Automata with Timers

Véronique Bruyère, Guillermo A. Pérez, Gaëtan Staquet, Frits W. Vaandrager

Theoretical computer science
University of Mons

Formal Techniques in Software Engineering
University of Antwerp

September 20, 2023

UMONS Université de Mons

fnrs LA LIBERTÉ DE CHERCHER

University of Antwerp

Many computer systems have timing constraints:

- Network protocols;
- Schedulers;
- Embedded systems;
- In general, real-time systems.

1

Many computer systems have timing constraints:

- Network protocols;
- Schedulers;
- Embedded systems;
- In general, real-time systems.

Well-known model for these systems: timed automata.[1]

[1] Baier and Katoen, *Principles of model checking*, 2008; Clarke et al., *Handbook of Model Checking*, 2018

Many computer systems have timing constraints:

- Network protocols;
- Schedulers;
- Embedded systems;
- In general, real-time systems.

Well-known model for these systems: timed automata.[1]

In short: finite automata augmented with clocks that can be reset or used in guards along transitions and states.

[1] Baier and Katoen, *Principles of model checking*, 2008; Clarke et al., *Handbook of Model Checking*, 2018

Many computer systems have timing constraints:

- Network protocols;
- Schedulers;
- Embedded systems;
- In general, real-time systems.

Well-known model for these systems: timed automata.[1]

In short: finite automata augmented with clocks that can be reset or used in guards along transitions and states.

BUT timed automata are hard to construct and understand.

[1] Baier and Katoen, *Principles of model checking*, 2008; Clarke et al., *Handbook of Model Checking*, 2018

We focus on properties that can be represented with timers: automata with timers.

Timed automata

▶

▶

▶

▶

▶

Automata with timers

▶

▶

▶

▶

▶

2

We focus on properties that can be represented with timers: automata with timers.

Timed automata

- Clocks go from 0 to infinity;
-
-
-
-

Automata with timers

- Timers go from a value set by the transition to 0;
-
-
-
-

2

We focus on properties that can be represented with timers: automata with timers.

Timed automata	Automata with timers
▶ Clocks go from 0 to infinity;	▶ Timers go from a value set by the transition to 0;
▶ We know the current value of the clocks;	▶ We do not know the current value of the timers;
▶	▶
▶	▶
▶	▶

2

We focus on properties that can be represented with timers: automata with timers.

Timed automata

- Clocks go from 0 to infinity;
- We know the current value of the clocks;
- Timed automata are more expressive;
-
-

Automata with timers

- Timers go from a value set by the transition to 0;
- We do not know the current value of the timers;
- Automata with timers are more restrictive;
-
-

2

We focus on properties that can be represented with timers: automata with timers.

Timed automata

- Clocks go from 0 to infinity;
- We know the current value of the clocks;
- Timed automata are more expressive;
- Learning (à la Angluin[2]) timed automata is challenging;
-

Automata with timers

- Timers go from a value set by the transition to 0;
- We do not know the current value of the timers;
- Automata with timers are more restrictive;
- Future work: learning algorithm;
-

[2] Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987

We focus on properties that can be represented with timers: automata with timers.

Timed automata

- Clocks go from 0 to infinity;
- We know the current value of the clocks;
- Timed automata are more expressive;
- Learning (à la Angluin[2]) timed automata is challenging;
- Well-known model.

Automata with timers

- Timers go from a value set by the transition to 0;
- We do not know the current value of the timers;
- Automata with timers are more restrictive;
- Future work: learning algorithm;
- This work studies some properties of automata with timers.

[2] Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987

An automaton with timers (AT) is a tuple $\mathcal{A} = (X, I, Q, q_0, \chi, \delta)$ where

- X is the set of timers,
- I is the set of actions,

An automaton with timers (AT) is a tuple $\mathcal{A} = (X, I, Q, q_0, \chi, \delta)$ where

- X is the set of timers,
- I is the set of actions,
- Q is the finite set of states,
- $q_0 \in Q$ is the initial state,

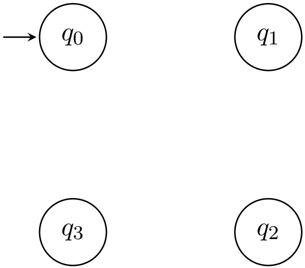


Figure 1: An AT.

