When are timed automata determinizable?

Christel Baier [1] Nathalie Bertrand [2] Patricia Bouyer [3] Thomas Brihaye [4]

[1] Technische Universität Dresden – Germany
[2] IRISA – INRIA Rennes – France
[3] LSV – CNRS & ENS Cachan – France
[4] Université de Mons – Belgium

June 12, 2009

Outline

Verification and formal languages

Real systems $\mathit{Syst} \rightsquigarrow \mathcal{A}_{\mathit{Syst}}$

Specification $\mathit{Spec} \rightsquigarrow \mathcal{A}_{\varphi}$

Verification and formal languages

Real systems $\mathit{Syst} \rightsquigarrow \mathcal{A}_{\mathit{Syst}}$

Specification $\mathit{Spec} \rightsquigarrow \mathcal{A}_{\varphi}$

The question $\mathit{Syst} \models \mathit{Spec}$? $\rightsquigarrow$ $\mathcal{L}(\mathcal{A}_{\mathit{Syst}}) \subseteq \mathcal{L}(\mathcal{A}_{\varphi})$?

Verification and formal languages

Real systems $\mathit{Syst} \rightsquigarrow \mathcal{A}_{\mathit{Syst}}$

Specification $\mathit{Spec} \rightsquigarrow \mathcal{A}_{\varphi}$

The question $\mathit{Syst} \models \mathit{Spec}$? $\rightsquigarrow \mathcal{L}(\mathcal{A}_{\mathit{Syst}}) \subseteq \mathcal{L}(\mathcal{A}_{\varphi})$?

Importance to have efficient algorithms to check language inclusion!

Verification and formal languages

Real systems $\mathit{Syst} \rightsquigarrow \mathcal{A}_{\mathit{Syst}}$

Specification $\mathit{Spec} \rightsquigarrow \mathcal{A}_{\varphi}$

The question $\mathit{Syst} \models \mathit{Spec}$? $\rightsquigarrow \mathcal{L}(\mathcal{A}_{\mathit{Syst}}) \subseteq \mathcal{L}(\mathcal{A}_{\varphi})$?

Importance to have efficient algorithms to check language inclusion!

Two special instances:

- The emptiness problem: $\mathcal{L}(\mathcal{A}) \subseteq \emptyset$
- The universality problem: $\Sigma^* \subseteq \mathcal{L}(\mathcal{A})$

Verification and formal languages (finite automata)

Real systems $\mathit{Syst} \rightsquigarrow \mathcal{A}_{\mathit{Syst}}$

Specification $\mathit{Spec} \rightsquigarrow \mathcal{A}_{\varphi}$

The question $\mathit{Syst} \models \mathit{Spec}$? $\rightsquigarrow \mathcal{L}(\mathcal{A}_{\mathit{Syst}}) \subseteq \mathcal{L}(\mathcal{A}_{\varphi})$?

Importance to have efficient algorithms to check language inclusion!
(PSPACE-complete)

Two special instances:

- The emptiness problem: $\mathcal{L}(\mathcal{A}) \subseteq \emptyset$ (NL-complete)
- The universality problem: $\Sigma^* \subseteq \mathcal{L}(\mathcal{A})$ (PSPACE-complete)

Verification and formal languages (finite automata)

Real systems $\quad Syst \rightsquigarrow \mathcal{A}_{Syst}$

Specification $\quad Spec \rightsquigarrow \mathcal{A}_{\varphi}$

The question $\quad Syst \models Spec\ ? \rightsquigarrow \mathcal{L}(\mathcal{A}_{Syst}) \subseteq \mathcal{L}(\mathcal{A}_{\varphi})\ ?$

Importance to have efficient algorithms to check language inclusion!
(PSPACE-complete)

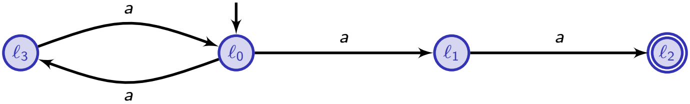
Two special instances:

- The emptiness problem: $\mathcal{L}(\mathcal{A}) \subseteq \emptyset$ (NL-complete)
- The universality problem: $\Sigma^* \subseteq \mathcal{L}(\mathcal{A})$ (PSPACE-complete)

Every finite automaton is determinizable
into an exponential size finite automaton.

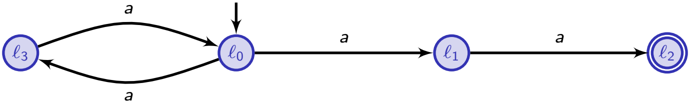
Determinizing finite automata (on finite words)

Example: $\mathcal{L}(\mathcal{A}) = (aa)^* aa$

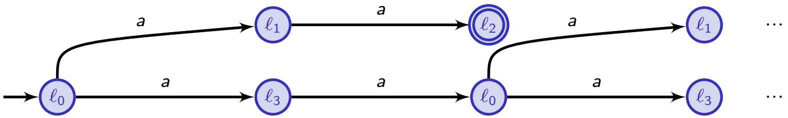


Determinizing finite automata (on finite words)

Example: $\mathcal{L}(\mathcal{A}) = (aa)^*aa$

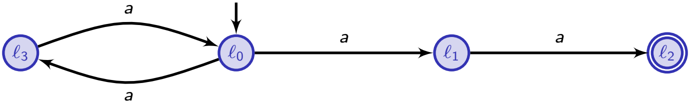


Unfolding $\mathcal{A}$

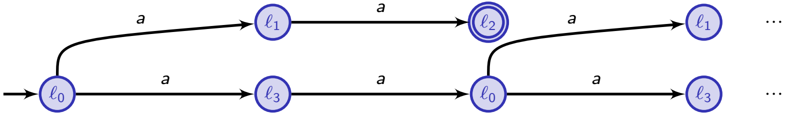


Determinizing finite automata (on finite words)

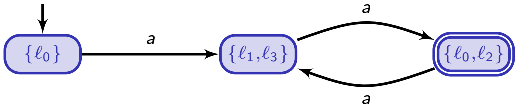
Example: $\mathcal{L}(\mathcal{A}) = (aa)^* aa$



Unfolding $\mathcal{A}$

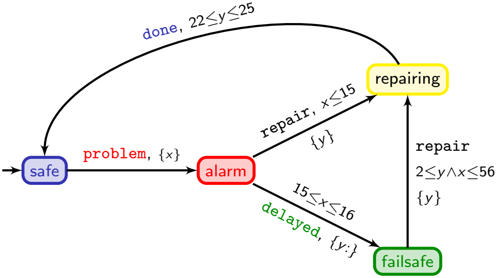


A deterministic version of $\mathcal{A}$



Outline

What is a timed automaton?



	safe	$\xrightarrow{23}$	safe	$\xrightarrow{\texttt{problem}}$	alarm	$\xrightarrow{15.6}$	alarm	$\xrightarrow{\texttt{delayed}}$	failsafe	
x	0		23		0		15.6		15.6	···
y	0		23		23		38.6		0	

