-valued) valuation of the formula clocks of* ϕ*. That* ρ *(at position zero) in* $\mathcal{A}$ *satisfies* ϕ *under valuation* ζ*, denoted with* $\mathcal{A}, \rho \models_\zeta \phi$*, is defined inductively on* ϕ *as follows:*

$$\begin{aligned}
\mathcal{A}, \rho \models_\zeta p &\quad\Leftrightarrow\quad p \in \ell(s_0) \\
\mathcal{A}, \rho \models_\zeta c \sim n &\quad\Leftrightarrow\quad \zeta(c) \sim n \\
\mathcal{A}, \rho \models_\zeta c.\phi_s &\quad\Leftrightarrow\quad \mathcal{A}, \rho \models_{\zeta[c\leftarrow 0]} \phi_s \\
\mathcal{A}, \rho \models_\zeta \neg\phi_s &\quad\Leftrightarrow\quad \mathcal{A}, \rho \not\models_\zeta \phi_s \\
\mathcal{A}, \rho \models_\zeta \phi_s \vee \phi'_s &\quad\Leftrightarrow\quad \mathcal{A}, \rho \models_\zeta \phi_s \text{ or } \mathcal{A}, \rho \models_\zeta \phi'_s \\
\mathcal{A}, \rho \models_\zeta \mathbf{E}\phi_p &\quad\Leftrightarrow\quad \exists \rho' \in \mathit{MaxRuns}(s_0).\ \mathcal{A}, \rho' \models_\zeta \phi_p \\
\mathcal{A}, \rho \models_\zeta \mathbf{A}\phi_p &\quad\Leftrightarrow\quad \forall \rho' \in \mathit{MaxRuns}(s_0).\ \mathcal{A}, \rho' \models_\zeta \phi_p
\end{aligned}$$

$$\begin{aligned}
\mathcal{A}, \rho \models_\zeta c.\phi_p &\quad\Leftrightarrow\quad \mathcal{A}, \rho \models_{\zeta[c\leftarrow 0]} \phi_p \\
\mathcal{A}, \rho \models_\zeta \neg\phi_p &\quad\Leftrightarrow\quad \mathcal{A}, \rho \not\models_\zeta \phi_p \\
\mathcal{A}, \rho \models_\zeta \phi_p \vee \phi'_p &\quad\Leftrightarrow\quad \mathcal{A}, \rho \models_\zeta \phi_p \text{ or } \mathcal{A}, \rho \models_\zeta \phi'_p \\
\mathcal{A}, \rho \models_\zeta \phi_p \,\mathbf{U}\, \phi'_p &\quad\Leftrightarrow\quad \exists k > 0 \text{ s.t. } \mathcal{A}, \rho_{\geq k} \models_{\zeta_k} \phi'_p \text{ and} \\
&\qquad\qquad \forall l \in [1, k-1].\ \mathcal{A}, \rho_{\geq l} \models_{\zeta_l} \phi_p, \\
&\qquad\qquad\quad \text{where } \zeta_0 = \zeta \text{ and } \zeta_{i+1} = \zeta_i + w_i
\end{aligned}$$

3.2.3 Expressiveness of TCTL^* and TCTL_c^*

It should be clear that formula clocks can encode constraints that decorate modalities:

$$\phi \,\mathbf{U}_I\, \psi \equiv c.(\phi \,\mathbf{U}\, (\psi \wedge c \in I)).$$

It follows that TCTL^*_c embeds TCTL^*, TCTL_c embeds TCTL, and TPTL embeds MTL.

The converse is also true, thanks to the discrete nature of time: in the same way as for $\mathsf{WLTL}_\sim$ and WLTL, we can enumerate all possible combinations of delays that amount to the required value in order to express any TCTL_c formula in TCTL [AH93].

3.2.4 Model checking of TCTL^* and TCTL^*_c

We now turn to model checking, proving decidability results for all logics we defined.

Theorem 83. *Model checking* TCTL *and* TCTL_c *on discrete-time timed automata is* PSPACE*-complete.*

Proof. For TCTL, this can be proved in the same lines as for Theorem 70. An algorithm for TCTL_c can be obtained by adapting the algorithm, handling formula clocks together with the clocks of the automaton. □

Theorem 84. *Model checking* MTL *and* TPTL *on discrete-time timed automata is* $\mathsf{EXPSPACE}$*-complete.*

Proof. We describe the algorithm for TPTL: it extends the so called *tableaux* construction (which we presented as resulting in a Büchi automaton, see page 24) for LTL and PLTL. Given a formula $\phi \in \mathsf{TPTL}$, we build a corresponding Büchi automaton $\mathcal{B}_\phi$ characterising the timed words satisfying ϕ (for the discrete-time semantics).

As for LTL, the states of the automaton will be subsets of subformulas (more precisely, formulas from the closure) of the formula ϕ being checked. But the closure here has to also take delays into account: to this aim, we add new atomic propositions δ_k, for $k \geq 0$ (hence infinitely many for the moment; we will reduce to a finite number later). Proposition δ_k will label any state that is reached after a delay of k time units.

We assume that our input formula (and all subformulas we will consider) are of the form $x.\phi$. This can be achieved by adding a reset of a fresh clock, if not already present. We also assume $x.\phi$ to be closed. This again can be obtained by initially resetting all clocks. Finally, we assume that any clock is reset at most once in ϕ, even if it means renaming some of the clocks. Now, given a formula $x.\psi$ s.t. x is free in ϕ, and an integer k, we define the shift of $x.\psi$ by k inductively as follows:

- when $k = 0$, $x.\psi^0$ equals $x.\psi$;
- when $k + 1 > 0$, $x.\psi^{k+1}$ is obtained from $x.\psi^k$ by replacing x with $x + 1$ in ψ. Then formulas of the form $x \geq -c$ and $x \leq -c - 1$ are replaced with true and false, respectively (for nonnegative c).

Thanks to the last step, the set of shihfted formulas remain finite: any constraint will eventually be changed to true or false.

Example. *If* ϕ *is* $\mathbf{F}\,(a \wedge \mathbf{F}\,(b \wedge x \geq 4))$*, then* $x.\phi^1$ *is* $\mathbf{F}\,(a \wedge \mathbf{F}\,(b \wedge x \geq 3))$*, and* $x.\phi^4$ *is* $\mathbf{F}\,(a \wedge \mathbf{F}\,b)$*.*

This construction enjoys the following property:

Lemma 85. *Let* $\rho =$ *be a path. Let* $x.\psi$ *be a closed* TPTL *formula, and* ζ *be a valuation of the clocks of* ψ*. Then* $\rho \models_\zeta x.\psi^k$ *iff* $\rho \models_{\zeta[x\leftarrow k]} \psi$*.*

The proof is by induction on the structure of the formula, and by induction on k.

As for LTL, we now define the *closure* of a formula ϕ. Roughly, it contains all the subformulas of ϕ, as well as all intermediary properties to be checked. Formally, the closure of $x.\phi$ (remember that all our formulas are assumed to begin with a clock reset) is the smallest set of formulas containing $x.\phi$ and such that

- $x.\neg\phi_1$ is in the closure if, and only if, $x.\phi_1$ is;
- if $x.(\phi_1 \vee \phi_2)$ is in the closure, then so are $x.\phi_1$ and $x.\phi_2$;

- if $x.(\phi_1 \wedge \phi_2)$ is in the closure, then so are $x.\phi_1$ and $x.\phi_2$;
- if $x.z.\phi_1$ is in the closure, then so is $x.\phi_1[z \leftarrow x]$, which is obtained by replacing all free occurrences of z with x in ϕ_1;
- if $x.\mathbf{X}\,\phi_1$ is in the closure, then so are all formulas $x.\phi_1^k$ for k;
- if $x.\phi_1 \,\mathbf{U}\, \phi_2$ is in the closure, then so are $x.\mathbf{X}\,\phi_2$, $x.\mathbf{X}\,\phi_2$ and $x.\mathbf{X}\,(\phi_1 \,\mathbf{U}\, \phi_2)$.

Notice that, as we assumed $x.\phi$ to be closed, any formula in the closure is also closed.

One can easily check the following result:

Lemma 86. *Writing M for the maximal constant appearing in ϕ, then the closure of $x.\phi$ has at most $4M \cdot |\phi|$ formulas.*

We now define the set of states of our automaton: as for LTL, a state will be a maximal set of consistent formulas in the closure, with the following definition of consistency.

Definition 87. *A set of formulas is consistent if the following constraints are met:*

- *it contains exactly one atomic proposition of the form δ_k;*
- *it contains all formulas $x.(x \sim c)$ for which $0 \sim c$ is true;*
- *it contains $x.$true and does not contain $x.$false;*
- *it does not contain a formula and its negation;*
- *it contains $x.(\phi_1 \vee \phi_2)$ if, and only if, it contains at least one of $x.\phi_1$ and $x.\phi_2$;*
- *it contains $x.(\phi_1 \wedge \phi_2)$ if, and only if, it contains both $x.\phi_1$ and $x.\phi_2$;*
- *it contains $x.(\phi_1 \,\mathbf{U}\, \phi_2)$ if, and only if, it contains $x.\mathbf{X}\,\phi_2$, or $x.\mathbf{X}\,\phi_1$ and $x.\mathbf{X}\,(\phi_1 \,\mathbf{U}\, \phi_2)$;*
- *it contains $x.z.\phi_1$ if, and only if, it contains $x.\phi_1[z \leftarrow x]$.*

A consistent set is maximal if it cannot be consistently extended. Maximal consistent sets contain either a subformula or its negation, so that there are at most $2^{4M\cdot|\phi|}$ such sets. This will be the state space of our automaton.