An automaton with timers (AT) is a tuple $\mathcal{A} = (X, I, Q, q_0, \chi, \delta)$ where

- X is the set of timers,
- I is the set of actions,
- Q is the finite set of states,
- $q_0 \in Q$ is the initial state,
- $\chi : Q \to \mathcal{P}(X)$ gives the active timers of each state,

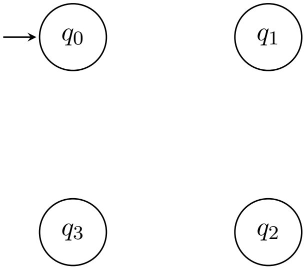


Figure 1: An AT.

An automaton with timers (AT) is a tuple $\mathcal{A} = (X, I, Q, q_0, \chi, \delta)$ where

- X is the set of timers,
- I is the set of actions,
- Q is the finite set of states,
- $q_0 \in Q$ is the initial state,
- $\chi : Q \to \mathcal{P}(X)$ gives the active timers of each state,
- δ is the transition function.

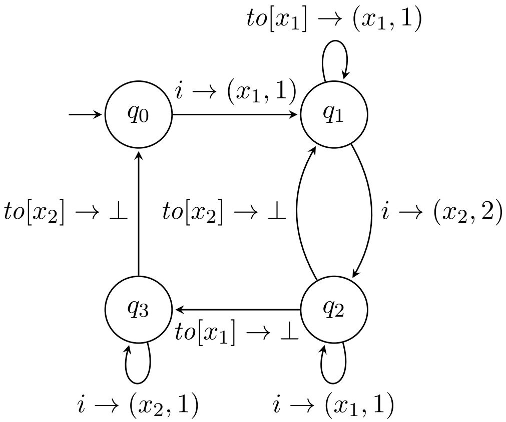


Figure 1: An AT.

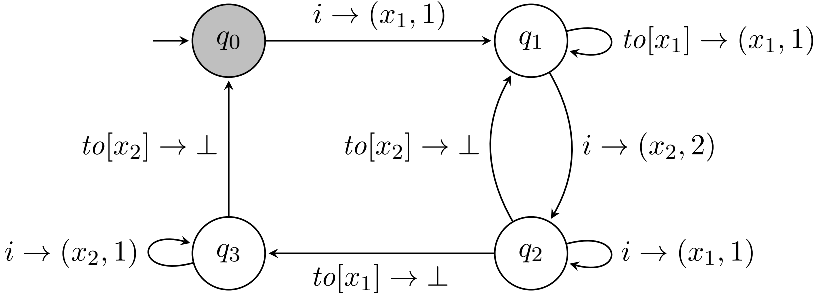


Figure 2: The same AT.

$(q_0, \emptyset)$

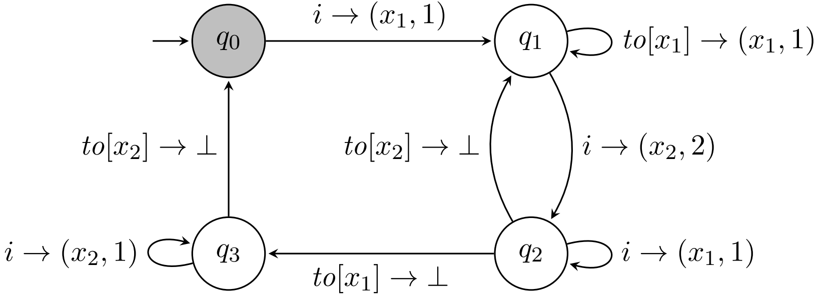


Figure 2: The same AT.

$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset)$$

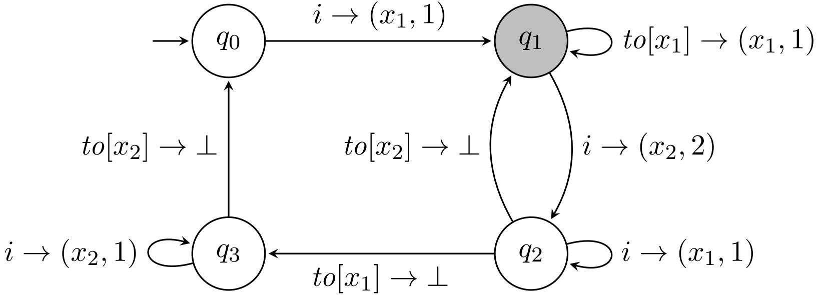


Figure 2: The same AT.