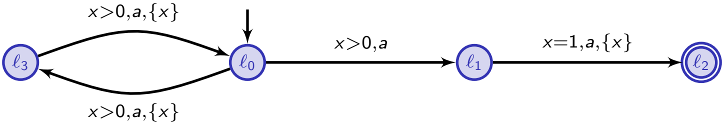
	failsafe	$\xrightarrow{2.3}$	failsafe	$\xrightarrow{\texttt{repair}}$	repairing	$\xrightarrow{22.1}$	repairing	$\xrightarrow{\texttt{done}}$	safe
···	15.6		17.9		17.9		40		40
	0		2.3		0		22.1		22.1

$\rightsquigarrow$ It reads the timed word $(\texttt{problem}, 23)(\texttt{delayed}, 38.6)(\texttt{repair}, 40.9)(\texttt{done}, 63)$

Timed languages accepted by timed automata

Example

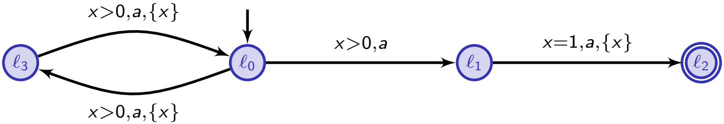
Let $\mathcal{A}$ be the following timed automaton:



Timed languages accepted by timed automata

Example

Let $\mathcal{A}$ be the following timed automaton:

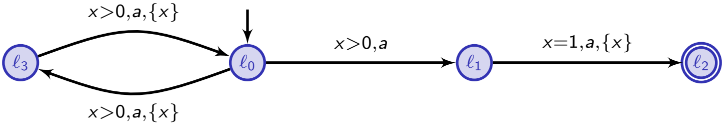


$$\mathcal{L}(\mathcal{A}) = \{(a, t_1)(a, t_2)\cdots(a, t_{2n}) \mid n \geq 1,\ 0 < t_1 < t_2 < \cdots < t_{2n-1} \text{ and } t_{2n} - t_{2n-2} = 1\}$$

Timed languages accepted by timed automata

Example

Let $\mathcal{A}$ be the following timed automaton:



$$\mathcal{L}(\mathcal{A}) = \{(a,t_1)(a,t_2)\cdots(a,t_{2n}) \mid n \geq 1,\ 0 < t_1 < t_2 < \cdots < t_{2n-1} \text{ and } t_{2n} - t_{2n-2} = 1\}$$

The timed word $w = (a,0.2)(a,0.5)(a,1.2)(a,1.5)$ is in $\mathcal{L}(\mathcal{A})$.

Results on timed automata [AD90,AD94]

Emptiness problem

The emptiness problem is PSPACE-complete for timed automata.

Results on timed automata [AD90,AD94]

Emptiness problem

The emptiness problem is PSPACE-complete for timed automata.

Universality problem

The universality problem is undecidable for timed automata.

Results on timed automata [AD90,AD94]

Emptiness problem

The emptiness problem is PSPACE-complete for timed automata.

Universality problem

The universality problem is undecidable for timed automata.

Inclusion problem

The (language) inclusion problem is undecidable for timed automata.

Results on timed automata [AD90,AD94]

Emptiness problem

The emptiness problem is PSPACE-complete for timed automata.

Universality problem

The universality problem is undecidable for timed automata.

Inclusion problem

The (language) inclusion problem is undecidable for timed automata.

$\rightsquigarrow$ prevents using timed automata as a specification language

Timed automata and determinism

Deterministic timed automaton

A timed automaton $\mathcal{A}$ is deterministic whenever for every timed word w, there is at most one initial run (starting from $(\ell_0, 0)$) which reads u.

Timed automata and determinism

Deterministic timed automaton

A timed automaton $\mathcal{A}$ is deterministic whenever for every timed word w, there is at most one initial run (starting from $(\ell_0, 0)$) which reads u.

Theorem [AD94]

Checking universality (and language inclusion) is PSPACE-complete for deterministic timed automata.

Timed automata and determinism

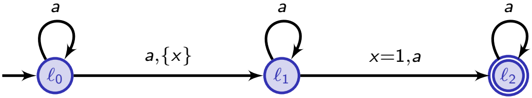
Deterministic timed automaton

A timed automaton $\mathcal{A}$ is deterministic whenever for every timed word w, there is at most one initial run (starting from $(\ell_0, 0)$) which reads u.

Theorem [AD94]

Checking universality (and language inclusion) is PSPACE-complete for deterministic timed automata.

There exist timed automata that are not determinizable [AD90]



$$\mathcal{L}(\mathcal{A}) = \{(a, t_1) \ldots (a, t_n) \mid n \geq 2 \text{ and } \exists i < j \text{ s.t. } t_j - t_i = 1\}$$

Timed automata and determinism

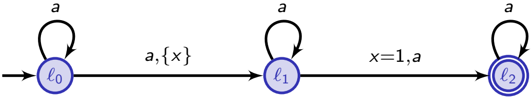
Deterministic timed automaton

A timed automaton $\mathcal{A}$ is deterministic whenever for every timed word w, there is at most one initial run (starting from $(\ell_0, 0)$) which reads u.

Theorem [AD94]

Checking universality (and language inclusion) is PSPACE-complete for deterministic timed automata.

There exist timed automata that are not determinizable [AD90]



$$\mathcal{L}(\mathcal{A}) = \{(a, t_1) \ldots (a, t_n) \mid n \geq 2 \text{ and } \exists i < j \text{ s.t. } t_j - t_i = 1\}$$

Theorem [Tri03,Fin06]

We cannot decide whether a timed automaton can be determinized.

Event-clock timed automata [AFH94]

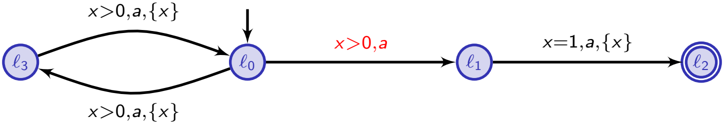
Event-clock timed automata

An event-clock timed automaton is a timed automaton that contains only event-recording clocks: for every letter $a \in \Sigma$, there is a clock x_a, which is reset at every occurrence of an a.

Event-clock timed automata [AFH94]

Event-clock timed automata

An event-clock timed automaton is a timed automaton that contains only event-recording clocks: for every letter $a \in \Sigma$, there is a clock x_a, which is reset at every occurrence of an a.

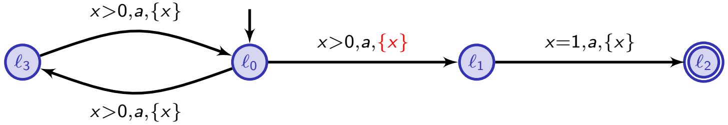


$\mathcal{A}$ is not an event-clock timed automaton

Event-clock timed automata [AFH94]

Event-clock timed automata

An event-clock timed automaton is a timed automaton that contains only event-recording clocks: for every letter $a \in \Sigma$, there is a clock x_a, which is reset at every occurrence of an a.



$\mathcal{A}$ is an event-clock timed automaton

Event-clock timed automata [AFH94]

Event-clock timed automata

An event-clock timed automaton is a timed automaton that contains only event-recording clocks: for every letter $a \in \Sigma$, there is a clock x_a, which is reset at every occurrence of an a.

