Automata with Timers[*]

Véronique Bruyère[1], Guillermo A. Pérez[2], Gaëtan Staquet[1,2], and Frits W. Vaandrager[3]

[1] University of Mons, Belgium
{veronique.bruyere,gaetan.staquet}@umons.ac.be
[2] University of Antwerp – Flanders Make, Belgium
guillermo.perez@uantwerpen.be
[3] Radboud University, The Netherlands
f.vaandrager@cs.ru.nl

Abstract. In this work, we study properties of deterministic finite-state automata with timers, a subclass of timed automata proposed by Vaandrager et al. as a candidate for an efficiently learnable timed model. We first study the complexity of the configuration reachability problem for such automata and establish that it is PSPACE-complete. Then, as simultaneous timeouts (we call these, races) can occur in timed runs of such automata, we study the problem of determining whether it is possible to modify the delays between the actions in a run, in a way to avoid such races. The absence of races is important for modelling purposes and to streamline learning of automata with timers. We provide an effective characterization of when an automaton is race-avoiding and establish that the related decision problem is in 3EXP and PSPACE-hard.

Keywords: Timed systems · model checking · reachability

1 Introduction

Timed automata were introduced by Alur & Dill [4] as finite-state automata equipped with real-valued clock variables for measuring the time between state transitions. These clock variables all increase at the same rate when time elapses, they can be reset along transitions, and be used in guards along transitions and in invariant predicates for states. Timed automata have become a framework of choice for modeling and analysis of real-time systems, equipped with a rich theory, supported by powerful tools, and with numerous applications [8].

Interestingly, whereas the values of clocks in a timed automaton *increase* over time, designers of real-time systems (e.g. embedded controllers and network protocols) typically use timers to enforce timing constraints, and the values of these timers *decrease* over time. If an application starts a timer with a certain

value t, then this value decreases over time and after t time units — when the value has become 0 — a timeout event occurs. It is straightforward to encode the behavior of timers using a timed automaton. Timed automata allow one to express a richer class of behaviors than what can be described using timers, and can for instance express that the time between two events is contained in an interval $[t - d, t + d]$. Moreover, timed automata can express constraints on the timing between arbitrary events, not just between start and timeout of timers.

However, the expressive power of timed automata entails certain problems. For instance, one can easily define timed automata models in which time stops at some point (timelocks) or an infinite number of discrete transitions occurs in a finite time (Zeno behavior). Thus timed automata may describe behavior that cannot be realized by any physical system. Also, learning [6, 14] of timed automata models in a black-box setting turns out to be challenging [5, 11, 12]. For a learner who can only observe the external events of a system and their timing, it may be really difficult to infer the logical predicates (invariants and guards) that label the states and transitions of a timed automaton model of this system. As a result, all known learning algorithms for timed automata suffer from combinatorial explosions, which severely limits their practical usefulness.

For these reasons, it is interesting to consider variations of timed automata whose expressivity is restricted by using timers instead of clocks. Vaandrager et al. [17] study deterministic Mealy machines with a single timer (MM1T). In an MM1T, a transition may start a timer by setting it to a certain constant. Whenever a timer reaches zero, it produces an observable timeout symbol that triggers a transition in the automaton. Vaandrager et al. provide a black-box active learning algorithm for MM1Ts, and evaluate it on a number of realistic applications, showing that it outperforms the timed automata based approaches of Aichernig et al. [2] and An et al. [5]. However, whereas MM1Ts only support a single timer, the genetic programming approach of [2] is able to learn models with multiple clocks/timers.

If we want to extend the learning algorithm of [17] to a setting with multiple timers, we need to deal with the issue of *races*, i.e., situations where multiple timers reach zero (and thus timeout) simultaneously. If a race occurs, then (despite the automaton being deterministic!) the automaton can process the simultaneous timeouts in various orders, leading to nondeterministic behavior. This means that during learning of an automaton with multiple timers, a learner needs to offer the inputs at specific times in order to avoid the occurrence of races. As long as there are no races, the behavior of the automaton will be deterministic, and a learner may determine, for each timeout, by which preceding input it was caused by slightly *wiggling* the timing of inputs and check whether the timing timeout changes in a corresponding manner.

Contribution In this work, we take the one-timer definition from [17] and extend it to multiple timers while — to avoid overcomplicating the model — keeping the restriction that every transition can start or restart at most one timer. We first study the complexity of the configuration reachability problem for this model and establish that it is PSPACE-complete. Then, we turn our attention to the

problem of determining whether it is possible to wiggle the delays between the inputs in a run, in a way to avoid races. The importance of the latter is twofold. First, automata with timers may not be an attractive modelling formalism in the presence of behaviors that do not align with those of the real-world systems they are meant to abstract. Second, the absence of races is a key property used in the learning algorithm for automata with a single timer. In this direction, we provide an effective characterization of when an automaton is race-avoiding and establish that the related decision problem is in 3EXP and PSPACE-hard. In a more pragmatic direction, while again leveraging our characterization, we show that with fixed input and timer sets, the problem is in PSPACE. Finally, we also give some simple yet sufficient conditions for an automaton to be race-avoiding.

2 Preliminaries

An *automaton with timers* uses a finite set X of *timers*. Intuitively, a timer can be *started* to any integer value to become *active*. Subsequently, its value is decremented as time elapses (i.e., at the same fixed rate for all timers). When the value of a timer reaches 0, it *times out* and it is no longer active. Active timers can also be *stopped*, rendering them inactive, too. Such an automaton, along any transition, can stop a number of timers and update a *single* timer.

Some definitions are in order. We write $TO[X]$ to denote the set $\{to[x] \mid x \in X\}$ of *timeouts of* X. We denote by I a finite set of *inputs*. We write $\hat{I}$ to denote the set $I \cup TO[X]$ of *actions*: either reading an input (an *input-action*), or processing a timeout (a *timeout-action*). Finally, we denote by $U = (X \times \mathbb{N}^{>0}) \cup \{\bot\}$ the set of *updates*, where (x, c) means that timer x is started with value c, and $\bot$ stands for no timer update.

Definition 1 (Automaton with timers). *An automaton with timers (AT, for short) is a tuple $\mathcal{A} = (X, I, Q, q_0, \chi, \delta)$ where:*

- *X is a finite set of timers, I a finite set of inputs,*
- *Q is a finite set of states, with $q_0 \in Q$ the initial state,*
- *$\chi \colon Q \to \mathcal{P}(X)$, with $\chi(q_0) = \emptyset$, is a total function that assigns a finite set of active timers to each state,*
- *$\delta \colon Q \times \hat{I} \to Q \times U$ is a partial transition function that assigns a state and an update to each state-action pair, such that*
   - *$\delta(q, i)$ is defined iff either $i \in I$ or there is a timer $x \in \chi(q)$ with $i = to[x]$,*
   - *if $\delta(q, i) = (q', u)$ with $i = to[x]$ and $u = (y, c)$, then $x = y$ (when processing a timeout $to[x]$, only the timer x can be restarted).*

Moreover, any transition t of the form $\delta(q, i) = (q', u)$ must be such that

- *if $u = \bot$, then $\chi(q') \subseteq \chi(q)$ (all timers active in q' were already active in q in case of no timer update); moreover, if $i = to[x]$, then $x \notin \chi(q')$ (when the timer x times out and is not restarted, then x becomes inactive in q');*
- *if $u = (x, c)$, then $x \in \chi(q')$ and $\chi(q') \setminus \{x\} \subseteq \chi(q)$ ((re)starting the timer x makes it active in q').*
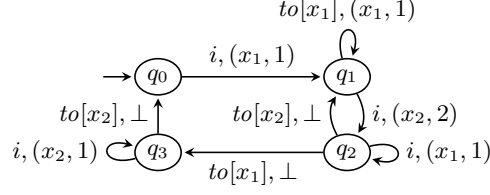
Fig. 1: An automaton with two timers x_1, x_2, such that $\chi(q_0) = \emptyset$, $\chi(q_1) = \{x_1\}$, $\chi(q_2) = \{x_1, x_2\}$, and $\chi(q_3) = \{x_2\}$.

When a timer x is active in q and $i \neq to[x]$, we say that the transition t stops x if x is inactive in q', and that t discards x if t stops x or restarts x. We write $q \xrightarrow{i}_u q'$ if $\delta(q,i) = (q',u)$.

Example 1. An AT $\mathcal{A}$ is shown in Figure 1 with set $X = \{x_1, x_2\}$ of timers and with set $I = \{i\}$ of inputs. In the initial state q_0, no timer is active, while x_1 is active in q_1 and q_2, and x_2 is active in q_2 and q_3. That is, $\chi(q_0) = \emptyset, \chi(q_1) = \{x_1\}, \chi(q_2) = \{x_1, x_2\}$, and $\chi(q_3) = \{x_2\}$. Timer updates are shown in the transitions. For instance, x_1 is started with value 1 when going from q_0 to q_1. The transition looping on q_2 discards x_1 and restarts it with value 1.

2.1 Timed semantics

The semantics of an AT $\mathcal{A}$ is defined via an infinite-state labeled transition system that describes all possible configurations and transitions between them.

A *valuation* is a partial function $\kappa\colon X \to \mathbb{R}^{\geq 0}$ that assigns nonnegative real numbers to timers. For $Y \subseteq X$, we write $\mathsf{Val}(Y)$ for the set of all valuations κ such that $\mathsf{dom}(\kappa) = Y$.[4] A *configuration* of $\mathcal{A}$ is a pair (q, κ) where $q \in Q$ and $\kappa \in \mathsf{Val}(\chi(q))$. The *initial configuration* is the pair (q_0, κ_0) where κ_0 is the empty valuation since $\chi(q_0) = \emptyset$. If $\kappa \in \mathsf{Val}(Y)$ is a valuation in which all timers from Y have a value of at least $d \in \mathbb{R}^{\geq 0}$, then d units of time may elapse. We write $\kappa - d \in \mathsf{Val}(Y)$ for the valuation that satisfies $(\kappa - d)(x) = \kappa(x) - d$, for all $x \in Y$. The following rules specify the transitions between configurations $(q, \kappa), (q', \kappa')$.

$$\frac{\forall x\colon \kappa(x) \geq d}{(q, \kappa) \xrightarrow{d} (q, \kappa - d)} \tag{1}$$

$$\frac{q \xrightarrow{i}_{\perp} q', \quad i = to[x] \Rightarrow \kappa(x) = 0, \quad \forall y \in \chi(q')\colon \kappa'(y) = \kappa(y)}{(q, \kappa) \xrightarrow{i}_{\perp} (q', \kappa')} \tag{2}$$

---

[4] Notation $\mathsf{dom}(f)$ means the domain of the partial function f.

$$q \xrightarrow[(x,c)]{i} q', \quad i = to[x] \Rightarrow \kappa(x) = 0, \quad \forall y \in \chi(q') \colon \kappa'(y) = \begin{cases} c & \text{if } y = x \\ \kappa(y) & \text{otherwise} \end{cases}$$

$$(q,\kappa) \xrightarrow[(x,c)]{i} (q',\kappa') \tag{3}$$

Transitions of type (1) are called *delay transitions* (delay zero is allowed); those of type (2) and (3) are called *discrete transitions* (*timeout transitions* when $i = to[x]$ and *input transitions* otherwise). A *timed run* of $\mathcal{A}$ is a sequence of configurations such that delay and discrete transitions alternate. The set $truns(\mathcal{A})$ of timed runs is defined inductively as follows.

- The sequence $(q_0, \kappa_0) \xrightarrow{d} (q_0, \kappa_0 - d)$ is in $truns(\mathcal{A})$.
- Suppose $\rho(q, \kappa)$ is a timed run ending with configuration (q, κ), then $\rho' = \rho(q, \kappa) \xrightarrow{i}{}_{u} (q', \kappa') \xrightarrow{d} (q', \kappa' - d)$ is in $truns(\mathcal{A})$.

A timed run is also written as $\rho = (q_0, \kappa_0) \, d_1 \, i_1/u_1 \, \ldots \, d_n \, i_n/u_n \, d_{n+1} \, (q, \kappa)$ such that only the initial configuration (q_0, κ_0) and the last configurations (q, κ) of ρ are given. The *untimed trace* of a timed run ρ, denoted $untime(\rho)$, is the alternating sequence of states and actions from ρ, that is, $untime(\rho) = q_0 \, i_1 \, \ldots \, i_n \, q$ (we omit the valuations, the delays, and the updates).

Example 2. A sample timed run ρ of the AT of Example 1 is given below. Notice the transition with delay zero, indicating that two actions occur at the same time.

$$\rho = (q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow[(x_1,1)]{i} (q_1, x_1 = 1) \xrightarrow{1} (q_1, x_1 = 0) \xrightarrow[(x_2,2)]{i} (q_2, x_1 = 0, x_2 = 2)$$

$$\xrightarrow{0} (q_2, x_1 = 0, x_2 = 2) \xrightarrow[\perp]{to[x_1]} (q_3, x_2 = 2) \xrightarrow{2} (q_3, x_2 = 0) \xrightarrow[\perp]{to[x_2]} (q_0, \emptyset) \xrightarrow{0.5} (q_0, \emptyset).$$

The untimed trace of ρ is $untime(\rho) = q_0 \, i \, q_1 \, i \, q_2 \, to[x_1] \, q_3 \, to[x_2] \, q_0$.

2.2 Blocks and races

In this section, given an AT $\mathcal{A}$, we focus on its timed runs $\rho = (q_0, \kappa_0) \, d_1 \, i_1/u_1 \cdots d_n \, i_n/u_n \, d_{n+1} \, (q, \kappa)$ such that their first and last delays are non-zero and no timer times out in their last configuration, i.e., $d_1 > 0, d_{n+1} > 0$ and $\kappa(x) \neq 0$ for all $x \in \chi(q)$.[5] Such runs are called *padded*, and we denote by $ptruns(\mathcal{A})$ the set of all padded timed runs of $\mathcal{A}$. To have a good intuition about padded timed runs, their decomposition into *blocks* is helpful and will be often used in the proofs. A block is composed of an input i that starts a timer x and of the succession of time-outs and restarts of x, that i induces inside a timed run. Let us formalize this notion. Consider a padded timed run $\rho = (q_0, \kappa_0) \, d_1 \, i_1/u_1 \, \ldots \, d_n \, i_n/u_n \, d_{n+1} \, (q, \kappa)$ of an AT. Let k, k' be such that $1 \leq k < k' \leq n$. We say that i_k *triggers* $i_{k'}$ if there is a timer x such that:

---

[5] The reason for this choice will be clarified at the end of this section.