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

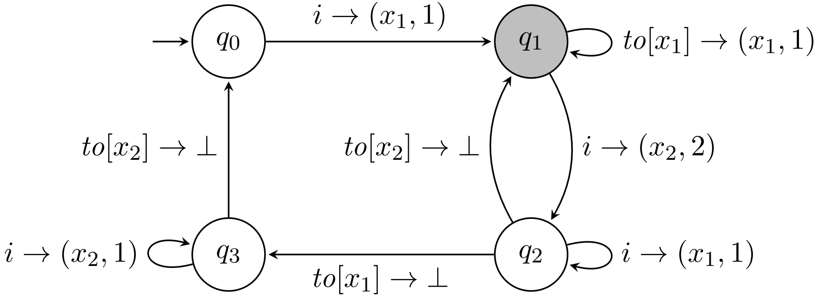


Figure 2: The same AT.

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

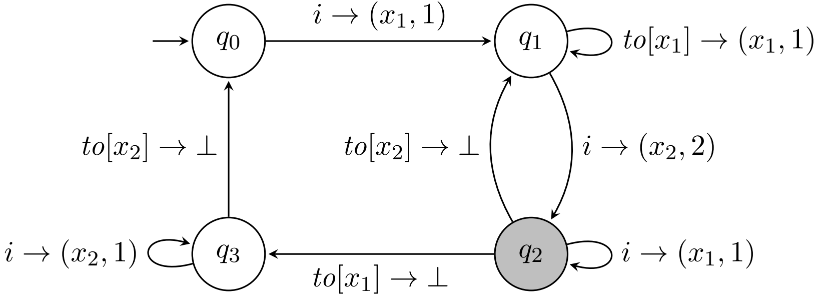


Figure 2: The same AT.

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

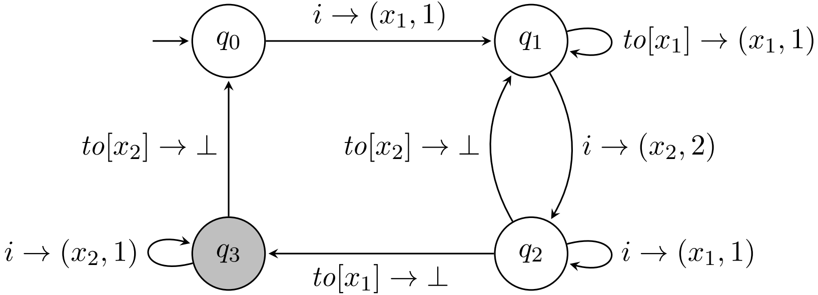


Figure 2: The same AT.

$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow[x_1,1]{i} (q_1, x_1 = 1) \xrightarrow{1} (q_1, x_1 = 0) \xrightarrow[x_2,2]{i} (q_2, x_1 = 0, x_2 = 2)$$
$$\xrightarrow{0} (q_2, x_1 = 0, x_2 = 2) \xrightarrow[\bot]{to[x_1]} (q_3, x_2 = 2)$$

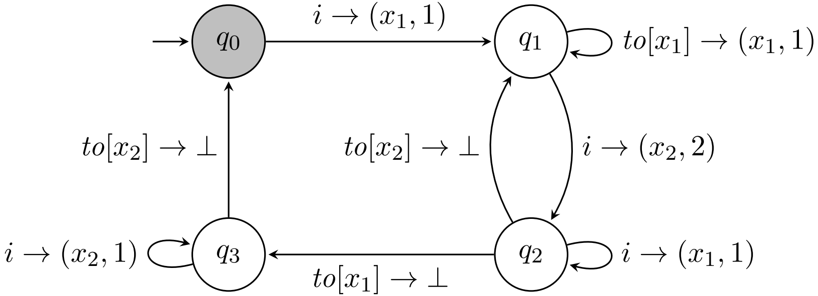


Figure 2: The same AT.

$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow[x_1,1]{i} (q_1, x_1 = 1) \xrightarrow{1} (q_1, x_1 = 0) \xrightarrow[x_2,2]{i} (q_2, x_1 = 0, x_2 = 2)$$
$$\xrightarrow{0} (q_2, x_1 = 0, x_2 = 2) \xrightarrow[\bot]{to[x_1]} (q_3, x_2 = 2) \xrightarrow{2} (q_3, x_2 = 0) \xrightarrow[\bot]{to[x_2]} (q_0, \emptyset) \xrightarrow{0.5} (q_0, \emptyset).$$

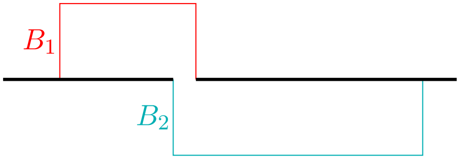


Figure 3: Block representation of the execution.