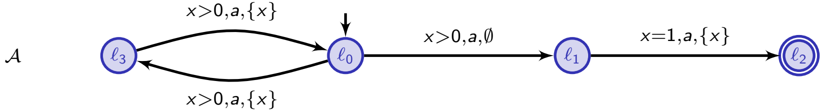
Theorem

- Event-clock timed automata are determinizable.
- Checking universality (and language inclusion) is PSPACE-complete for event-clock timed automata.

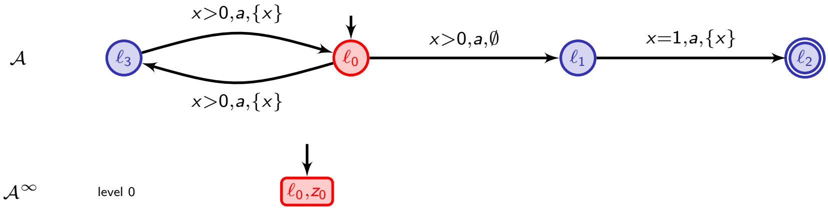
Outline

1. General framework
2. Timed automata
3. Towards a determinization procedure for timed automata...
   - Unfolding
   - Region equivalence
   - Symbolic determinization
   - Reducing the number of clocks
   - Reducing the number of locations
4. When can we apply the procedure?
5. Conclusion

Unfolding

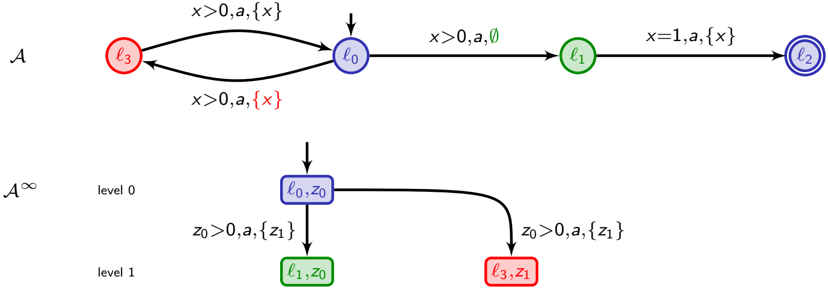


Unfolding

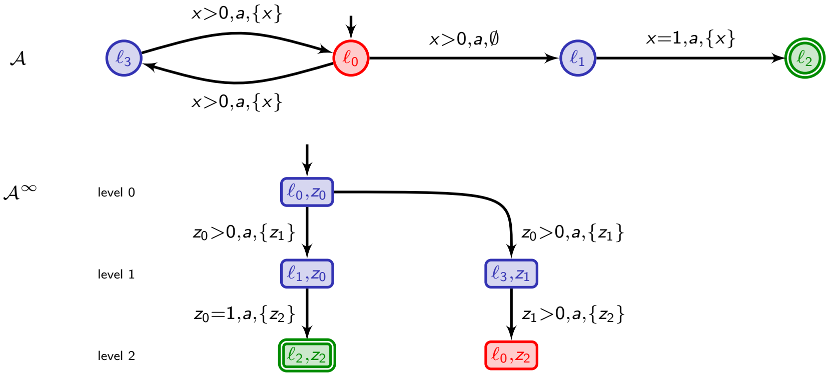


$\mathcal{A}$
$x>0, a, \{x\}$
$x>0, a, \{x\}$
$x>0, a, \emptyset$
$x=1, a, \{x\}$
ℓ_3
ℓ_0
ℓ_1
ℓ_2
$\mathcal{A}^\infty$
level 0
ℓ_0, z_0

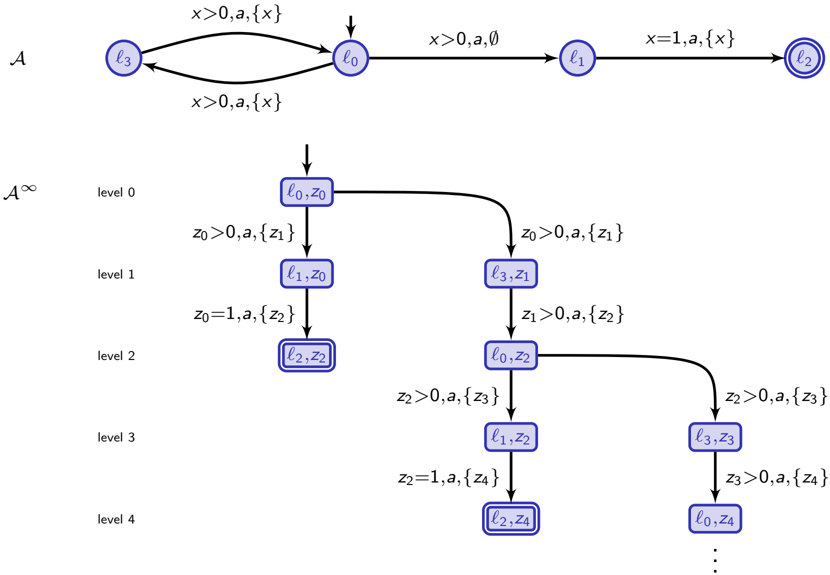
Unfolding


$\mathcal{A}$
$x>0,a,\{x\}$
$x>0,a,\emptyset$
$x=1,a,\{x\}$
ℓ_3
ℓ_0
ℓ_1
ℓ_2
$x>0,a,\{x\}$
$\mathcal{A}^\infty$
level 0
ℓ_0,z_0
$z_0>0,a,\{z_1\}$
$z_0>0,a,\{z_1\}$
level 1
ℓ_1,z_0
ℓ_3,z_1

Unfolding


$\mathcal{A}$
$x>0, a, \{x\}$
$x>0, a, \{x\}$
$x>0, a, \emptyset$
$x=1, a, \{x\}$
ℓ_3
ℓ_0
ℓ_1
ℓ_2
$\mathcal{A}^\infty$
level 0
level 1
level 2
ℓ_0, z_0
$z_0>0, a, \{z_1\}$
$z_0>0, a, \{z_1\}$
ℓ_1, z_0
ℓ_3, z_1
$z_0=1, a, \{z_2\}$
$z_1>0, a, \{z_2\}$
ℓ_2, z_2
ℓ_0, z_2

Unfolding


$\mathcal{A}$
$x>0,a,\{x\}$
$x>0,a,\{x\}$
$x>0,a,\emptyset$
$x=1,a,\{x\}$
ℓ_3
ℓ_0
ℓ_1
ℓ_2
$\mathcal{A}^\infty$
level 0
level 1
level 2
level 3
level 4
ℓ_0,z_0
$z_0>0,a,\{z_1\}$
$z_0>0,a,\{z_1\}$
ℓ_1,z_0
ℓ_3,z_1
$z_0=1,a,\{z_2\}$
$z_1>0,a,\{z_2\}$
ℓ_2,z_2
ℓ_0,z_2
$z_2>0,a,\{z_3\}$
$z_2>0,a,\{z_3\}$
ℓ_1,z_2
ℓ_3,z_3
$z_2=1,a,\{z_4\}$
$z_3>0,a,\{z_4\}$
ℓ_2,z_4
ℓ_0,z_4
⋮

Properties of the unfolding

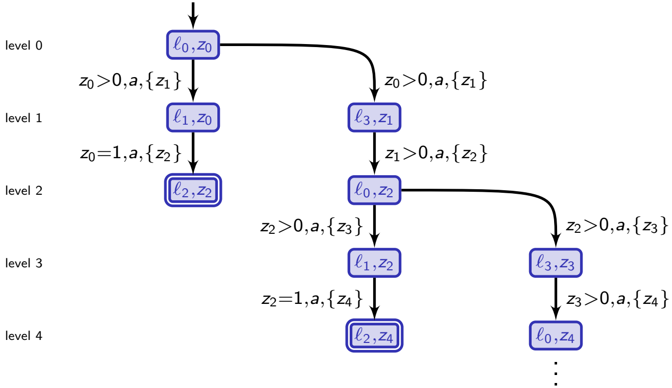
Advantage of the unfolding: "input-determinacy"

Given a finite timed word w, there is a unique valuation v_w such that every initial run reading w ends in a configuration (n, v_w) with $level(n) = |w|$.