We now define the transitions of the automaton: there is a transition from state S to state S' if, and only if, for any formula of he form $x.\mathbf{X}\,\psi$ in the closure of ϕ, it holds

$$x.\mathbf{X}\,\psi \in S \qquad \Longleftrightarrow \qquad x.\psi^k \in S'$$

where k is the integer such that $\delta_k \in S'$. Such a transition is labelled with the set of atomic propositions appearing in S, together with the integer k such that $\delta_k \in S'$. This achieves what we have in mind: the transition from S to S' will have duration k, and all conditions that S imposes to S' (those of the form $z.\mathbf{X}\,\psi$) have to holds true in S', taking into account the fact that some delay has elapsed during the transition.

Notice that we do not require here that a delay transition stays in the same state: our construction allows "merged" transitions (representing the merge of a delay and an action transition) as well as separate delay and action transitions. The exact semantics will be enforced when we do the product of our automaton with the semantics of the timed automaton to be checked (which we only do symbolically).

It remains to define the acceptance condition. As for LTL, it is defined as a generalized Büchi condition imposing that, for any subformula of the form $\alpha \,\mathbf{U}\, \beta$, some state containing either $x.\neg(\alpha \,\mathbf{U}\, \beta)$ or $x.\beta$ has to be visited infinitely many times.

The correctness of our construction is expressed as follows:

Lemma 88. *Let ρ be a run of a timed automaton under discrete time (writing $(s_i)_{i\geq 0}$ for the sequence of states, and w_i for the duration of the transition from s_i to s_{i+1}). Then the word $(s_i, w_i)_{i\geq 0}$ is accepted by $\mathcal{A}_\phi$ from some state containing $x.\phi$ if, and only if, $\rho \models x.\phi$ (with initial clock valuation mapping all clocks to zero).*

The proof is similar as for LTL: first assume that there is a path in the automaton. Write (S_i) for the sequence of states visited along that accepting run. We prove that any formula appearing in S_i holds true at position i along ρ. Given the choice of the initial state, this will prove one direction of our result.

As the base case, it holds $s_i = S_i \cap \mathrm{AP}$, so that the above result holds for atomic propositions. The cases of formulas $z.z \sim c$ and of boolean combinations are straightforward. Until formulas are handled in the same way as for LTL. We now focus on subformulas involving clocks.

Consider a formula of the form $z.\mathbf{X}\,\phi_1$, assumed to be in S_i. Then $z.\phi_1^k$ is in S_{i+1}, where k is the integer such that $\delta_k \in S_{i+1}$. From the induction hypothesis, $\rho, i+1 \models z.\phi_1^k$ (the formula being closed, this holds for any clock valuation ζ). From Lemma 85, it holds $\rho, i+1 \models_{\zeta[z\leftarrow k]} \phi_1$, for any valuation ζ. It follows that $\rho, i \models_\zeta z.\mathbf{X}\,\phi_1$ for any ζ, as required.

Finally, assume that $z.x.\phi_1 \in S_i$. Then $z.\phi_1[x \leftarrow z]$ is in S_i, so that $\rho, i \models z.\phi_1[x \leftarrow z]$, which is equivalent to $\rho, i \models z.x.\phi_1$.

We now prove the converse implication. We write S_i for the set of formulas in the closure of ϕ that hold true at position i along ρ. These sets are maximally consistent. One can easily check that there is a transition between S_i and s_{i+1}, thanks to Lemma 85, and that the acceptance condition is fulfilled, which concludes our proof. $\square$

3.3 Exercises

Exercice 8 $\star$ In weighted games, prove that winning strategies for $p\,\mathbf{U}^{\leq i}_{\leq n}\,q$ might need memory (in other terms, there might be winning strategies but no memoryless winning strategies).

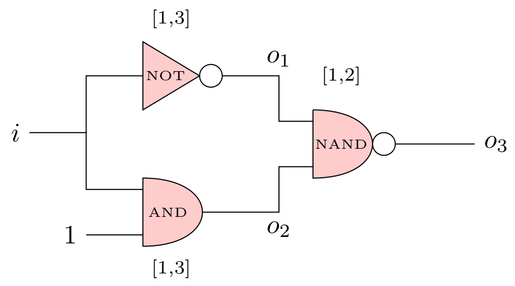
Exercice 9 $\star$ Prove that any weighted labelled transition system can be represented as a timed automaton with discrete-time semantics. Does it apply to any weighted labelled transition system? Is the translation possible for all the semantics we defined?

4. Timed temporal logics

4.1 Discrete *vs* dense time

We illustrate the usefulness of dense time with an example of digital circuits [BS91]. Consider the circuit described on Fig. 18, where the intervals decorating each gate indicate the delay

Figure 18: Example of a circuit



needed for this gate to 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]).

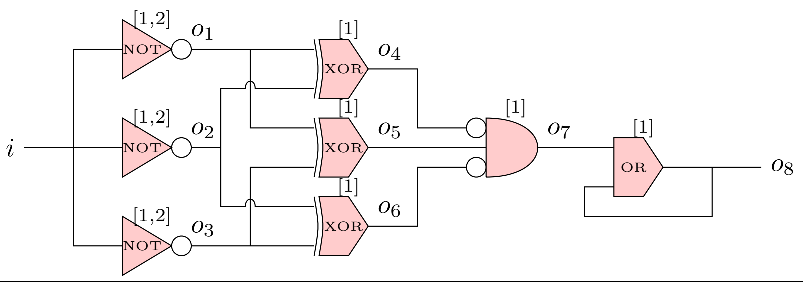
When i turns to 1, one possible sequence of values (before stabilization) is

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

the last set of values corresponding to the stable configuration.

Now, consider the (more complex) circuit depicted on Fig. 19 (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

Figure 19: A circuit that is not 1-discretizable



discrete-time does *not* 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. 19 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.

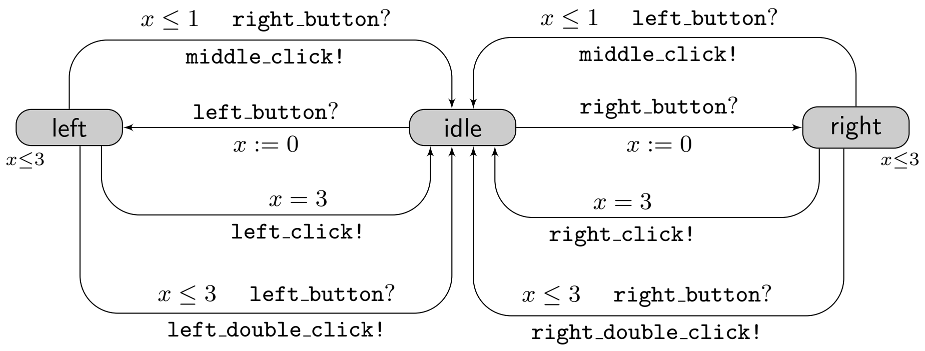
4.2 Timed automata

We reuse the definitions of Section 3.2, but with $\mathbb{R}_{\geq 0}$ as the timeline (hence clock valuations take values in $\mathbb{R}_{\geq 0}$). We recall the syntax of timed automata:

Definition 89. *Let AP be a finite set of atomic propositions. A* timed automaton *[AD94] over AP is a 5-tuple $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \mathsf{Inv}\rangle$ where*

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

Figure 20: *Clicks* and *double-clicks* of a mouse



Example. *Fig. 20 depicts a timed automaton representing the behaviour of a computer mouse when a user hits te left- or right buttons: depending on the delays between the actions, the mouse interprets them as either simple clicks, or as double-clicks. We also encode middle-button clicks, when the two buttons are pressed at almost the same time.*

As for the models we defined in the previous sections, the semantics of timed automata can be defined in terms of labelled transition systems:

Definition 90. *Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \mathsf{Inv}\rangle$ be a timed automaton over AP and alphabet Σ. The* semantics *of $\mathcal{A}$, denoted with $[\![\mathcal{A}]\!]$, is the (infinite-state) weighted labelled transition system $[\![\mathcal{A}]\!] = \langle Q, R, l\rangle$ where*

- *$Q = \{(s, \nu) \in S \times \mathbb{T}^{\mathfrak{C}} \mid \nu \models \mathsf{Inv}(s)\}$;*
- *R is the union of two sets of transitions:*
  - delay transitions*: $((s, \nu), d, (s', \nu')) \in R$ iff (s, ν) and (s', ν') belong to Q, $s = s'$, and $d \in \mathbb{R}_{>0}$ is s.t. $\nu' = \nu + d$;*
  - action transitions*: $((s, \nu), 0, (s', \nu')) \in R$ iff (s, ν) and (s', ν') belong to Q and there exists $t = (s, g, \sigma, r, s') \in T$ s.t. $\nu \models g$, $\nu' = \nu[r \to 0]$;*
- *$l((s, \nu)) = \ell(s)$ for all $(s, \nu) \in Q$.*

Following this definition, the notion of a run in a (continuous-time) timed automaton follows.

4.3 Deciding reachability

As usual, we begin with solving the most basic problem, namely reachability. This is well-known to be decidable, using the *region abstraction* of [AD94].