- i_k (re)starts x, that is, $u_k = (x, c)$,
- $i_{k'}$ is the action $to[x]$, and
- there is no ℓ with $k < \ell < k'$ such that $i_\ell = to[x]$ or i_ℓ discards x.

Note that $i_{k'}$ may restart x or not, and if it does, x later times out or is discarded.

Definition 2 (Block). *Let $\rho = (q_0, \kappa_0)\ d_1\ i_1/u_1\ \ldots\ d_n\ i_n/u_n\ d_{n+1}\ (q, \kappa)$ be a padded timed run of an AT. A* block *of ρ is a pair $B = (k_1 k_2 \ldots k_m, \gamma)$ such that $i_{k_1}, i_{k_2}, \ldots, i_{k_m}$ is a maximal subsequence of actions of ρ such that $i_{k_1} \in I$, i_{k_ℓ} triggers $i_{k_{\ell+1}}$ for all $1 \le \ell < m$, and γ is the* timer fate *of B defined as:*

$$\gamma = \begin{cases} \bot & \textit{if } i_{k_m} \textit{ does not restart } x, \\ \bullet & \textit{if } i_{k_m} \textit{ restarts } x \textit{ which is discarded by some } i_\ell, \textit{ with } k_m < \ell \le n, \\ & \textit{when its value is zero,} \\ \times & \textit{otherwise.} \end{cases}$$

In the timer fate definition, consider the case where i_{k_m} restarts x. For the purposes of Section 4.1, it is convenient to know whether x is later discarded or not, and in case it is discarded, whether this occurs when its value is zero ($\gamma = \bullet$). Hence, $\gamma = \times$ covers both situations where x is discarded with non-zero value, and x is still active in the last configuration (q, κ) of ρ. Notice that in the latter case, x has also non-zero value in (q, κ) as ρ is padded. When no confusion is possible, we denote a block by a sequence of inputs rather than the corresponding sequence of indices, that is, $B = (i_{k_1} i_{k_2} \ldots i_{k_m}, \gamma)$. In the sequel, we use notation $i \in B$ to denote an action i belonging to the sequence of B.

By definition of an AT, recall that the same timer x is restarted along a block B. Hence we also say that B is an *x-block*. Note also that the sequence of a block can be composed of a single input $i \in I$.

As this notion of blocks is not trivial but plays a great role in this paper, let us give several examples illustrating multiple situations.

Example 3. Consider the timed run ρ of Example 2 from the AT $\mathcal{A}$ depicted in Figure 1. It has two blocks: an x_1-block $B_1 = (i\ to[x_1], \bot)$ and an x_2-block $B_2 = (i\ to[x_2], \bot)$, both represented in Figure 2a.[6] In this visual representation of the blocks, time flows left to right and is represented by the thick horizontal line. A "gap" in that line indicates that the time is stopped, i.e., the delay between two consecutive actions is zero. We draw a vertical line for each input, and join together inputs belonging to a block by a horizontal (non-thick) line.

Consider another timed run σ from $\mathcal{A}$:

$$\sigma = (q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow[(x_1, 1)]{i} (q_1, x_1 = 1) \xrightarrow{1} (q_1, x_1 = 0) \xrightarrow[(x_1, 1)]{to[x_1]} (q_1, x_1 = 1)$$

$$\xrightarrow{0} (q_1, x_1 = 1) \xrightarrow[(x_2, 2)]{i} (q_2, x_1 = 1, x_2 = 2) \xrightarrow{1} (q_2, x_1 = 0, x_2 = 1)$$

$$\xrightarrow[\bot]{to[x_1]} (q_3, x_2 = 1) \xrightarrow{1} (q_3, x_2 = 0) \xrightarrow[\bot]{to[x_2]} (q_0, \emptyset) \xrightarrow{0.5} (q_0, \emptyset).$$

---

[6] When using the action indices in the blocks, we have $B_1 = (1\,3, \bot)$ and $B_2 = (2\,4, \bot)$.

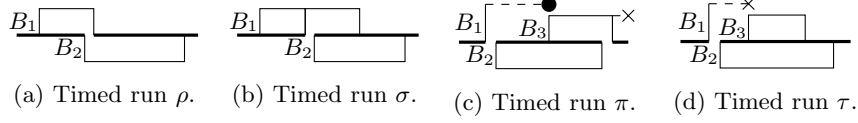(a) Timed run ρ.   (b) Timed run σ.   (c) Timed run π.   (d) Timed run τ.

Fig. 2: Block representations of four timed runs.

This timed run has also two blocks represented in Figure 2b, such that $B_1 = (i\ to[x_1]\ to[x_1], \bot)$ with x_1 timing out twice.

We conclude this example with two other timed runs, π and τ, such that some of their blocks have a timer fate $\gamma \neq \bot$. Let π and τ be the timed runs:

$$\pi = (q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow[(x_1,1)]{i} (q_1, x_1 = 1) \xrightarrow{0} (q_1, x_1 = 1) \xrightarrow[(x_2,2)]{i} (q_2, x_1 = 1, x_2 = 2)$$

$$\xrightarrow{1} (q_2, x_1 = 0, x_2 = 1) \xrightarrow[(x_1,1)]{i} (q_2, x_1 = 1, x_2 = 1) \xrightarrow{1} (q_2, x_1 = 0, x_2 = 0)$$

$$\xrightarrow[\bot]{to[x_2]} (q_1, x_1 = 0) \xrightarrow{0} (q_1, x_1 = 0) \xrightarrow[(x_1,1)]{to[x_1]} (q_1, x_1 = 1) \xrightarrow{0.5} (q_1, x_1 = 0.5)$$

$$\tau = (q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow[(x_1,1)]{i} (q_1, x_1 = 1) \xrightarrow{0} (q_1, x_1 = 1) \xrightarrow[(x_2,2)]{i} (q_2, x_1 = 1, x_2 = 2)$$

$$\xrightarrow{0.5} (q_2, x_1 = 0.5, x_2 = 1.5) \xrightarrow[(x_1,1)]{i} (q_2, x_1 = 1, x_2 = 1.5) \xrightarrow{1} (q_2, x_1 = 0, x_2 = 0.5)$$

$$\xrightarrow[\bot]{to[x_1]} (q_3, x_2 = 0.5) \xrightarrow{0.5} (q_3, x_2 = 0) \xrightarrow[\bot]{to[x_2]} (q_0, \emptyset) \xrightarrow{0.5} (q_0, \emptyset).$$

The run π has three blocks $B_1 = (i, \bullet)$ (x_1 is started by i and then discarded while its value is zero), $B_2 = (i\ to[x_2], \bot)$, and $B_3 = (i\ to[x_1], \times)$ (x_1 is again started in B_3 but π ends before x_1 reaches value zero). Those blocks are represented in Figure 2c, where we visually represent the timer fate of B_1 (resp. B_3) by a dotted line finished by $\bullet$ (resp. $\times$). Finally, the run τ has its blocks depicted in Figure 2d. This time, x_1 is discarded before reaching zero, i.e., $B_1 = (i, \times)$.

As illustrated by the previous example, blocks satisfy the following property.

Lemma 1. *Let $\rho = (q_0, \kappa_0)\ d_1\ i_1/u_1\ \ldots\ d_n\ i_n/u_n\ d_{n+1}\ (q, \kappa)$ be a padded timed run of an AT. Then, the sequences of the blocks of ρ form a partition of the set of indices $\{1, \ldots, n\}$ of the actions of ρ.*

Along a timed run of an AT $\mathcal{A}$, it can happen that a timer times out at the same time that another action takes place. This leads to a sort of *nondeterminism*, as $\mathcal{A}$ can process those concurrent actions in any order. This situation appears in Example 3 each time a gap appears in the time lines of Figure 2. We call these situations *races* that we formally define as follows.

Definition 3 (Race). *Let B, B' be two blocks of a padded timed run ρ with timer fates γ and γ'. We say that B and B' participate in a race if:*
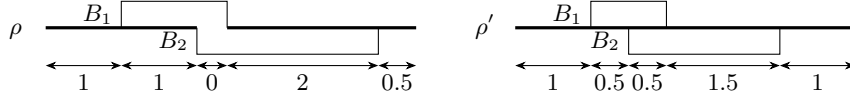
Fig. 3: Modifying the delays in order to remove a race.

- *either there exist actions $i \in B$ and $i' \in B'$ such that the sum of the delays between i and i' in ρ is equal to zero, i.e., no time elapses between them,*
- *or there exists an action $i \in B$ that is the first action along ρ to discard the timer started by the last action $i' \in B'$ and $\gamma' = \bullet$, i.e., the timer of B' (re)started by i' reaches value zero when i discards it.*

We also say that the actions i and i' participate in this race.

The first case of the race definition appears in Figure 2a, while the second case appears in Figure 2c (see the race in which blocks B_1 and B_3 participate). The nondeterminism is highlighted in Figures 2a and 2b where two actions (i and $to[x]$) occur at the same time but are processed in a different order in each figure. Unfortunately, imposing a particular way of resolving races (i.e. imposing a particular action order) may seem arbitrary when modelling real-world systems. It is therefore desirable for the set of sequences of actions along timed runs to be independent to the resolution of races.

Definition 4 (Race-avoiding). *An AT $\mathcal{A}$ is* race-avoiding *iff for all padded timed runs $\rho \in ptruns(\mathcal{A})$ with races, there exists some $\rho' \in ptruns(\mathcal{A})$ with no races such that $untime(\rho') = untime(\rho)$.*

Example 4. Let us come back to the timed run ρ of Example 2 that contains a race (see Figure 2a). By moving the second occurrence of action i slightly earlier in ρ, we obtain the timed run ρ':

$$\rho' = (q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow[(x_1,1)]{i} (q_1, x_1 = 1) \xrightarrow{0.5} (q_1, x_1 = 0.5) \xrightarrow[(x_2,2)]{i} (q_2, x_1 = 0.5, x_2 = 2)$$

$$\xrightarrow{0.5} (q_2, x_1 = 0, x_2 = 1.5) \xrightarrow[\perp]{to[x_1]} (q_3, x_2 = 1.5) \xrightarrow{1.5} (q_3, x_2 = 0) \xrightarrow[\perp]{to[x_2]} (q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset).$$

Notice that $untime(\rho') = q_0 \ i \ q_1 \ i \ q_2 \ to[x_1] \ q_3 \ to[x_2] \ q_0 = untime(\rho')$. Moreover, ρ' contains no races as indicated in Figure 3.

Notice that several blocks could participate in the same race. The notion of block has been defined for padded timed runs only, as we do not want to consider runs that end *abruptly* during a race (some pending timeouts may not be processed at the end of the run, for instance). Moreover, it is always possible for the first delay to be positive as no timer is active in the initial state. Finally, non-zero delays at the start and the end of the runs allow to move blocks as introduced in Example 4 and further detailed in Section 4.1.

In Section 4, we study whether it is decidable that an AT is race-avoiding, and how to eliminate races in a race-avoiding AT while keeping the same traces. Before, we study the (classical) reachability problem in Section 3.

3   Reachability

The *reachability problem* asks, given an AT $\mathcal{A}$ and a state q, whether there exists a timed run $\rho \in trans(\mathcal{A})$ from the initial configuration to some configuration (q, κ). In this section, we argue that this problem is PSPACE-complete.

Theorem 1. *The reachability problem for ATs is* PSPACE-*complete.*

For hardness, we reduce from the acceptance problem for linear-bounded Turing machines (LBTM, for short), as done for timed automata, see e.g. [1]. In short, given an LBTM $\mathcal{M}$ and a word w of length n, we construct an AT that uses n timers x_i, $1 \le i \le n$, such that the timer x_i encodes the value of the i-th cell of the tape of $\mathcal{M}$. We also rely on a timer x that is always (re)started at one, and is used to synchronize the x_i timers and the simulation of $\mathcal{M}$. The simulation is split into *phases*: The AT first seeks the symbol on the current cell i of the tape (which can be derived from the moment at which the timer x_i times out, using the number of times x timed out since the beginning of the phase). Then, the AT simulates a transition of $\mathcal{M}$ by restarting x_i, reflecting the new value of the i-th cell. Finally, the AT can reach a designated state iff $\mathcal{M}$ is in an accepting state. Therefore, the reachability problem is PSPACE-hard.

For membership, we follow the classical argument used to establish that the reachability problem for timed automata is in PSPACE: We first define *region automata* for ATs (which are a simplification of region automata for timed automata) and observe that reachability in an AT reduces to reachability in the corresponding region automaton. The region automaton is of size exponential in the number of timers and polynomial in the number of states of the AT. Hence, the reachability problem for ATs is in PSPACE via standard arguments.

We define region automata for ATs much like they are defined for timed automata [3, 4, 7]. Let $\mathcal{A} = (X, I, Q, q_0, \chi, \delta)$ be an AT. For a timer $x \in X$, c_x denotes the largest constant to which x is updated in $\mathcal{A}$. Let $C = \max_{x \in X} c_x$. Two valuations κ and κ' are said *timer-equivalent*, noted $\kappa \cong \kappa'$, iff $\mathsf{dom}(\kappa) = \mathsf{dom}(\kappa')$ and the following hold for all $x_1, x_2 \in \mathsf{dom}(\kappa)$:

- $\lfloor \kappa(x_1) \rfloor = \lfloor \kappa'(x_1) \rfloor$,
- $\mathrm{frac}(\kappa(x_1)) = 0$ iff $\mathrm{frac}(\kappa'(x_1)) = 0$,
- $\mathrm{frac}(\kappa(x_1)) \le \mathrm{frac}(\kappa(x_2))$ iff $\mathrm{frac}(\kappa'(x_1)) \le \mathrm{frac}(\kappa'(x_2))$.

A *timer region* for $\mathcal{A}$ is an equivalence class of timer valuations induced by $\cong$. We lift the relation to configurations: $(q, \kappa) \cong (q', \kappa')$ iff $\kappa \cong \kappa'$ and $q = q'$. Finally, $[\![(q, \kappa)]\!]_{\cong}$ denotes the equivalence class of (q, κ).