Properties of the unfolding

Advantage of the unfolding: "input-determinacy"

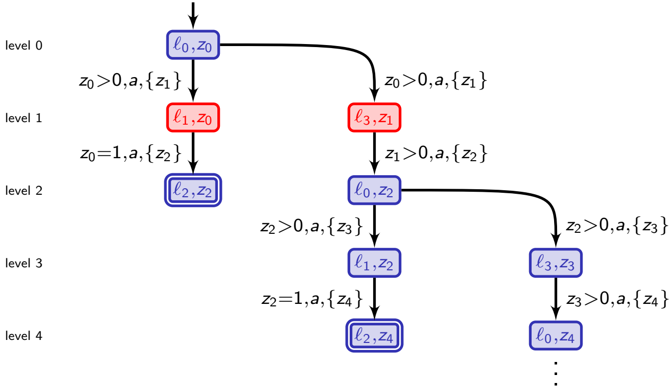
Given a finite timed word w, there is a unique valuation v_w such that every initial run reading w ends in a configuration (n, v_w) with $level(n) = |w|$.



Properties of the unfolding

Advantage of the unfolding: "input-determinacy"

Given a finite timed word w, there is a unique valuation v_w such that every initial run reading w ends in a configuration (n, v_w) with $level(n) = |w|$.

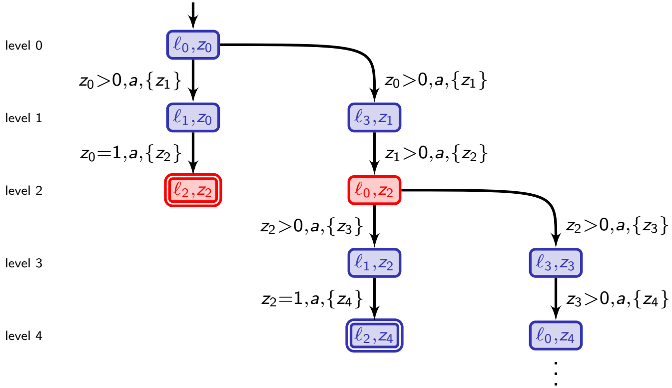


$w = (a, 0.2)$ $\quad\rightsquigarrow\quad$ $v_w = (\underset{z_0}{0.2}, \underset{z_1}{0})$

Properties of the unfolding

Advantage of the unfolding: "input-determinacy"

Given a finite timed word w, there is a unique valuation v_w such that every initial run reading w ends in a configuration (n, v_w) with $level(n) = |w|$.

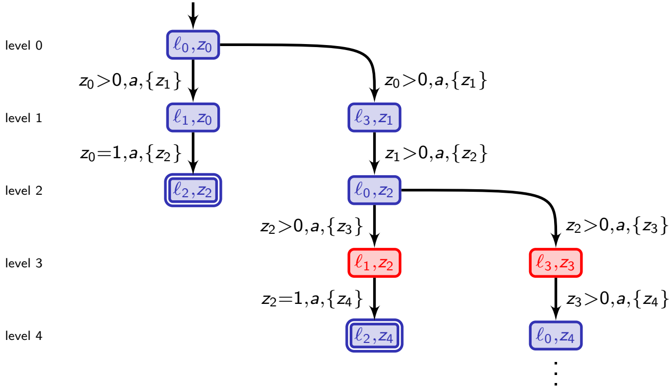


$w = (a, 0.2)(a, 0.5) \quad \rightsquigarrow \quad v_w = (\underset{z_0}{0.5}, \underset{z_1}{0.3}, \underset{z_2}{0})$

Properties of the unfolding

Advantage of the unfolding: "input-determinacy"

Given a finite timed word w, there is a unique valuation v_w such that every initial run reading w ends in a configuration (n, v_w) with $level(n) = |w|$.

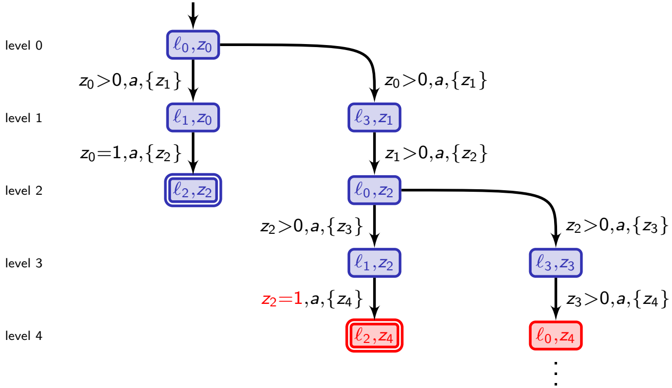


$$w = (a, 0.2)(a, 0.5)(a, 1.2) \qquad \rightsquigarrow \quad v_w = (\underset{z_0}{1.2}, \underset{z_1}{1}, \underset{z_2}{0.7}, \underset{z_3}{0})$$

Properties of the unfolding

Advantage of the unfolding: "input-determinacy"

Given a finite timed word w, there is a unique valuation v_w such that every initial run reading w ends in a configuration (n, v_w) with $level(n) = |w|$.



$$w = (a, 0.2)(a, 0.5)(a, 1.2)(a, 1.5) \rightsquigarrow v_w = (\underset{z_0}{1.5}, \underset{z_1}{1.3}, \underset{z_2}{1}, \underset{z_3}{0.3}, \underset{z_4}{0})$$

Properties of the unfolding

Advantage of the unfolding: "input-determinacy"

Given a finite timed word w, there is a unique valuation v_w such that every initial run reading w ends in a configuration (n, v_w) with $\mathit{level}(n) = |w|$.

Drawbacks of the unfolding

- $\mathcal{A}^\infty$ has infinitely many locations.
- $\mathcal{A}^\infty$ has infinitely many clocks.
- $\mathcal{A}^\infty$ is not deterministic.

Properties of the unfolding

Advantage of the unfolding: "input-determinacy"

Given a finite timed word w, there is a unique valuation v_w such that every initial run reading w ends in a configuration (n, v_w) with $level(n) = |w|$.

