An introduction to timed systems

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France

Outline

Time!

Context: verification of critical systems

Time

- naturally appears in real systems (for ex. protocols, embedded systems)
- appears in properties (for ex. bounded response time)

  "Will the airbag oben within 5ms after the car crashes?"

$\leadsto$ Need of models and specification languages integrating timing aspects

Adding timing informations

- **Untimed case:** sequence of observable events
  a: send message b: receive message

$$a\ b\ a\ b\ a\ b\ a\ b\ a\ b\ \cdots = (a\ b)^\omega$$

Adding timing informations

- **Untimed case:** sequence of observable events
  a: send message b: receive message

$$a\ b\ a\ b\ a\ b\ a\ b\ a\ b\ \cdots = (a\ b)^\omega$$

- **Timed case:** sequence of **dated** observable events

$$(a, d_1)\ (b, d_2)\ (a, d_3)\ (b, d_4)\ (a, d_5)\ (b, d_6)\ \cdots$$

d_1: date at which the first a occurs
d_2: date at which the first b occurs, ...

Adding timing informations

- **Untimed case:** sequence of observable events
  a: send message b: receive message

$$a\ b\ a\ b\ a\ b\ a\ b\ a\ b\ \cdots = (a\ b)^\omega$$

- **Timed case:** sequence of **dated** observable events

$$(a, d_1)\ (b, d_2)\ (a, d_3)\ (b, d_4)\ (a, d_5)\ (b, d_6)\ \cdots$$

  d_1: date at which the first a occurs
  d_2: date at which the first b occurs, ...
  - **Discrete-time semantics:** dates are *e.g.* taken in $\mathbb{N}$
    Ex: $(a, 1)(b, 3)(c, 4)(a, 6)$

Adding timing informations

- **Untimed case:** sequence of observable events
  a: send message b: receive message

$$a\ b\ a\ b\ a\ b\ a\ b\ a\ b\ \cdots = (a\ b)^\omega$$

- **Timed case:** sequence of **dated** observable events

$$(a, d_1)\ (b, d_2)\ (a, d_3)\ (b, d_4)\ (a, d_5)\ (b, d_6)\ \cdots$$

  d_1: date at which the first a occurs
  d_2: date at which the first b occurs, ...
  - **Discrete-time semantics:** dates are *e.g.* taken in $\mathbb{N}$
    Ex: $(a, 1)(b, 3)(c, 4)(a, 6)$
  - **Dense-time semantics:** dates are *e.g.* taken in $\mathbb{Q}_+$, or in $\mathbb{R}_+$
    Ex: $(a, 1.28).(b, 3.1).(c, 3.98)(a, 6.13)$

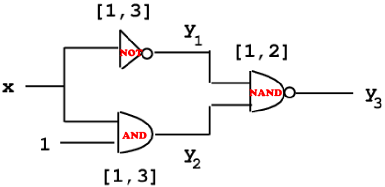
A case for dense-time

Time domain: discrete (*e.g.* $\mathbb{N}$) or dense (*e.g.* $\mathbb{Q}_+$ or $\mathbb{R}_+$)

- A compositionality problem with discrete time
- Dense-time is a more general model than discrete time
- But, can we not always discretize?

A digital circuit [Alu91]

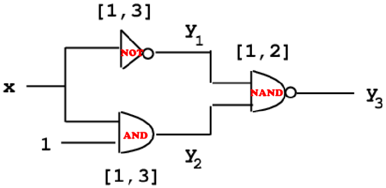
Discussion in the context of reachability problems for asynchronous digital circuits [BS91]



[Alu91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis, 1991*.
[BS91] Brzozowski, Seger. Advances in asynchronous circuit theory *BEATCS'91*.

A digital circuit [Alu91]

Discussion in the context of reachability problems for asynchronous digital circuits [BS91]

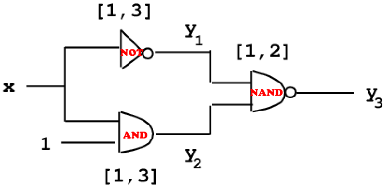


Start with x=0 and y=[101] (stable configuration)

[Alu91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis, 1991*.
[BS91] Brzozowski, Seger. Advances in asynchronous circuit theory *BEATCS'91*.

A digital circuit [Alu91]

Discussion in the context of reachability problems for asynchronous digital circuits [BS91]



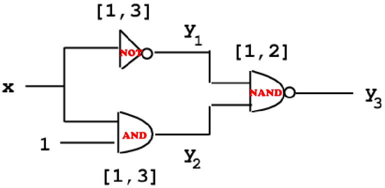
Start with x=0 and y=[101] (stable configuration)

The input x changes to 1. The corresponding stable state is y=[011]

[Alu91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis, 1991.*
[BS91] Brzozowski, Seger. Advances in asynchronous circuit theory *BEATCS'91.*

A digital circuit [Alu91]

Discussion in the context of reachability problems for asynchronous digital circuits [BS91]



Start with x=0 and y=[101] (stable configuration)

The input x changes to 1. The corresponding stable state is y=[011]

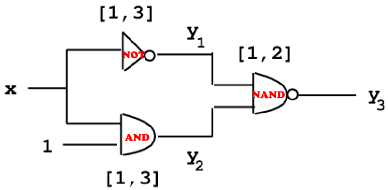
However, many possible behaviours, e.g.

$$[101] \xrightarrow[1.2]{y_2} [111] \xrightarrow[2.5]{y_3} [110] \xrightarrow[2.8]{y_1} [010] \xrightarrow[4.5]{y_3} [011]$$

[Alu91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis, 1991*.
[BS91] Brzozowski, Seger. Advances in asynchronous circuit theory *BEATCS'91*.

A digital circuit [Alu91]

Discussion in the context of reachability problems for asynchronous digital circuits [BS91]



Start with x=0 and y=[101] (stable configuration)

The input x changes to 1. The corresponding stable state is y=[011]

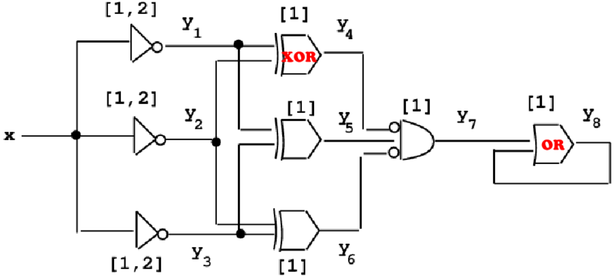
However, many possible behaviours, e.g.

$$[101] \xrightarrow[1.2]{y_2} [111] \xrightarrow[2.5]{y_3} [110] \xrightarrow[2.8]{y_1} [010] \xrightarrow[4.5]{y_3} [011]$$

Reachable configurations: $\{[101], [111], [110], [010], [011], [001]\}$

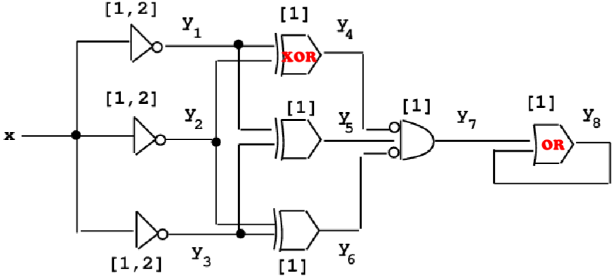
[Alu91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis, 1991.*
[BS91] Brzozowski, Seger. Advances in asynchronous circuit theory *BEATCS'91.*

Is discretizing sufficient? An example [Alu91]



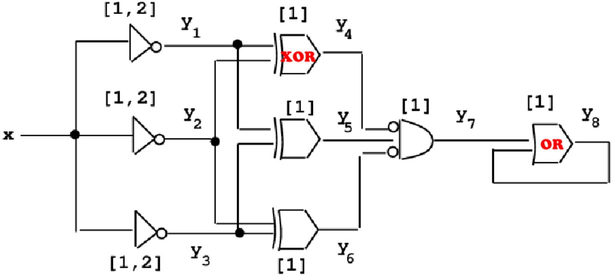
- This digital circuit **is not** 1-discretizable.

Is discretizing sufficient? An example [Alu91]



- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, x is set to 1)

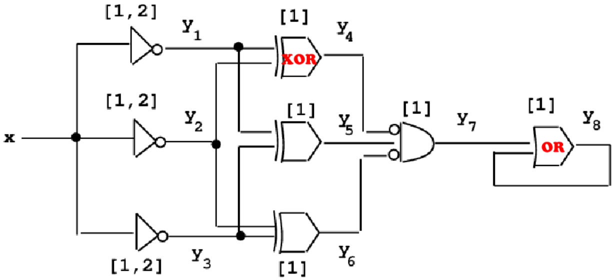
Is discretizing sufficient? An example [Alu91]



- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, x is set to 1)

$$[11100000] \xrightarrow[1]{y_1} [01100000] \xrightarrow[1.5]{y_2} [00100000] \xrightarrow[2]{y_3, y_5} [00001000] \xrightarrow[3]{y_5, y_7} [00000010] \xrightarrow[4]{y_7, y_8} [00000001]$$

Is discretizing sufficient? An example [Alu91]

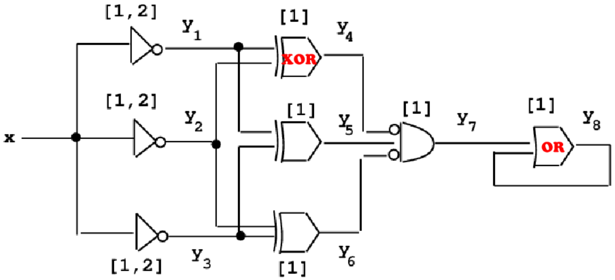


- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, x is set to 1)

$$[11100000] \xrightarrow[1]{y_1} [01100000] \xrightarrow[1.5]{y_2} [00100000] \xrightarrow[2]{y_3, y_5} [00001000] \xrightarrow[3]{y_5, y_7} [00000010] \xrightarrow[4]{y_7, y_8} [00000001]$$

$$[11100000] \xrightarrow[1]{y_1, y_2, y_3} [00000000]$$

Is discretizing sufficient? An example [Alu91]



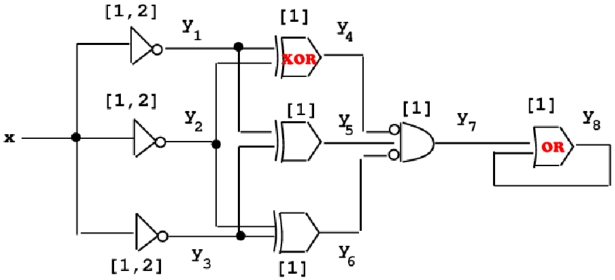
- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, x is set to 1)

$$[11100000] \xrightarrow[1]{y_1} [01100000] \xrightarrow[1.5]{y_2} [00100000] \xrightarrow[2]{y_3, y_5} [00001000] \xrightarrow[3]{y_5, y_7} [00000010] \xrightarrow[4]{y_7, y_8} [00000001]$$

$$[11100000] \xrightarrow[1]{y_1, y_2, y_3} [00000000]$$

$$[11100000] \xrightarrow[1]{y_1} [01111000] \xrightarrow[2]{y_2, y_3, y_4, y_5} [00000000]$$