Definition 91. *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 $R \cap [\![g]\!]_{\mathfrak{C}} = \varnothing$, or $R \subseteq [\![g]\!]_{\mathfrak{C}}$;*
- *for every $R, R' \in \mathcal{R}$ and every $Y \subseteq \mathfrak{C}$, letting $R[Y \to 0] = \{v[Y \to 0] \mid v \in R\}$, it holds either $R[Y \to 0] \cap R' = \varnothing$ or $R[Y \to 0] \subseteq R'$.*

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 92. *Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \mathsf{Inv}\rangle$ be a timed automaton, $[\![\mathcal{A}]\!] = \langle Q, R, l\rangle$ 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*

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

is a bisimulation relation on $[\![\mathcal{A}]\!]$.

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

Let (s, ν) and (s', ν') be two states of $[\![\mathcal{A}]\!]$. 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 $[\![\mathcal{A}]\!]$. 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, $\mu\models\mathsf{Inv}(s)$. By definition of $\mathcal{R}$ being compatible with $\mathcal{G}$, this entails that
  - for any $\rho\in[\![\nu]\!]$, there exists $d'\in\mathbb{T}$ s.t. $\rho+d'\in[\![\mu]\!]$,
  - $[\![\mu]\!]\subseteq[\![\mathsf{Inv}(s)]\!]$.

  Since $(s,\nu)\simeq_{\mathcal{R}}(s',\nu')$, we also have $s'=s$ and $\nu'\in[\![\nu]\!]$. Hence for some $d'\in\mathbb{T}$, it holds $\nu'+d'\in[\![\mu]\!]$. Letting $t'=s$ and $\mu'=\nu'+d'$, we have $((s',\nu'),(t',\mu'))\in R$ (because $\mu'\models\mathsf{Inv}(s)$) 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 $\mu=\nu[R\to 0]\models\mathsf{Inv}(t)$. Since $(s,\nu)\simeq_{\mathcal{R}}(s',\nu')$, we have $s'=s$ and $\nu'\in[\![\nu]\!]$. Since $\mathcal{R}$ is compatible with $\mathcal{G}$ and $\nu\models g$, it holds $[\![\nu]\!]\subseteq[\![g]\!]$, hence $\nu'\models g$. Let $t'=t$, and $\mu'=\nu'[R\to 0]$. We have $\mu=\nu[R\to 0]$, meaning that $[\![\nu]\!][R\to 0]\cap[\![\mu]\!]\neq\varnothing$. by definition of $\mathcal{R}$ being compatible with $\mathcal{G}$, this entails that $[\![\nu]\!][R\to 0]\subseteq[\![\mu]\!]$. Since $\nu'\in[\![\nu]\!]$, we deduce that $\mu'=\nu'[R\to 0]\in[\![\mu]\!]$. Hence $\mu'\models\mathsf{Inv}(t)$, so that there exists a transition $((s',\nu'),(t',\mu'))\in R$ and $(t,\mu)\simeq_{\mathcal{R}}(t',\mu')$.

□

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 93. *Let $\mathcal{A}=\langle S,\mathfrak{C},T,\ell,\mathit{Inv}\rangle$ 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 94. *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. $\nu(c)\le M_c$ (hence also $\nu'(c)\le M_c$), 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 $\nu(c)\le M_c$ and $\nu(c')\le M_c$ (hence also $\nu'(c)\le M_c$ and $\nu'(c')\le M_{c'}$), it holds $\langle\nu(c)\rangle\le\langle\nu(c')\rangle$ iff $\langle\nu'(c)\rangle\le\langle\nu'(c')\rangle$.*

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 $\mathcal{G}^{\mathbf{M}}_{\mathfrak{C}}=\{c\sim n_c\mid c\in\mathfrak{C},\sim\in\{<,\le,=,\ge,>\},0\le n_c\le M_c\}$. 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 $\mathcal{G}^{\mathbf{M}}_{\mathfrak{C}}$, and ν be a valuation. Assume that $[\![\nu]\!]\cap[\![g]\!]\neq\varnothing$: there is a valuation $\mu\in[\![\nu]\!]$ s.t. $\mu(c)\sim n$. Pick any other valuation $\mu'\in[\![\nu]\!]$: if $\mu'(c)>M_c$, then also $\mu(c)>M_c$. Since $n\le M_c$, it must be the case that $\sim\in\{>,\ge\}$, hence $\mu'(c)\sim n$.

  Otherwise, we have $\mu'(c)\le M_c$, and $\mu(c)\le M_c$. 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 $[\![\nu]\!][R\to 0]\cap[\![\nu']\!]\neq\varnothing$. Pick a valuation $\mu\in[\![\nu]\!][R\to 0]$.

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

  - for $r \in R$, it must be the case that $\nu'(r) = 0$, and also $\mu[R \to 0](r) = 0$, so that none of them is strictly more than M_r. For a clock $c \notin R$, $\mu[R \to 0](c) > M_c$ iff $\mu(c) > M_c$ iff $\rho(c) > M_c$ iff $\rho[R \to 0](c) > M_c$ iff $\nu'(c) > M_c$.
  - the second property for clocks in R is obvious. For clocks not in R, $\lfloor \mu[R \to 0](c) \rfloor = \lfloor \mu(c) \rfloor = \lfloor \rho(c) \rfloor = \lfloor \rho[R \to 0](c) \rfloor = \lfloor \nu'(c) \rfloor$, and $\langle \mu[R \to 0](c) \rangle = 0$ iff $\langle \mu(c) \rangle = 0$ iff $\langle \rho(c) \rangle = 0$ iff $\langle \rho[R \to 0](c) \rangle = 0$ iff $\langle \nu'(c) \rangle = 0$.
  - finally, for two clocks c and c' s.t. $\mu[R \to 0](c) \leq M_c$ and $\mu[R \to 0](c') \leq M_{c'}$ and $\langle \mu[R \to 0](c) \rangle \leq \langle \mu[R \to 0](c') \rangle$, we again have several cases:
    * if c and c' are in R, then $\langle \rho[R \to 0](c) \rangle \leq \langle \rho[R \to 0](c') \rangle$, and similarly for ν'.
    * if $c \in R$ and $c' \notin R$, then $\langle \rho[R \to 0](c) \rangle = 0$, 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 $\langle \rho[R \to 0](c) \rangle \leq \langle \rho[R \to 0](c') \rangle$, 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 $[\![\nu]\!]$ and $[\![\nu']\!]$ for the corresponding equivalence classes. Assume that for some $\mu \in [\![\nu]\!]$, there is a $t \in \mathbb{T}$ s.t. $\mu + t \in [\![\nu']\!]$. Pick $\mu' \in [\![\nu]\!]$: we exhibit a $t' \in \mathbb{T}$ s.t. $\mu' + t' \in [\![\nu']\!]$ 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 $[\![\nu]\!] = [\![\nu']\!]$. Then for any $t' \in \mathbb{T}$, $(\mu' + t')(c) > M_c$ for all $c \in \mathfrak{C}$, which entails $\mu' + t' \in [\![\nu']\!]$.

- 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 $\mu' + t' \in [\![\nu']\!]$, which we leave to the reader.

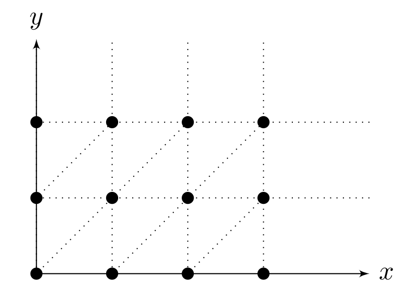
- finally, if for some $c \in \mathfrak{C}$, we have that $(\mu + t)(c) \leq M_c$, but no such clock is mapped to an integer value by $\mu + t$, then we distinguish two cases:

  - 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 $[\![\mu + t]\!] = [\![\mu]\!]$, 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 $[\![\mu + u]\!] = [\![\mu' + u']\!]$. We are now left with the case where we depart from a region $[\![\mu]\!]$ 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. □

The figure on the right is a schematic representation of **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 21: **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 95. *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. □

Definition 96. *Let* $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \mathsf{Inv} \rangle$ *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* $[\![\mathcal{A}]\!]$*, with* $\nu \in r$ *and* $\nu' \in r'$*;*
- $l(s, r) = \ell(s)$ *for all* $s \in S$ *and* $r \in \mathcal{R}$*.*

Proposition 97. *Let* $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \mathsf{Inv} \rangle$ *be a timed automaton, and* $\mathcal{B}$ *be its region automaton. Let* (s, ν) *be a state of* $[\![\mathcal{A}]\!]$*, and* $(s, [\![\nu]\!])$ *be the corresponding state in* $\mathcal{B}$*. Then there exists a bisimulation (over the union of* $[\![\mathcal{A}]\!]$ *and* $\mathcal{B}$*) containing* $((s, \nu), (s, [\![\nu]\!]))$*.*

Proof. Consider the set

$$R = \{((t, \mu), (t, [\![\mu]\!])) \mid \mu \models \mathsf{Inv}(t)\}.$$

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

Theorem 98. *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 $S \colon \mathbb{N} \to \mathbb{N}$ 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 $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \mathsf{Inv}\rangle$ s.t.

- $S = Q \times \{1, ..., S(|w|)\} \times \{0, ..., 2 \times S(|w|)\}$, 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);
- $\mathfrak{C} = \{x_i \mid i \in \{1, ..., S(|w|)\} \cup \{t\}$. 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 $i \in \{2, ..., S(|w|)\}$. 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 bisimulation[5] 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 weighted labelled transition systems, since we heavily use clocks here—reachability in (weighted) labelled transition systems is NLOGSPACE-complete).