Drawbacks of the unfolding

- $\mathcal{A}^\infty$ has infinitely many locations.
- $\mathcal{A}^\infty$ has infinitely many clocks.
- $\mathcal{A}^\infty$ is not deterministic.

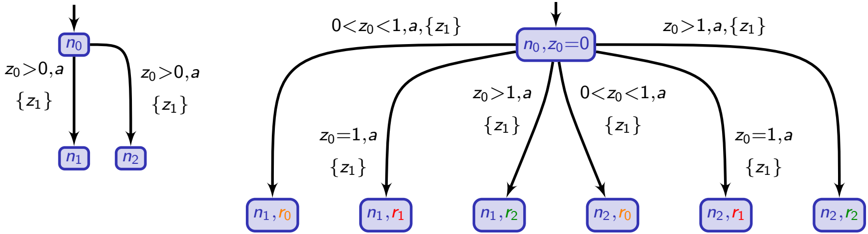
Lemma

$\mathcal{A}$ and $\mathcal{A}^\infty$ are strongly timed bisimilar.
In particular $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}^\infty)$.

Region equivalence on $\mathcal{A}^\infty$

The standard region equivalence naturally extends to $\mathcal{A}^\infty$,

at level i we only consider region over $\{z_1, \ldots, z_i\}$.

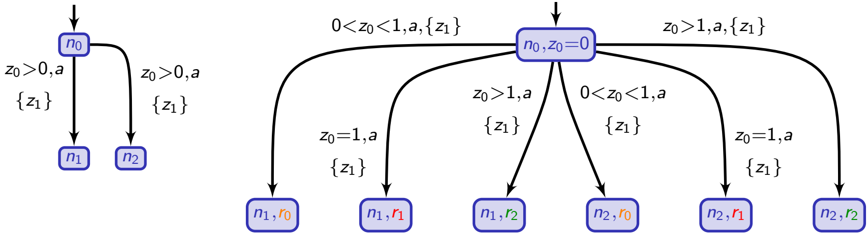


where $r_0 = (0 = z_1 < z_0 < 1)$, $r_1 = (0 = z_1 < z_0 = 1)$, $r_2 = (0 = z_1 < 1 < z_0)$.

Region equivalence on $\mathcal{A}^\infty$

The standard region equivalence naturally extends to $\mathcal{A}^\infty$,

at level i we only consider region over $\{z_1, \ldots, z_i\}$.

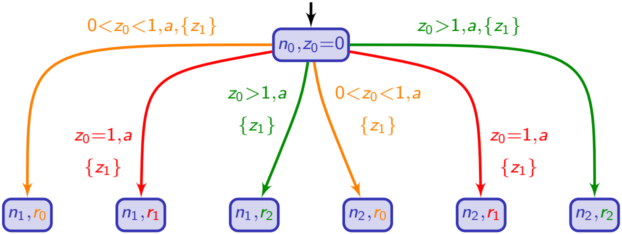


where $r_0 = (0 = z_1 < z_0 < 1)$, $r_1 = (0 = z_1 < z_0 = 1)$, $r_2 = (0 = z_1 < 1 < z_0)$.

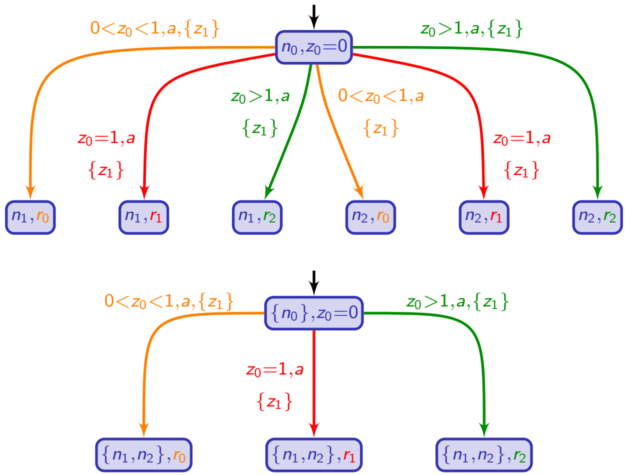
Lemma

$\mathcal{A}^\infty$ and $R(\mathcal{A}^\infty)$ are strongly timed bisimilar.
In particular, $\mathcal{L}(\mathcal{A}^\infty) = \mathcal{L}(R(\mathcal{A}^\infty))$.

Symbolic determinization of $R(\mathcal{A}^\infty)$



Symbolic determinization of $R(\mathcal{A}^\infty)$



Properties of the symbolic determinization

Advantage of SymbDet($R(\mathcal{A}^\infty)$)

SymbDet($R(A^\infty)$) is deterministic!

Properties of the symbolic determinization

Advantage of SymbDet($R(\mathcal{A}^\infty)$)

SymbDet($R(A^\infty)$) is deterministic!

Drawbacks of SymbDet($R(\mathcal{A}^\infty)$)

- SymbDet($R(\mathcal{A}^\infty)$) has infinitely many locations.
- SymbDet($R(\mathcal{A}^\infty)$) has infinitely many clocks.

Properties of the symbolic determinization

Advantage of SymbDet($R(\mathcal{A}^\infty)$)

SymbDet($R(A^\infty)$) is deterministic!

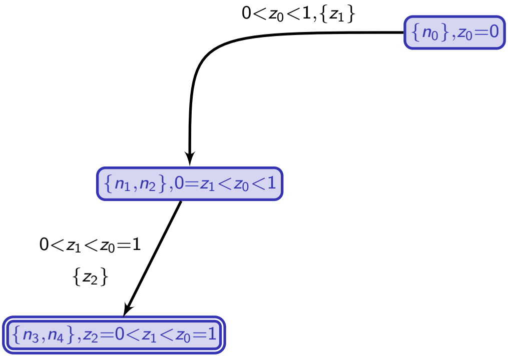
Drawbacks of SymbDet($R(\mathcal{A}^\infty)$)

- SymbDet($R(\mathcal{A}^\infty)$) has infinitely many locations.
- SymbDet($R(\mathcal{A}^\infty)$) has infinitely many clocks.

Lemma

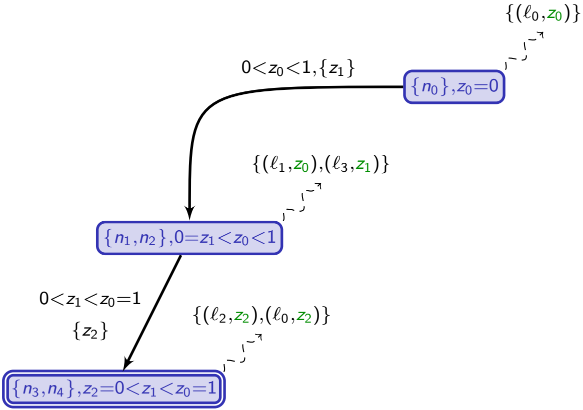
$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\text{SymbDet}(R(\mathcal{A}^\infty)))$.