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

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

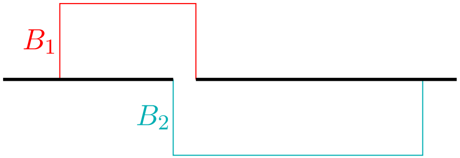


Figure 3: Block representation of the execution.

$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow[x_1,1]{i} (q_1, x_1 = 1) \xrightarrow{1} (q_1, x_1 = 0) \xrightarrow[x_2,2]{i} (q_2, x_1 = 0, x_2 = 2)$$
$$\xrightarrow{0} (q_2, x_1 = 0, x_2 = 2) \xrightarrow[\bot]{to[x_1]} (q_3, x_2 = 2) \xrightarrow{2} (q_3, x_2 = 0) \xrightarrow[\bot]{to[x_2]} (q_0, \emptyset) \xrightarrow{0.5} (q_0, \emptyset).$$

We have concurrent actions.

We can avoid this concurrency and still see the same sequence of actions.

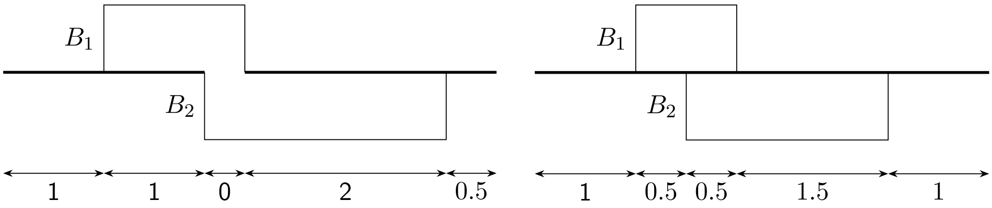


Figure 4: Idea: wiggle delays between actions.

We can avoid this concurrency and still see the same sequence of actions.

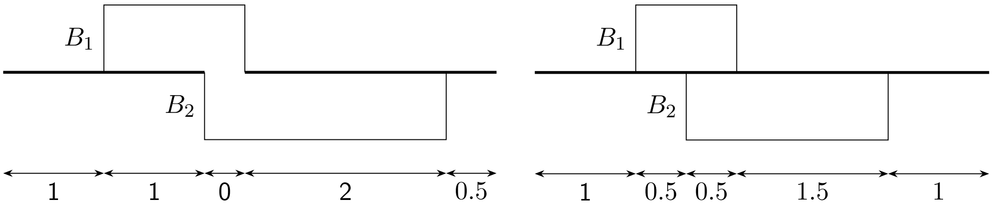


Figure 4: Idea: wiggle delays between actions.

Is it always possible?

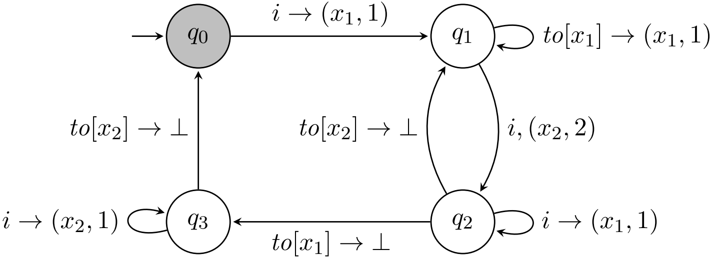


Figure 5: The same AT.

$(q_0, \emptyset)$

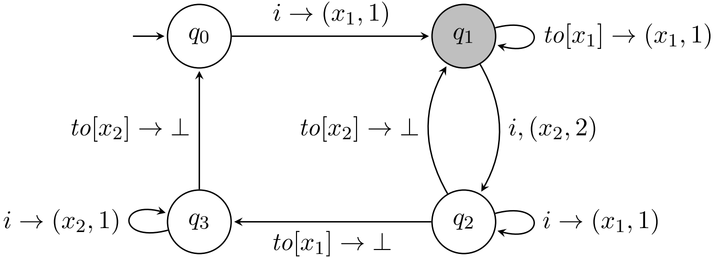


Figure 5: The same AT.