We are now able to define a finite automaton called the *region automaton* of $\mathcal{A}$ and denoted $\mathcal{R}$. The alphabet of $\mathcal{R}$ is $\Sigma = \{\tau\} \cup \hat{I}$ where τ is a special symbol used in non-zero delay transitions. Formally, $\mathcal{R}$ is the finite automaton (Σ, S, s_0, Δ) where:

- $S = \{(q, \kappa) \mid q \in Q, \kappa \in \mathsf{Val}(\chi(q))\}_{/\cong}$, i.e., the quotient of the configurations by $\cong$, is the set of states,

- $s_0 = (q_0, [\![\kappa_0]\!]_{\cong})$ with κ_0 the empty valuation, is the initial state,
- the set of transitions $\Delta \subseteq S \times \Sigma \times S$ includes $([\![(q, \kappa)]\!]_{\cong}, \tau, [\![(q, \kappa')]\!]_{\cong})$ if $(q, \kappa) \xrightarrow{d} (q, \kappa')$ in $\mathcal{A}$ whenever $d > 0$, and $([\![(q, \kappa)]\!]_{\cong}, i, [\![(q', \kappa')]\!]_{\cong})$ if $(q, \kappa) \xrightarrow[u]{i} (q', \kappa')$ in $\mathcal{A}$.

It is easy to check that the timer-equivalence relation on configurations is a *(strong) time-abstracting bisimulation* [9, 16]. That is, for all $(q_1, \kappa_1) \cong (q_2, \kappa_2)$ the following holds:

- if $(q_1, \kappa_1) \xrightarrow[u]{i} (q_1', \kappa_1')$, then there is $(q_2, \kappa_2) \xrightarrow[u]{i} (q_2', \kappa_2')$ with $(q_1', \kappa_1') \cong (q_2', \kappa_2')$,
- if $(q_1, \kappa_1) \xrightarrow{d_1} (q_1, \kappa_1')$, then there exists $(q_2, \kappa_2) \xrightarrow{d_2} (q_2, \kappa_2')$ where $d_1, d_2 > 0$ may differ such that $(q_1, \kappa_1') \cong (q_2, \kappa_2')$, and
- the above also holds if (q_1, κ_1) and (q_2, κ_2) are swapped.

Using this property, we can prove the following about $\mathcal{R}$.

Lemma 2. *Let $\mathcal{A} = (X, I, Q, q_0, \chi, \delta)$ be an AT and $\mathcal{R}$ be its region automaton.*

1. *The size of $\mathcal{R}$ is linear in $|Q|$ and exponential in $|X|$. That is, $|S|$ is smaller than or equal to $|Q| \cdot |X|! \cdot 2^{|X|} \cdot (C+1)^{|X|}$.*
2. *There is a timed run ρ of $\mathcal{A}$ that begins in (q, κ) and ends in (q', κ') iff there is a run ρ' of $\mathcal{R}$ that begins in $[\![(q, \kappa)]\!]_{\cong}$ and ends in $[\![(q', \kappa')]\!]_{\cong}$.*

Example 5. Let us consider the AT $\mathcal{A}$ of Figure 1 and the timed run π given in Example 3. The corresponding run π' in the region automaton $\mathcal{R}$ is

$$(q_0, [\![\emptyset]\!]_{\cong}) \xrightarrow{\tau} (q_0, [\![\emptyset]\!]_{\cong}) \xrightarrow{i} (q_1, [\![x_1 = 1]\!]_{\cong}) \xrightarrow{i} (q_2, [\![x_1 = 1, x_2 = 2]\!]_{\cong})$$

$$\xrightarrow{\tau} (q_2, [\![x_1 = 0, x_2 = 1]\!]_{\cong}) \xrightarrow{i} (q_2, [\![x_1 = 1, x_2 = 1]\!]_{\cong}) \xrightarrow{\tau} (q_2, [\![x_1 = 0, x_2 = 0]\!]_{\cong})$$

$$\xrightarrow{to[x_2]} (q_1, [\![x_1 = 0]\!]_{\cong}) \xrightarrow{to[x_1]} (q_1, [\![x_1 = 1]\!]_{\cong}) \xrightarrow{\tau} (q_1, [\![0 < x_1 < 1]\!]_{\cong}).$$

Notice that the transitions with delay zero of π do not appear in π'.

4   Race-avoiding ATs

In this section, we study whether an AT $\mathcal{A}$ is race-avoiding, i.e., whether for any padded timed run ρ of $\mathcal{A}$ with races, there exists another run ρ' with no races such that $untime(\rho) = untime(\rho')$. We are able to prove the next theorem.

Theorem 2. *Deciding whether an AT is race-avoiding is* PSPACE*-hard and in* 3EXP. *It is in* PSPACE *if the sets of actions I and of timers X are fixed.*

Our approach is, given $\rho \in ptruns(\mathcal{A})$, to study how to slightly move blocks along the time line of ρ in a way to get another $\rho' \in ptruns(\mathcal{A})$ where the races are eliminated while keeping the actions in the same order as in ρ. We call this action *wiggling*. Let us first give an example and then formalize this notion.

Example 6. We consider again the AT of Figure 1. We have seen in Example 4 and Figure 3 that the block B_2 of ρ can be slightly moved to the left to obtain the timed run ρ' with no race such that $untime(\rho) = untime(\rho')$. Figure 3 illustrates how to move B_2 by changing some of the delays.

In contrast, this is not possible for the timed run π of Example 3. Indeed looking at Figure 2c, we see that it is impossible to move block B_2 to the left due to its race with B_1 (remember that we need to keep the same action order). It is also not possible to move it to the right due to its race with B_3. Similarly, it is impossible to move B_1 neither to the right (due to its race with B_2), nor to the left (otherwise its timer will time out instead of being discarded by B_3). Finally, one can also check that block B_3 cannot be moved.

Given a padded timed run $\rho = (q_0, \kappa_0) \ d_1 \ i_1/u_1 \ \ldots \ d_n \ i_n/u_n \ d_{n+1}(q, \kappa) \in ptruns(\mathcal{A})$ and a block $B = (k_1 \ldots k_m, \gamma)$ of ρ participating in a race, we say that we can *wiggle* B if for some ϵ, we can move B to the left (by $\epsilon < 0$) or to the right (by $\epsilon > 0$), and obtain a run $\rho' \in ptruns(\mathcal{A})$ such that $untime(\rho) = untime(\rho')$ and B no longer participates in any race. More precisely, we have $\rho' = (q_0, \kappa_0) \ d'_1 \ i_1/u_1 \ \ldots \ d'_n \ i_n/u_n \ d'_{n+1} \ (q, \kappa')$ such that

- for all $i_{k_\ell} \in B$ with $k_\ell > 1$, if $i_{k_\ell - 1} \notin B$ (the action before i_{k_ℓ} in ρ does not belong to B), then $d'_{k_\ell} = d_{k_\ell} + \epsilon$,
- if there exists $i_{k_\ell} \in B$ with $k_\ell = 1$ (the first action of B is the first action of ρ), then $d'_1 = d_1 + \epsilon$,
- for all $i_{k_\ell} \in B$ with $k_\ell < n$, if $i_{k_\ell + 1} \notin B$ (the action after i_{k_ℓ} in ρ does not belong to B), then $d'_{k_\ell + 1} = d_{k_\ell + 1} - \epsilon$,
- if there exists $i_{k_\ell} \in B$ with $k_\ell = n$ (the last action of B is the last action of ρ), then $d'_{n+1} = d_{n+1} - \epsilon$,
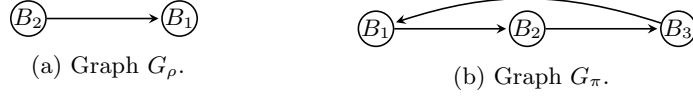- for all other d'_k, we have $d'_k = d_k$.

As $\rho' \in ptruns(\mathcal{A})$ and $untime(\rho) = untime(\rho')$, we must have $d'_k \geq 0$ for all k and $d'_1, d'_{n+1} > 0$.

We say that we can wiggle ρ, or that ρ is *wigglable*, if it is possible to wiggle its blocks, *one block at a time*, to obtain $\rho' \in ptruns(\mathcal{A})$ with no races such $untime(\rho) = untime(\rho')$. Hence if all padded timed runs with races of an AT $\mathcal{A}$ are wigglable, then $\mathcal{A}$ is race-avoiding.

In the next sections, we first associate a graph with any $\rho \in ptruns(\mathcal{A})$ in a way to characterize when ρ is wigglable thanks to this graph. We then state the equivalence between the race-avoiding characteristic of an AT and the property that all $\rho \in ptruns(\mathcal{A})$ can be wiggled (Theorem 3). This allows us to provide logic formulas to determine whether an AT has an unwigglable run, and then to prove the upper bound of Theorem 2. We also discuss its lower bound. Finally, we discuss some sufficient hypotheses for a race-avoiding AT.

4.1   Wiggling a run

In this section, given an AT $\mathcal{A}$ and a padded timed run $\rho \in ptruns(\mathcal{A})$, we study the conditions required to be able to wiggle ρ. For this purpose, we define the

(a) Graph G_ρ.

(b) Graph G_π.

Fig. 4: Block graphs of the timed runs ρ and π of Example 3.

following graph G_ρ associated with ρ. When two blocks B and B' of ρ participate in a race, we write $B \prec B'$ if there exist actions $i \in B$ and $i' \in B'$ such that i, i' participate in this race and, according to Definition 3:

- either i occurs before i' along ρ and the total delay between i and i' is zero.
- or the timer of B' (re)started by i' reaches value zero when i discards it.

We define the *block graph* $G_\rho = (V, E)$ of ρ where V is the set of blocks of ρ and E has an edge (B, B') iff $B \prec B'$.

Example 7. Let $\mathcal{A}$ be the AT from Figure 1, and ρ and π be the timed runs from Example 3, whose block decomposition is represented in Figures 2a and 2c. For the run ρ, it holds that $B_2 \prec B_1$ leading to the block graph G_ρ depicted in Figure 4a. For the run π, we get the block graph G_π depicted in Figure 4b.

Notice that G_ρ is acyclic while G_π is cyclic. By the following proposition, this difference is enough to characterize that ρ can be wiggled and π cannot.

Proposition 1. *Let $\mathcal{A}$ be an AT and $\rho \in ptruns(\mathcal{A})$ be a padded timed run with races. Then, ρ can be wiggled iff G_ρ is acyclic.*

Intuitively, a block cannot be moved to the left (resp. right) if it has a predecessor (resp. successor) in the block graph, due to the races in which it participates. Hence, if a block has both a predecessor and a successor, it cannot be wiggled (see Figures 2c and 4b for instance). Then, the blocks appearing in a cycle of the block graph cannot be wiggled. The other direction holds by observing that we can do a topological sort of the blocks if the graph is acyclic. We then wiggle the blocks, one by one, according to that sort.

The next corollary will be useful in the following sections. It is illustrated by Figure 5 with the simple cycle $(B_0, B_1, B_2, B_3, B_4, B_0)$.

Corollary 1. *Let $\mathcal{A}$ be an AT and $\rho \in ptruns(\mathcal{A})$ be a padded timed run with races. Suppose that G_ρ is cyclic. Then there exists a cycle $\mathcal{C}$ in G_ρ such that*

- *any block of $\mathcal{C}$ participate in exactly two races described by this cycle,*
- *for any race described by $\mathcal{C}$, exactly two blocks participate in the race,*
- *the blocks $B = (k_1 \ldots k_m, \gamma)$ of $\mathcal{C}$ satisfy either $m \geq 2$, or $m = 1$ and $\gamma = \bullet$.*

From the definition of wiggling, we know that if all padded timed runs with races of an AT $\mathcal{A}$ are wigglable, then $\mathcal{A}$ is race-avoiding. The converse also holds as stated in the next theorem. By Proposition 1, this means that an AT is race-avoiding iff the block graph of all its padded timed run is acyclic.
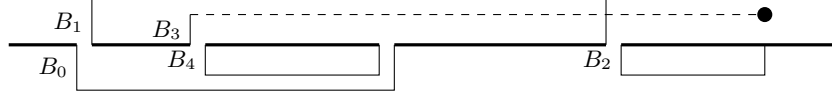
Fig. 5: Races of a padded timed run ρ with $B_\ell \prec B_{\ell+1 \bmod 5}$, $0 \leq \ell \leq 4$.

Theorem 3. *An AT $\mathcal{A}$ is race-avoiding*

- *iff any padded timed run $\rho \in ptruns(\mathcal{A})$ with races can be wiggled,*
- *iff for any padded timed run $\rho \in ptruns(\mathcal{A})$, its graph G_ρ is acyclic.*

Let us sketch the missing proof. By modifying $\rho \in ptruns(\mathcal{A})$ to explicitly encode when a timer is discarded, one can show the races of ρ cannot be avoided if the block graph of ρ is cyclic as follows. Given two actions i, i' of this modified run, it is possible to define the *relative elapsed time* between i and i', noted reltime(i, i'), from the sum d of all delays between i and i': if i occurs before i', then reltime$(i, i') = d$, otherwise reltime$(i, i') = -d$. Lifting this to a sequence of actions from ρ is defined naturally. Then, one can observe that the relative elapsed time of a cyclic sequence of actions is zero, i.e., reltime$(i_1, i_2, \ldots, i_k, i_1) = 0$. Finally, from a cycle of G_ρ as described in Corollary 1, we extract a cyclic sequence of actions and prove, thanks to the concept of relative elapsed time, that any run ρ' such that $untime(\rho) = untime(\rho')$ must contain some races.

4.2   Existence of an unwigglable run

In this section, we give the intuition for the announced complexity bounds for the problem of deciding whether an AT $\mathcal{A}$ is race-avoiding (Theorem 2).

Let us begin with the 3EXP-membership. The crux of our approach is to use the characterization of the race-avoiding property given in Theorem 3, to work with a slight modification of the region automaton $\mathcal{R}$ of $\mathcal{A}$, and to construct a finite-state automaton whose language is the set of runs of $\mathcal{R}$ whose block graph is cyclic. Hence deciding whether $\mathcal{A}$ is race-avoiding amounts to deciding whether the language accepted by the latter automaton is empty. To do so, we construct a *monadic second-order* (MSO, for short; see [10,15] for an introduction) formula that is satisfied by words w labeling a run ρ of $\mathcal{R}$ iff the block graph of ρ is cyclic.

Our *modification* of $\mathcal{R}$ is best seen as additional annotations on the states and transitions of $\mathcal{R}$. We extend the alphabet Σ as follows: *(i)* we add a timer to each action $i \in \hat{I}$ to remember the updated timers; *(ii)* we also use new symbols $di[x]$, $x \in X$, with the intent of explicitly encoding in $\mathcal{R}$ when the timer x is discarded while its value is zero. Therefore, the modified alphabet is $\Sigma = \{\tau\} \cup (\hat{I} \times \hat{X}) \cup \{di[x] \mid x \in X\}$ where $\hat{X} = X \cup \{\bot\}$. As a transition in $\mathcal{A}$ can discard more than one timer, we store the set D of discarded timers in the states of $\mathcal{R}$, as well as outgoing transitions labeled by $di[x]$, for all discarded timers x. For this, the states of $\mathcal{R}$ become $S = \{(q, [\![\kappa]\!]_\cong, D) \mid q \in Q, \kappa \in \mathsf{Val}(\chi(q)), D \subseteq X\}$ and Δ is modified in the natural way so that D is updated as required. Note

that the size of this modified $\mathcal{R}$ is only larger than what is stated in Lemma 2 by a factor of $2^{|X|}$.