4.4 Timed temporal logics

Syntactically, our logics will precisely correspond to the logics we defined for the discrete-time semantics of timed automata, namely TCTL, TCTL$_c$, MTL and TPTL. We will also define MITL in the sequel, which is a syntactic fragment of MTL. Because we have two different semantics for runs of a timed automaton, we define two semantics for our temporal logics.

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

4.4.1 Timed-word semantics

Based on the semantics defined at Definition 90, we would naturally define the set of runs of the timed automaton as the set of runs of its semantics. This however is not adequate for our purposes: indeed, from any state where a positive delay can elapse, there is an infinite run visiting only that state: it suffices to take only delay transitions whose durations sum up to some total value compatible with the invariant in that state. Such behaviours are called *Zeno runs*. We'd better try to avoid them as much as possible, as they correspond to runs along which time converges, which has no real practical meaning.

Instead, a run in a timed automaton is a run in its underlying weighted labelled transition system in which no two consecutive delay transitions are allowed. In the sequel, we consider a slight variant, consisting in merging delay- and action transitions (even if it means allowing delays of duration zero). This avoids observing each location twice, as is the case when individual delay transitions are allowed.

The timed-word semantics of TCTL, TCTL_c, MTL and TPTL are defined for continuous-time timed automata in the very same way as we defined them for discrete-time timed automata, using the weighted labelled transition systems defining their semantics. The only difference lies in the fact that timestamps are non-negative reals instead of integers.

4.4.2 Timed-signal semantics

A run in the above semantics can then be seen as a sequence of "timed states": we go form one timed state to the next one by applying a delay transition followed with an action transition. Such a timed-state sequence can equivalently be seen as a timed word. It might look surprising that, even though we have moved to *continuous time*, we are still left with discrete behaviours. This is due to our moving to transition systems, whose semantics is discrete. While the above semantics does make sense, we also consider a different, really continuous one.

Definition 99. *Let $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \mathit{Inv}\rangle$ be a timed automaton over AP and alphabet Σ. The set of* continuous runs *(or* signals*) of $\mathcal{A}$ is the set of functions f mapping $\mathbb{R}_{\geq 0}$ to S such that there exists a run in the semantics of $\mathcal{A}$, seen as a timed-state sequence $(s_i, t_i)_{i\in\mathbb{N}}$, such that $f(t) = s_k$ whenever $t_k \leq t < t_{k+1}$.*

Of course, for our temporal-logic purposes, we will most often see such signals as maps from $\mathbb{R}_{\geq 0}$ to 2^{AP}, in the obvious way.

With this definition, runs of a timed automaton still have countably many transitions, but they have a continuous nature. We will see later how this affects expressiveness of timed temporal logics, which we now define. Notice that we could have gone one step further and allowed arbitrarily many transitions along a signal (instead of countably many). We could also have allowed to visit several states at the same time, hence modelling zero-delay transitions. We leave these technical details to the interested reader.

As signals become our notion of "words" in the continuous setting, we have to define preifxes and suffixes of signals, in the obvious way: in particular, given $t \in \mathbb{R}_{\geq 0}$ and a signal ρ, the suffix of ρ after t is the signal $\sigma_{\geq t}$ defined by $\sigma_{\geq t}(t') = \sigma(t + t')$.

We now define the semantics of TPTL (that of MTL will follow thanks to the natural translation into TPTL) over timed signals. We will not extend this definition to branching-time temporal logics, as it would require defining trees over the reals.

Let ρ be a timed signal. The timed-signal semantics of TPTL is defined as follows:

$$\begin{aligned}
\rho \models_\zeta p \quad &\Leftrightarrow \quad p \in \ell(s_0)\\
\rho \models_\zeta c \sim n \quad &\Leftrightarrow \quad \zeta(c) \sim n\\
\rho \models_\zeta c.\phi \quad &\Leftrightarrow \quad \rho \models_{\zeta[c\leftarrow 0]} \phi\\
\rho \models_\zeta \neg\phi \quad &\Leftrightarrow \quad \rho \not\models_\zeta \phi\\
\rho \models_\zeta \phi \vee \phi' \quad &\Leftrightarrow \quad \rho \models_\zeta \phi \text{ or } \rho \models_\zeta \phi'\\
\rho \models_\zeta \phi \,\mathbf{U}\, \phi' \quad &\Leftrightarrow \quad \exists t > 0 \text{ s.t. } \rho_{\geq t} \models_{\zeta_t} \phi' \text{ and } \forall u \in (0,t).\ \rho_{\geq u} \models_{\zeta_u} \phi,\\
&\qquad\qquad \text{where } \zeta_t = \zeta + t.
\end{aligned}$$

4.5 Expressiveness questions

One natural question one may ask is whether, in the continuous-time case, formula clock do bring more expressiveness to temporal logics. The answer has been open for a long time, with the following formula as a possible witness that TPTL is strictly more expressive than MTL:

$$x.\mathbf{F}\,(a \wedge \mathbf{F}\,(b \wedge x \leq 2)).$$

This formula states that we will have an a followed with a b, with the latter occurring less than two time units from the initial time. Naive attempts to express this property in MTL include $\mathbf{F}_{\leq 1}\,(a \wedge \mathbf{F}_{\leq 1}\,b)$ and $\mathbf{F}\,(a \wedge \mathbf{F}_{\leq 2}\,b)$, which obviously is not equivalent to the original formula.

It turns out that the above formula *can* be expressed in MTL only in the timed-signal semantics, using the following equivalent formula:

$$\mathbf{F}_{\leq 1}\,(a \wedge \mathbf{F}_{\leq 1}\,b) \quad \vee \quad \mathbf{F}_{<1}\,a \wedge \mathbf{F}_{=1}\,\mathbf{F}_{\leq 1}\,b \quad \vee \quad \mathbf{F}_{\leq 1}\,(\mathbf{F}_{<1}\,a \wedge \mathbf{F}_{=1}\,b)$$

Why would this disjunction be equivalent to the original formula? It distinguishes three cases: either both events a and b occurs within the first time unit (which is over-approximated by the first disjunct), or a occurs in the first time unit and b in the second one (second disjunct), or both occur in the second half (in which case we consider the time point lying one time unit before b). Because of the latter conjunct, the translation is only correct under the signal semantics: it requires imposing a formula at a time point where no event occurs.

When considering the timed-word semantics, it can be proved that no MTL formula can express the above property. The prove is quite involved, and out of the scope of these notes. Under the signal semantics, a more complex formula can be used to prove that TPTL is also more expressive than MTL.

Theorem 100. *In continuous time,* TPTL *is strictly more expressiven than* MTL*.*

4.6 Model checking timed temporal logics

4.6.1 TCTL model checking

As usual, the main ingredient in the model-checking algorithm for TCTL is a kind of labelling of the states with the subformulas they satisfy. In the timed setting, however, there are infinitely many states, and the "states" to be considered are in fact the regions. Hence we have to prove that two equivalent valuations satisfy the same TCTL formulas. We prove this in the context of logics with formula clocks, which is stronger but requires more technicalities.

Lemma 101. *Let $\mathcal{A}$ be a timed automaton, with set of clocks $\mathfrak{C}$. Let X be a disjoint finite set of clocks (intended to be formula clocks), and extend the set clocks of clocks of $\mathcal{A}$ with X (this is just syntactic, and we require that clocks in X are never used in guards and never reset). Fix some maximal constants for each of these extra clocks. Now, pick two equivalent valuations ν and ν' of clocks in $\mathfrak{C} \cup X$, and two runs ρ and ρ' starting at (s,ν) and (s,ν') and visiting the same sequences of regions. Then for any formula $\phi \in \mathsf{TBTL}_c$ involving clocks in X that are only compared to integers less than or equal to their associated maximal constants,*

$$\rho \models_\nu \phi \quad \iff \quad \rho' \models_{\nu'} \phi.$$

Proof. We prove this result by induction on ϕ. For atomic propositions, the result is straightforward as we start from the same state. For clock constraints, the result follows from the fact that ν and ν' are region equivalent. For boolean combinations of subformulas, the result follows by induction.

Now, assume $\phi = \mathbf{E}\psi$. Then $\rho \models_\nu \phi$ implies the existence of a run π from (s,ν) s.t. $\pi \models_\nu \psi$. It then suffices to pick a path π' from (s,ν') visiting the same regions as π, so that by induction we get $\pi' \models_{\nu'} \psi$, which proves our result.

For the case where $\phi = x.\psi$, we may assume w.l.o.g. that ψ begins with a path quantifier. So we assume $\rho \models_\nu x.\,\mathbf{E}\psi'$. This means that there is a path π from $(s,\nu[x \leftarrow 0])$ along which

ψ' holds. Now, pick a path π' from $(s, \nu'[x \leftarrow 0])$ visiting the same sequence of regions as π. Following the induction hypothesis, $\pi' \models_{\nu'[x\leftarrow]} \psi'$, so that $\rho' \models_{\nu'} x.\mathbf{E}\psi'$, as expected.