Is discretizing sufficient? An example [Alu91]



- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, x is set to 1)

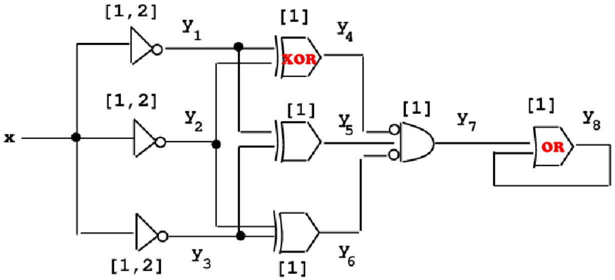
$[11100000] \xrightarrow[1]{y_1} [01100000] \xrightarrow[1.5]{y_2} [00100000] \xrightarrow[2]{y_3,y_5} [00001000] \xrightarrow[3]{y_5,y_7} [00000010] \xrightarrow[4]{y_7,y_8} [00000001]$

$[11100000] \xrightarrow[1]{y_1,y_2,y_3} [00000000]$

$[11100000] \xrightarrow[1]{y_1} [01111000] \xrightarrow[2]{y_2,y_3,y_4,y_5} [00000000]$

$[11100000] \xrightarrow[1]{y_1,y_2} [00100000] \xrightarrow[2]{y_3,y_5,y_6} [00001100] \xrightarrow[3]{y_5,y_6} [00000000]$

Is discretizing sufficient? An example [Alu91]



- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, x is set to 1)

$[11100000] \xrightarrow[1]{y_1} [01100000] \xrightarrow[1.5]{y_2} [00100000] \xrightarrow[2]{y_3, y_5} [00001000] \xrightarrow[3]{y_5, y_7} [00000010] \xrightarrow[4]{y_7, y_8} \boxed{[00000001]}$

$[11100000] \xrightarrow[1]{y_1, y_2, y_3} \boxed{[00000000]}$

$[11100000] \xrightarrow[1]{y_1} [01111000] \xrightarrow[2]{y_2, y_3, y_4, y_5} \boxed{[00000000]}$

$[11100000] \xrightarrow[1]{y_1, y_2} [00100000] \xrightarrow[2]{y_3, y_5, y_6} [00001100] \xrightarrow[3]{y_5, y_6} \boxed{[00000000]}$

Is discretizing sufficient?

Theorem [BS91]

For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time (with granularity $\frac{1}{k}$).

[BS91] Brzozowski, Seger. Advances in asynchronous circuit theory *BEATCS'91*.

Is discretizing sufficient?

Theorem [BS91]

For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time (with granularity $\frac{1}{k}$).

Claim

Finding a correct granularity is as difficult as computing the set of reachable states in dense-time.

[BS91] Brzozowski, Seger. Advances in asynchronous circuit theory *BEATCS'91*.

Is discretizing sufficient?

Theorem [BS91]

For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time (with granularity $\frac{1}{k}$).

Claim

Finding a correct granularity is as difficult as computing the set of reachable states in dense-time.

Further counter-example

There exist systems for which no granularity exists. (see later)

[BS91] Brzozowski, Seger. Advances in asynchronous circuit theory *BEATCS'91*.

Is discretizing sufficient?

Theorem [BS91]

For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time (with granularity $\frac{1}{k}$).

Claim

Finding a correct granularity is as difficult as computing the set of reachable states in dense-time.

Further counter-example

There exist systems for which no granularity exists. (see later)

Hence, we better consider a dense-time domain!

[BS91] Brzozowski, Seger. Advances in asynchronous circuit theory *BEATCS'91*.

Outline

A plethora ot models...

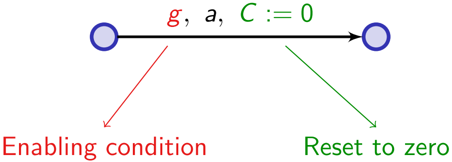
- ... for real-time systems:
    - timed circuits,
    - time(d) Petri nets,
    - timed process algebra,
    - timed automata,
    - ...
- ... and for real-time properties:
    - timed observers,
    - real-time logics: MTL, TPTL, TCTL, QTL, MITL...

A plethora ot models...

- ... for real-time systems:
    - timed circuits,
    - time(d) Petri nets,
    - timed process algebra,
    - timed automata,
    - ...
- ... and for real-time properties:
    - timed observers,
    - real-time logics: MTL, TPTL, TCTL, QTL, MITL...

Timed automata [AD90]

- A finite control structure + variables (clocks)
- A transition is of the form:



- An enabling condition (or guard) is:

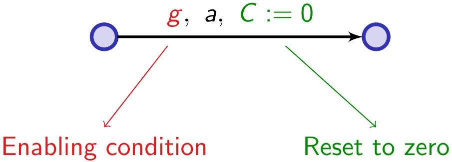
$$g \quad ::= \quad x \sim c \;\mid\; x - y \sim c \;\mid\; g \wedge g$$

where $\sim \in \{<, \leq, =, \geq, >\}$

[AD90] Alur, Dill. Automata for modeling real-time systems *(ICALP'90)*.

Timed automata [AD90]

- A finite control structure + variables (clocks)
- A transition is of the form:



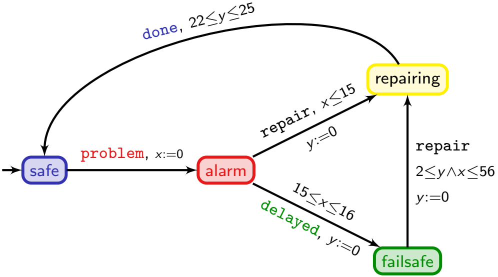
- An enabling condition (or guard) is:

$$g \quad ::= \quad x \sim c \quad | \quad x - y \sim c \quad | \quad g \wedge g$$

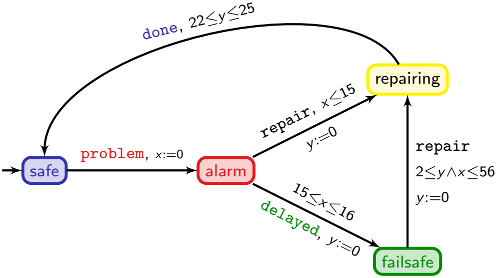
where $\sim \in \{<, \leq, =, \geq, >\}$

[AD90] Alur, Dill. Automata for modeling real-time systems *(ICALP'90)*.

An example of a timed automaton

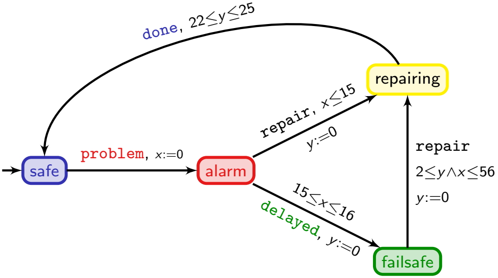


An example of a timed automaton



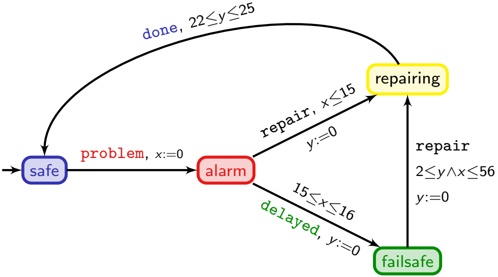
	safe
x	0
y	0

An example of a timed automaton



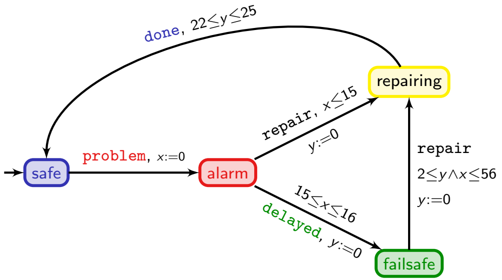
	safe	$\xrightarrow{23}$	safe
x	0		23
y	0		23

An example of a timed automaton



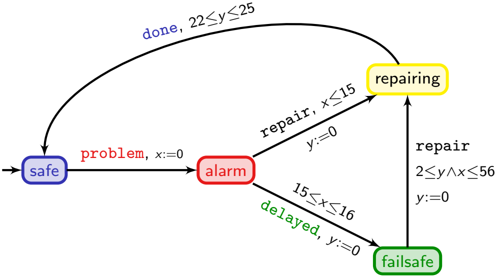
	safe	$\xrightarrow{23}$	safe	$\xrightarrow{\texttt{problem}}$	alarm
x	0		23		0
y	0		23		23

An example of a timed automaton



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

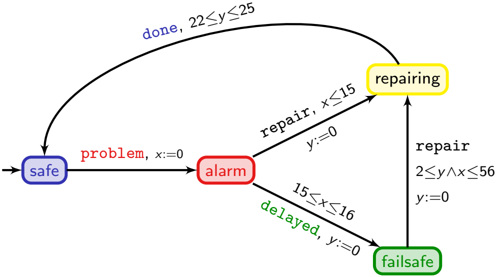
An example of a timed automaton



	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
···	15.6
	0

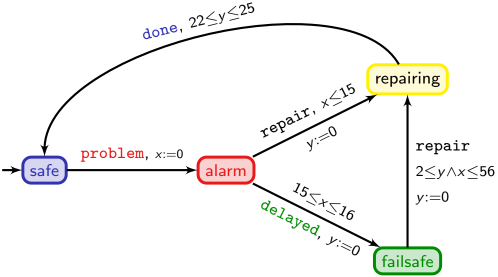
An example of a timed automaton



	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
⋯	15.6		17.9
	0		2.3

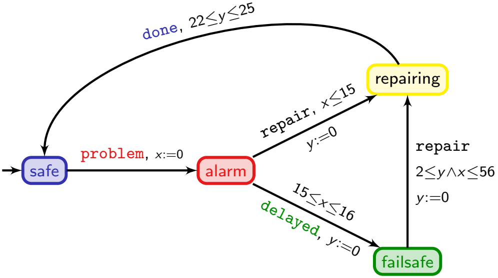
An example of a timed automaton



	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
⋯	15.6		17.9		17.9
	0		2.3		0

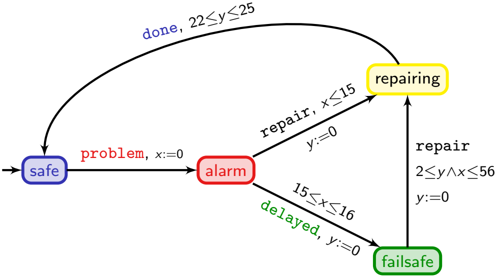
An example of a timed automaton



	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
⋯	15.6		17.9		17.9		40
	0		2.3		0		22.1

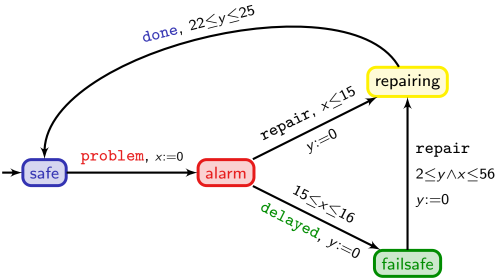
An example of a timed automaton



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

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

An example of 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

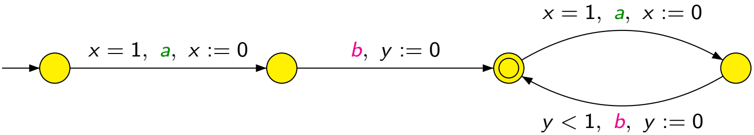
This run read the timed word
$(\texttt{problem}, 23)(\texttt{delayed}, 38.6)(\texttt{repair}, 40.9), (\texttt{done}, 63)$.