Note that any x-block $(i_{k_1}, \ldots, i_{k_m}, \gamma)$ of a timed run ρ in $\mathcal{A}$ is translated into the sequence of symbols $(i'_{k_1}, \ldots, i'_{k_m}, \gamma')$ in the corresponding run ρ' of the modified $\mathcal{R}$ with an optional symbol γ' such that:

- $i'_{k_\ell} = (i_{k_\ell}, x)$, for $1 \leq \ell < m$,
- $i'_{k_m} = (i_{k_m}, \bot)$ if $\gamma = \bot$, and (i_{k_m}, x) otherwise,
- $\gamma' = di[x]$ if $\gamma = \bullet$, and γ' does not exist otherwise.

It follows that, instead of considering padded timed runs $\rho \in ptruns(\mathcal{A})$ and their block graph G_ρ, we work with their corresponding (padded) runs, blocks, and block graphs in the modified region automaton $\mathcal{R}$ of $\mathcal{A}$.

Example 8. The run π' of Example 5 becomes

$$(q_0, [\![\emptyset]\!]_\cong, \emptyset) \xrightarrow{\tau} (q_0, [\![\emptyset]\!]_\cong, \emptyset) \xrightarrow{(i,x_1)} (q_1, [\![x_1=1]\!]_\cong, \emptyset) \xrightarrow{(i,x_2)} (q_2, [\![x_1=1, x_2=2]\!]_\cong, \emptyset)$$

$$\xrightarrow{\tau} (q_2, [\![x_1=0, x_2=1]\!]_\cong, \emptyset) \xrightarrow{(i,x_1)} (q_2, [\![x_1=1, x_2=1]\!]_\cong, \{x_1\})$$

$$\xrightarrow{di[x_1]} (q_2, [\![x_1=1, x_2=1]\!]_\cong, \emptyset) \xrightarrow{\tau} (q_2, [\![x_1=0, x_2=0]\!]_\cong, \emptyset)$$

$$\xrightarrow{(to[x_2],\bot)} (q_1, [\![x_1=0]\!]_\cong, \emptyset) \xrightarrow{(to[x_1],x_1)} (q_1, [\![x_1=1]\!]_\cong, \emptyset) \xrightarrow{\tau} (q_1, [\![0<x_1<1]\!]_\cong, \emptyset).$$

The transition with label $di[x_1]$ indicates that the timer x_1 is discarded in the original timed run while its value equals zero (see the race in which blocks B_1 and B_3 participate in Figure 2c). The three blocks of π become $B'_1 = ((i, x_1), di[x_1])$, $B'_2 = ((i, x_2), (to[x_2], \bot))$, and $B'_3 = ((i, x_1), (to[x_1], x_1))$ in π'. The fact that in π, B_1 and B_2 participate in a race (with a zero-delay between their respective actions i and i), is translated in π' with the non existence of the τ-symbol between the symbols (i, x_1) and (i, x_2) in B'_1 and B'_2 respectively.

Lemma 3. *Let $\mathcal{A}$ be an AT and $\mathcal{R}$ be its modified region automaton. We can construct an MSO formula Φ of size linear in I and X such that a word labeling a run ρ of $\mathcal{R}$ satisfies Φ iff ρ is a padded run that cannot be wiggled. Moreover, the formula Φ, in prenex normal form, has three quantifier alternations.*

The formula Φ of this lemma describes the existence of a cycle $\mathcal{C}$ of blocks $B_0, B_1, \ldots, B_{k-1}$ such that $B_\ell \prec B_{\ell+1 \bmod k}$ for any $0 \leq \ell \leq k-1$, as described in Corollary 1 (see Figure 5). To do so, we consider the actions (i.e., symbols of the alphabet Σ of $\mathcal{R}$) participating in the races of $\mathcal{C}$: $i_0, i_1, \ldots, i_{k-1}$ and $i'_0, i'_1, \ldots, i'_{k-1}$ such that for all ℓ, i_ℓ, i'_ℓ belong to the same block, and $i'_\ell, i_{\ell+1 \bmod k}$ participate in a race. One can write MSO formulas expressing that two actions participate in a race (there is no τ transition between them), that two actions belong to the same block, and, finally, the existence of these two action sequences.

From the formula Φ of Lemma 3, by the Büchi-Elgot-Trakhtenbrot theorem, we can construct a finite-state automaton whose language is the set of all words satisfying Φ. Its size is triple-exponential. We then compute the intersection $\mathcal{N}$

of this automaton with $\mathcal{R}$ — itself exponential in size. Finally, the language of $\mathcal{N}$ is empty iff each padded timed run of $\mathcal{A}$ can be wiggled, and this can be checked in polynomial time with respect to the triple-exponential size of $\mathcal{N}$, thus showing the 3EXP-membership of Theorem 2. Notice that when we fix the sets of inputs I and of timers X, as the formula Φ becomes of constant size, the automaton $\mathcal{N}$ has now exponential size. Checking its emptiness can be done "on the fly", yielding a nondeterministic decision procedure which requires a polynomial space only. We thus obtain that, under fixed inputs and timers, deciding whether an AT is race-avoiding is in PSPACE.

The complexity lower bound of Theorem 2 follows from the PSPACE-hardness of the reachability problem for ATs (see the intuition given in Section 3). We can show that any run in the AT constructed from the given LBTM and word w can be wiggled. Once the designated state for the reachability reduction is reached, we add a widget that forces a run that cannot be wiggled. Therefore, as the only way of obtaining a run that cannot be wiggled is to reach a specific state (from the widget), the problem whether an AT is race-avoiding is PSPACE-hard. Notice that this hardness proof is no longer valid if we fix the sets I and X.

4.3   Sufficient hypotheses

Let us discuss some sufficient hypotheses for an AT $\mathcal{A}$ to be race-avoiding.

1. If every state in $\mathcal{A}$ has at most one active timer, then $\mathcal{A}$ is race-avoiding. Up to renaming the timers, we actually have a single-timer AT in this case.
2. If we modify the notion of timed run by imposing non-zero delays everywhere in the run, then $\mathcal{A}$ is race-avoiding. Indeed, the only races that can appear are when a zero-valued timer is discarded, and it is impossible to form a cycle in the block graph with only this kind of races. Imposing a non-zero delay before a timeout is debateable. Nevertheless, imposing a non-zero delay before inputs only is not a sufficient hypothesis (we have counter-examples).
3. Let us fix a total order $<$ over the timers, and modify the semantics of an AT to enforce that, in a race, any action of an x-block is processed before an action of a y-block, if $x < y$ (x is preemptive over y). Then the AT is race-avoiding. Towards a contradiction, assume there are blocks $B_0, B_1, \ldots, B_{k-1}$ forming a cycle as described in Corollary 1, where each B_i is an x_i-block. By the order and the races, we get $x_0 \leq x_1 \leq \ldots \leq x_{k-1} \leq x_0$, i.e., we have a single timer (as in the first hypothesis). Hence, it is always possible to wiggle, which is a contradiction.

5   Conclusion and future work

In this paper, we studied automata with timers. We proved that the reachability problem for ATs is PSPACE-complete. Moreover, given a padded timed run in an AT, we defined a decomposition of its actions into blocks, and provided a way to remove races (concurrent actions) inside the run by wiggling blocks one by one.

We also proved that this notion of wiggling is necessary and sufficient to decide whether an AT is race-avoiding. Finally, we showed that deciding whether an AT is race-avoiding is in 3EXP and PSPACE-hard.

For future work, it may be interesting to tighten the complexity bounds for the latter decision problem, both when fixing the sets I and X and when not. A second important direction, which we plan to pursue, is to work on a learning algorithm for ATs, as initiated in [17] with Mealy machines with one timer. This would allow us to construct ATs from real-world systems, such as network protocols, in order to verify that these systems behave as expected.

References

1. Aceto, L., Laroussinie, F.: Is your model checker on time? on the complexity of model checking for timed modal logics. J. Log. Algebraic Methods Program. **52-53**, 7–51 (2002). `https://doi.org/10.1016/S1567-8326(02)00022-X`
2. Aichernig, B.K., Pferscher, A., Tappler, M.: From Passive to Active: Learning Timed Automata Efficiently. In: Lee, R., Jha, S., Mavridou, A. (eds.) NFM'20. LNCS, vol. 12229, pp. 1–19. Springer (2020)
3. Alur, R.: Timed automata. In: Computer Aided Verification: 11th International Conference, CAV'99 Trento, Italy, July 6–10, 1999 Proceedings 11. pp. 8–22. Springer (1999)
4. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). `https://doi.org/10.1016/0304-3975(94)90010-8`
5. An, J., Chen, M., Zhan, B., Zhan, N., Zhang, M.: Learning one-clock timed automata. In: Biere, A., Parker, D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12078, pp. 444–462. Springer (2020). `https://doi.org/10.1007/978-3-030-45190-5_25`
6. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). `https://doi.org/10.1016/0890-5401(87)90052-6`
7. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
8. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Ouaknine, J., Worrell, J.: Model checking real-time systems. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 1001–1046. Springer (2018). `https://doi.org/10.1007/978-3-319-10575-8_29`
9. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.): Handbook of Model Checking. Springer (2018). `https://doi.org/10.1007/978-3-319-10575-8`
10. Grädel, E., Thomas, W., Wilke, T.: Automata, logics, and infinite games: a guide to current research, vol. 2500. Springer (2003)
11. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. Theor. Comput. Sci. **411**(47), 4029–4054 (2010). `https://doi.org/10.1016/j.tcs.2010.07.008`
12. Grinchtein, O., Jonsson, B., Pettersson, P.: Inference of event-recording automata using timed decision trees. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4137, pp. 435–449. Springer (2006). `https://doi.org/10.1007/11817949_29`

13. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
14. Howar, F., Steffen, B.: Active automata learning in practice - an annotated bibliography of the years 2011 to 2016. In: Bennaceur, A., Hähnle, R., Meinke, K. (eds.) Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers. Lecture Notes in Computer Science, vol. 11026, pp. 123–148. Springer (2018). `https://doi.org/10.1007/978-3-319-96562-8_5`
15. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, Volume 3: Beyond Words, pp. 389–455. Springer (1997). `https://doi.org/10.1007/978-3-642-59126-6_7`
16. Tripakis, S., Yovine, S.: Analysis of timed systems using time-abstracting bisimulations. Formal Methods Syst. Des. **18**(1), 25–68 (2001). `https://doi.org/10.1023/A:1008734703554`
17. Vaandrager, F., Ebrahimi, M., Bloem, R.: Learning Mealy machines with one timer. Information and Computation p. 105013 (2023). `https://doi.org/https://doi.org/10.1016/j.ic.2023.105013`

A   Proof of PSPACE lower bound of Theorem 1

Theorem 1. *The reachability problem for ATs is* PSPACE-*complete.*

A *linear-bounded Turing machine* (LBTM, for short) $\mathcal{M} = (\Sigma, Q, q_0, q_F, T)$ is a nondeterministic Turing machine which can only use $|w|$ cells of the tape to determine whether an input w is accepted. Formally, Σ is a finite alphabet, Q is a finite set of states, q_0 and q_F are the initial and final states respectively, and $T \subseteq Q \times \Sigma \times \Sigma \times \{L, R\} \times Q$ is the set of transitions. A configuration of $\mathcal{M}$ is a triple $(q, w, i) \in Q \times \Sigma^* \times \mathbb{N}^{>0}$ where q denotes the current control state, $w = w_1 \ldots w_n$ is the content of the tape, and i is the position of the tape head. We say a transition $(q, \alpha, \alpha', D, q') \in T$ is enabled in a configuration (q, w, i) if $w_i = \alpha$. In that case, taking the transition results in the machine reaching the new configuration (q', w', i') with $w'_i = \alpha'$, $w'_j = w_j$ for all $j \neq i$, and $i' = i + 1$ if $D = R$ or $i' = i - 1$ otherwise. A given word w is said to be accepted by $\mathcal{M}$ if there is a sequence of transitions from $(q_0, w, 1)$ to a configuration of the form (q_F, w', i). Deciding whether a given LBTM accepts a given word is PSPACE-complete [13].

Proof (of Theorem 1 (lower bound)). We show that the acceptance problem for LBTMs can be reduced in polynomial time to the reachability problem for ATs. The proof is inspired by the one presented by Aceto and Laroussinie for the fact that reachability is PSPACE-hard for timed automata [1, Section 3.1].

Let $\mathcal{M} = (\Sigma, Q, q_0, q_F, T)$ be an LBTM and let $w \in \Sigma^*$ be an input word with $|w| = n$. From $\mathcal{M}$ and w we are going to build, in polynomial time, an AT $\mathcal{A}_{\mathcal{M},w}$ such that w is accepted by $\mathcal{M}$ iff there exists a timed run that ends in a specific state r_{done} of $\mathcal{A}_{\mathcal{M},w}$.[7]

Let $\Sigma = \{a_1, \ldots, a_k\}$. Then, every $2k + 2$ time units the AT will simulate a single step of the LBTM. This is what we call a *phase*.

Let $T = \{t_1, \ldots, t_m\}$ be the set of transitions of $\mathcal{M}$. Then, $\mathcal{A}_{\mathcal{M},w}$ has inputs $I = \{go\} \cup T$ and timers $X = \{x, x_1, \ldots, x_n\}$. Now, a state of $\mathcal{A}_{\mathcal{M},w}$ is an element of $\{r_0, \ldots, r_n, r_{done}, r_{sink}\}$ (with r_0 the initial state) or a tuple $\langle q, i, symbol, clock \rangle$, where:

- $q \in Q$ records the current state of the LBTM,
- $i \in \{1, \ldots, n\}$ records the current position of the tape head,
- $symbol \in \{0, \ldots, k\}$ records the index of the last symbol read from the tape (0 when no symbol has been read or the last read symbol has been processed),
- $clock \in \{0, 1, \ldots, 2k + 1\}$.