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

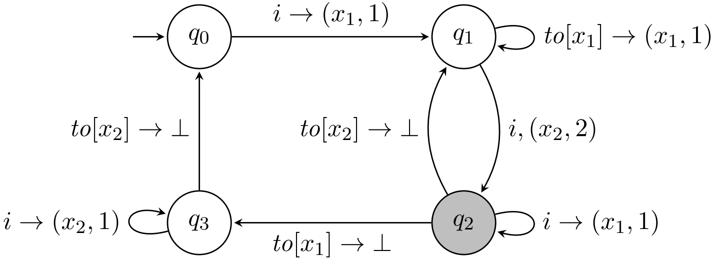


Figure 5: The same AT.

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

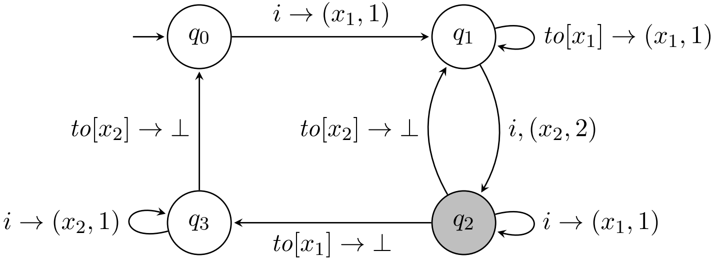


Figure 5: The same AT.

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

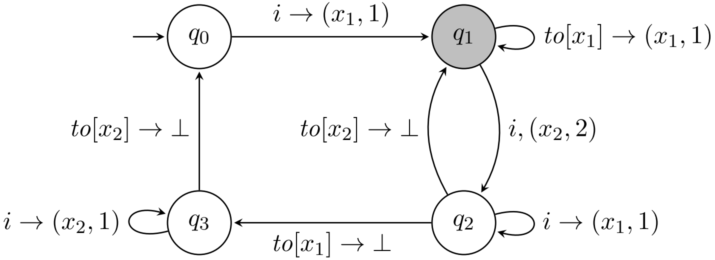


Figure 5: The same AT.

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

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

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

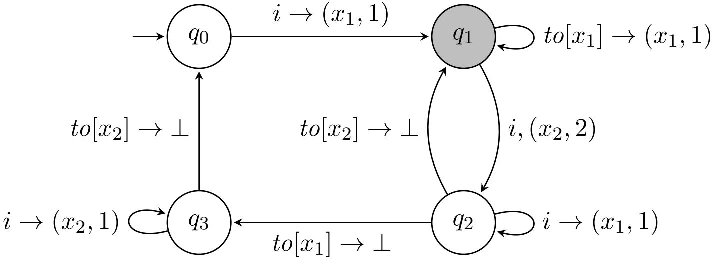


Figure 5: The same AT.

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

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

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

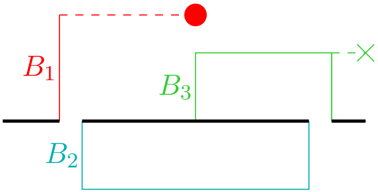


Figure 6: Block representation of the timed run.

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

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

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

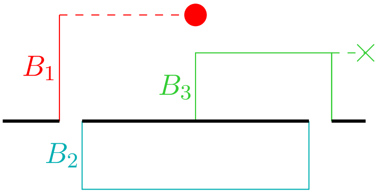


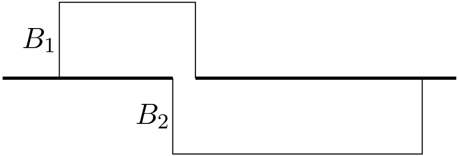
Figure 6: Block representation of the timed run.

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

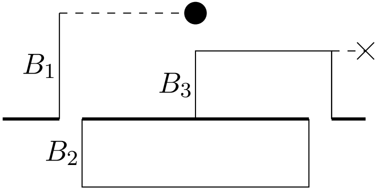
$$\xrightarrow{1} (q_2, x_1 = 0, x_2 = 1) \xrightarrow[x_1,1]{i} (q_2, x_1 = 1, x_2 = 1) \xrightarrow{1} (q_2, x_1 = 0, x_2 = 0)$$

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

We cannot avoid this concurrency and still see the same sequence of actions.

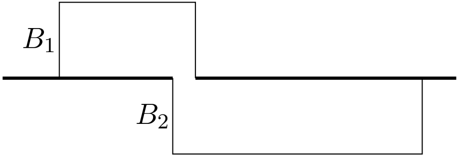


(a) Can be avoided.

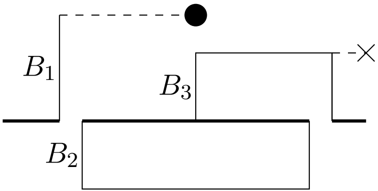


(b) Cannot be avoided.