Timed automata semantics

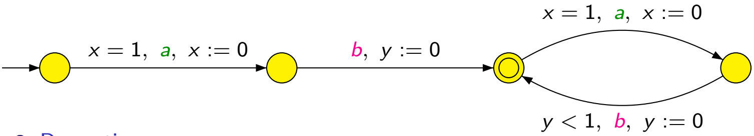
- $\mathcal{A} = (\Sigma, L, X, \longrightarrow)$ is a TA
- **Configurations:** $(\ell, v) \in L \times T^X$ where T is the time domain

  v is called the (clock) valuation
- **Timed transition system:**
  - **action transition**: $(\ell, v) \xrightarrow{a} (\ell', v')$ if $\exists \ell \xrightarrow{g,a,r} \ell' \in \mathcal{A}$ s.t. $\begin{cases} v \models g \\ v' = v[r \leftarrow 0] \end{cases}$
  - **delay transition**: $(\ell, v) \xrightarrow{\delta(d)} (\ell, v + d)$ if $d \in T$

Discrete *vs* dense-time semantics

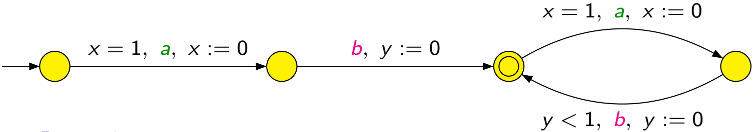


Discrete *vs* dense-time semantics



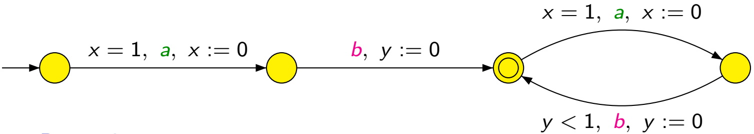
- Dense-time:
  $L_{dense} = \{((ab)^\omega, \tau) \mid \forall i,\ \tau_{2i-1} = i \text{ and } \tau_{2i} - \tau_{2i-1} > \tau_{2i+2} - \tau_{2i+1}\}$

Discrete *vs* dense-time semantics



- Dense-time:
  $L_{dense} = \{((ab)^\omega, \tau) \mid \forall i,\ \tau_{2i-1} = i \text{ and } \tau_{2i} - \tau_{2i-1} > \tau_{2i+2} - \tau_{2i+1}\}$
- Discrete-time: $L_{discrete} = \emptyset$

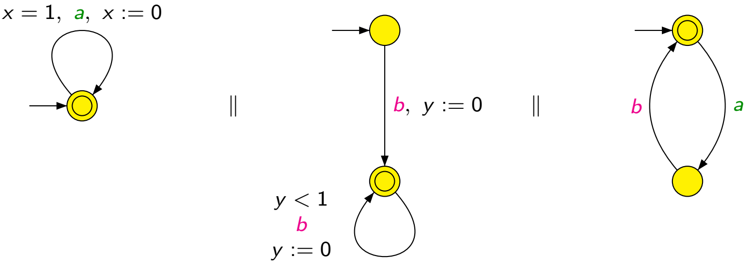
Discrete *vs* dense-time semantics



- Dense-time:
  $L_{dense} = \{((ab)^\omega, \tau) \mid \forall i,\ \tau_{2i-1} = i \text{ and } \tau_{2i} - \tau_{2i-1} > \tau_{2i+2} - \tau_{2i+1}\}$

- Discrete-time: $L_{discrete} = \emptyset$

However, it does result from the following parallel composition:

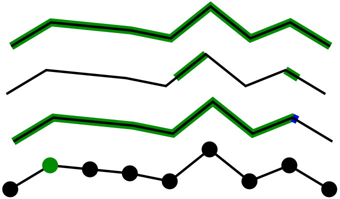


Classical verification problems

- reachability of a control state
- $\mathcal{S} \sim \mathcal{S}'$: bisimulation, etc...
- $L(\mathcal{S}) \subseteq L(\mathcal{S}')$: language inclusion
- $\mathcal{S} \models \varphi$ for some formula φ: model-checking
- $\mathcal{S} \parallel A_T$ + reachability: testing automata
- ...

Classical temporal logics

Path formulas:



$\mathbf{G}\,\varphi$ — "Always"

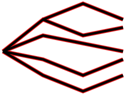
$\mathbf{F}\,\varphi$ — "Eventually"

$\varphi\,\mathbf{U}\,\varphi'$ — "Until"

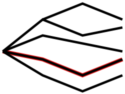
$\mathbf{X}\,\varphi$ — "Next"

State formulas:

$\mathbf{A}\psi$



$\mathbf{E}\psi$



$\rightsquigarrow$ LTL: Linear Temporal Logic [Pnu77],
CTL: Computation Tree Logic [EC82]

[Pnu77] Pnueli. The temporal logic of programs *(FoCS'77)*.
[EC82] Emerson, Clarke. Using branching time temporal logic to synthesize synchronization skeletons *(Science of Computer Programming 1982)*.

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

[ACD90] Alur, Courcoubetis, Dill. Model-checking for real-time systems *(LICS'90)*.
[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time *(Information and Computation)*.
[HNSY94] Henzinger, Nicollin, Sifakis, Yovine. Symbolic model-checking for real-time systems *(ACM Transactions on Computational Logic)*.

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$\mathbf{AG}\,(\texttt{problem} \Rightarrow \mathbf{AF}\ \texttt{alarm})$$

[ACD90] Alur, Courcoubetis, Dill. Model-checking for real-time systems *(LICS'90)*.
[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time *(Information and Computation)*.
[HNSY94] Henzinger, Nicollin, Sifakis, Yovine. Symbolic model-checking for real-time systems *(ACM Transactions on Computational Logic)*.

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$\mathbf{AG}\,(\texttt{problem} \Rightarrow \mathbf{AF}\ \texttt{alarm})$$

How can we express:

"any problem is followed by an alarm **within 20 time units**"?

[ACD90] Alur, Courcoubetis, Dill. Model-checking for real-time systems *(LICS'90)*.
[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time *(Information and Computation)*.
[HNSY94] Henzinger, Nicollin, Sifakis, Yovine. Symbolic model-checking for real-time systems *(ACM Transactions on Computational Logic)*.

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$\mathbf{AG}\,(\texttt{problem} \Rightarrow \mathbf{AF}\ \texttt{alarm})$$

How can we express:

"any problem is followed by an alarm **within 20 time units**"?

- Temporal logics with **subscripts**. ex: CTL $+$ $\left|\begin{array}{l} \mathbf{E}\varphi\,\mathbf{U}_{\sim k}\,\psi \\ \mathbf{A}\varphi\,\mathbf{U}_{\sim k}\,\psi \end{array}\right.$

[ACD90] Alur, Courcoubetis, Dill. Model-checking for real-time systems *(LICS'90)*.
[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time *(Information and Computation)*.
[HNSY94] Henzinger, Nicollin, Sifakis, Yovine. Symbolic model-checking for real-time systems *(ACM Transactions on Computational Logic)*.

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$\mathbf{AG}\,(\texttt{problem} \Rightarrow \mathbf{AF}\ \texttt{alarm})$$

How can we express:

"any problem is followed by an alarm **within 20 time units**"?

- Temporal logics with **subscripts**.

$$\mathbf{AG}\,(\texttt{problem} \Rightarrow \mathbf{AF}_{\leq 20}\ \texttt{alarm})$$

[ACD90] Alur, Courcoubetis, Dill. Model-checking for real-time systems *(LICS'90)*.
[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time *(Information and Computation)*.
[HNSY94] Henzinger, Nicollin, Sifakis, Yovine. Symbolic model-checking for real-time systems *(ACM Transactions on Computational Logic)*.

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$\mathbf{AG}\,(\texttt{problem} \Rightarrow \mathbf{AF}\ \texttt{alarm})$$

How can we express:

"any problem is followed by an alarm **within 20 time units**"?

- Temporal logics with **subscripts**.

$$\mathbf{AG}\,(\texttt{problem} \Rightarrow \mathbf{AF}_{\leq 20}\ \texttt{alarm})$$

- Temporal logics with **clocks**.

$$\mathbf{AG}\,(\texttt{problem} \Rightarrow (x\ \texttt{in}\ \mathbf{AF}\,(x \leq 20 \wedge \texttt{alarm})))$$

[ACD90] Alur, Courcoubetis, Dill. Model-checking for real-time systems *(LICS'90)*.
[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time *(Information and Computation)*.
[HNSY94] Henzinger, Nicollin, Sifakis, Yovine. Symbolic model-checking for real-time systems *(ACM Transactions on Computational Logic)*.

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$\mathbf{AG}\,(\texttt{problem} \Rightarrow \mathbf{AF}\ \texttt{alarm})$$

How can we express:

"any problem is followed by an alarm **within 20 time units**"?

- Temporal logics with **subscripts**.

$$\mathbf{AG}\,(\texttt{problem} \Rightarrow \mathbf{AF}_{\leq 20}\ \texttt{alarm})$$

- Temporal logics with **clocks**.

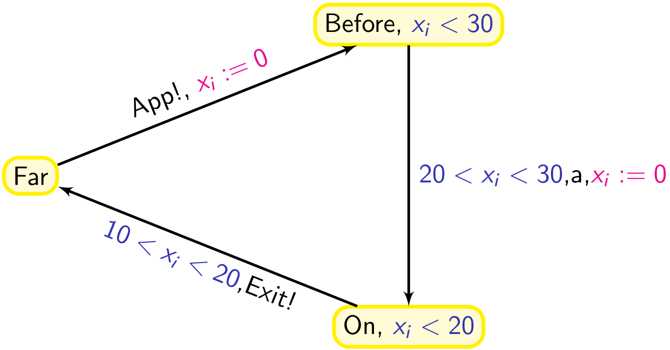
$$\mathbf{AG}\,(\texttt{problem} \Rightarrow (x\ \texttt{in}\ \mathbf{AF}\,(x \leq 20 \wedge \texttt{alarm})))$$

$\rightsquigarrow$ TCTL: Timed CTL **[ACD90,ACD93,HNSY94]**

[ACD90] Alur, Courcoubetis, Dill. Model-checking for real-time systems *(LICS'90)*.
[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time *(Information and Computation)*.
[HNSY94] Henzinger, Nicollin, Sifakis, Yovine. Symbolic model-checking for real-time systems *(ACM Transactions on Computational Logic)*.