The AT starts with an initialization phase in which it goes through states r_0 to r_n to start all the timers via a sequence of *go*-inputs. Execution begins with a *go*-input that starts timer x: $r_0 \xrightarrow[(x,1)]{go} r_1$. In order to reach state r_{done}, all the

---

[7] In the following, we rely on the notion of blocks to pass on the intuition. While blocks are only defined on padded timed runs, it is easy to check that any timed run reaching r_{done} can be turned into a padded run.
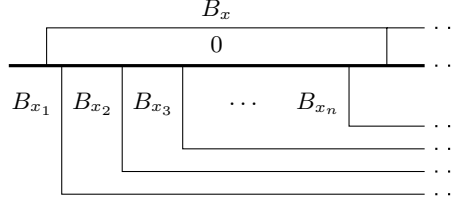
Fig. 6: Starting blocks.

other timers need to be started before timer x times out. We use timer x_i, for $1 \leq i \leq n$, to record the value of the i-th tape cell: if this value is symbol $a_j \in \Sigma$, then timer x_i will time out when $\lfloor clock/2 \rfloor = j$ (it will become clear later how this is possible). Timer x_i is started in state r_i and set to its appropriate value:

- if $i < n$, we have the transition $r_i \xrightarrow[(x_i, 2j)]{go} r_{i+1}$ that initializes timer x_i with value $2j$,
- if $i = n$, we have the transition $r_n \xrightarrow[(x_n, 2j)]{go} \langle q_0, 1, 0, 0 \rangle$ that initializes timer x_n with the same value $2j$ and starts the computation of the LBTM.

All timeout transitions from r_i, $1 \leq i \leq n$, go to r_{sink}. This ensures that — in order to reach state r_{done}— all timers x_i are initialized. Hence, all timed runs in $\mathcal{A}_{\mathcal{M},w}$ that reach r_{done} have the same starting blocks, as depicted in Figure 6: an x-block B_x and n x_i-blocks B_{x_i}.[8]

We use timer x to advance the value of $clock$ that runs cyclically from 0 to $2k + 1$:

$$clock > 0 \lor symbol = 0 \quad \Rightarrow$$
$$\langle q, i, symbol, clock \rangle \xrightarrow[(x,1)]{to[x]} \langle q, i, symbol, (clock + 1) \bmod 2k + 2 \rangle. \tag{4}$$

(It will become clear later why the condition on this transition is required.) When timer x_ℓ times out, for some $\ell \neq i$, then we just restart it so that it will times out at exactly the same point in the next phase:
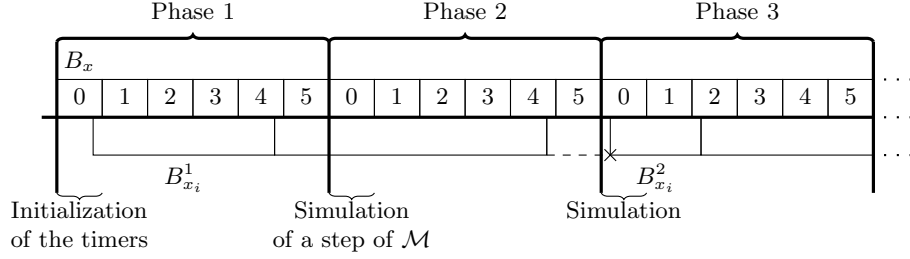
$$\langle q, i, symbol, clock \rangle \xrightarrow[(x_\ell, 2k+2)]{to[x_\ell]} \langle q, i, symbol, clock \rangle.$$

When timer x_i times out, then we restart it in the same way, but in addition we store the index of the symbol that it encodes in the state of the AT:

$$\langle q, i, symbol, clock \rangle \xrightarrow[(x_i, 2k+2)]{to[x_i]} \langle q, i, \lfloor clock/2 \rfloor, clock \rangle.$$

In order to see why this is true, suppose timer x_i has been started at time d with value $2j$. Then, $d \in [0, 1]$ (see Figure 6) and x_i will expire at time $d' = d + 2j$,

---

[8] Notice that some of these blocks participate in a race if their timers are initialized at the same time. For instance, all timers can be started at the same time $d = 0$.

Fig. 7: The beginning of a timed run that reaches r_{done} with $k = 2$.

so $d' \in [2j, 2j + 1]$. At this time, the value of *clock* will be either $2j$ or $2j + 1$, and thus $j = \lfloor clock/2 \rfloor$.

When the value of *clock* becomes 0 again (a next phase begins), the LBTM $\mathcal{M}$ has read a symbol from the tape, so *symbol* > 0, and $\mathcal{M}$ may (nondeterministically) take a transition. For each transition $t = (q, \alpha, \alpha', q', L)$ of $\mathcal{M}$, with $\alpha = a_{symbol}$ and $\alpha' = a_j$, the AT has a transition:

$$\langle q, i, symbol, 0 \rangle \xrightarrow[(x_i, 2j)]{t} \langle q', i - 1, 0, 0 \rangle.$$

The AT also has transitions, mutatis mutandis, for each $t = (q, \alpha, \alpha', q', R)$ of the LBTM. In order to ensure that these transitions are taken before timer x times out, we add transitions:

$$symbol > 0 \quad \Rightarrow \quad \langle q, i, symbol, 0 \rangle \xrightarrow{to[x]} r_{sink}. \tag{5}$$

The condition *clock* $> 0 \lor symbol = 0$ from (4) and the condition *symbol* > 0 from (5) both ensure that we see first action t and then action $to[x]$ in a timed run that reaches r_{done}.

In Figure 7, we fix $k = 2$, and, for a timed run ρ that reaches r_{done}, we indicate the sequence of phases, with the cyclic value of *clock* from 0 to $2k + 1$. We also indicate the block B_x such that timer x is restarted each time it times out along ρ. Finally, we indicate two x_i-blocks, $B_{x_i}^1$ and $B_{x_i}^2$, such that in $B_{x_i}^1$, timer x_i is started in phase 1, restarted during this phase, and restarted again during phase 2, until it is discarded when *clock* $= 0$ in phase 3; and in $B_{x_i}^2$, timer x_i is started with a new value dictated by the processed transition of the LBTM. Note that there may be other blocks (for timers x_j, with $j \neq i$) which are not represented in the figure.

As soon as the LBTM reaches q_F, the AT may proceed to its final state r_{done}:

$$\langle q_F, i, symbol, clock \rangle \xrightarrow{go} r_{done}.$$

For all states of the AT, outgoing transitions for actions that have not been specified lead to r_{sink}.

Finally, we define the active timers in each state of the AT as follows: $\chi(r_0) = \chi(r_{sink}) = \chi(r_{done}) = \emptyset$, $\chi(r_i) = \{x, x_1, x_2, \ldots, x_{i-1}\}$ for $1 \leq i \leq n$, and $\chi(r) = \{x, x_1, x_2, \ldots, x_n\}$ for all the other states r.

It is clear that $\mathcal{A}_{\mathcal{M},w}$ can be constructed from $\mathcal{M}$ and w in polynomial time and that r_{done} is reachable in the AT iff the LBTM accepts w. $\qquad\square$

B   Proof of Lemma 2

Lemma 2. *Let $\mathcal{A} = (X, I, Q, q_0, \chi, \delta)$ be an AT and $\mathcal{R}$ be its region automaton.*

1. *The size of $\mathcal{R}$ is linear in $|Q|$ and exponential in $|X|$. That is, $|S|$ is smaller than or equal to $|Q| \cdot |X|! \cdot 2^{|X|} \cdot (C+1)^{|X|}$.*
2. *There is a timed run ρ of $\mathcal{A}$ that begins in (q, κ) and ends in (q', κ') iff there is a run ρ' of $\mathcal{R}$ that begins in $[\![(q, \kappa)]\!]_{\cong}$ and ends in $[\![(q', \kappa')]\!]_{\cong}$.*

Proof (of Lemma 2). For the first statement of the lemma, recall that each state of the region automaton is of the form $(q, [\![\kappa]\!]_{\cong})$. The number of states q is equal to $|Q|$. Concerning the number of region classes $[\![\kappa]\!]_{\cong}$, $(C+1)^{|X|}$ is related to the integer parts of the timers between 0 and C, $2^{|X|}$ to which timers have a zero fractional part, and $|X|!$ to the order of these fractional parts.

The second statement of the lemma follows from the definition of the region automaton $\mathcal{R}$. Notice that delay transitions with delay 0 in $\mathcal{A}$ disappear in $\mathcal{R}$.
$\qquad\square$

C   Proof of PSPACE upper bound of Theorem 1

Theorem 1. *The reachability problem for ATs is* PSPACE-*complete.*

Proof (upper bound of Theorem 1). To decide the reachability problem for ATs, by Lemma 2, we can simulate a run of the corresponding region automaton. Instead of constructing the region automaton in full, we can do so "on the fly". This yields a nondeterministic decision procedure for the reachability problem which, due to the form $(q, [\![\kappa]\!]_{\cong})$ of the states of $\mathcal{R}$, requires polynomial space only. Since NPSPACE = PSPACE, we obtain the upper bound stated in Theorem 1. $\qquad\square$

D   Proof of Proposition 1

Proposition 1. *Let $\mathcal{A}$ be an AT and $\rho \in ptruns(\mathcal{A})$ be a padded timed run with races. Then, ρ can be wiggled iff G_ρ is acyclic.*

Before establishing this result, we prove the following intermediate result.

Lemma 4. *Let G_ρ be the block graph of $\rho \in ptruns(\mathcal{A})$ and B be a block in this graph. It is impossible to wiggle B iff B has at least one predecessor and at least one successor in G_ρ.*

Proof. We prove the lemma by showing both directions.

$\Leftarrow$ Suppose that B has a block B' as predecessor and B'' as successor, that is, $B' \prec B$ and $B \prec B''$. Notice that it may be that $B' = B''$. However, $B \neq B'$ and $B \neq B''$ by the definition of races. Let us prove that it is impossible to wiggle B by arguing that it is not feasible to move B to the right nor to the left. Given $B' \prec B$, let us prove that we cannot move B to the left. We have two cases for the actions $i \in B$ and $i' \in B'$ that participate in the race:

- Action i' occurs before action i along ρ and the sum of the delays between these two actions is zero. Thus the delay d before i in ρ is equal to zero and it is impossible to have $d + \epsilon \geq 0$ for any $\epsilon < 0$. This implies that no movement of B to the left is possible.
- The timer x of B (re)started by i reaches value zero when i' discards it. By moving B to the left by some $\epsilon < 0$, x times out and therefore an action $to[x]$ occurs, while it does not occur in ρ (as i' discards x). As we want to keep the same untimed trace as for ρ, it is impossible to move B to the left.

With symmetrical arguments, we obtain that we cannot move B to the right, as $B \prec B''$. We conclude that we cannot wiggle B.

$\Rightarrow$ We prove this direction by contraposition. We first assume that B has no predecessor (however it may have successors C). We argue that B can be wiggled by moving it to the left. From the definition of a race, we obtain that:

- Since ρ is a padded timed run, the first delay d_1 of ρ is non-zero.
- For each action i' before some action $i \in B$ such that $i' \notin B$, the delay $d_{i'}$ between i' and i must be non-zero (as B has no predecessor).
- The timer fate γ_B of B is either $\bot$ or $\times$. Indeed, assume by contradiction that $\gamma_B = \bullet$. Recall that, by definition of a padded run, timers cannot have a zero value at the end of a run. Therefore, there must exist a block B'' that discards the timer of B while its valuation is zero. Thus, we have that $B'' \prec B$ which is not possible.

From these observations, we conclude that there is enough room to move B to the left. Indeed, it is possible to choose some $\epsilon < 0$ with $|\epsilon|$ small enough, such that $d_1 + \epsilon > 0$, $d_{i'} + \epsilon > 0$ for all the delays $d_{i'}$ mentioned above, and in a way that if $\gamma_B = \times$ then the new timer fate of B is still equal to $\times$. In this way we produce a timed run ρ' such that $untime(\rho) = untime(\rho')$.

It remains to explain that the blocks C participating in a race with B in ρ no longer participate in such a race in ρ'. Let C be one of these blocks, hence $B \prec C$. We have again two cases:

- There exist $i \in B$ and $i' \in C$ such that i occurs before i' in ρ and the total delay between them is zero. In the timed run ρ', this delay becomes equal to $-\epsilon > 0$ and B, C no longer participate in a race.
- The timer fate γ_C of C is equal to $\bullet$ and the timer of C is discarded by B along ρ. In ρ', we get that $\gamma_C = \times$ and B, C no longer participate in a race.

It follows that if B has no predecessor, we can wiggle it. With symmetrical arguments, if B has no successor, we can also wiggle it. Hence, the lemma holds. $\square$

Now, we proceed to proving Proposition 1.

Proof (of Proposition 1). We prove the equivalence by showing both directions.

$\Rightarrow$ We prove this direction by contraposition. Suppose that G_ρ has a cycle that we can assume to be simple, i.e., there are $k > 1$ distinct blocks B_ℓ, $0 \leq \ell \leq k - 1$ such that $B_\ell \prec B_{\ell+1 \bmod k}$. As every block B_ℓ has a predecessor and a successor in this cycle, we cannot wiggle B_ℓ by Lemma 4. Thus, ρ cannot be wiggled as it is impossible to resolve the races in which the blocks B_ℓ participate.
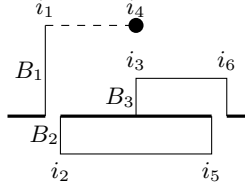
$\Leftarrow$ Assume G_ρ is acyclic. Hence, we can compute a topological sort of G_ρ restricted to the blocks participating in the races of ρ. Let B be the greatest block with respect to this sort, that is, B has no successor and it has predecessors. By Lemma 4, we can wiggle B by moving it slightly to the right, thus eliminating the races between B and all the other blocks. We thus obtain a new timed run ρ' such that $untime(\rho) = untime(\rho')$ and $G_{\rho'}$ has the same vertices as G_ρ and strictly less edges (B becomes an isolated vertex). We repeat this process until the blocks of the graph are all isolated, meaning that the resulting timed run has no races and the same untimed trace as ρ. $\square$

E Proof of Corollary 1

Corollary 1. *Let $\mathcal{A}$ be an AT and $\rho \in ptruns(\mathcal{A})$ be a padded timed run with races. Suppose that G_ρ is cyclic. Then there exists a cycle $\mathcal{C}$ in G_ρ such that*

- *any block of $\mathcal{C}$ participate in exactly two races described by this cycle,*
- *for any race described by $\mathcal{C}$, exactly two blocks participate in the race,*
- *the blocks $B = (k_1 \ldots k_m, \gamma)$ of $\mathcal{C}$ satisfy either $m \geq 2$, or $m = 1$ and $\gamma = \bullet$.*

Proof. As G_ρ is cyclic, we consider a cycle of minimal length. First notice that a block B of this cycle can only appear once per race. Indeed, the value at which a timer is (re)started in the block is positive, thus imposing non-zero delays between two actions of B (i.e., two actions of B can not participate in a common race). Second, by minimality of its length, the cycle is simple, implying that each of its blocks participates in exactly two races, one with its unique successor (in the cycle) and another one with its unique predecessor. Third, assume that three blocks B_1, B_2 and B_3 participate in a common race, in that order. By the previous remark, they are pairwise distinct, and it must be that $B_1 \prec B_2$, $B_2 \prec B_3$ and $B_1 \prec B_3$. It follows that we get a smaller cycle by eliminating B_2, which is a contradiction. Finally, assume that the cycle contains some block $B = (k_1, \gamma)$ with $\gamma \neq \bullet$. Let $B_1 \prec B$ (resp. $B \prec B_2$) be the predecessor (resp. successor) of B in the cycle. Due to the form of B, the three blocks B_1, B_2 and B participate in the same race, which is impossible. $\square$

Fig. 8: The extended run of π and its block decomposition.