Finally, we handle the case of the "until" modality. If $\rho \models_\nu \alpha \,\mathbf{U}\, \beta$, then for some delay t, $\rho_{\geq t} \models_{\nu+t} \beta$ and for all intermediary position u, $\rho_{\geq u} \models_{\nu+u} \alpha$. Since ρ' visits the same sequence of regions as ρ, there is a delay t' such that $\nu + t$ and $\nu' + t'$ end up in the same region, and such that for all intermediary position u', there is some $0 < u < t$ such that $\nu' + u'$ is in the same region as $\nu + u$. Applying the induction hypothesis, this proves that $\rho' \models_{\nu'} \alpha \,\mathbf{U}\, \beta$. ☐

Theorem 102 ([ACD93]). *TCTL and TBTL_c model-checking on timed automata can be achieved in exponential time, and are PSPACE-complete.*

Proof. Following our previous lemma, the algorithm exponential-time algorithm consists in labelling regions (extended with formula clocks) with the subformulas they satisfy. This amounts to applying the CTL model-checking algorithm on the exponential-size region automaton.

In order to make the algorithm cope with the polynomial-space restriction, we make it run on-the-fly, checking subformulas on demand and guessing paths in the region automaton for witnessing path-quantifier formulas.

Hardness is easy since reachability in timed automata is already PSPACE-complete. ☐

4.7 MTL model checking

Unfortunately, our series of nice decidability results stops here: with MTL and continuous time, we have hit the border. As we will see later, decidability can be recovered in some circumstances.

Theorem 103 ([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: 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:

$$t \wedge \neg a_1 \wedge \neg a_2 \wedge \mathbf{G}\,(t \iff (\neg t)\,\mathbf{U}_{=1}\,(t \wedge \neg a_1 \wedge \neg a_2)) \tag{4}$$

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

$$\mathbf{G}\,\Big(\bigvee_{q\in Q} q \wedge \bigwedge_{q\neq q'} (\neg q \vee \neg q')\Big) \tag{5}$$

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

$$\bigwedge_{q\in Q} \mathbf{G}\,(q \Rightarrow (q \wedge \neg t)\,\mathbf{U}\,t) \tag{6}$$

- a_1 and a_2 do not overlap:
$$\mathbf{G}\,(\neg a_1 \vee \neg a_2) \tag{7}$$

- occurrences and absences of a_1 (resp. a_2), except the last ones, are repeated one time unit later:
$$\mathbf{G}\left[(a_1 \wedge \neg(a_1\,\mathbf{U}\,(\neg a_1 \wedge \neg a_1\,\mathbf{U}\,t))) \Rightarrow \mathbf{F}_{=1}\,a_1\right] \wedge \mathbf{G}\left[(\neg a_1 \wedge \neg(\neg a_1\,\mathbf{U}\,t)) \Rightarrow \mathbf{F}_{=1}\,\neg a_1\right] \tag{8}$$
The same formula can be written for a_2;

- transitions are applied correctly: assume that there is a transition of the form
```
q:  if c₁>0 then c₁:=c₁-1; goto q' else goto q''
```
This could be encoded as follows:
$$\begin{aligned}\mathbf{G}\Big[(t \wedge q \wedge (\neg t\,\mathbf{U}\,a_1)) \Rightarrow (&\mathbf{F}_{=1}\,q' \wedge \\ &\mathbf{F}_{<1}\,(\neg a_1 \wedge (\neg a_1 \wedge \neg t)\,\mathbf{U}\,[a_1 \wedge (a_1 \wedge \neg t)\,\mathbf{U}\,(\neg a_1 \wedge (\neg a_1 \wedge \neg t)\,\mathbf{U}\,t)] \wedge \\ &\mathbf{F}_{=1}\,(\neg a_1 \wedge \neg a_1\,\mathbf{U}\,t)) \wedge \mathbf{G}_{<1}\,(a_2 \Leftrightarrow \mathbf{F}_{=1}\,a_2))\Big]\end{aligned} \tag{9}$$
and
$$\mathbf{G}\Big[(t \wedge q \wedge (\neg a_1\,\mathbf{U}\,t)) \Rightarrow (\mathbf{F}_{=1}\,q'' \wedge \mathbf{G}_{<1}\,(a_1 \iff \mathbf{F}_{=1}\,a_1) \wedge \mathbf{G}_{<1}\,(a_2 \iff \mathbf{F}_{=1}\,a_2))\Big] \tag{10}$$
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 a 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.

In the end, the three-counter machine does not halt if, and only if, there is a computation visiting infinitely many configurations containing only y's. □

Hopefully, when dropping timing constraints from the logic, we recover decidability:

Theorem 104. 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. □

In the undecidability proof for MTL model checking, it clearly appears that punctuality is the key ingredient: we use it for copying configurations of the two-counter machine from one time unit to the next one. And indeed, when relaxing punctuality, we get:

Theorem 105. *Write* MITL *for the fragment of* MTL *where constraining intervals are required to be non-singular. The model-checking problem for* MITL *on timed automata is* EXPSPACE*-complete.*

Proof. The full proof of this result is very complex (see [AFH96]), but we try to convey the main ideas here. The general idea is to associate with a formula ϕ of MITL, a timed automaton $\mathcal{B}_\phi$ accepting precisely the timed state sequences satisfying ϕ. The construction of this automaton roughly follows the construction of the Büchi automaton for LTL. For a formula of the form $\mathbf{F}_I\, p$, the automaton will set a clock to zero, and store in its control state that it has to visit a p-state when the value of this clock is in I. At first sight, this might require infinitely many clocks: if checking $\mathbf{G}\,(q \Rightarrow \mathbf{F}_I\, p)$, we have to start a new clock each time we see a q-state. This is where the fact that I is non-punctual will help: several q-states will use the same p-state as a witness, hence the same clock.

As a first step, we simplify our input formula, turning it into some normal form, where it only involves atomic propositions and their negations, conjunction and disjunction, and the following six constructs:

- $\mathbf{F}_I\, \psi$ where $I = (0, b\rangle$ for some finite b;
- $\mathbf{G}_I\, \psi$ where $I = (0, b\rangle$ for some finite b;
- $\psi_1\, \mathbf{U}_I\, \psi_2$ where $I = \langle a, b\rangle$ with $0 < a < b < \infty$;
- $\psi_1\, \mathbf{R}_I\, \psi_2$ where $I = \langle a, b\rangle$ with $0 < a < b < \infty$;

- $\psi_1 \mathbf{U} \psi_2$;
- $\mathbf{G} \psi$.

In order to come to such a form, we apply a series of transformations:

- unbounded intervals are only $(0, \infty)$: this is achieved by the following equivalences:

$$\alpha \mathbf{U}_{(a,\infty)} \beta \equiv \mathbf{G}_{(0,a]} (\alpha \wedge \alpha \mathbf{U} \beta)$$
$$\alpha \mathbf{U}_{[a,\infty)} \beta \equiv \mathbf{G}_{(0,a)} \alpha \wedge \mathbf{G}_{(0,a]} (\beta \vee (\alpha \wedge \alpha \mathbf{U} \beta))$$

- bounded intervals with left end-point zero only label unary modalities: for this we do as follows:

$$\alpha \mathbf{U}_{(0,b\rangle} \beta \equiv \mathbf{F}_{(0,b\rangle} \beta \wedge \alpha \mathbf{U} \beta$$
$$\alpha \mathbf{R}_{(0,b\rangle} \beta \equiv \mathbf{G}_{(0,b\rangle} \beta \vee \alpha \mathbf{R} \beta$$

- eliminate unconstrained “release” modalities: we use

$$\alpha \mathbf{R} \beta \equiv \mathbf{G} \beta \vee \beta \mathbf{U} (\alpha \wedge \beta).$$

Notice that during this transformations, the DAG-size of the formula (*i.e.*, its number of distinct subformulas) did not increase by more than a linear polynomial, and the maximal constant is unchanged.

Now, the states of our automaton will be the consistent sets of subformulas of the input formula (assumed to be in normal form). The states will also contain a number of clock constraints for the eventualities that are about to be fulfilled.

- When the automaton enters a state containing a formula of the form $\zeta = \mathbf{F}_I \psi$ (where $I = (0, b\rangle$ for some finite b), it will set a clock x_ζ of zero. Formula ζ will appear in the state as long as the eventuality has not been fulfilled. If, at some point before the eventuality is fulfilled, formula ζ has to be checked again, then we will not reset clock x_ζ: indeed, we are already expecting a witnessing ψ-state, which will serve as a witness for both occurrences. This way, as long as x_ζ is “running”, all other occurrences of ζ will automatically be fulfilled.

- The situation of $\zeta = \mathbf{G}_I \psi$ is symmetric: clock x_ζ is reset to zero each time the formula has to be enforced, and ψ has to hold true as long as clock x_ζ is in I. Again, on ly one clock is sufficient for the whole automaton.

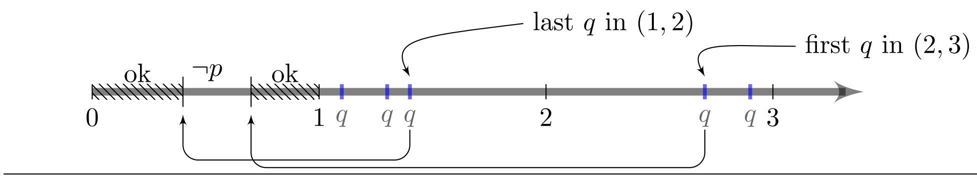
- Formulas for the form $\alpha \mathbf{U}_{\langle a,b\rangle} \beta$, with $0 < a < b < \infty$, are the most difficult to handle.