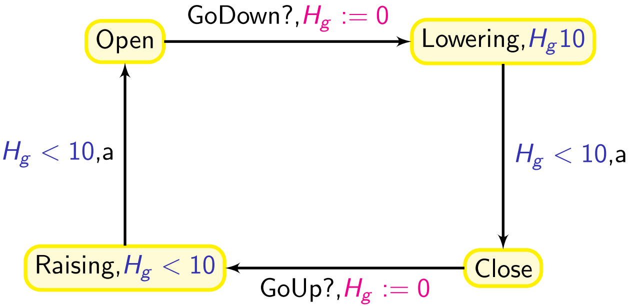
The train crossing example (1)

Train$_i$ with $i = 1, 2, \ldots$



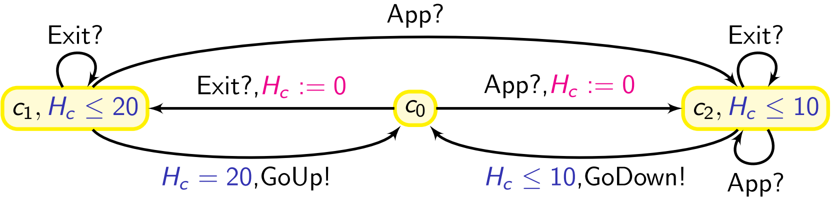
The train crossing example (2)

The gate:



The train crossing example (3)

The controller:



The train crossing example (4)

We use the synchronization function f:

Train_1	Train_2	Gate	Controller	
App!	.	.	App?	App
.	App!	.	App?	App
Exit!	.	.	Exit?	Exit
.	Exit!	.	Exit?	Exit
a	.	.	.	a
.	a	.	.	a
.	.	a	.	a
.	.	GoUp?	GoUp!	GoUp
.	.	GoDown?	GoDown!	GoDown

to define the parallel composition $(\text{Train}_1 \parallel \text{Train}_2 \parallel \text{Gate} \parallel \text{Controller})$

NB: the parallel composition does not add expressive power!

The train crossing example (5)

Some properties one could check:

- Is the gate closed when a train crosses the road?

The train crossing example (5)

Some properties one could check:

- Is the gate closed when a train crosses the road?

$$\mathbf{AG}\,(\text{train.On} \Rightarrow \text{gate.Close})$$

The train crossing example (5)

Some properties one could check:

- Is the gate closed when a train crosses the road?

$$\mathbf{AG}\,(\text{train.On} \Rightarrow \text{gate.Close})$$

- Is the gate always closed for less than 5 minutes?

The train crossing example (5)

Some properties one could check:

- Is the gate closed when a train crosses the road?

$$\mathbf{AG}\,(\text{train.On} \Rightarrow \text{gate.Close})$$

- Is the gate always closed for less than 5 minutes?

$$\neg\mathbf{EF}\,(\text{gate.Close} \wedge \mathbf{E}(\text{gate.Close}\ \mathbf{U}_{>5\text{ min}}\ \neg\text{gate.Close}))$$

Another example: A Fischer protocol

A mutual exclusion protocol with a shared variable *id* [AL94].

[AL94] Abadí, Lamport. An old-fashioned recipe for real time *(ACM Transactions on Programming Languages and Systems)*.

Another example: A Fischer protocol

A mutual exclusion protocol with a shared variable *id* [AL94].

Process i:

a : await ($id = 0$);
b : set id to i;
c : await ($id = i$);
d : enter critical section.

$\rightsquigarrow$ a max. delay k_1 between a and b
a min. delay k_2 between b and c

[AL94] Abadí, Lamport. An old-fashioned recipe for real time *(ACM Transactions on Programming Languages and Systems)*.

Another example: A Fischer protocol

A mutual exclusion protocol with a shared variable *id* [AL94].

Process i:

a : await ($id = 0$);
b : set id to i;
c : await ($id = i$);
d : enter critical section.

$\leadsto$ a max. delay k_1 between a and b
a min. delay k_2 between b and c

$\leadsto$ See the demo with the tool Uppaal
(can be downloaded freely on http://www.uppaal.com/)

[AL94] Abadí, Lamport. An old-fashioned recipe for real time *(ACM Transactions on Programming Languages and Systems)*.

Outline

Verification

Emptiness problem: is the language accepted by a timed automaton empty?

- basic reachability/safety properties (final states)
- basic liveness properties (ω-regular conditions)

[AD90] Alur, Dill. Automata for modeling real-time systems *(ICALP'90)*.
[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

Verification

Emptiness problem: is the language accepted by a timed automaton empty?

- Problem: the set of configurations is infinite
  $\leadsto$ classical methods for finite-state systems cannot be applied

[AD90] Alur, Dill. Automata for modeling real-time systems *(ICALP'90)*.
[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

Verification

Emptiness problem: is the language accepted by a timed automaton empty?

- Problem: the set of configurations is infinite
  $\rightsquigarrow$ classical methods for finite-state systems cannot be applied

- Positive key point: variables (clocks) increase at the same speed

[AD90] Alur, Dill. Automata for modeling real-time systems *(ICALP'90)*.
[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

Verification

Emptiness problem: is the language accepted by a timed automaton empty?

- Problem: the set of configurations is infinite
  $\rightsquigarrow$ classical methods for finite-state systems cannot be applied

- Positive key point: variables (clocks) increase at the same speed

Theorem [AD90,AD94]

The emptiness problem for timed automata is decidable and PSPACE-complete.

[AD90] Alur, Dill. Automata for modeling real-time systems *(ICALP'90)*.
[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

Verification

Emptiness problem: is the language accepted by a timed automaton empty?

- Problem: the set of configurations is infinite
  $\rightsquigarrow$ classical methods for finite-state systems cannot be applied
- Positive key point: variables (clocks) increase at the same speed

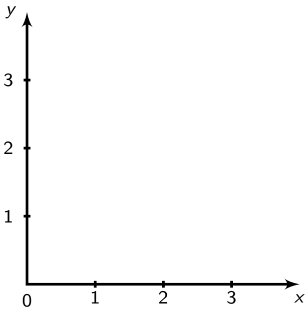
Theorem [AD90,AD94]

The emptiness problem for timed automata is decidable and PSPACE-complete.

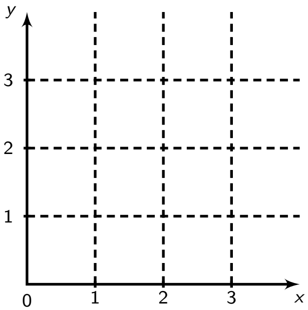
Method: construct a finite abstraction

[AD90] Alur, Dill. Automata for modeling real-time systems *(ICALP'90)*.
[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

The region abstraction

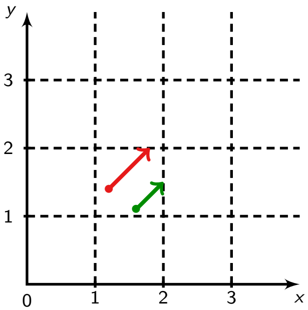


The region abstraction



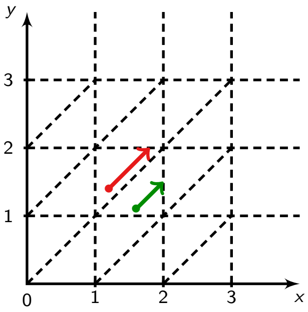
- "compatibility" between regions and constraints

The region abstraction



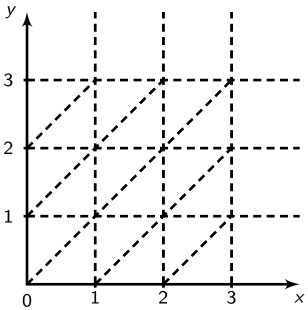
- “compatibility” between regions and constraints
- “compatibility” between regions and time elapsing

The region abstraction



- "compatibility" between regions and constraints
- "compatibility" between regions and time elapsing

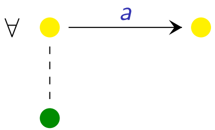
The region abstraction



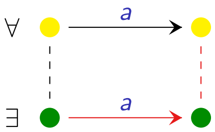
- "compatibility" between regions and constraints
- "compatibility" between regions and time elapsing

$\rightsquigarrow$ an equivalence of finite index
a time-abstract bisimulation

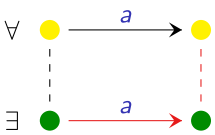
Time-abstract bisimulation

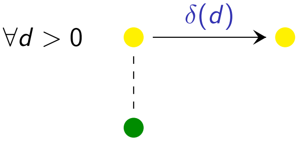


Time-abstract bisimulation

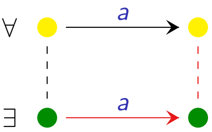


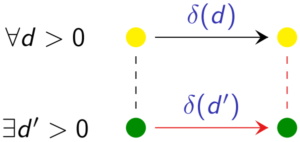
Time-abstract bisimulation



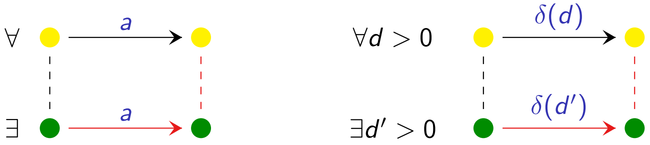


Time-abstract bisimulation



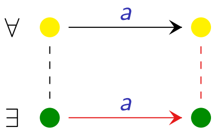


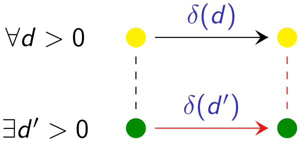
Time-abstract bisimulation



$$(\ell_0, v_0) \xrightarrow{a_1, t_1} (\ell_1, v_1) \xrightarrow{a_2, t_2} (\ell_2, v_2) \xrightarrow{a_3, t_3} \ldots$$

Time-abstract bisimulation





$$
\begin{tikzcd}
(\ell_0, v_0) \arrow[r, "a_1, t_1"] \arrow[d] & (\ell_1, v_1) \arrow[r, "a_2, t_2"] \arrow[d] & (\ell_2, v_2) \arrow[r, "a_3, t_3"] \arrow[d] & \ldots \\
(\ell_0, R_0) \arrow[r, "a_1"] & (\ell_1, R_1) \arrow[r, "a_2"] & (\ell_2, R_2) \arrow[r, "a_3"] & \ldots
\end{tikzcd}
$$

with $v_i \in R_i$ for all i.