F   Proof of Theorem 3

Theorem 3. *An AT $\mathcal{A}$ is race-avoiding*

– *iff any padded timed run $\rho \in ptruns(\mathcal{A})$ with races can be wiggled,*
– *iff for any padded timed run $\rho \in ptruns(\mathcal{A})$, its graph G_ρ is acyclic.*

For the first equivalence, we only need to prove one implication of this result, as by definition of wiggling, an AT $\mathcal{A}$ is race-avoiding if all its padded timed runs with races are wigglable. The second equivalence is then a consequence of Proposition 1.

For this purpose, we need to introduce some new notions. Given a padded timed run $\rho = (q_0, \kappa_0)\ d_1\ i_1/u_1\ \ldots\ d_n\ i_n/u_n\ d_{n+1}\ (q, \kappa)$, we extend it with additional transitions indicating when a timer has been discarded in the following way.[9] Let $(q_{\ell-1}, \kappa_{\ell-1})\ d_\ell\ i_\ell/u_\ell\ (q_\ell, \kappa_\ell)$ be any transition of ρ such that the set $D = \{y_1, \ldots, y_m\}$ of timers discarded by i_ℓ is not empty. Then we replace (q_ℓ, κ_ℓ) by the sequence of transitions

$$(q_\ell, \kappa_\ell)\ 0\ j_1/\bot\ (q_\ell, \kappa_\ell)\ 0\ j_2/\bot\ (q_\ell, \kappa_\ell)\ \cdots\ (q_\ell, \kappa_\ell)\ 0\ j_m/\bot\ (q_\ell, \kappa_\ell)$$

such that

– for all k, $1 \leq k \leq m$, if y_k was discarded by i_ℓ when its value was zero, then $j_k = \bullet$, otherwise $j_k = \times$,
– each delay is zero, and
– each update is $\bot$.

We denote by $ext(\rho)$ the resulting extended run, such that symbols $\bullet$ and $\times$ are also called actions.

Example 9. Let us come back to the timed run π of Example 3 (see also Figure 2c). Its transition $(q_2, x_1 = 1, x_2 = 2)\ 1\ i/(x_1, 1)\ (q_2, x_1 = 1, x_2 = 1)$ discards timer x_1 when its value is zero. Therefore in $ext(\pi)$, we have $(q_2, x_1 = 1, x_2 = 1)$ being replaced by $(q_2, x_1 = 1, x_2 = 1)\ 0\ \bullet/\bot\ (q_2, x_1 = 1, x_2 = 1)$. This extended run and its block decomposition are depicted in Figure 8, such that $i_1, i_2, \ldots, i_7$ is the sequence of actions along $ext(\pi)$ with $i_4 = \bullet$.

---

[9] In Section 4.2, this was done in the modified region automaton $\mathcal{R}$ of $\mathcal{A}$ with the new symbols $di[x]$ indicating that the timer x was discarded when its value was zero. We here also consider the case when x is discarded with a non-zero value. To get more intuition about the proof of Theorem 3, we recommend reading Appendix G.

Let ρ be a padded timed run and $\mathrm{ext}(\rho)$ be its extended run. Given two actions i and i' of $\mathrm{ext}(\rho)$, the *relative elapsed time* between i and i', denoted by $\mathrm{reltime}_\rho(i, i')$, is defined as follows from the sum d of all delays between i and i' in $\mathrm{ext}(\rho)$: if i occurs before i', then $\mathrm{reltime}_\rho(i, i') = d$, otherwise $\mathrm{reltime}_\rho(i, i') = -d$. Notice that the relative elapsed time is sensitive to the order of the actions along $\mathrm{ext}(\rho)$, and if i and i' participate in a race, then $\mathrm{reltime}_\rho(i, i') = \mathrm{reltime}_\rho(i', i) = 0$. We naturally lift this definition to a sequence of actions $i_1, i_2, \ldots, i_k$ as

$$\mathrm{reltime}_\rho(i_1, i_2, \ldots, i_k) = \sum_{\ell=1}^{k-1} \mathrm{reltime}_\rho(i_\ell, i_{\ell+1}).$$

The following lemma is trivial, as the relative elapsed time between two actions has a sign that depends on the relative position of the actions. It is illustrated by Example 10 below.

Lemma 5. *If the sequence $i_1, i_2, \ldots, i_k$ is a cycle (that is, $k \geq 3$ and $i_k = i_1$), then $\mathrm{reltime}_\rho(i_1, i_2, \ldots, i_k) = 0$.*

Example 10. We consider again the timed run π and its extended run $\mathrm{ext}(\pi)$. Recall that π cannot be wiggled as G_π is cyclic (see Figure 4b). From this cycle and the block decomposition of $\mathrm{ext}(\pi)$ (see Figure 8), we extract the following sequence of actions: $i_1, i_2, i_5, i_6, i_3, i_4, i_1$. Notice that it is a cycle such that any two consecutive actions are either in the same block, or participate in a race, and are enumerated in a way to "follow" the cycle $B_1 \prec B_2 \prec B_3 \prec B_1$ of G_π. For instance, the first two actions i_1, i_2 describes the race $B_1 \prec B_2$, then i_2, i_5 both belong to B_2, then i_5, i_6 describes the race $B_2 \prec B_3$, etc. We have $\mathrm{reltime}_\pi(i_1, i_2, i_5, i_6, i_3, i_4, i_1) = 0 + 2 + 0 - 1 + 0 - 1 = 0$.

We now proceed to the proof of Theorem 3.

Proof (of Theorem 3). The second equivalence holds by Proposition 1. Let us focus on proving that $\mathcal{A}$ is race-avoiding iff any padded timed run $\rho \in ptruns(\mathcal{A})$ is wigglable. As, by definition, it is obvious that $\mathcal{A}$ is race-avoiding if any padded timed run is wigglable, we show the other direction.

Towards a contradiction, assume $\mathcal{A}$ is race-avoiding and there exists $\rho_1 \in ptruns(\mathcal{A})$ with races that is not wigglable. Since $\mathcal{A}$ is race-avoiding, there exists another padded timed run ρ_2 without races and such that $untime(\rho_1) = untime(\rho_2)$. We consider the two extended runs $\mathrm{ext}(\rho_1)$ and $\mathrm{ext}(\rho_2)$.

By Proposition 1, there must exist a cycle $\mathcal{C}$ in the block graph of ρ_1. We assume that $\mathcal{C}$ is as described in Corollary 1 and we study it on $\mathrm{ext}(\rho_1)$ (instead of ρ_1). That is, $\mathcal{C}$ is composed of $k > 1$ distinct blocks B_ℓ, $0 \leq \ell \leq k-1$, such that $B_\ell \prec B_{\ell+1 \bmod k}$, and

- any block B_ℓ participates in exactly two races described by $\mathcal{C}$,
- for any race described by $\mathcal{C}$, exactly two blocks participate in the race,
- the blocks of the cycle have at least two actions (of which one can be ●).

We thus have the following sequence of actions from $\mathrm{ext}(\rho_1)$

$$\mathcal{S}_1 = i'_0, i_1, i'_1, \ldots, i_{k-1}, i'_{k-1}, i_0, i'_0$$

that is a cycle and such that for all ℓ, $0 \le \ell < k$ (see also Example 10):

- i_ℓ and i'_ℓ are the two symbols of B_ℓ that participate in (different) races of $\mathcal{C}$,
- $i'_\ell \in B_\ell$ and $i_{\ell+1 \bmod k} \in B_{\ell+1 \bmod k}$ participate in a race of $\mathcal{C}$, that is, $\mathrm{reltime}_{\rho_1}(i'_\ell, i_{\ell+1 \bmod k}) = 0$,
- i'_ℓ occurs before $i_{\ell+1 \bmod k}$ in $\mathrm{ext}(\rho_1)$ (since $B_\ell \prec B_{\ell+1 \bmod k}$).

By Lemma 5, we have $\mathrm{reltime}_{\rho_1}(\mathcal{S}_1) = 0$. Therefore,

$$\mathrm{reltime}_{\rho_1}(\mathcal{S}_1) = \sum_{\ell=0}^{k-1} \mathrm{reltime}_{\rho_1}(i_\ell, i'_\ell) = 0. \tag{6}$$

Let us now study $\mathrm{ext}(\rho_2)$ knowing that $\mathit{untime}(\rho_1) = \mathit{untime}(\rho_2)$. Both padded timed runs ρ_1 and ρ_2 (and thus $\mathrm{ext}(\rho_1)$ and $\mathrm{ext}(\rho_2)$) must have the same block decomposition. Indeed recall that $\mathcal{A}$ is deterministic and we see the same actions. Hence, it must be that ρ_1 and ρ_2 follow the same transitions, with the same updates alongside both runs. We then have the same sequences of triggered actions, and therefore the same block decomposition in both runs.

We can thus consider the blocks of $\mathcal{C}$ seen as blocks in $\mathrm{ext}(\rho_2)$, and the sequence $\mathcal{S}_2 = j'_0, j_1, j'_1 \ldots, j_{k-1}, j'_{k-1}, j_0, j'_0$ from $\mathrm{ext}(\rho_2)$ that corresponds to the sequence $\mathcal{S}_1$ from $\mathrm{ext}(\rho_1)$. We have the following properties for all $\ell, 0 \le \ell < k$:

- if $i_\ell \in \hat{I}$ (resp. $i'_\ell \in \hat{I}$), then $j_\ell = i_\ell$ ($j'_\ell = i'_\ell$),
- if $i_\ell = \bullet$ (resp. $i'_\ell = \bullet$), then $j_\ell = \times$ ($j'_\ell = \times$), as ρ_2 has no races and $\mathit{untime}(\rho_1) = \mathit{untime}(\rho_2)$,
- if $i_\ell, i'_\ell \in \hat{I}$, then $\mathrm{reltime}_{\rho_1}(i_\ell, i'_\ell) = \mathrm{reltime}_{\rho_2}(j_\ell, j'_\ell)$, as $i_\ell = j_\ell, i'_\ell = j'_\ell$ are in the same block in both $\mathrm{ext}(\rho_1)$ and $\mathrm{ext}(\rho_2)$,
- if one among i_ℓ, i'_ℓ is equal to $\bullet$, then $|\mathrm{reltime}_{\rho_1}(i_\ell, i'_\ell)| > |\mathrm{reltime}_{\rho_2}(j_\ell, j'_\ell)|$ as the corresponding action in $\mathrm{ext}(\rho_2)$ is equal to $\times$ (the timer has been discarded earlier in ρ_2 than in ρ_1),
- we have $\mathrm{reltime}_{\rho_2}(j'_\ell, j_{\ell+1 \bmod k}) \ne 0$, as ρ_2 has no races,
- moreover, $\mathrm{reltime}_{\rho_2}(j'_\ell, j_{\ell+1 \bmod k}) > 0$ because i'_ℓ occurs before $i_{\ell+1 \bmod k}$ in $\mathrm{ext}(\rho_1)$ and $\mathit{untime}(\rho_1) = \mathit{untime}(\rho_2)$.

Let us first assume that no action $\bullet$ ever appears in $\mathcal{S}_1$. Hence, there is no action $\times$ in $\mathcal{S}_2$. As $\mathcal{S}_2$ is a cycle, by Lemma 5, we have that $\mathrm{reltime}_{\rho_2}(\mathcal{S}_2) = 0$. It follows by (6) that

$$0 = \mathrm{reltime}_{\rho_2}(\mathcal{S}_2) = \sum_{\ell=0}^{k-1} \mathrm{reltime}_{\rho_2}(j'_\ell, j_{\ell+1 \bmod k}) + \sum_{\ell=0}^{k-1} \mathrm{reltime}_{\rho_2}(j_\ell, j'_\ell)$$

$$> 0 + \sum_{\ell=0}^{k-1} \mathrm{reltime}_{\rho_1}(i_\ell, i'_\ell) \tag{7}$$

$$= \mathrm{reltime}_{\rho_1}(\mathcal{S}_1) = 0.$$

This leads to a contradiction.

Let us now assume that there exists at least one action $\bullet$ in $\mathcal{S}_1$. Consider any ℓ, $0 \leq \ell < k$, such that one action among i_ℓ, i'_ℓ is equal to $\bullet$. Necessarily, $i_\ell = \bullet$ and i'_ℓ occurs before i_ℓ in ρ_1, that is as $\mathrm{reltime}_{\rho_1}(i_\ell, i'_\ell) < 0$. Indeed, given the two races $B_{\ell-1 \bmod k} \prec B_\ell \prec B_{\ell+1 \bmod k}$, $\bullet$ participates in the first race and appears at the end of B_ℓ. It follows that

$$\mathrm{reltime}_{\rho_1}(i_\ell, i'_\ell) < \mathrm{reltime}_{\rho_2}(j_\ell, j'_\ell) < 0.$$

Therefore, we get the same inequalities as in (7), leading again to a contradiction. This completes the proof. □

G   Proof of Lemma 3

Lemma 3. *Let $\mathcal{A}$ be an AT and $\mathcal{R}$ be its modified region automaton. We can construct an MSO formula Φ of size linear in I and X such that a word labeling a run ρ of $\mathcal{R}$ satisfies Φ iff ρ is a padded run that cannot be wiggled. Moreover, the formula Φ, in prenex normal form, has three quantifier alternations.*

Before giving the proof, some definitions are in order: Sets of finite words (word structures, to be precise) over an alphabet Σ can be defined by sentences in MSO with the signature $(<, \{Q_a\}_{a \in \Sigma})$. Intuitively, we interpret the formula over the word $w \in \Sigma^*$ with variables being positions that take values in $\mathbb{N}$, that can be ordered with $<$, and the predicates $Q_a(p)$ indicating whether the p-th symbol of the word (structure) is a. The formulas also use variables P being sets of positions, and $P(p)$ meaning that p is a position belonging to P. We recall that a formula is in *prenex normal form* if it can be written as $Q_1 v_1 Q_2 v_2 \ldots Q_n v_n F$ with F a formula without quantifiers, Q_i a quantifier and v_i a variable for all $1 \leq i \leq n$. We suppose the reader is familiar with the rules to put a formula into a prenex normal form. By *quantifier alternations*, we mean alternating blocks of existential or universal quantifiers, respectively denoted by $\exists^*$ and $\forall^*$.