Notion of active clocks

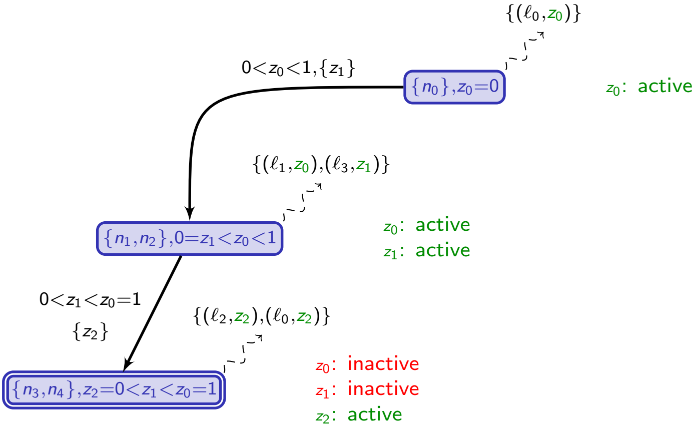


Notion of active clocks

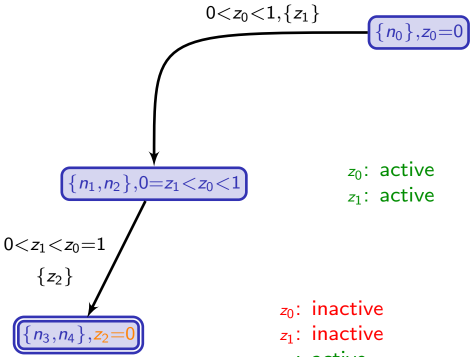
Remember where nodes come from!



Notion of active clocks



Notion of active clocks

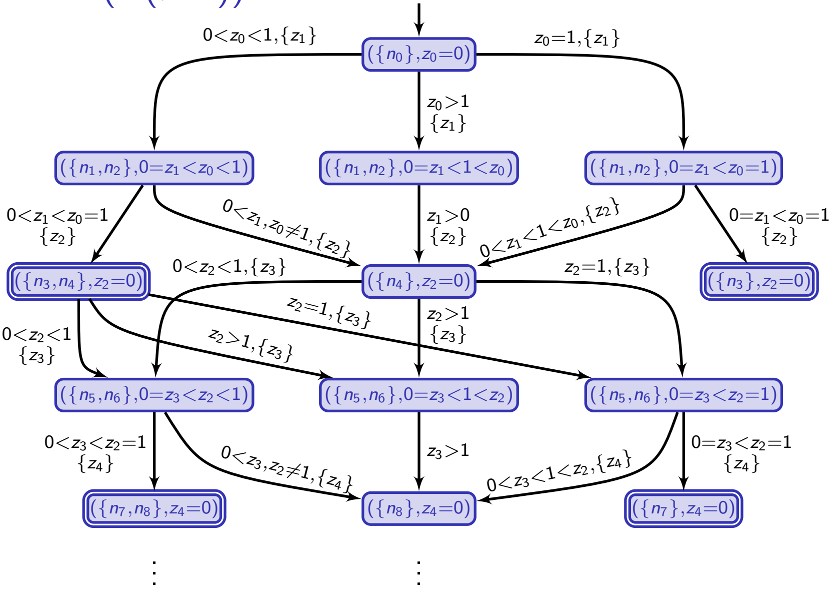


z_0: active

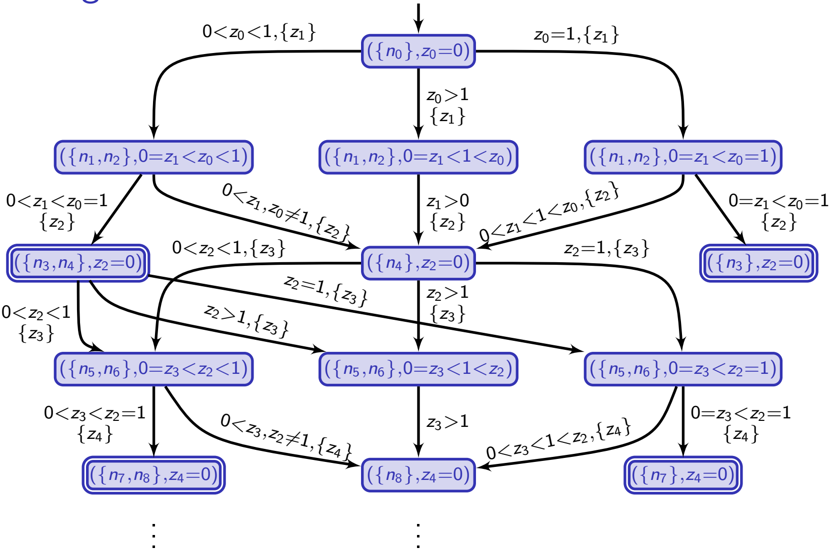
z_0: active
z_1: active

z_0: inactive
z_1: inactive
z_2: active

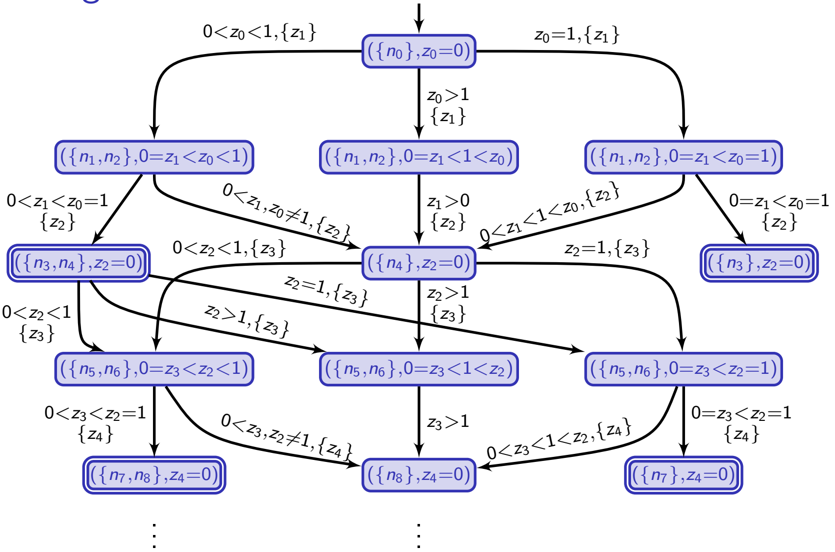
SymbDet($R(\mathcal{A}^\infty)$)



Reducing the number of clocks

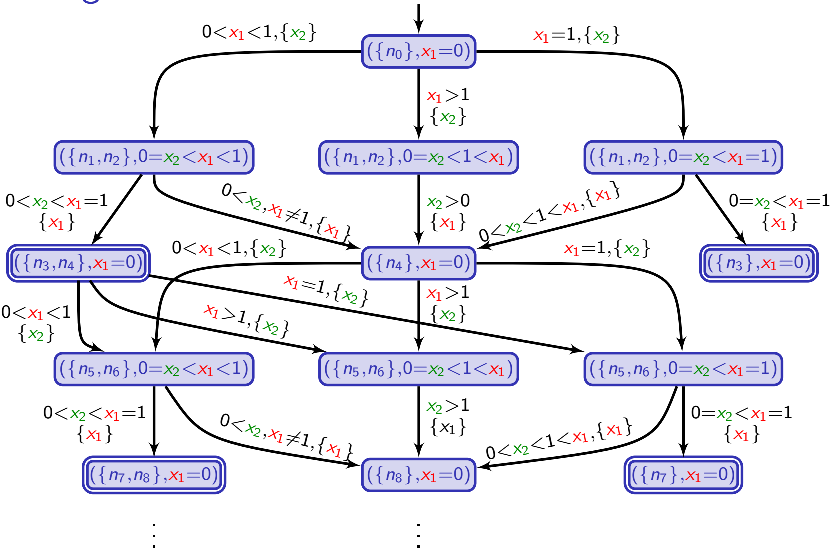


Reducing the number of clocks



Two clocks are sufficient to get full timing information! $\rightsquigarrow$ $\begin{cases} z_{2i} & \mapsto x_1 \\ z_{2i+1} & \mapsto x_2 \end{cases}$