  Consider formula $\mathbf{G}_{(0,1)} (p \Rightarrow \mathbf{F}_{[1,2]} q)$. The naive approach of reseting one clock each time a p-state is encountered does not work, as it requires unboundedly many clocks. Instead, the idea consists in guessing, for the whole interval $(0, 1)$, the dates t_1 and t_2 such that the first q-state in $(1, 2)$ occurs at $t_1 + 1$, and the last q-state in $(2, 3)$ occurs at $t_2 + 2$ (these are assumed to exist; we also assume that q is false at time 2, as otherwise the formula is automatically true). Then any p-state that occurs before t_1 in $(0, 1)$ will have a corresponding q-state after some delay in $[1, 2]$, and so will any p-state that occurs after t_2 in $(0, 1)$. When $t_1 \geq t_2$, then the whole interval $(0, 1)$ is covered; otherwise, the automaton has to ensure that no p-state occurs between t_1 and t_2 (besides checking that the guessed values for t_1 and t_2 are correct, of course).

  In case q holds true on an *open* interval $(t_1 + 1, t'_1 + 1)$, the situation is a bit more complex: there is no last point where q holds, and we would then have to guess both t_1 and t'_1. The other arguments are similar.

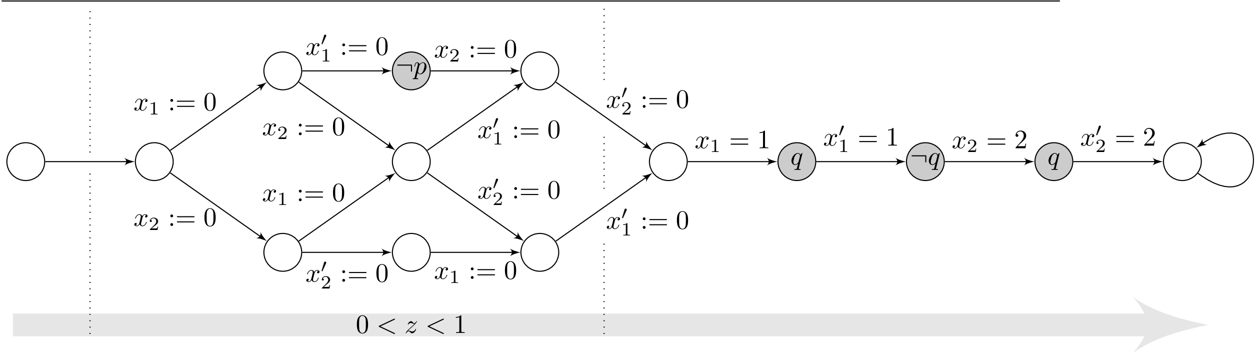
  In the end, for the formula above, the automaton would like the one depicted on Fig. 23. In this automaton, we start clocks x_1 and x'_1 at guessed dates t_1 and t'_1, and clocks x_2 and x'_2 at dates t_2 and t'_2; we impose that p has to be false between t'_1 and t_2,

Figure 22: Witnessing $\mathbf{G}_{(0,1)}\,(p \Rightarrow \mathbf{F}_{[1,2]}\,q)$ with only two occurrences of q



when it is the case that $t'_1 < t_2$. We then check that the guessed dates are indeed correct. Notice that the automaton uses an extra clock z to enforce that clocks are reset before time 1. Notice also that there are special cases that should be handled separately: cases where q never holds true in $[1,2]$ or in $[2,3]$, and the case where q holds at time 2. These are easy to handle.

Figure 23: A timed automaton capturing $\mathbf{G}_{(0,1)}\,(p \Rightarrow \mathbf{F}_{[1,2]}\,q)$



The details for handling the general case are a bit tedious, but follow the same idea: each "until" formula uses four clock per unit-length interval, and we can reuse clocks one they exceed the upper bound b of the constraining interval of the formula being checked. The automaton will then have to handle all subformaulas at the same time, in quite the same manner as the automaton for an LTL formula propagates the obligations to be checked from one state to the next one.

- The case of formulas of the form $\alpha\,\mathbf{R}_I\,\beta$ is handled in a very similar way, and we omit it.

In the end, our automaton has size doubly-exponential, since states are subsets of subformulas and clock constraints, and there can be as many as $4M \cdot |\phi|$ clock constraints. The model-checking algorithm then checks for the existence of a run in the product of this automaton with the input timed automaton. Notice that this does not require building the whole automaton $\mathcal{B}_\phi$, but instead can be achieved on-the-fly, using only exponential space. □

Remark. *In case the constraining intervals are of the form $\langle 0, a\rangle$ or $\langle a, +\infty)$, the construction above becomes much simple: we completely get rid of formulas of the form $\psi_1\,\mathbf{U}_I\,\psi_2$ and $\psi_1\,\mathbf{R}_I\,\psi_2$, so that we now only need one clock per subformula (instead of $4M$, where M is the maximal constant). The resulting algorithm runs in polynomial space.*

Remark. *Recently, it has been proved that banning punctual intervals is not the only way to recover decidability of model checking. Using very different techniques, it has been proved that* FlatMTL *(existential) model checking is also decidable: in* FlatMTL*, subformulas of the form $\alpha\,\mathbf{U}_I\,\beta$ are allowed if either I is bounded, or $\alpha \in$* MITL *(and symmetrically, $\alpha\,\mathbf{R}_I\,\beta$ is allowed if I is bounded or β is in* MITL*). In other terms, this requires that punctual constraints are only imposed for a bounded amount of time [BMOW07].*

5. Weighted timed automata

5.1 Timed automata with linear observers

Definition 106 ([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* $\mathcal{W} = \langle S, \mathfrak{C}, T, \ell, \mathsf{Inv}, \mathsf{Var}, \mathsf{rate}, \mathsf{upd}\rangle$ *where*

- $\mathcal{A} = \langle S, \mathfrak{C}, T, \ell, \mathsf{Inv}\rangle$ *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*);*
- $\mathsf{rate}\colon S \to \mathbb{R}^{\mathsf{Var}}$ *indicates the derivative of the observer variables in each state.*
- $\mathsf{upd}\colon T \to \mathbb{R}^{\mathsf{Var}}$ *associates with each transition the values to be added to each observer upon firing this transition;*

Definition 107. *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 90. The semantics of* $\mathcal{W}$ *is the infinite-state weighted automaton* $\langle V, E, \lambda\rangle$ *where*

- $V = \{(s, \nu, \mu) \in S \times \mathbb{T}^{\mathfrak{C}} \times \mathbb{R}^{\mathsf{Var}} \mid (s, \nu) \in Q\}$*;*
- $E \subseteq V \times \mathbb{R}^{\mathsf{Var}} \times V$ *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.*
    * $\nu' = \nu + d$*;*
    * $c = \mathsf{rate}(s) \times d$*;*
    * $\mu' = \mu + c$*.*
  - 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.* $\nu \models g$*,* $\nu' = \nu[r \to 0]$*,* $c = \mathsf{upd}(t)$ *and* $\mu' = \mu + c$*.*
- λ *labels a each state* (s, ν, μ) *following* $\ell(s)$*.*

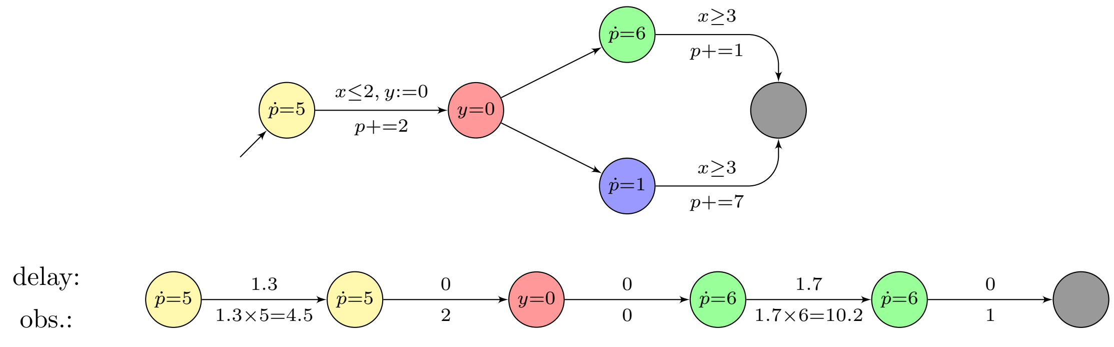
Remark. *It must be noted that the existence of a transition never depends on* μ *in the semantics above: contrary to hybrid systems, and as their name indicates, observers are just there to observe the evolution of some quantities along the executions of the underlying timed automaton, without modifying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

Example. *Fig. 24 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. 24, 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

Figure 24: Example of a timed automaton with linear observers



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

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

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.

5.2 Optimal reachability

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 108.