Moreover, recall that the modifications applied on the region automaton imply the following property (see Section 4.2). Given a timed run ρ of an AT, by Lemma 2, there exists an equivalent run ρ' in the region automaton such that any x-block $(i_{k_1} \ldots i_{k_m}, \gamma)$ of ρ is translated into the sequence of symbols $(i'_{k_1}, \ldots, i'_{k_m}, \gamma')$ in ρ' with an optional symbol γ' such that:

- $i'_{k_\ell} = (i_{k_\ell}, x)$, for $1 \leq \ell < m$,
- $i'_{k_m} = (i_{k_m}, \bot)$ if $\gamma = \bot$, and (i_{k_m}, x) otherwise,
- $\gamma' = di[x]$ if $\gamma = \bullet$, and γ' does not exist otherwise.

In the sequel, we again call $\mathcal{R}$ the modified region automaton.

We are going to describe a formula Φ such that a word labeling a run ρ of $\mathcal{R}$ satisfies Φ iff ρ is a padded run that cannot be wiggled. To define Φ, we use Proposition 1. Recall that it characterizes an unwigglable run ρ by a

cyclic block graph G_ρ. We also focus on the particular cycle of G_ρ as described in Corollary 1. Step by step, we create MSO formulas expressing the following statements about a run ρ of $\mathcal{R}$:

1. Two symbols belong to the same block (see the above property about the translation of x-blocks in $\mathcal{R}$ and the particular case of zero-valued timers that are discarded).
2. Two blocks participate in a race. Rather, we express that two symbols, one in each block, participate in a race.
3. The run is a padded run that cannot be wiggled. Rather, we express that there exists a cycle in G_ρ whose form is as in Corollary 1.

Some useful predicates. We define four predicates to help us write the MSO formulas. The formula $\mathrm{First}(p, P)$ expresses that a position p is the first element of a set P, while $\mathrm{Last}(p, P)$ states that p is the last element of P. Finally, $\mathrm{Next}(p, P, q)$ expresses that q is the successor of p in P with regards to $<$. More formally,

$$\mathrm{First}(p, P) := P(p) \wedge \forall q\colon q < p \rightarrow \neg P(q) \tag{8}$$

$$\mathrm{Last}(p, P) := P(p) \wedge \forall q\colon q > p \rightarrow \neg P(q) \tag{9}$$

$$\mathrm{Next}(p, P, q) := p < q \wedge P(p) \wedge P(q) \wedge \forall r\colon p < r < q \rightarrow \neg P(r). \tag{10}$$

The last useful predicate, $\mathrm{Partition}(P, P_1, P_2)$, states that there exist sets of positions P, P_1, P_2 such that $P = P_1 \uplus P_2$, the first position of P is in P_1, the last one is in P_2, and the positions of P alternate between P_1 and P_2:

$$
\begin{aligned}
\mathrm{Partition}(P, P_1, P_2) := & \forall r\colon P(r) \leftrightarrow (P_1(r) \vee P_2(r)) \\
& \wedge \exists p, q\colon \mathrm{First}(p, P) \wedge \mathrm{Last}(q, P) \wedge P_1(p) \wedge P_2(q) \\
& \wedge \forall r\colon P_1(r) \leftrightarrow \neg P_2(r) \\
& \wedge \forall r, s\colon \mathrm{Next}(r, P, s) \rightarrow (P_1(r) \leftrightarrow P_2(s)).
\end{aligned}
\tag{11}
$$

Two symbols belong to the same block. We give here an MSO formula expressing that two positions $p < q$ are labeled by symbols belonging to the same x-block. Thus, the formula $\mathrm{Block}_x(p, q, P)$, with P a set of positions labeled by consecutive symbols of an x-block, states that P must respect the following constraints:

- We have $p < q$, with $p, q \in P$.
- The position p is labeled by either $(i, x) \in \Sigma$ (meaning we start an x-block B), or $(to[x], x) \in \Sigma$ (meaning we are in the block B),
- The input at position q is either $(to[x], x)$ (we are in the block B), or $(to[x], \bot)$ (we are at the end of B and we do not restart x), or $di[x]$ (we finish B by discarding its timer while its value is zero),
- Every other position $r \in P$ such that $p < r < q$ is labeled by $(to[x], x)$ (we restart x to keep the block active),

- There is no position $r \notin P$ between p and q such that r is labeled by some symbol of Σ among $(to[x], \cdot), (\cdot, x)$, or $di[x]$ (the $\cdot$ indicates "any value"). That is, any intermediate position cannot affect x.

Formally, we have:

$$\text{Affects}_x(r) := Q_{(to[x],x)}(r) \vee Q_{(to[x],\bot)}(r) \vee \bigvee_{i \in I} Q_{(i,x)}(r) \vee Q_{di[x]}(r) \qquad (12)$$

$$\begin{aligned}
\text{Block}_x(p, q, P) := \ & p < q \wedge P(p) \wedge P(q) \\
& \wedge \big(\bigvee_{i \in I} Q_{(i,x)}(p) \vee Q_{(to[x],x)}(p) \big) \\
& \wedge \big(Q_{(to[x],x)}(q) \vee Q_{(to[x],\bot)}(q) \vee Q_{di[x]}(q) \big) \\
& \wedge \forall r \colon \big(p < r < q \wedge P(r) \big) \to Q_{(to[x],x)}(r) \\
& \wedge \forall r \colon \big(p < r < q \wedge \neg P(r) \big) \to \neg\text{Affects}_x(r).
\end{aligned} \qquad (13)$$

Two symbols participate in a race. The formula $\text{Race}(p,q)$ states that two positions $p < q$ are labeled by symbols that participate in a race, that is, there is no position labeled by τ between p and q.

$$\text{Race}(p, q) := p < q \wedge \neg\big(\exists r \colon p < r < q \wedge Q_\tau(r) \big). \qquad (14)$$

The run is unwigglable. Finally, we give a formula Φ that expresses that a word is the label of a padded run ρ that cannot be wiggled, i.e., that highlights a cycle (as in Corollary 1) in the block graph of ρ. The idea of that formula is as follows. There are positions $p_1 < q_1 < p_2 < q_2 < \cdots < p_m < q_m$ such that each pair p_k, q_k participate in a race. Moreover, for any q_k belonging to a block B_k, there must exist a p_ℓ that also belongs to B_k. Notice that p_ℓ is not necessarily after q_k. See Figure 5 for an illustration of that scenario with $m = 5$.

The formula Φ states that there exist sets of positions P, P_1, P_2 such that:

- We have $P_1 \uplus P_2$ forming a partition of P as described previously,
- For any $p \in P_1$ and $q \in P_2$ such that q is the next element after p in P, we have $\text{Race}(p, q)$.
- For any $q \in P_2$, there must exist a p in P_1 such that p and q belong to the same block. That is, there must exist a timer x and a set P' such that $\text{Block}_x(p, q, P')$ if $p < q$ or $\text{Block}_x(q, p, P')$ if $p > q$.

Finally, in order to describe a padded run ρ, the first and last positions of the word must be labeled with τ (i.e., a non-zero delay). These positions do not belong to P.

$$\text{InBlock}(p, q, P') := \big(p < q \wedge \bigvee_x \text{Block}_x(p, q, P') \big) \vee \big(p > q \wedge \bigvee_x \text{Block}_x(q, p, P') \big) \qquad (15)$$

$$\begin{aligned}
\varPhi := \exists P, P_1, P_2 \colon \Big(& \mathrm{Partition}(P, P_1, P_2) \\
& \wedge \forall p, q \colon \big(P_1(p) \wedge \mathrm{Next}(p, P, q) \big) \to \mathrm{Race}(p, q) \\
& \wedge \forall q \colon P_2(q) \to \big(\exists p, P' \colon P_1(p) \wedge \mathrm{InBlock}(p, q, P') \big) \Big) \\
& \wedge Q_\tau(1) \wedge \big(\exists r \colon Q_\tau(r) \wedge (\forall r' \colon r' < r) \big).
\end{aligned} \tag{16}$$

Correctness. Now that we have constructed the formula $\varPhi$, let us show that it correctly encodes that the word labeling a run ρ in $\mathcal{R}$ satisfies $\varPhi$ iff ρ is a padded run that is not wigglable.

$\Leftarrow$ Assume ρ in $\mathcal{R}$ is a padded run that is not wigglable. Let $w \in \varSigma^*$ be its labeling. By Proposition 1, we know that the block graph G_ρ is cyclic. Moreover, by Corollary 1, there exists a cycle whose blocks satisfy the following properties: exactly two blocks participate in any race of the cycle, any block participates in exactly two races, and any block has a size at least equal to two[10]. We consider this particular cycle $(B_0, \ldots, B_{m-1}, B_0)$.

From the races in which the blocks B_k participate, we define the sets P_1, P_2 of positions, and, thus, $P = P_1 \uplus P_2$ as follows. For every $B_k \prec B_{k+1 \bmod m}$ in the cycle, consider $a \in B_k$ and $b \in B_{k+1 \bmod m}$ that are the two symbols of w participating in a race. We add the position of a in P_1 and the position of b in P_2 (see Figure 5 to get intuition). Thus, the positions in P alternate between P_1 and P_2, the first element of P is in P_1, and the last is in P_2. Moreover, for any $p \in P_1$ and $q \in P_2$ such that q is the successor of p in P, it holds that $\mathrm{Race}(p, q)$. That is, the second line of formula (16) about races is satisfied by w.

We have to show that the third line of (16) is also satisfied by w, that is, for any position q in P_2, there exists a position p in P_1 such that p and q belong to the same block. Let $q \in P_2$. By construction, q belongs to some block B_k of the cycle. By Corollary 1, B_k participate in exactly two races of the cycle, one as described above with $\mathrm{Race}(p, q)$, and another one with $\mathrm{Race}(p', q')$ for some other positions $p' \in P_1$ and $q' \in P_2$. Necessarily, p' is a position of a symbol in B_k (and not q' by choice of the cycle), such that either $p' < q$ or $p' > q$. Thus, the third line of (16) is satisfied by w.

Finally, since ρ is padded, it must be that its first and last delays are positive, i.e., the corresponding positions are labeled by τ. Hence the last line of (16) is satisfied by w. We conclude that w satisfies all conjuncts of (16) and then also the formula $\varPhi$.

$\Rightarrow$ Assume now that the label w of a run ρ in $\mathcal{R}$ satisfies formula $\varPhi$. Since the last line of (16) forces the first and last symbols of w to be τ, the formula describes a padded run of $\mathcal{R}$.

Let P, P_1, and P_2 be the sets that satisfy the formula $\varPhi$. Let $P_1 = \{p_1, \ldots, p_m\}$ and $P_2 = \{q_1, \ldots, q_m\}$ be such that $\mathrm{Next}(p_k, P, q_k)$ is satisfied for all k. Then, by (14), it holds that $\mathrm{Race}(p_k, q_k)$ is also satisfied, i.e., the symbols of w labeling the positions p_k, q_k participate in a race. The third line of (16) implies that there are at most m blocks involved in these races. Notice that there can be less than

---

[10] Recall the way blocks in $\mathcal{A}$ are translated into blocks in $\mathcal{R}$.

m blocks, as for q, q' in P_2 with $q \neq q'$, we could have the same $p \in P_1$ that makes sub-formula InBlock satisfied in (16).

From formula Φ, we are going to construct a part of the block graph G_ρ of ρ that is cyclic. We proceed inductively as follows:

- Take an arbitrary position $q_{k_0} \in P_2$. There exists $p_{k_1} \in P_1$ such that $\text{Block}_{x_1}(q_{k_0}, p_{k_1}, P_1')$ or $\text{Block}_{x_1}(p_{k_1}, q_{k_0}, P_1')$ is satisfied. Call B_1 the related x_1-block.
- Let $q_{k_1} \in P_2$. Then, there exists $p_{k_2} \in P_1$ such that $\text{Block}_{x_2}(q_{k_1}, p_{k_2}, P_2')$ or $\text{Block}_{x_2}(p_{k_2}, q_{k_1}, P_2')$ is satisfied. For the related x_2-block B_2, we have $B_1 \prec B_2$ as p_{k_1}, q_{k_1} participate in a race, and $p_{k_1} \in B_1, q_{k_1} \in B_2$.
- Let $q_{k_2} \in P_2$. Then, there exists $p_{k_3} \in P_1$ such that $\text{Block}_{x_3}(q_{k_2}, p_{k_3}, P_3')$ or $\text{Block}_{x_3}(p_{k_3}, q_{k_2}, P_3')$ is satisfied. For the related x_3-block B_3, we have $B_2 \prec B_3$.
- We repeat this process until we obtain a cycle. This situation necessarily arises as the number of blocks is bounded by m.

This shows that G_ρ is cyclic.

Prenex form of Φ. Formula $\text{Block}_x(p, q, B)$, see (13), can be easily rewritten in prenex normal form that starts with a block $\forall^*$ of universal quantifiers. Similarly $\text{Race}(p, q)$, see (14), requires a single universal quantifier. Let us consider the formula Φ putting aside the quantifiers $\exists P, P_1, P_2$, see (16). Notice that formulas (8), (9), and (10) all use a single universal quantifier. The first conjunct of (16) uses $\text{Partition}(P, P_1, P_2)$, see (11), that can be rewritten with three blocks $\forall^* \exists^* \forall^*$. The second (resp. last) conjunct can be rewritten with two blocks $\forall^* \exists^*$ (resp. $\exists^* \forall^*$), and the third one with three blocks $\forall^* \exists^* \forall^*$. Hence, we obtain that the quantifiers of the prenex normal form of Φ are $\exists^* \forall^* \exists^* \forall^*$, that is, with three quantifier alternations, as expected.