Reducing the number of clocks



Two clocks are sufficient to get full timing information! $\rightsquigarrow$ $\begin{cases} z_{2i} & \mapsto x_1 \\ z_{2i+1} & \mapsto x_2 \end{cases}$

Properties of the clock reduction

Given $\gamma \in \mathbb{N}$, we say that SymbDet($R(\mathcal{A}^\infty)$) is γ-clock-bounded if in every node, the number of active clocks is bounded by γ.

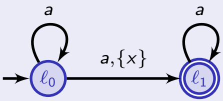
Properties of the clock reduction

Given $\gamma \in \mathbb{N}$, we say that SymbDet($R(\mathcal{A}^\infty)$) is γ-clock-bounded if in every node, the number of active clocks is bounded by γ.
In our case: $\gamma = 2$.

Properties of the clock reduction

Given $\gamma \in \mathbb{N}$, we say that SymbDet($R(\mathcal{A}^\infty)$) is γ-clock-bounded if in every node, the number of active clocks is bounded by γ.
In our case: $\gamma = 2$.

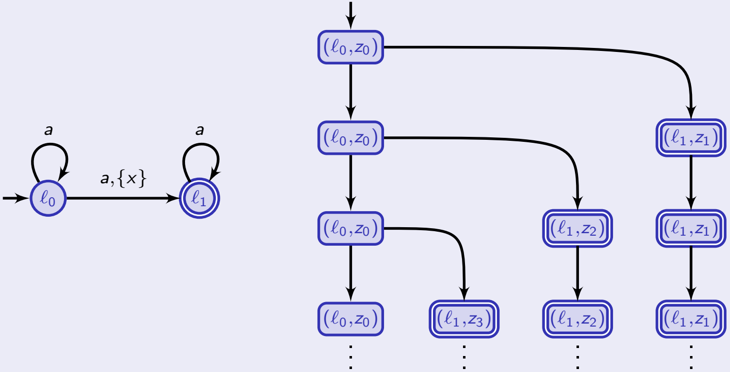
Not all SymbDet($R(\mathcal{A}^\infty)$) are γ-clock-bounded



Properties of the clock reduction

Given $\gamma \in \mathbb{N}$, we say that SymbDet($R(\mathcal{A}^\infty)$) is γ-clock-bounded if in every node, the number of active clocks is bounded by γ.
In our case: $\gamma = 2$.

Not all SymbDet($R(\mathcal{A}^\infty)$) are γ-clock-bounded



Properties of the clock reduction

Given $\gamma \in \mathbb{N}$, we say that $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is γ-clock-bounded if in every node, the number of active clocks is bounded by γ.
In our case: $\gamma = 2$.

In case $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is γ-clock-bounded, we construct using a deterministic policy $\Gamma_\gamma(\mathsf{SymbDet}(R(\mathcal{A}^\infty)))$, an equivalent timed system with clocks $\{x_1, \ldots, x_\gamma\}$.

Properties of the clock reduction

Given $\gamma \in \mathbb{N}$, we say that SymbDet($R(\mathcal{A}^\infty)$) is γ-clock-bounded if in every node, the number of active clocks is bounded by γ.
In our case: $\gamma = 2$.

In case SymbDet($R(\mathcal{A}^\infty)$) is γ-clock-bounded, we construct using a deterministic policy Γ_γ(SymbDet($R(\mathcal{A}^\infty)$)), an equivalent timed system with clocks $\{x_1, \ldots, x_\gamma\}$.

Advantages of Γ_γ(SymbDet($R(\mathcal{A}^\infty)$))

- Γ_γ(SymbDet($R(\mathcal{A}^\infty)$)) is deterministic!
- Γ_γ(SymbDet($R(\mathcal{A}^\infty)$)) has finitely many clocks.

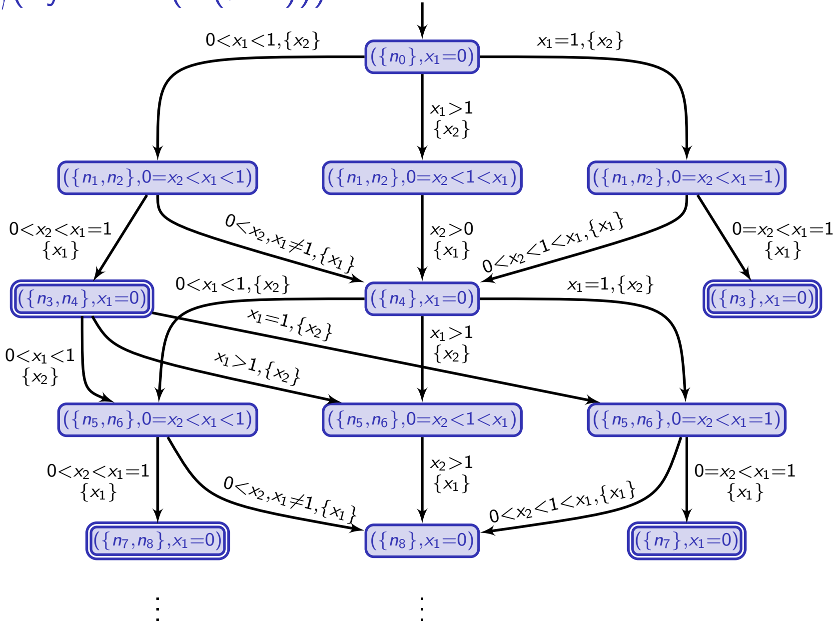
Drawback of Γ_γ(SymbDet($R(\mathcal{A}^\infty)$))

Γ_γ(SymbDet($R(\mathcal{A}^\infty)$)) has infinitely many locations.