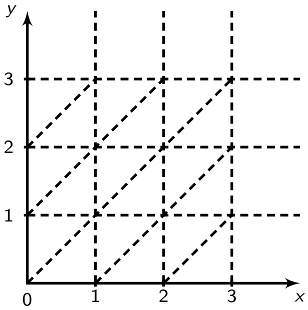
Time-abstract bisimulation

$$\begin{tikzcd} \forall & \bullet \arrow[r, "a"] \arrow[d, dash, dashed] & \bullet \arrow[d, dash, dashed] \\ \exists & \bullet \arrow[r, "a"] & \bullet \end{tikzcd}
\qquad
\begin{tikzcd} \forall d > 0 & \bullet \arrow[r, "\delta(d)"] \arrow[d, dash, dashed] & \bullet \arrow[d, dash, dashed] \\ \exists d' > 0 & \bullet \arrow[r, "\delta(d')"] & \bullet \end{tikzcd}$$

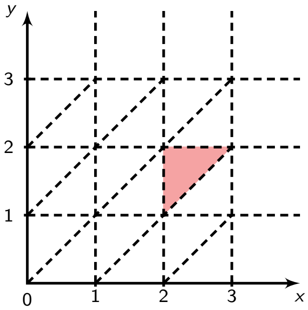
$$\begin{tikzcd} (\ell_0, v_0) \arrow[r, "a_1, t_1"] \arrow[d, leftrightarrow] & (\ell_1, v_1) \arrow[r, "a_2, t_2"] \arrow[d, leftrightarrow] & (\ell_2, v_2) \arrow[r, "a_3, t_3"] \arrow[d, leftrightarrow] & \dots \\ (\ell_0, R_0) \arrow[r, "a_1"] & (\ell_1, R_1) \arrow[r, "a_2"] & (\ell_2, R_2) \arrow[r, "a_3"] & \dots \end{tikzcd}$$

with $v_i \in R_i$ for all i.

The region abstraction

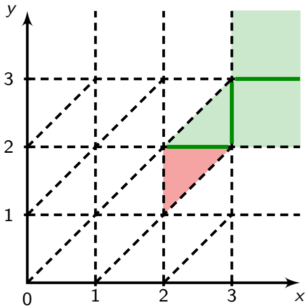


The region abstraction



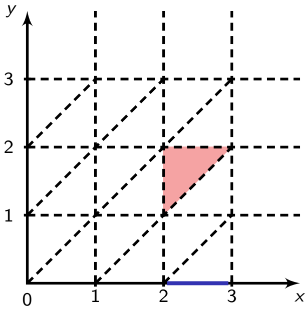
$$\begin{cases} 2<x<3 \\ 1<y<2 \\ \{x\}<\{y\} \end{cases}$$

The region abstraction



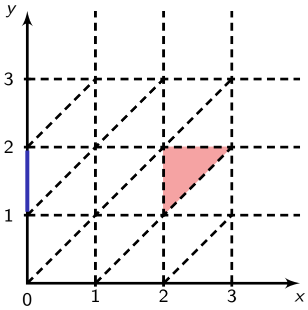
time successors

The region abstraction



reset of clock y

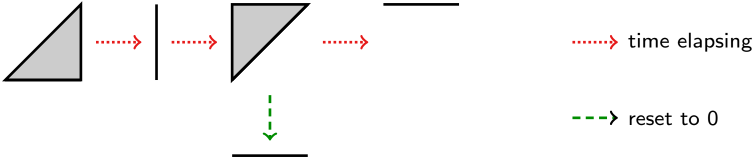
The region abstraction



reset of clock x

The region graph

A finite graph representing time elapsing and reset of clocks:



Region automaton $\equiv$ finite bisimulation quotient

timed automaton $\bigotimes$ region graph

Region automaton $\equiv$ finite bisimulation quotient

timed automaton $\bigotimes$ region graph

$\ell \xrightarrow{g,a,C:=0} \ell'$ is transformed into:

$(\ell, R) \xrightarrow{a} (\ell', R')$ if there exists $R'' \in \mathsf{Succ}_t^*(R)$ s.t.

- $R'' \subseteq g$
- $[C \leftarrow 0]R'' \subseteq R'$

Region automaton $\equiv$ finite bisimulation quotient

timed automaton $\bigotimes$ region graph

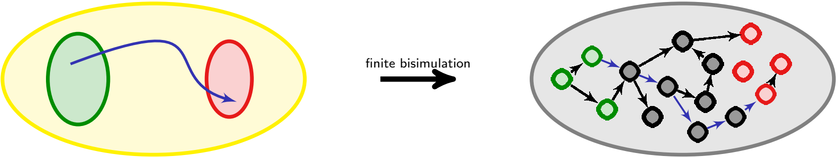
$\ell \xrightarrow{g,a,C:=0} \ell'$ is transformed into:

$(\ell, R) \xrightarrow{a} (\ell', R')$ if there exists $R'' \in \mathrm{Succ}_t^*(R)$ s.t.

- $R'' \subseteq g$
- $[C \leftarrow 0]R'' \subseteq R'$

$\mathcal{L}$**(reg. aut.) = UNTIME(**$\mathcal{L}$**(timed aut.))**

where $\mathrm{UNTIME}((a_1, t_1)(a_2, t_2)\dots) = a_1 a_2 \dots$

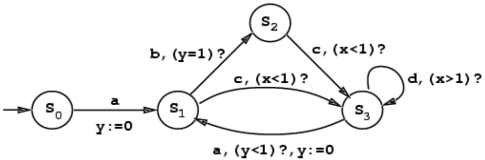


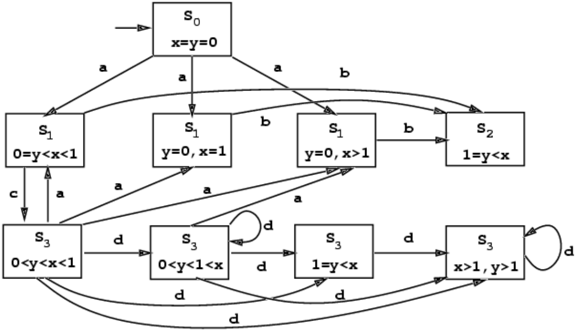
timed automaton

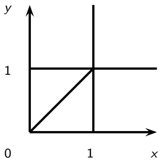
large (but finite) automaton
(region automaton)

$$\mathcal{L}(\textbf{reg. aut.}) = \textbf{UNTIME}(\mathcal{L}(\textbf{timed aut.}))$$

An example [AD94]







PSPACE membership

The size of the region graph is in $\mathcal{O}(|X|!.2^{|X|})$

- **One configuration:** a discrete location + a region

PSPACE membership

The size of the region graph is in $\mathcal{O}(|X|!.2^{|X|})$

- **One configuration:** a discrete location + a region
  - a discrete location: log-space

PSPACE membership

The size of the region graph is in $\mathcal{O}(|X|!.2^{|X|})$

- **One configuration:** a discrete location + a region
  - a discrete location: log-space
  - a region:
    - an interval for each clock
    - an interval for each pair of clocks

PSPACE membership

The size of the region graph is in $\mathcal{O}(|X|!.2^{|X|})$

- **One configuration:** a discrete location + a region
  - a discrete location: log-space
  - a region:
    - an interval for each clock
    - an interval for each pair of clocks

$\rightsquigarrow$ requires polynomial space

PSPACE membership

The size of the region graph is in $\mathcal{O}(|X|!.2^{|X|})$

- **One configuration:** a discrete location + a region
  - a discrete location: log-space
  - a region:
    - an interval for each clock
    - an interval for each pair of clocks

$\leadsto$ requires polynomial space

- By guessing a path of length at most exponential: needs only to store two consecutive configurations

PSPACE membership

The size of the region graph is in $\mathcal{O}(|X|!.2^{|X|})$

- **One configuration:** a discrete location + a region
  - a discrete location: log-space
  - a region:
    - an interval for each clock
    - an interval for each pair of clocks

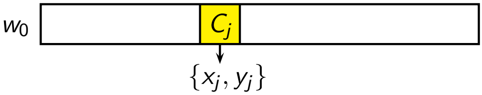
$\rightsquigarrow$ requires polynomial space

- By guessing a path of length at most exponential: needs only to store two consecutive configurations

$\rightsquigarrow$ in NPSPACE, thus in PSPACE

PSPACE-hardness

$$\left.\begin{array}{c}\mathcal{M}\ \text{LBTM}\\ w_0 \in \{a,b\}^*\end{array}\right\} \rightsquigarrow \begin{array}{l}A_{\mathcal{M},w_0}\ \text{s.t. } \mathcal{M} \text{ accepts } w_0 \text{ iff the final state}\\ \text{of } A_{\mathcal{M},w_0} \text{ is reachable}\end{array}$$



C_j contains an "a" if $x_j = y_j$
C_j contains a "b" if $x_j < y_j$

(these conditions are invariant by time elapsing)

LBTM: linearly bounded Turing machine (a witness for PSPACE-complete problems)

PSPACE-hardness (cont.)

If $q \xrightarrow{\alpha,\alpha',\delta} q'$ is a transition of $\mathcal{M}$, then for each position i of the tape, we have a transition

$$(q,i) \xrightarrow{g,r:=0} (q',i')$$

where:

- g is $x_i = y_i$ (resp. $x_i < y_i$) if $\alpha = a$ (resp. $\alpha = b$)
- $r = \{x_i, y_i\}$ (resp. $r = \{x_i\}$) if $\alpha' = a$ (resp. $\alpha' = b$)
- $i' = i + 1$ (resp. $i' = i - 1$) if δ is right and $i < n$ (resp. left)

Enforcing time elapsing: on each transition, add the condition $t = 1$ and clock t is reset.

Initialization: init $\xrightarrow{t=1, r_0:=0}$ $(q_0, 1)$ where $r_0 = \{x_i \mid w_0[i] = b\} \cup \{t\}$

Termination: $(q_f, i) \longrightarrow$ end

The case of single-clock timed automata

Exercise [LMS04]

Think of the special case of single-clock timed automata. Can we do better than PSPACE?

[LMS04] Laroussinie, Markey, Schnoebelen. Model checking timed automata with one or two clocks *(CONCUR'04)*.

Consequence of region automata construction

Region automata:

correct finite (and exponential) abstraction for checking reachability/Büchi-like properties.

Consequence of region automata construction

Region automata:

correct finite (and exponential) abstraction for checking reachability/Büchi-like properties.

However...

everything can not be reduced to finite automata...

A model not far from undecidability

Some bad news...

- Language universality is undecidable [AD90]
- Language inclusion is undecidable [AD90]
- Complementability is undecidable [Tri03,Fin06]
- ...

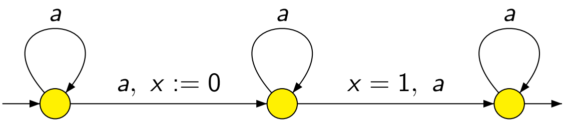
[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata *(FORMATS'03)*.
[Fin06] Finkel. Undecidable problems about timed automata *(FORMATS'06)*.
[AM04] Alur, Madhusudan. Decision problems for timed automata: A survey *(SFM-04:RT))*.