Figure 7: Some concurrency can be avoided, some not.



(a) Can be avoided.



(b) Cannot be avoided.

Figure 7: Some concurrency can be avoided, some not.

Can we characterize when it is possible to remove the concurrency?

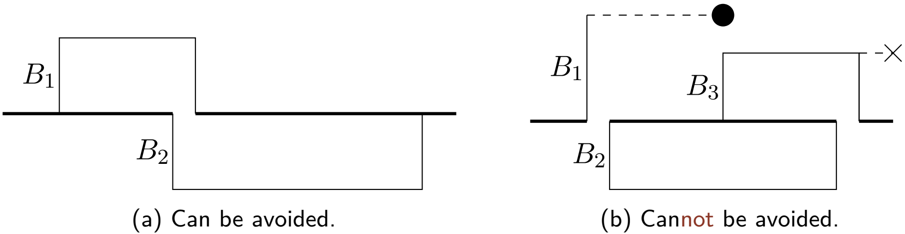


Figure 7: Some concurrency can be avoided, some not.

Can we characterize when it is possible to remove the concurrency?

Yes!

We studied two problems.

We studied two problems.

Theorem 1 (Contribution)

Fix an automaton and a state q. Deciding whether there exists an execution of the automaton that reaches q is PSPACE*-complete.*

We studied two problems.

Theorem 1 (Contribution)

Fix an automaton and a state q. Deciding whether there exists an execution of the automaton that reaches q is PSPACE-*complete.*

- Hardness: reduction from Linear Bounded Turing Machine.
- Membership: region automaton.

We studied two problems.

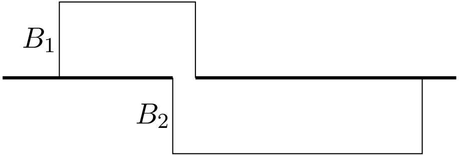
Theorem 1 (Contribution)

Fix an automaton and a state q. Deciding whether there exists an execution of the automaton that reaches q is PSPACE*-complete.*

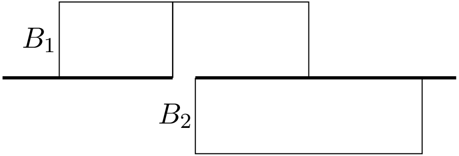
- Hardness: reduction from Linear Bounded Turing Machine.
- Membership: region automaton.

Theorem 2 (Contribution)

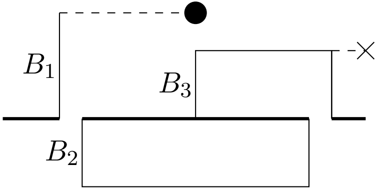
Deciding whether an AT contains an execution in which some concurrency cannot be avoided is PSPACE*-hard and in* 3EXP.



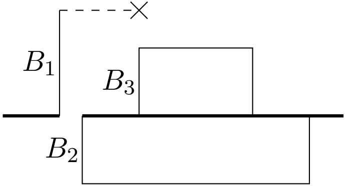
(a) Can be wiggled.



(b) Can be wiggled.



(c) Cannot be wiggled.



(d) Can be wiggled.

Figure 8: Not all runs can be wiggled.

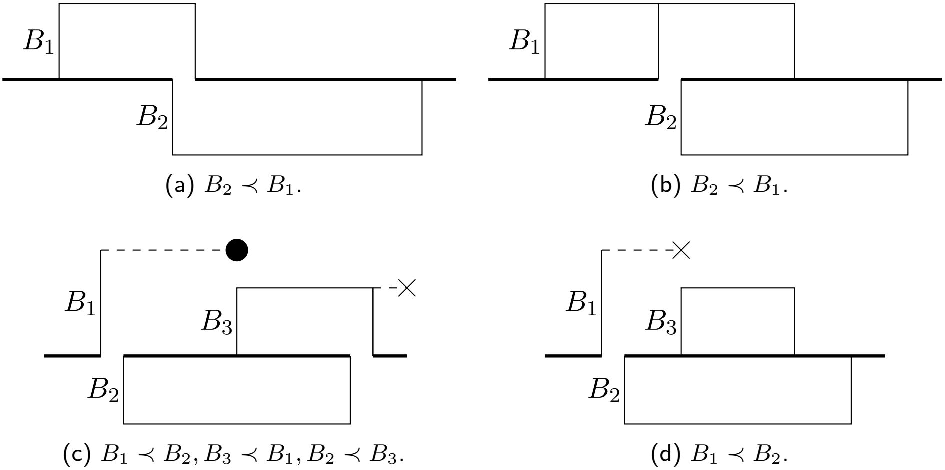
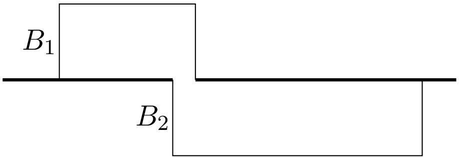
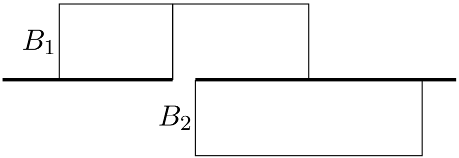