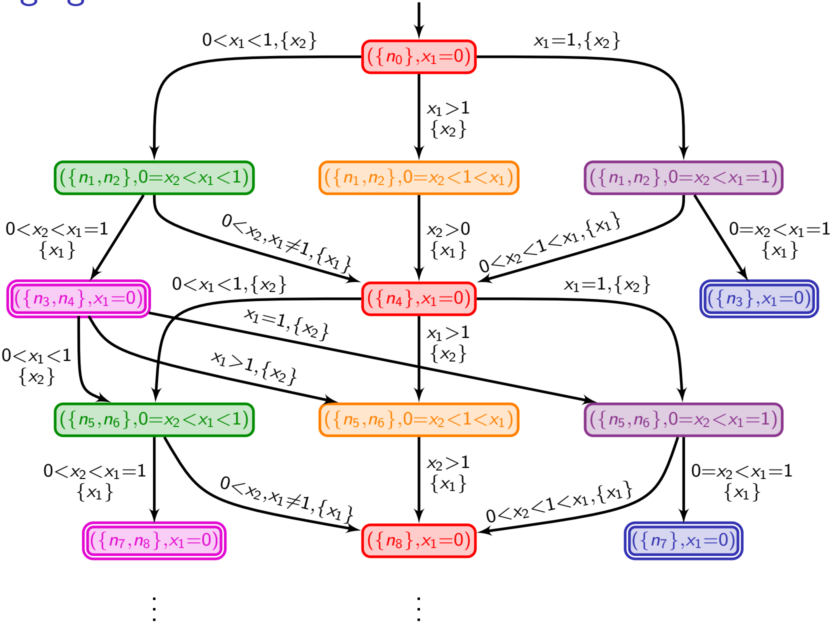
Lemma

$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\Gamma_\gamma(\text{SymbDet}(R(\mathcal{A}^\infty))))$.

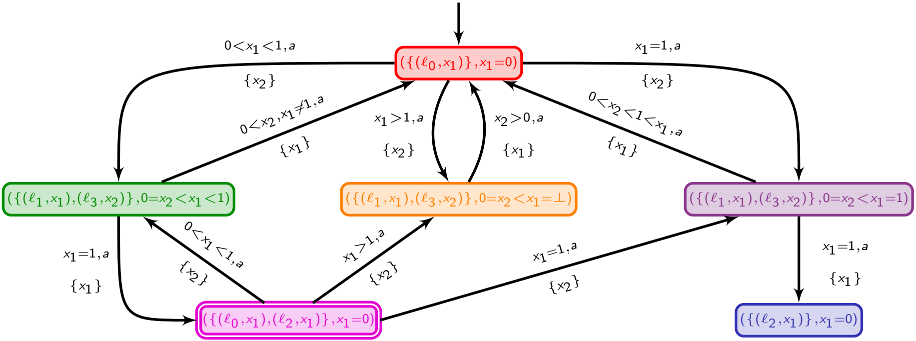
$\Gamma_\gamma(\mathsf{SymbDet}(R(\mathcal{A}^\infty)))$



Merging “identical” nodes



A deterministic timed automaton equivalent to $\mathcal{A}$



Properties of the location reduction

In case $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is γ-clock-bounded, we define $\mathcal{B}_{\mathcal{A},\gamma}$ obtained by merging the nodes of $\Gamma_\gamma(\mathsf{SymbDet}(R(\mathcal{A}^\infty)))$ with "the same labels".

Theorem

In case $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is γ-clock-bounded, $\mathcal{B}_{\mathcal{A},\gamma}$ is a deterministic timed automaton such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B}_{\mathcal{A},\gamma})$.

Outline

When can we apply our procedure?

We need to have that $\mathsf{SymbDet}(R(\mathcal{A}^{\infty}))$ is γ-clock bounded.

When can we apply our procedure?

We need to have that $\mathrm{SymbDet}(R(\mathcal{A}^\infty))$ is γ-clock bounded.

The p-assumption

Given $p \in \mathbb{N}$, $\mathcal{A}$ satisfies the *p-assumption* if for every $n \geq p$, for every run

$$\varrho = (\ell_0, v_0) \xrightarrow{\tau_1, a_1} (\ell_1, v_1) \ldots \xrightarrow{\tau_n, a_n} (\ell_n, v_n)$$

for every clock $x \in X$, either x is reset along ϱ or $v_n(x) > M$.

When can we apply our procedure?

We need to have that $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is γ-clock bounded.

The p-assumption

Given $p \in \mathbb{N}$, $\mathcal{A}$ satisfies the p-assumption if for every $n \geq p$, for every run

$$\varrho = (\ell_0, v_0) \xrightarrow{\tau_1, a_1} (\ell_1, v_1) \ldots \xrightarrow{\tau_n, a_n} (\ell_n, v_n)$$

for every clock $x \in X$, either x is reset along ϱ or $v_n(x) > M$.

If $\mathcal{A}$ satisfies the p-assumption then $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is p-clock bounded.

When can we apply our procedure?

We need to have that $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is γ-clock bounded.

The p-assumption

Given $p \in \mathbb{N}$, $\mathcal{A}$ satisfies the p-assumption if for every $n \geq p$, for every run

$$\varrho = (\ell_0, v_0) \xrightarrow{\tau_1, a_1} (\ell_1, v_1) \ldots \xrightarrow{\tau_n, a_n} (\ell_n, v_n)$$

for every clock $x \in X$, either x is reset along ϱ or $v_n(x) > M$.

If $\mathcal{A}$ satisfies the p-assumption then $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is p-clock bounded.

Classes to which the procedure applies

- Event-clock timed automata (with $\gamma = |\Sigma|$)
- Strongly non-Zeno timed automata (since they satisfy the p-assumption)
- timed automata with integer resets [SPKM08]

Hardness issues

We can prove EXPSPACE-hardness of:

- the universality problem for timed automata satisfying the p-assumption and for timed automata with integer resets;
- the inclusion problem for strongly non-Zeno timed automata.

The results

Summary of the complexity results

	size of the det. TA	universality problem	inclusion problem
TA_p	doubly exp.	EXPSPACE-compl.	EXPSPACE-compl.
SnZTA	doubly exp.	trivial	EXPSPACE-compl.
ECTA [AFH94]	exp.	PSPACE-compl.	PSPACE-compl.
IRTA [SPKM08]	doubly exp.	EXPSPACE-compl.	EXPSPACE-compl.

The results

Summary of the complexity results

	size of the det. TA	universality problem	inclusion problem
TA_p	doubly exp.	EXPSPACE-compl.	EXPSPACE-compl.
SnZTA	doubly exp.	trivial	EXPSPACE-compl.
ECTA [AFH94]	exp.	PSPACE-compl.	PSPACE-compl.
IRTA [SPKM08]	doubly exp.	EXPSPACE-compl.	EXPSPACE-compl.

Remark

In case $\mathcal{A}$ has one clock, $\mathrm{SymbDet}(R(\mathcal{A}^\infty))$ allows to recover the decidability of the universality problem in one-clock TA [OW04].

Outline

Conclusion

What we have done

- We have described a procedure to determinize timed automata...
- ... which terminates for several subclasses of timed automata
  - event-clock timed automata
  - timed automata with integer resets
  - strongly non-Zeno timed automata
  - ...
- We recover known results, but also describe new determinizable classes of timed automata.
- This procedure gives optimal complexity bounds.

Conclusion

What we have done

- We have described a procedure to determinize timed automata...
- ... which terminates for several subclasses of timed automata
    - event-clock timed automata
    - timed automata with integer resets
    - strongly non-Zeno timed automata
    - ...
- We recover known results, but also describe new determinizable classes of timed automata.
- This procedure gives optimal complexity bounds.

What we will do now

- We want to see whether other determinizable classes (open timed automata) could fit our framework.
- We will extend to infinite timed words (with a Safra-like construction mixed with our procedure?)