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 $(s,\mathbf{0}_{\mathfrak{C}},0)$ to (s',ν',μ') in $\mathcal{W}$ along which the observer variable increases by at most m?*

Theorem 109. *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 110. *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 111. *The* corner-point abstraction *of a* TALO *is the weighted graph $\langle Q,R,l\rangle$ where*

- *$Q=\{(s,r,\nu)\mid(r,\nu)$ is a corner point$\}$;*
- *$R\colon Q\times\mathbb{R}^{\mathcal{V}}\times Q$ 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 $c = \textsf{rate}(s)$;*
  - *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 $\textsf{upd}(t) = c$, $r \subseteq g$ and $\nu' = \nu[r \to 0]$.*

- *the labelling function follows that of the* TALO*.*

Lemma 112. *Let f be a function defined on a compact convex set $A \subset \mathbb{R}^n$ of the form*

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

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 $\textsf{Cl}(A)$ (the topological closure of 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 $g_\alpha(x_2, \ldots, x_n) = f(\alpha, x_2, \ldots, x_n)$, 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 $(x_2, \ldots, x_n)$ is in the border of A', then $(\alpha, x_2, \ldots, x_n)$ is in the border of A. That $(x_2, \ldots, x_n)$ is in the border of A' means that for all $\varepsilon > 0$, there is a point $(y_2, \ldots, y_n)$ not in A' and with $\|(x_2, \ldots, x_n) - (y_2, \ldots, y_n)\|_\infty < \varepsilon$. Now, $(\alpha, y_2, \ldots, y_n) \notin A$, since A' is the projection of $A \cap \{x_1 = \alpha\}$ on the last $n - 1$ coordinates. Moreover the distance between $(\alpha, x_2, \ldots, x_n)$ and $(\alpha, y_2, \ldots, y_n)$ is less than ε. Hence $(\alpha, x_2, \ldots, x_n)$ 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. □

Lemma 113. *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 $(d_1, \ldots, d_n)$, 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: $\nu_{2j}(x) = t_j - t_{k+1}$ where k is the largest index less than j with $x \in R_k$ if any, and 0 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

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

From Lemma 112, 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 $\gamma_{2j}(x) = \alpha_j - \alpha_{k+1}$ where k is the largest index less than j with $x \in R_k$ if any, and 0 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 $\gamma_{2j} \in \mathsf{Cl}r_{2j}$. Moreover, α has integer values, thus also γ_{2j} has integer values.

From this, we can built 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:

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

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 $m + \varepsilon$. ☐

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. ☐

5.3 Quantitative model-checking on priced-timed automata

Definition 114. WCTL *extends* CTL *as follows:*

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

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*

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

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

Theorem 115. 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 $\mathsf{Add}^+(x, \{z\})$ and $\mathsf{Add}^-(x, \{z\})$, displayed on Fig. 25.

Those automata clearly satisfy the following Lemma:

Figure 25: Automata $\mathsf{Add}^+(x,\{z\})$ and $\mathsf{Add}^-(x,\{z\})$

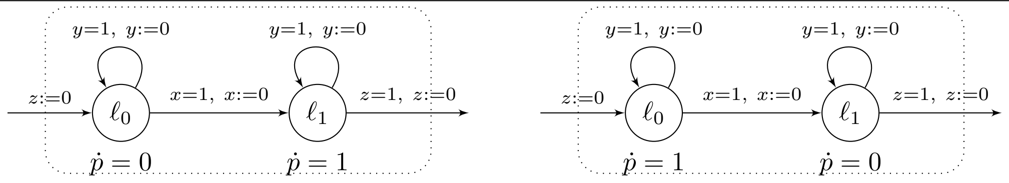
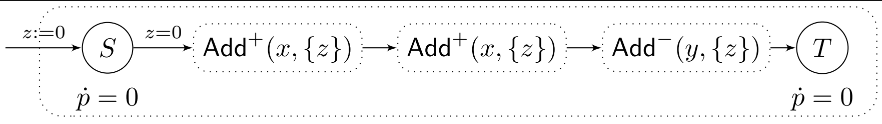


Figure 26: Automaton $\mathsf{Test}(y=2x,\{z\})$



$z{:=}0$
S
$z{=}0$
$\mathsf{Add}^+(x,\{z\})$
$\mathsf{Add}^+(x,\{z\})$
$\mathsf{Add}^-(y,\{z\})$
T
$\dot p = 0$
$\dot p = 0$

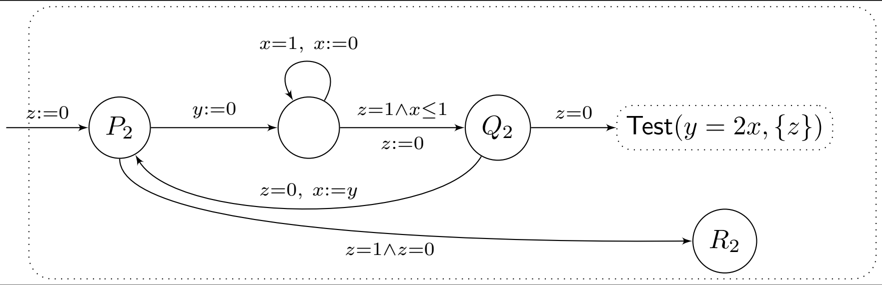
Lemma 116. *If a run enters location ℓ_0 of $\mathsf{Add}^+(x,\{z\})$ (resp. $\mathsf{Add}^-(x,\{z\})$) with $x = \alpha_0 \in [0,1]$, $y = \beta_0 \in [0,1]$ and $\mu(p) = \gamma_0$, it then leaves location ℓ_1 with the same values for x and y, and with $\mu(p) = \gamma_0 + \alpha_0$ (resp. $\mu(p) = \gamma_0 + 1 - \alpha_0$).*

Module $\mathsf{Test}(y=2x,\{z\})$ is the (deterministic) automaton displayed on Fig. 26. It sets the cost to $2x+1-y$. Let $\varphi_1 = S \wedge \mathbf{EF}_{p\le 1} T \wedge \mathbf{EF}_{p\ge 1} T$. The following Lemma clearly holds:

Lemma 117. *Formula φ_1 holds in S along module $\mathsf{Test}(y=2x,\{z\})$ with $x = \alpha_0 \in [0,1]$ and $y = \beta_0 \in [0,1]$ if, and only if, $\beta_0 = 2\alpha_0$.*

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

Figure 27: Automaton $\mathsf{Power}_2(x,\{y,z\})$



Module $\mathsf{Power}_2(x,\{y,z\})$ is displayed on Fig. 27. 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 $\varphi_2 = P_2 \wedge \mathbf{E}((Q_2 \to \mathbf{E}(Q_2 \,\mathbf{U}\, \varphi_1)) \,\mathbf{U}\, R_2)$. We have the following Lemma:

Lemma 118. *Formula φ_2 holds in P_2 in module $\mathsf{Power}_2(x,\{y,z\})$ with $x = \alpha_0 \in (0,1]$ if, and only if, there exists a non-negative integer d s.t. $\alpha_0 = 1/2^d$.*

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

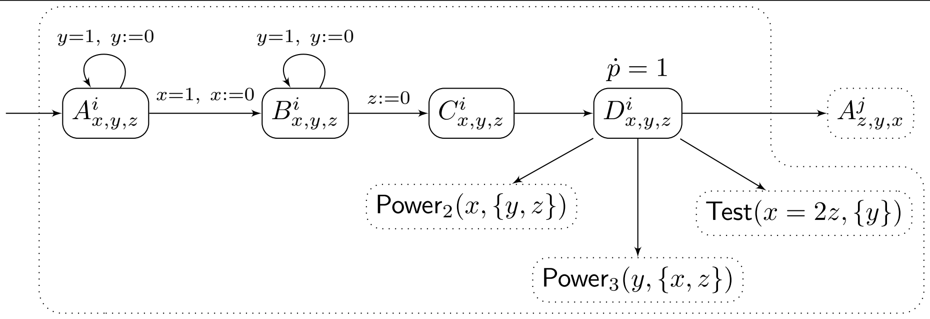
We now tackle the main part of the reduction. Let $\mathcal{M}$ be such a two-counter machine. We build a weighted timed automaton $\mathcal{W}_{\mathcal{M}}$ (with three clocks and one stopwatch observer) and a WCTL-formula Φ such that given q_0, a well-chosen state of $\mathcal{W}_{\mathcal{M}}$, we 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 $\{x_1, x_2\} \to \{x, y, z\}$.

Consider the following instruction of the two-counter machine:

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

We assume that the initial value of c_1 is stored in clock x, whereas that of c_2 is stored in y. To p_i, we associate the automaton $\mathsf{Aut}^i_{1,+}(x, y, z)$ as in Fig. 28. In that figure (and in all the other ones), observer rates which are omitted are equal to zero. The subscript $1, +$ is a remainder that instruction p_i deals with counter stored in the first clock (x here) and is an incrementation (we might omit it when it is not necessary), the tuple (x, y, z) indicates which clocks encode counters c_1 and c_2: here, c_1 is initially stored in x and c_2 is initially stored in y. At the end of this module, the new values of c_1 and c_2 are stored in z and y, resp.; that's why we swap x and z when leaving the module (transition from $D^i_{x,y,z}$ to $A^j_{z,y,x}$).

Figure 28: Incrementing a counter



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

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

The last point will be ensured through a global WCTL-formula stating that the value of the observer is unchanged in location $D^i_{x,y,z}$. The second point is obtained by a module $\mathsf{Test}(x = 2z, \{y\})$, together with a WCTL-formula ϕ_1. Finally, according to Lemma 119 below, the first point is enforced by checking that the values of x and y when entering $D^i_{x,y,z}$ are $1/2^n$ and $1/3^m$ for some integers n and m. Those conditions are ensured by modules Power_2 and Power_3 and the associated formulas ϕ_2 and ϕ_3.