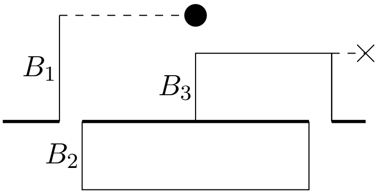
Figure 9: Define an order $\prec$ over the blocks, based on races.



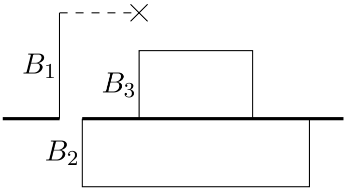
(a) $B_2 \prec B_1$.



(b) $B_2 \prec B_1$.



(c) $B_1 \prec B_2, B_3 \prec B_1, B_2 \prec B_3$.



(d) $B_1 \prec B_2$.

Figure 9: Define an order $\prec$ over the blocks, based on races.

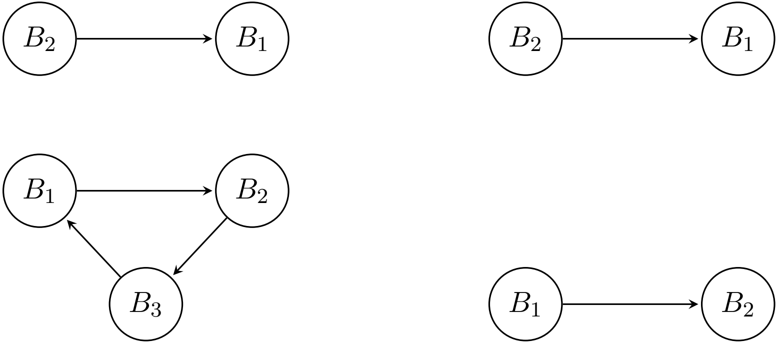


Figure 10: Block graphs defined from the blocks and $\prec$.

Proposition 3 (Contribution)

A timed run ρ can be wiggled if and only if its block graph is acyclic.

$\Rightarrow$ By contraposition, we have a cycle.
If a block has...

- A predecessor? It cannot move left.
- A successor? It cannot move right.
- Both? It cannot move at all.

Thus, ρ cannot be wiggled since we have a cycle.

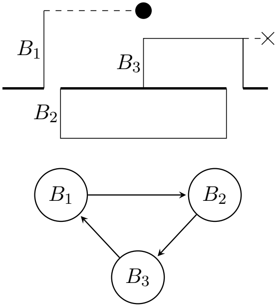


Figure 11: We have a cycle.

Proposition 3 (Contribution)

A timed run ρ can be wiggled if and only if its block graph is acyclic.

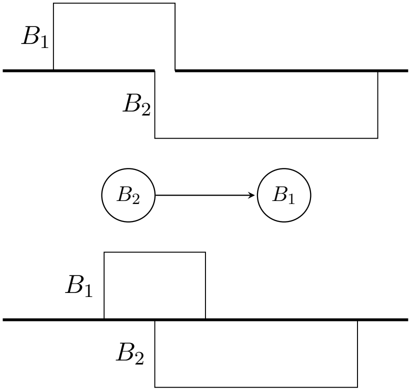


Figure 12: We change delays.

$\Leftarrow$ The graph is acyclic. Compute its topological sort and move the "last" block to the right.

$\hookrightarrow$ obtain ρ' with the same sequence of actions as ρ but ρ' contains strictly less races.

Repeat until all races are removed.

Theorem 4 (Contribution)

An AT contains a run that cannot be wiggled if and only if the block graph of that run is cyclic.

Theorem 4 (Contribution)

An AT contains a run that cannot be wiggled if and only if the block graph of that run is cyclic.

This can be encoded in an MSO formula that is satisfied if

Theorem 4 (Contribution)

An AT contains a run that cannot be wiggled if and only if the block graph of that run is cyclic.