A model not far from undecidability

Some bad news...

- Language universality is undecidable [AD90]
- Language inclusion is undecidable [AD90]
- Complementability is undecidable [Tri03,Fin06]
- ...

An example of non-determinizable/non-complementable timed aut.:



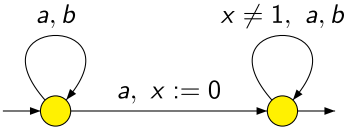
[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata *(FORMATS'03)*.
[Fin06] Finkel. Undecidable problems about timed automata *(FORMATS'06)*.
[AM04] Alur, Madhusudan. Decision problems for timed automata: A survey *(SFM-04:RT))*.

A model not far from undecidability

Some bad news...

- Language universality is undecidable [AD90]
- Language inclusion is undecidable [AD90]
- Complementability is undecidable [Tri03,Fin06]
- ...

An example of non-determinizable/non-complementable timed aut.: [AM04]



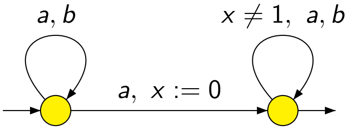
[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata *(FORMATS'03)*.
[Fin06] Finkel. Undecidable problems about timed automata *(FORMATS'06)*.
[AM04] Alur, Madhusudan. Decision problems for timed automata: A survey *(SFM-04:RT))*.

A model not far from undecidability

Some bad news...

- Language universality is undecidable [AD90]
- Language inclusion is undecidable [AD90]
- Complementability is undecidable [Tri03,Fin06]
- ...

An example of non-determinizable/non-complementable timed aut.: [AM04]



UNTIME $\left(\overline{L} \cap \{(a^* b^*, \tau) \mid \text{all } a's \text{ happen before 1 and no two } a's \text{ simultaneously}\}\right)$ is not regular **(exercise!)**

[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata *(FORMATS'03)*.
[Fin06] Finkel. Undecidable problems about timed automata *(FORMATS'06)*.
[AM04] Alur, Madhusudan. Decision problems for timed automata: A survey *(SFM-04:RT))*.

The two-counter machine

Definition

A two-counter machine is a finite set of instructions over two counters (c and d):

- Incrementation:
  $\texttt{(p):}\quad c := c+1;\ \texttt{goto (q)}$
- Decrementation:
  $\texttt{(p):}\quad \texttt{if } c > 0 \texttt{ then } c := c-1;\ \texttt{goto (q) else goto (r)}$

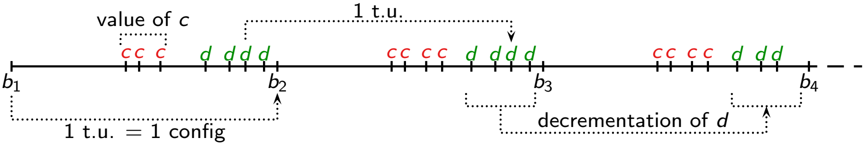
Theorem **[Minsky 67]**

The halting problem for two counter machines is undecidable.

Undecidability of universality

Theorem [AD90]

Universality of timed automata is undecidable.

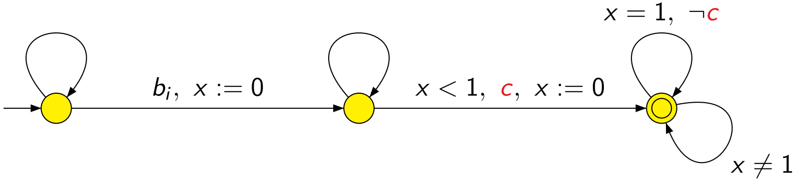


- one configuration is encoded in one time unit
- number of c's: value of counter c
- number of d's: value of counter d
- one time unit between two corresponding c's (resp. d's)

$\rightsquigarrow$ **We encode "non-behaviours" of a two-counter machine**

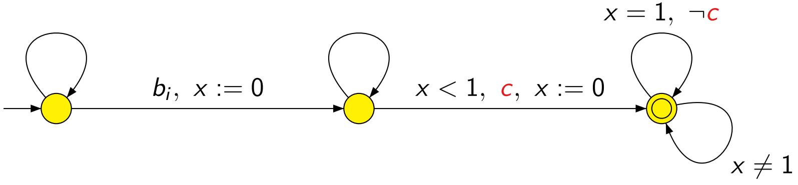
Example

Module to check that if instruction i does not decrease counter c, then all actions c appearing less than 1 t.u. after b_i has to be followed by an other c 1 t.u. later.



Example

Module to check that if instruction i does not decrease counter c, then all actions c appearing less than 1 t.u. after b_i has to be followed by an other c 1 t.u. later.



The union of all small modules is not universal
iff
The two-counter machine has a recurring computation

Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
    [Bérard,Diekert,Gastin,Petit 1998] [Choffrut,Goldwurm 2000]
    [Bouyer,Dufourd,Fleury,Petit 2004] $\cdots$

Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
    [Bérard,Diekert,Gastin,Petit 1998] [Choffrut,Goldwurm 2000]
    [Bouyer,Dufourd,Fleury,Petit 2004] $\cdots$
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
    [Alur,Courcoubetis,Dill 1993] [Laroussinie,Larsen,Weise 1995]

Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
    [Bérard,Diekert,Gastin,Petit 1998] [Choffrut,Goldwurm 2000]
    [Bouyer,Dufourd,Fleury,Petit 2004] $\cdots$
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
    [Alur,Courcoubetis,Dill 1993] [Laroussinie,Larsen,Weise 1995]
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
    [Bouyer,Larsen,Markey,Rasmussen 2006] [Bouyer,Larsen,Markey 2007]

Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
    [Bérard,Diekert,Gastin,Petit 1998] [Choffrut,Goldwurm 2000]
    [Bouyer,Dufourd,Fleury,Petit 2004] $\cdots$
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
    [Alur,Courcoubetis,Dill 1993] [Laroussinie,Larsen,Weise 1995]
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
    [Bouyer,Larsen,Markey,Rasmussen 2006] [Bouyer,Larsen,Markey 2007]
  - o-minimal hybrid systems
    [Lafferriere,Pappas,Sastry 2000] [Brihaye 2005]

Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
    [Bérard,Diekert,Gastin,Petit 1998] [Choffrut,Goldwurm 2000] [Bouyer,Dufourd,Fleury,Petit 2004] $\cdots$
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
    [Alur,Courcoubetis,Dill 1993] [Laroussinie,Larsen,Weise 1995]
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
    [Bouyer,Larsen,Markey,Rasmussen 2006] [Bouyer,Larsen,Markey 2007]
  - o-minimal hybrid systems
    [Lafferriere,Pappas,Sastry 2000] [Brihaye 2005]
  - $\cdots$

Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
    [Bérard,Diekert,Gastin,Petit 1998] [Choffrut,Goldwurm 2000] [Bouyer,Dufourd,Fleury,Petit 2004] $\cdots$
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
    [Alur,Courcoubetis,Dill 1993] [Laroussinie,Larsen,Weise 1995]
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
    [Bouyer,Larsen,Markey,Rasmussen 2006] [Bouyer,Larsen,Markey 2007]
  - o-minimal hybrid systems
    [Lafferriere,Pappas,Sastry 2000] [Brihaye 2005]
  - $\cdots$
- Note however that it might be hard to prove there is a finite bisimulation quotient!

Outline

Outline

A general model: hybrid systems

[Hen96]

What is a hybrid system?

a discrete control (the mode of the system)
$+$ a continuous evolution within a mode (given by variables)

[Hen96] Henzinger. The theory of hybrid automata *(LICS'96)*.

A general model: hybrid systems

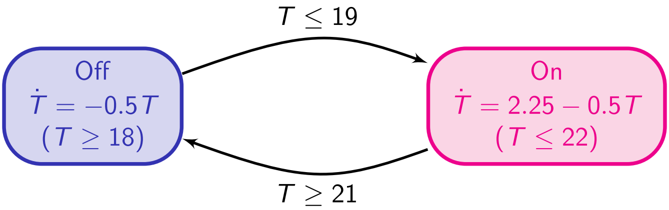
[Hen96]

What is a hybrid system?

a discrete control (the mode of the system)
$+$ a continuous evolution within a mode (given by variables)

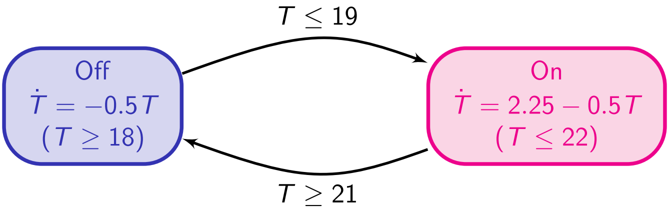
Example (The thermostat)

A simple thermostat, where T (the temperature) depends on the time:

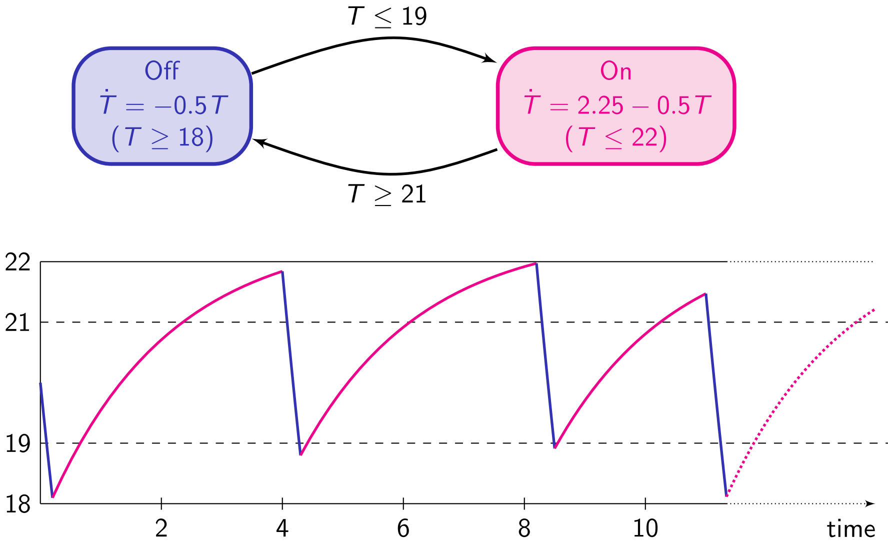


[Hen96] Henzinger. The theory of hybrid automata *(LICS'96)*.

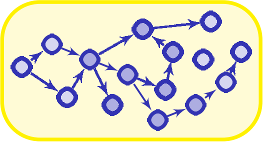
The thermostat example



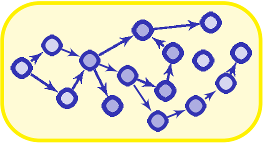
The thermostat example



Ok...

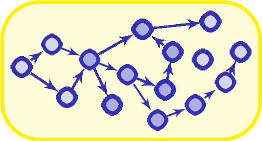


Ok...