Finally, by carefully examining the formulas, we notice that most of them have constant size except $\text{Affects}_x(r)$ and $\text{Block}_x(p, q, P)$ whose sizes are linear in $|I|$, and $\text{InBlock}(p, q, P')$ and Φ whose sizes are linear in $|I|$ and $|X|$.
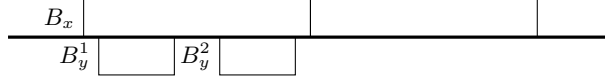
H   Proof of Theorem 2

Theorem 2. *Deciding whether an AT is race-avoiding is* PSPACE-*hard and in* 3EXP. *It is in* PSPACE *if the sets of actions I and of timers X are fixed.*

We begin by proving the upper bound.

H.1   Upper bound

We make use of the Büchi-Elgot-Trakhtenbrot theorem: A language is regular iff it can be defined as the set of all words satisfied by an MSO formula (with effective translations, see [10, 15]). First, from the formula Φ of Lemma 3, we can construct a finite-state automaton $\mathcal{N}$ whose language is the set of all

Fig. 9: Visualization of the forced buffer using timer y.

words satisfying Φ. Due to the reduction from non-deterministic to deterministic automaton, each quantifier alternation induces an exponential blowup in the automaton construction. Hence, the size of $\mathcal{N}$ is triple-exponential as Φ has three quantifier alternations. Second, we compute the intersection of $\mathcal{N}$ with $\mathcal{R}$ — itself exponential in size. This can be done in polynomial time in the sizes of both automata, i.e., in 3EXP. Finally, the language of the resulting automaton is empty iff there is no padded run of $\mathcal{A}$ that cannot be wiggled, and this can be checked in polynomial time with respect to the size of the (triple-exponential) automaton.

H.2   Lower bound

In this section, we prove that deciding whether all padded timed runs of a given AT are wigglable is a PSPACE-hard problem. The idea of the proof is to leverage the PSPACE-hardness proof for the reachability problem, see Theorem 1. We use the same notations as in the proof provided for the latter result in Appendix A.

Let $\mathcal{M}$ be an LBTM and w be an input word. Let $\mathcal{A}_{\mathcal{M},w}$ be the AT constructed from $\mathcal{M}$ and w in the proof of Theorem 1, such that the state r_{done} is reachable in the AT iff the LBTM accepts w. We are going to slightly modify the LBTM and the constructed AT such that any padded timed run ρ of $\mathcal{A}_{\mathcal{M},w}$ has an acyclic block graph G_ρ. Thanks to Proposition 1, this is equivalent to stating that ρ can be wiggled. Then, we give a widget that extends any padded timed run ρ reaching r_{done} into one that is unwigglable. Hence, the given LBTM accepts w if the constructed AT has some unwigglable padded timed run.

First, we modify the LBTM $\mathcal{M}$ and word $w = w_1 \ldots w_n$ to obtain a new LBTM $\mathcal{M}'$ that accepts w iff it does so while maintaining the invariant that no two cells of the tape of $\mathcal{M}'$ hold the same symbol. Let Σ be the alphabet of $\mathcal{M}$. We create a new word $w' = (w_1, 1)(w_2, 2) \ldots (w_n, n)$, and a new LBTM $\mathcal{M}'$ over the alphabet $\Sigma' = \Sigma \times \{1, \ldots, n\}$ such that $\mathcal{M}'$ simulates $\mathcal{M}$ by discarding the second component of each symbol of Σ', except that whenever $\mathcal{M}$ writes the symbol a at the position i on the tape, $\mathcal{M}'$ writes the symbol (a, i). This requires to store the current i in the state of $\mathcal{M}'$, inducing a polynomial blowup in n for the number of states of $\mathcal{M}'$. Thanks to the second component, we indeed have that every cell contains a symbol that is different from any other cell.

Second, we modify the construction of the AT $\mathcal{A}_{\mathcal{M}',w'}$ as follows. We add a new timer y which we use to force a block acting as a buffer before and after each action implying the timer x when the value of *clock* is equal to zero, as illustrated in Figure 9. Therefore, any input-action must take place between these two buffers. In order to have enough room for these buffers, we multiply by

three all the values at which the timers $x, x_1, \ldots, x_n$ are updated (in particular x is always (re)started with value 3 instead of 1). For the initialization of the timers $x, x_1, \ldots, x_n$ (at the start of phase 1), we add new states $\langle r_1, y_k \rangle$ and $\langle q_0, 1, 0, 0, y_k \rangle$, for $k = 1, 2$, and modify the transitions to force y to start and time out, then we force the initialization part and, finally, force y to start and time out again:

$$r_0 \xrightarrow[(x,3)]{go} \langle r_1, y_1 \rangle \xrightarrow[(y,1)]{go} \langle r_1, y_2 \rangle \xrightarrow[\perp]{to[y]} r_1 \ldots$$

$$\ldots r_n \xrightarrow[(x_n,6j)]{go} \langle q_0, 1, 0, 0, y_1 \rangle \xrightarrow[(y,1)]{go} \langle q_0, 1, 0, 0, y_2 \rangle \xrightarrow[\perp]{to[y]} \langle q_0, 1, 0, 0 \rangle$$

with $6j$ the value at which x_n must be set (see the proof of Theorem 1). We define $\chi(\langle r_1, y_1 \rangle) = \chi(r_1) = \{x\}$, $\chi(\langle r_1, y_2 \rangle) = \{x, y\}$, $\chi(\langle q_0, 1, 0, 0, y_1 \rangle) = \chi(\langle q_0, 1, 0, 0 \rangle) = \{x, x_1, \ldots, x_n\}$, and $\chi(\langle q_0, 1, 0, 0, y_2 \rangle) = \{x, x_1, \ldots, x_n, y\}$. In states where $clock = 0$ in the other phases, we do likewise in the following way. For each state $\langle q, i, symbol, 0 \rangle$ of the AT with $symbol > 0$, we add new states $\langle q, i, symbol, 0, y_k \rangle$ and $\langle q', j, 0, 0, y_k \rangle$, for $k = 1, 2$, and modify the transitions to force y to start and time out, then allow a t-transition[11], and, finally, force y to start and time out again:

$$\langle q, i, symbol, 0, y_1 \rangle \xrightarrow[(y,1)]{go} \langle q, i, symbol, 0, y_2 \rangle \xrightarrow[\perp]{to[y]} \langle q, i, symbol, 0 \rangle$$

$$\xrightarrow[(x_i,6j)]{t} \langle q', j, 0, 0, y_1 \rangle \xrightarrow[(y,1)]{go} \langle q', j, 0, 0, y_2 \rangle \xrightarrow[\perp]{to[y]} \langle q', j, 0, 0 \rangle.$$

Any $to[x]$-transition leading to $\langle q, i, symbol, 0 \rangle$ instead goes to $\langle q, i, symbol, 0, y_1 \rangle$. We then define $\chi(\langle q, i, symbol, 0, y_1 \rangle) = \chi(\langle q', j, 0, 0, y_1 \rangle) = \{x, x_1, \ldots, x_n\}$ and $\chi(\langle q, i, symbol, 0, y_2 \rangle) = \chi(\langle q', j, 0, 0, y_2 \rangle) = \{x, x_1, \ldots, x_n, y\}$. Any other transition leads to r_{sink}.

Third, let $\mathcal{A}'$ be the AT obtained with this modified construction. We now argue that any padded timed run $\rho \in ptruns(\mathcal{A}')$ can be wiggled. We do so by proving that the block graph of ρ is acyclic. In the sequel, we say that two block B, B' are *incomparable* if neither $B \prec B'$ nor $B' \prec B$.

1. Suppose first that ρ does not contain the state r_{sink}.
   - Recall that x is never discarded and every $to[x]$-transition restarts it. Thus, there exists a single x-block; we call it B_x. By construction, none of the $to[y]$-transitions update y. Therefore, we have strictly more than one y-block. Call them $B_y^1, B_y^2, \ldots, B_y^{m_y}$ in the order they are seen alongside ρ. For any odd i, it may be that $B_x \prec B_y^i$ (if the input-action starting y occurs at the same time x times out), $B_y^i \prec B_y^{i+1}$ (if the sum of the delays between the timeout of y in B_y^i and the *go*-transition starting B_y^{i+1} is zero), and $B_y^{i+1} \prec B_x$. However, it is impossible to have $B_x \prec B_y^i \prec B_y^{i+1} \prec B_x$, as x is (re)started with value 3 and there are exactly

---

[11] We use the notations of the proof of Theorem 1.

two units of time if $B_y^i \prec B_y^{i+1}$ (since y is always started with value 1). That is, we do not have a cycle. Moreover, since the y-blocks only appear when the *clock* component of the current state is zero, B_y^j and B_y^{j+1} are incomparable for every even j.

- Let us now focus on the timers x_i. Since a t-action can discard x_i, we may have multiple x_i-blocks, say $B_{x_i}^1, \ldots, B_{x_i}^{m_{x_i}}$ (again, in the order they are seen in ρ). Since x_i is updated such that it cannot time out while the *clock* component of the current state is zero (i.e., the timer fate of the corresponding block is $\times$) and only one t-transition can occur per phase of $\mathcal{M}'$, all of the x_i-blocks are pairwise incomparable. Moreover, thanks to the y-buffers, B_x and any block $B_{x_i}^j$ are incomparable. As stated before, it may happen that a t-transition of some block $B_{x_i}^j$ occurs concurrently with an action of a block B_y^ℓ (resp. $B_y^{\ell+1}$) with ℓ an odd number. By construction, $B_y^\ell \prec B_{x_i}^j$ (resp. $B_{x_i}^j \prec B_y^{\ell+1}$). Note that it is possible that $B_y^\ell \prec B_{x_i}^j \prec B_y^{\ell+1}$ if all blocks participate in the same race.
- We now consider two different timers x_i and x_k. Since the LBTM $\mathcal{M}'$ is such that no two cells contain the same value, it must be that x_i and x_k are always updated to time out in states containing different *clock* component in $\mathcal{A}'$ (as, otherwise, this would imply that the two cells of $\mathcal{M}'$ contain identical symbols). That is, it is not possible for two timers to time out concurrently. However, during the initialization, it may be that the actions starting $B_{x_1}^1, B_{x_2}^1, \ldots, B_{x_n}^1$ occur at the same time. We thus have $B_{x_1}^1 \prec B_{x_2}^1 \prec \cdots \prec B_{x_n}^1$ and the blocks $B_{x_i}^j$ and $B_{x_k}^\ell$ are incomparable with $j, \ell > 1$.

Using all these facts over the races and $\prec$, we deduce that G_ρ is acyclic, i.e., ρ can be wiggled.

2. Suppose now that ρ contains the state r_{sink}. Recall that $\chi(r_{sink}) = \emptyset$, meaning that the update of every transition ending in r_{sink} is $\bot$, every timer is stopped, and every new block started after reaching r_{sink} contains exactly one action. We thus focus on the prefix of the run up to the transition leading to r_{sink}. Call this last transition t^* with action i^*. By construction, it must be this prefix is a run that satisfies the constraints explained above (i.e., there is no cycle in the block graph induced by the prefix of the run). Let us show that adding t^* after the prefix does not induce a cycle in the block graph.

   Suppose first that i^* is an input. As the update of t^* is $\bot$, it must be that i^* is the only action of its block, and this block cannot appear in a cycle. Suppose now that i^* is a timeout-action.

   - Let us study the initialization or a simulation step (i.e., the start of a phase). Recall that no timer x_i can time out, i.e., we only have to consider the timers x and y. In this case, we must have $i^* = to[x]$. Indeed, this happens when a block B_y^ℓ is started too late, in a way that $to[x]$ occurs before y times out, or when such a block is not started at all. The only hope to have a cycle is to adapt to the current situation the discussion we made above about $B_x \prec B_y^\ell \prec B_y^{\ell+1} \prec B_x$ with ℓ odd. Here, as

$i^* = to[x]$ stops the timer y and x (resp. y) is (re)started with value 3 (resp. value 1), we get $B_x \prec B_y^\ell$ and $B_x \prec B_y^{\ell+1}$. We thus have no cycle.

- Otherwise, we consider a state in which the *clock* value is not zero (i.e., this is not the start of a phase). By construction, a $to[x]$-, or any $to[x_i]$-transition cannot lead to r_{sink}, and y is never started. Thus, i^* cannot be a timeout-action in this case.

Hence, we covered every case and never obtained a cycle, i.e., ρ is wigglable.

We have thus proved that each timed run $\rho \in ptruns(\mathcal{A}')$ can be wiggled.

Finally, we add a widget that forces an unwigglable run after r_{done}. To do so, we add new timers z, z' and states s_1 to s_4, and define the following transitions

$$r_{done} \xrightarrow[(z,1)]{go} s_1 \xrightarrow[(z',1)]{go} s_2 \xrightarrow[\perp]{to[z']} s_3 \xrightarrow[\perp]{to[z]} s_4.$$

We define $\chi(s_1) = \chi(s_3) = \{z\}$, $\chi(s_2) = \{z, z'\}$, and $\chi(s_4) = \emptyset$. Given a padded timed run ending in r_{done}, the only ways to extend it into a padded timed run reaching s_4 are by adding the following sequence of transitions, with any $d > 0$:

$$(r_{done}, \emptyset) \xrightarrow[(z,1)]{go} (s_1, z = 1) \xrightarrow{0} (s_1, z = 1) \xrightarrow[(z',1)]{go} (s_2, z = 1, z' = 1)$$

$$\xrightarrow{1} (s_2, z = 0, z' = 0) \xrightarrow[\perp]{to[z']} (s_3, z = 0) \xrightarrow{0} (s_3, z = 0) \xrightarrow[\perp]{to[z]} (s_4, \emptyset) \xrightarrow{d} (s_4, \emptyset).$$
$$(17)$$

The resulting padded timed run is not wigglable, as two blocks B_z and $B_{z'}$ have been added such that $B_z \prec B_{z'} \prec B_z$.

To conclude, it remains to prove that the given $\mathcal{M}'$ accepts the given w' iff there exists an unwigglable padded timed run in the AT $\mathcal{A}'$ extended with the widget. First, suppose that $\mathcal{M}'$ accepts w'. Then by the proof of Theorem 1, we know that there exists a timed run $\rho \in truns(\mathcal{A}')$ reaching r_{done}. As both r_0 and r_{done} do not have any active timer, the first and last delays of ρ can be made positive, thus making ρ padded. We then extend ρ with (17) and obtain an unwigglable run that is still padded. Second, suppose that there exists a padded timed run ρ that cannot be wiggled. We proved above that if ρ ends in some state of $\mathcal{A}'$, then ρ is wigglable. Therefore, by construction of the widget, ρ has to end with s_4, meaning that a prefix of ρ reaches r_{done}. It follows that w' is accepted by $\mathcal{M}'$. Thus, deciding whether all padded timed runs of an AT can be wiggled is a PSPACE-hard problem.