This can be encoded in an MSO formula that is satisfied if

- there exists a run of the region automaton

Theorem 4 (Contribution)

An AT contains a run that cannot be wiggled if and only if the block graph of that run is cyclic.

This can be encoded in an MSO formula that is satisfied if

- there exists a run of the region automaton that cannot be wiggled,

Theorem 4 (Contribution)

An AT contains a run that cannot be wiggled if and only if the block graph of that run is cyclic.

This can be encoded in an MSO formula that is satisfied if

- there exists a run of the region automaton that cannot be wiggled,
- i.e., there are concurrent actions

Theorem 4 (Contribution)

An AT contains a run that cannot be wiggled if and only if the block graph of that run is cyclic.

This can be encoded in an MSO formula that is satisfied if

- there exists a run of the region automaton that cannot be wiggled,
- i.e., there are concurrent actions inducing a cyclic block graph.

Theorem 4 (Contribution)

An AT contains a run that cannot be wiggled if and only if the block graph of that run is cyclic.

This can be encoded in an MSO formula that is satisfied if

- there exists a run of the region automaton that cannot be wiggled,
- i.e., there are concurrent actions inducing a cyclic block graph.

Can be written with three quantifiers alternations $\rightsquigarrow$ 3EXP.

Theorem 5 (Contribution)

Fix an automaton and a state q. Deciding whether there exists an execution of the automaton that reaches q is PSPACE*-complete.*

Theorem 6 (Contribution)

Deciding whether an AT contains an execution in which some concurrency cannot be avoided is PSPACE*-hard and in* 3EXP*.*

Thank you!

For all details, see Bruyère et al., "Automata with Timers", 2023.

References I

- Angluin, Dana. "Learning Regular Sets from Queries and Counterexamples". In: *Inf. Comput.* 75.2 (1987), pp. 87–106. DOI: 10.1016/0890-5401(87)90052-6.
- Baier, Christel and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9.
- Bruyère, Véronique et al. "Automata with Timers". In: *CoRR* abs/2305.07451 (2023). DOI: 10.48550/arXiv.2305.07451. arXiv: 2305.07451. URL: https://doi.org/10.48550/arXiv.2305.07451.
- Clarke, Edmund M. et al., eds. *Handbook of Model Checking*. Springer, 2018. ISBN: 978-3-319-10574-1. DOI: 10.1007/978-3-319-10575-8.

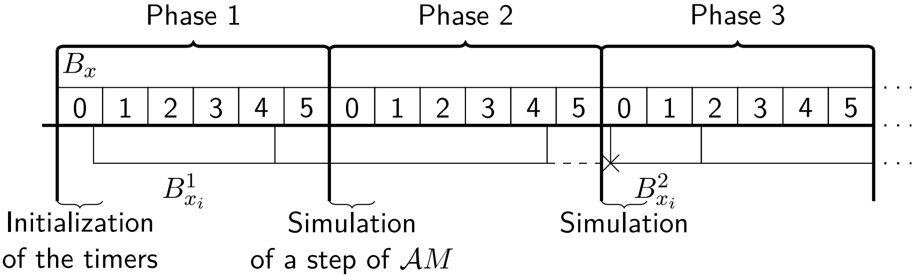


Figure 14: The beginning of a run for the reachability PSPACE-hardness proof.

Let $\mathcal{A} = (X, I, Q, q_0, \chi, \delta)$ be an automaton with timers. 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 $\mathrm{frac}(\kappa'(x_1)) = 0$,
- $\mathrm{frac}(\kappa(x_1)) \leq \mathrm{frac}(\kappa(x_2))$ iff $\mathrm{frac}(\kappa'(x_1)) \leq \mathrm{frac}(\kappa'(x_2))$.

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 $\mathcal{A}$.

Lemma 7

Let $\mathcal{A} = (X, I, Q, q_0, \chi, \delta)$ *be an automaton with timers 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* $[\![(q', \kappa')]\!]_{\cong}$*.*

Corollary 8

Let $\mathcal{A}$ be an automaton with timers 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 $\mathcal{C}$ participates in exactly two races described by this cycle,*
- *for any race described by $\mathcal{C}$, exactly two blocks of $\mathcal{C}$ participate in the race,*
- *the blocks $B = (k_1 \dots k_m, \gamma)$ of $\mathcal{C}$ satisfy either $m \geq 2$, or $m = 1$ and $\gamma = \bullet$.*