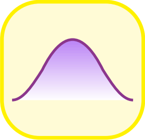


Easy...

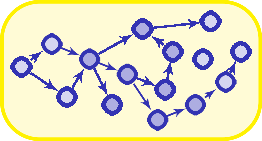
Ok...



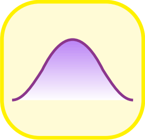
Easy...



Ok...

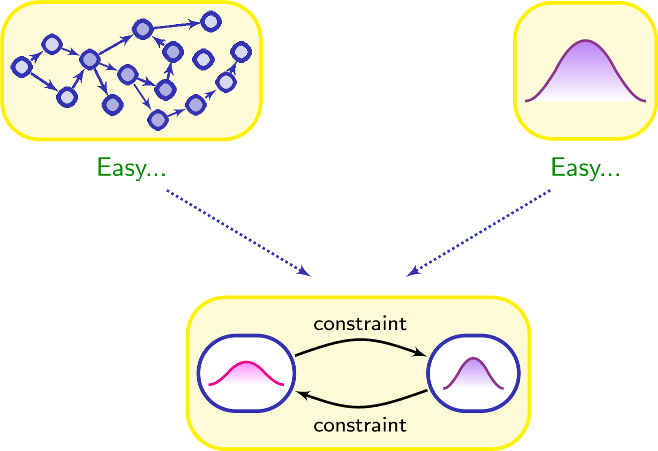


Easy...

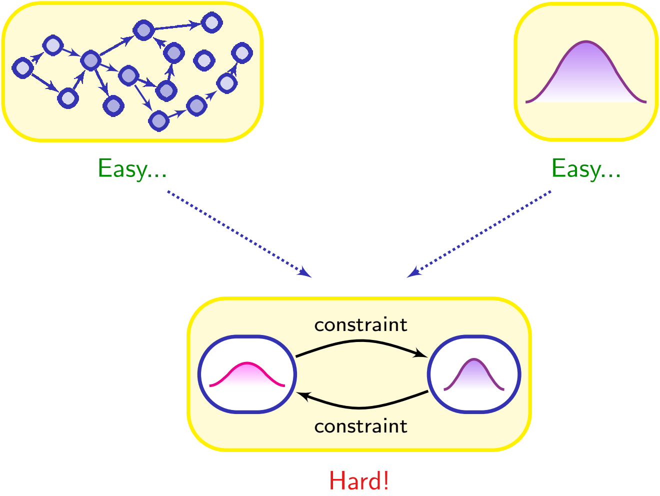


Easy...

Ok... but?



Ok... but?



What about decidability?

$\rightsquigarrow$ almost everything is undecidable

Negative results [HKPV95]

- The class of hybrid systems with clocks and only one variable having possibly two slopes $k_1 \neq k_2$ is undecidable.
- The class of *stopwatch* automata is undecidable.

[HKPV95] Henzinger, Kopke, Puri, Varaiya. What's decidable about hybrid automata? *(STOC'95)*.

Outline

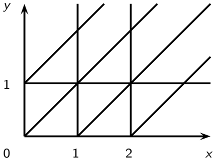
Role of diagonal constraints

$$\boxed{x - y \sim c \quad \text{and} \quad x \sim c}$$

Role of diagonal constraints

$$\boxed{x - y \sim c \quad \text{and} \quad x \sim c}$$

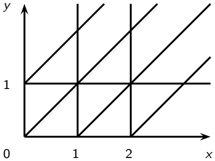
- **Decidability:** yes, using the region abstraction



Role of diagonal constraints

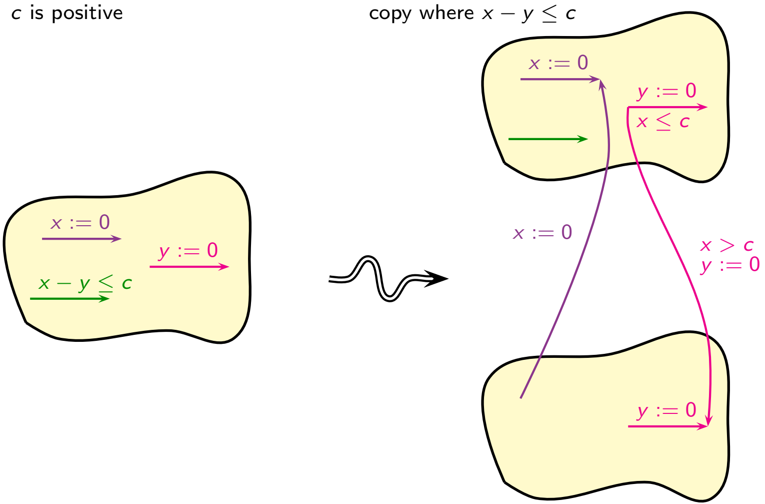
$$\boxed{x - y \sim c \quad \text{and} \quad x \sim c}$$

- **Decidability:** yes, using the region abstraction



- **Expressiveness:** no additional expressive power

Role of diagonal constraints (cont.)



copy where $x - y > c$

$\rightsquigarrow$ proof in [BDGP98]

[BDGP98] Bérard, Diekert, Gastin, Petit. Characterization of the expressive power of silent transitions in timed automata *(Fundamenta Informaticae)*.

Role of diagonal constraints (cont.)

Exercise [BC05]

Consider, for every positive integer n, the timed language:

$$\mathcal{L}_n = \{(a, t_1) \dots (a, t_{2^n}) \mid 0 < t_1 < \cdots < t_{2^n} < 1\}$$

1. Construct a timed automaton with diagonal constraints which recognizes $\mathcal{L}_n$. What is the size of this automaton?
2. Idem without diagonal constraints. Can you do better?
3. Conclude.

[BC05] Bouyer, Chevalier. On conciseness of extensions of timed automata *(Journal of Automata, Languages and Combinatorics)*.

Adding silent actions

$$\xrightarrow{g,\ \varepsilon,\ C := 0}$$

[BDGP98]

Adding silent actions

$$\xrightarrow{g, \varepsilon, C := 0}$$

[BDGP98]

- **Decidability:** yes

(actions have no influence on region automaton construction)

Adding silent actions

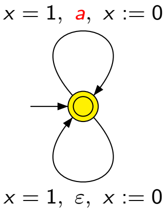
$$\xrightarrow{g,\, \varepsilon,\, C := 0}$$

[BDGP98]

- **Decidability:** yes

  (actions have no influence on region automaton construction)

- **Expressiveness:** strictly more expressive!



Adding additive constraints

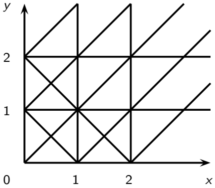
$\boxed{x + y \sim c \quad \text{and} \quad x \sim c}$ **[BD00]**

[BD00] Bérard, Dufourd. Timed automata and additive clock constraints *(Information Proecessing Letters)*.

Adding additive constraints

$\boxed{x + y \sim c \quad \text{and} \quad x \sim c}$ **[BD00]**

- **Decidability:** - for two clocks, decidable using the abstraction

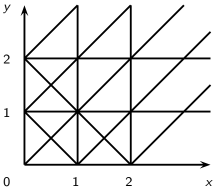


[BD00] Bérard, Dufourd. Timed automata and additive clock constraints *(Information Proecessing Letters)*.

Adding additive constraints

$\boxed{x + y \sim c \quad \text{and} \quad x \sim c}$ **[BD00]**

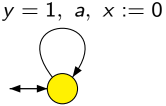
- **Decidability:** - for two clocks, decidable using the abstraction



  - for four clocks (or more), undecidable!

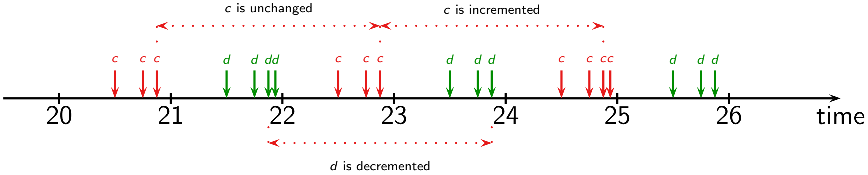
- **Expressiveness:** more expressive! (even using two clocks)

$$\{(a^n, t_1 \dots t_n) \mid n \geq 1 \text{ and } t_i = 1 - \frac{1}{2^i}\}$$



[BD00] Bérard, Dufourd. Timed automata and additive clock constraints *(Information Proecessing Letters)*.

Undecidability proof



$\rightsquigarrow$ simulation of
- decrementation of a counter
- incrementation of a counter

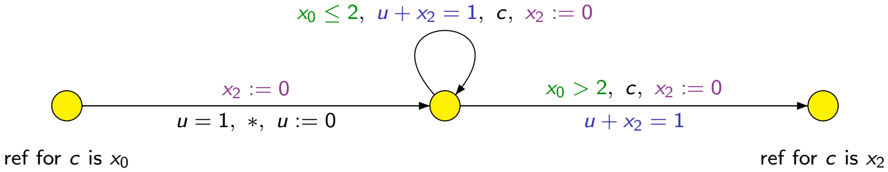
We will use 4 clocks:

- u, "tic" clock (each time unit)
- x_0, x_1, x_2: reference clocks for the two counters

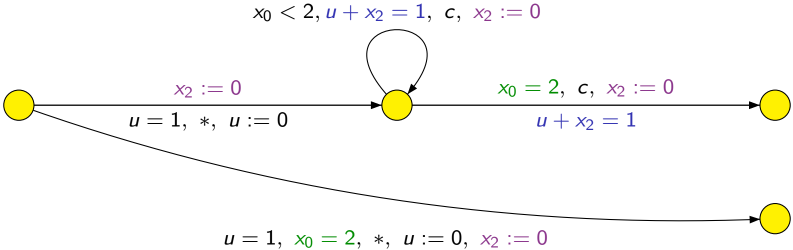
"x_i reference for c" $\equiv$ "the last time x_i has been reset is the last time action c has been performed"

Undecidability proof (cont.)

- **Incrementation of counter c:**

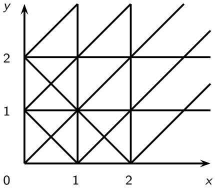


- **Decrementation of counter c:**



Adding constraints of the form $x + y \sim c$

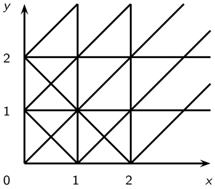
- Two clocks: decidable using the abstraction



- Four clocks (or more): undecidable!

Adding constraints of the form $x + y \sim c$

- Two clocks: decidable using the abstraction



- Three clocks: **open question**

- Four clocks (or more): undecidable!

Adding new operations on clocks

Several types of updates: $x := y + c$, $x :< c$, $x :> c$, etc...

Adding new operations on clocks

Several types of updates: $x := y + c$, $x :< c$, $x :> c$, etc...