Lemma 119. *If a run enters location $A^i_{x,y,z}$ with $x = 1/2^{c_1}$, $y = 1/3^{c_2}$ and enters location $D^i_{x,y,z}$ t time units later with the value of x of the form $1/2^n$ for some n, and the value of y of the form $1/3^m$ for some m, then $t = 1$, $n = c_1$ and $m = c_2$.*

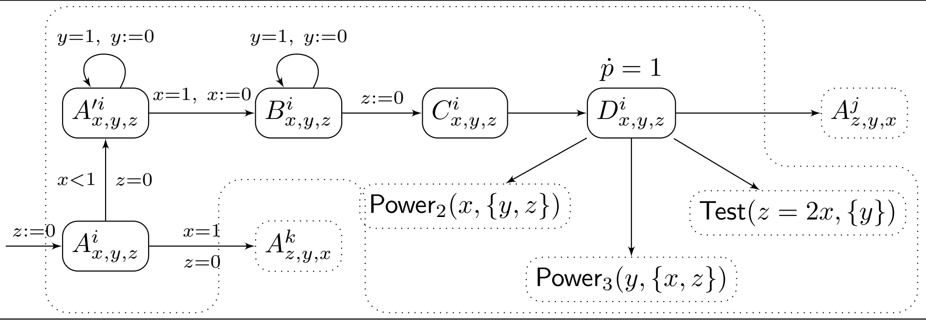
This lemma can easily be proved using elementary arithmetical manipulations. It plays a crucial role in our reduction: it explains how comparing clocks to powers of 1/2 and 1/3 gives a way to measure exactly 1 t.u., and thus why we encode the counters as powers of 1/2 and 1/3. Note that 2 and 3 could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

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

We now treat instruction:

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

Figure 29: Incrementing a counter



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

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 119 clearly also holds for $\mathsf{Aut}^i_{1,-}(x,y,z)$. Automaton $\mathsf{Aut}^i_{2,-}(x,y,z)$ is built in the same way.

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

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

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

Lemma 120. *$\mathcal{W}_{\mathcal{M}}, q_0 \models \Phi$ iff the two-counter machine $\mathcal{M}$ has a halting computation.*

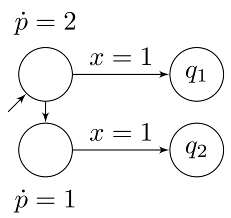
□

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

Theorem 121. *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. 30. Then formula $\mathbf{EF}_{\leq 1}\, q_1$ only holds in the initial state if clock x is larger than or equal to $1/2$.

Figure 30: A one-clock weighted timed automaton



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 122. *Let Φ be a* WCTL *formula and let $\mathcal{W}$ be a one-clock weighted timed automaton. Then there exist finitely many constants $0 = a_0 < a_1 < \ldots < a_n < a_{n+1} = +\infty$ 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,*

- *$\{a_0, ..., a_n\}$ 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. □

Definition 123. WLTL *extends* LTL *as follows:*

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

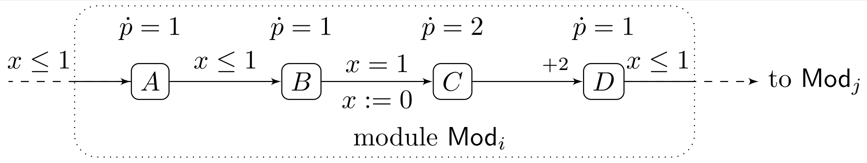
The semantics of the new "until" modality (along weighted timed words[6]) is the same as for WCTL.

Theorem 124. WLTL *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. 31 encodes incrementation of the first counter (*i.e.*, it divides the value of the clock by 2 between entrance and exit).

Figure 31: Module for incrementing c_1



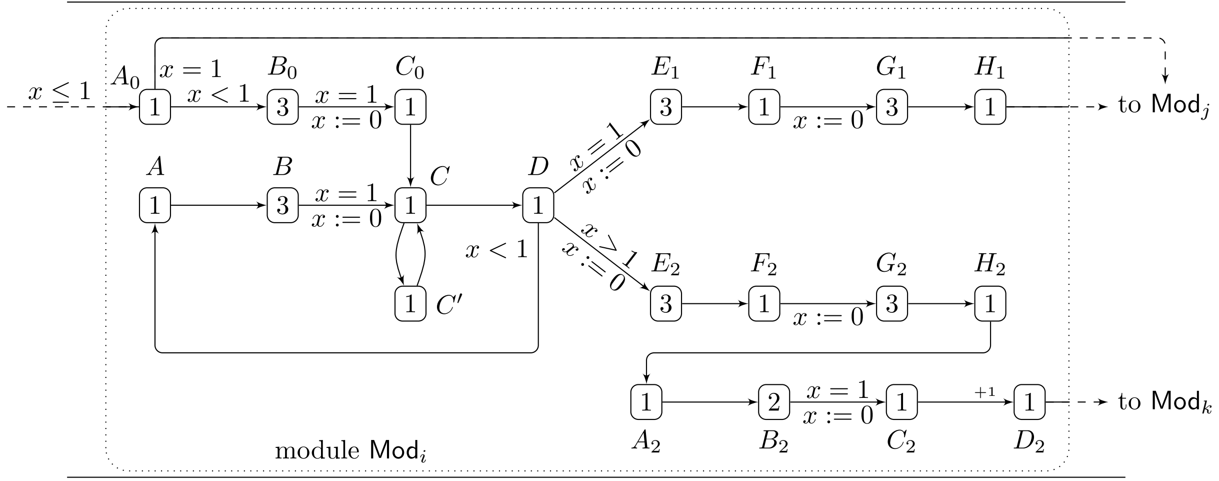
The following lemma is then easy to prove:

Lemma 125. *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 $x_1 = x_0/2$.*

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

Figure 32: Module testing/decrementing c_1 (the observer rates are indicated in the locations)



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 32.

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

Lemma 126. *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 127. *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 WLTL, as well as reachability of the halting state. This concludes the reduction, proving that WLTL model checking is undecidable. □

[6]We could of course also define the semantics over weighted signals.

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, Tómas 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.

[AHK97] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*, pages 100–109. IEEE Comp. Soc. Press, October 1997.

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, September 2002.

[AHKV98] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In Davide Sangiorgi and Robert de Simone, editors, *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer-Verlag, September 1998.

[AI03] Micah Adler and Neil Immerman. An $n!$ lower bound on formula size. *ACM Transactions on Computational Logic*, 4(3):296–314, July 2003.

[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.

[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.

[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.

[BFL$^+$08] Patricia Bouyer, Uli Fahrenberg, Kim Gulstrand Larsen, Nicolas Markey, and Jiři Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *Proceedings of the 6th Internationl Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'08)*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer-Verlag, September 2008.

[BGS92] José Balcázar, Joaquim Gabarró, and Miklós Sántha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4(1 supplement):638–648, November 1992.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Moel-Checking.* MIT Press, May 2008.

[BMOW07] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. The cost of punctuality. In *Proceedings of the 22nd Annual Symposium on Logic in Computer Science (LICS'07)*, pages 109–118. IEEE Comp. Soc. Press, July 2007.

[Boz08] Laura Bozzelli. The complexity of ctl* + linear past. In Roberto Amadio, editor, *Proceedings of the 11th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'08)*, volume 4962 of *Lecture Notes in Computer Science*, pages 186–200. Springer-Verlag, March-April 2008.

[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.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CGP00] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking.* MIT Press, 2000.

[EJ99] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, September 1999.

[ES84a] E. Allen Emerson and A. Prasad Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, June 1984.

[ES84b] 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.

[EVW02] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279–295, December 2002.

[GPSS80] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Conference Record of the 7th ACM Symposium on Principles of Programming Languages (POPL'80)*, pages 163–173. ACM Press, January 1980.

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

[Kam68] Hans W. Kamp. *Tense Logic and the Theory of Linear Order.* Phd thesis, Computer Science Department, University of California at Los Angeles, USA, 1968.

[Kir02] Daniel Kirsten. Alternating tree automata and parity games. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*, pages 153–167. Springer-Verlag, 2002.

[KP95] Orna Kupferman and Amir Pnueli. *Once* and *for all*. In *Proceedings of the 10th Annual Symposium on Logic in Computer Science (LICS'95)*, pages 25–35. IEEE Comp. Soc. Press, June 1995.

[Lan08] Martin Lange. A purely model-theoretic proof of the exponential succinctness gap between CTL$^+$ and CTL. *Information Processing Letters*, 108(5):308–312, November 2008.

[Lar95] François Laroussinie. About the expressive power of CTL combinators. *Information Processing Letters*, 54(6):343–345, June 1995.

[LMO07] François Laroussinie, Nicolas Markey, and Ghassan Oreiby. On the expressiveness and complexity of ATL. In Helmut Seidl, editor, *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'07)*, volume 4423 of *Lecture Notes in Computer Science*, pages 243–257. Springer-Verlag, March 2007.

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

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

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[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.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simonetta Ronchi Della Rocca, editors, *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP'89)*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer-Verlag, July 1989.

[Pri57] Arthur N. Prior. *Time and Modality.* Oxford University Press, 1957.

[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.

[SC85] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, July 1985.

[vG90] Rob van Glabbeek. The linear-time–branching-time spectrum. In Jos C. M. Baeten and Jan Willem Klop, editors, *Proceedings of the 1st International Conference on Concurrency Theory (CONCUR'90)*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, August 1990.

[Wika] Wikipedia. Ariane 5 flight 501.

[Wikb] Wikipedia. Mars climate orbiter.

[Wikc] Wikipedia. Pentium FDIV bug.

[Wikd] Wikipedia. Therac-25.

[Wil99] Thomas Wilke. CTL$^+$ is exponentially more succinct than CTL. In C. Pandu Rangan, Venkatesh Raman, and R. Ramanujam, editors, *Proceedings of the 19th Conferentce on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'99)*, volume 1738 of *Lecture Notes in Computer Science*, pages 110–121. Springer-Verlag, December 1999.