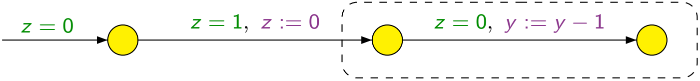
- The general model is undecidable.
  (simulation of a two-counter machine)

Adding new operations on clocks

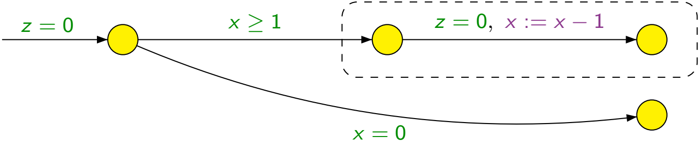
Several types of updates: $x := y + c$, $x :< c$, $x :> c$, etc...

- The general model is undecidable.
  (simulation of a two-counter machine)

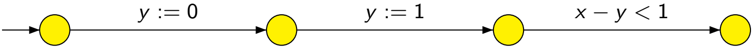
- Only decrementation also leads to undecidability
  - **Incrementation of counter x**



  - **Decrementation of counter x**



Decidability



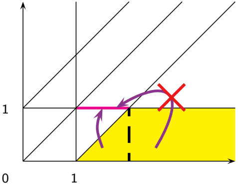


image by $y := 1$

$\rightsquigarrow$ the bisimulation property is not met

The classical region automaton construction is not correct.

Decidability (cont.)

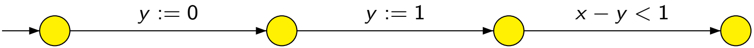
$$\begin{array}{lcl} \mathcal{A} & \rightsquigarrow & \text{Diophantine linear inequations system} \\ & \rightsquigarrow & \text{is there a solution?} \\ & \rightsquigarrow & \text{if yes, belongs to a decidable class} \end{array}$$

Examples:

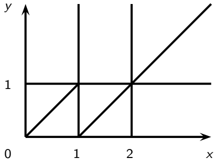
- constraint $x \sim c$ — $c \leq \max_x$
- constraint $x - y \sim c$ — $c \leq \max_{x,y}$
- update $x :\sim y + c$ — $\max_x \leq \max_y + c$
  and for each clock z, $\max_{x,z} \geq \max_{y,z} + c$, $\max_{z,x} \geq \max_{z,y} - c$
- update $x :< c$ — $c \leq \max_x$
  and for each clock z, $\max_z \geq c + \max_{z,x}$

The constants $(\max_x)$ and $(\max_{x,y})$ define a set of regions.

Decidability (cont.)



$$\left\{\begin{array}{l} \max_y \geq 0 \\ \max_x \geq 0 + \max_{x,y} \\ \max_y \geq 1 \\ \max_x \geq 1 + \max_{x,y} \\ \max_{x,y} \geq 1 \end{array}\right. \qquad \text{implies} \qquad \left\{\begin{array}{l} \max_x = 2 \\ \max_y = 1 \\ \max_{x,y} = 1 \\ \max_{y,x} = -1 \end{array}\right.$$

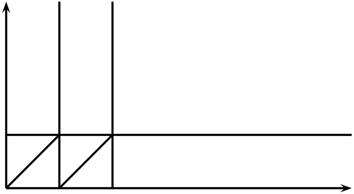


The bisimulation property is met.

What's wrong when undecidable?

Decrementation $x := x - 1$

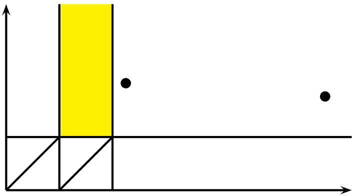
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

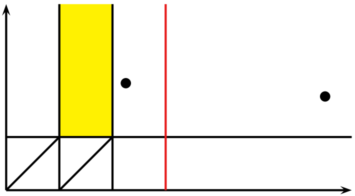
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

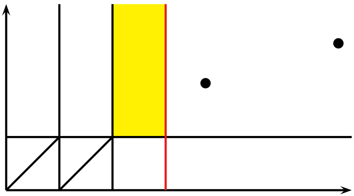
$\max_x \leq \max_x - 1$



What's wrong when undecidable?

Decrementation $x := x - 1$

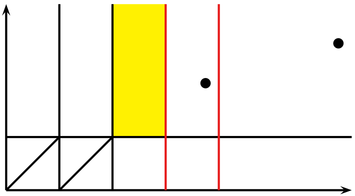
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

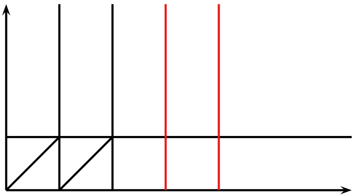
$\max_x \leq \max_x - 1$



What's wrong when undecidable?

Decrementation $x := x - 1$

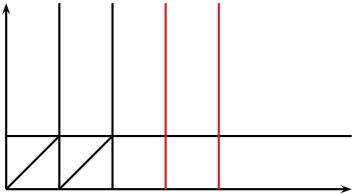
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

$$\max_x \leq \max_x - 1$$



etc...

Decidability (cont.)

<table>
<tr><th></th><th>Diagonal-free constraints</th><th>General constraints</th></tr>
<tr><td>$x := c,\ x := y$</td><td rowspan="3">PSPACE-complete</td><td>PSPACE-complete</td></tr>
<tr><td>$x := x + 1$</td><td rowspan="3">Undecidable</td></tr>
<tr><td>$x := y + c$</td></tr>
<tr><td>$x := x - 1$</td><td>Undecidable</td></tr>
<tr><td>$x :< c$</td><td rowspan="4">PSPACE-complete</td><td>PSPACE-complete</td></tr>
<tr><td>$x :> c$</td><td rowspan="4">Undecidable</td></tr>
<tr><td>$x :\sim y + c$</td></tr>
<tr><td>$y + c <: x :< y + d$</td></tr>
<tr><td>$y + c <: x :< z + d$</td><td>Undecidable</td></tr>
</table>

[BDFP00]

[BDFP00] Bouyer, Dufourd, Fleury, Petit. Updatable timed automata *(Theoretical Computer Science)*.

Outline

The example of alternating timed automata

Alternating timed automata $\equiv$ ATA

[LW05,OW05]

[LW05] Lasota, Walukiewicz. Alternating timed automata *(FoSSaCS'05)*.
[OW05] Ouaknine, Worrell. On the decidability of Metric Temporal Logic *(LICS'05)*.

The example of alternating timed automata

Alternating timed automata $\equiv$ ATA

[LW05,OW05]

Example

"No two a's are separated by 1 unit of time"

$$\begin{cases} \ell_0, a, \mathit{true} & \mapsto \quad \ell_0 \wedge (x := 0, \ell_1) \\ \ell_1, a, x \neq 1 & \mapsto \quad \ell_1 \\ \ell_1, a, x = 1 & \mapsto \quad \ell_2 \\ \ell_2, a, \mathit{true} & \mapsto \quad \ell_2 \end{cases}$$

$$\begin{cases} \ell_0 \text{ initial state} \\ \ell_0, \ell_1 \text{ final states} \\ \ell_2 \text{ losing state} \end{cases}$$

[LW05] Lasota, Walukiewicz. Alternating timed automata *(FoSSaCS'05)*.
[OW05] Ouaknine, Worrell. On the decidability of Metric Temporal Logic *(LICS'05)*.

The example of alternating timed automata

Alternating timed automata $\equiv$ ATA

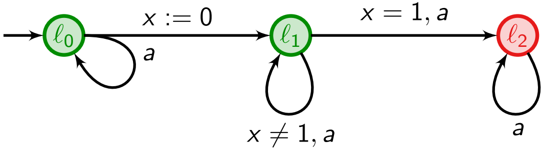
[LW05,OW05]

Example

"No two a's are separated by 1 unit of time"

$$\begin{cases} \ell_0, a, \mathit{true} & \mapsto \quad \ell_0 \wedge (x := 0, \ell_1) \\ \ell_1, a, x \neq 1 & \mapsto \quad \ell_1 \\ \ell_1, a, x = 1 & \mapsto \quad \ell_2 \\ \ell_2, a, \mathit{true} & \mapsto \quad \ell_2 \end{cases}$$

$$\begin{cases} \ell_0 \text{ initial state} \\ \ell_0, \ell_1 \text{ final states} \\ \ell_2 \text{ losing state} \end{cases}$$



[LW05] Lasota, Walukiewicz. Alternating timed automata *(FoSSaCS'05)*.
[OW05] Ouaknine, Worrell. On the decidability of Metric Temporal Logic *(LICS'05)*.

- nice closure properties

[Sch02] Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity *(Information Processing Letters)*.

- nice closure properties

  $\rightsquigarrow$ universality is as difficult as reachability

[Sch02] Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity *(Information Processing Letters)*.

- nice closure properties

  $\rightsquigarrow$ universality is as difficult as reachability

- more expressive than timed automata

[Sch02] Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity *(Information Processing Letters)*.

- nice closure properties

  $\rightsquigarrow$ universality is as difficult as reachability
- more expressive than timed automata

Theorem

- Emptiness of ATA is undecidable.
- Emptiness of one-clock ATA is decidable, but non-primitive recursive.
- Emptiness for Büchi properties of one-clock ATA is undecidable.
- Emptiness of one-clock ATA with ε-transitions is undecidable.

[Sch02] Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity *(Information Processing Letters)*.

- nice closure properties

  $\rightsquigarrow$ universality is as difficult as reachability
- more expressive than timed automata

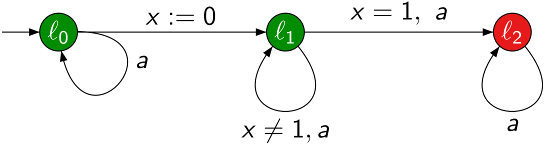
Theorem

- Emptiness of ATA is undecidable.
- Emptiness of one-clock ATA is decidable, but non-primitive recursive.
- Emptiness for Büchi properties of one-clock ATA is undecidable.
- Emptiness of one-clock ATA with ε-transitions is undecidable.

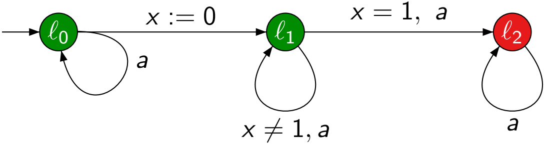
Lower bound: simulation of a lossy channel system... [Sch02]

[Sch02] Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity *(Information Processing Letters)*.

Example

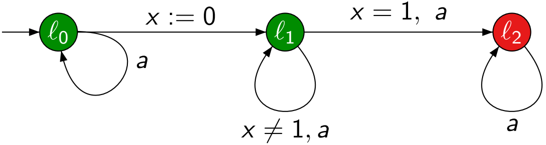


Example



Execution over timed word $(a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)$

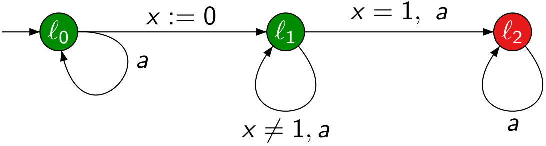
Example



Execution over timed word $(a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)$

$$\{\ (\ell_0, 0)\ \}$$

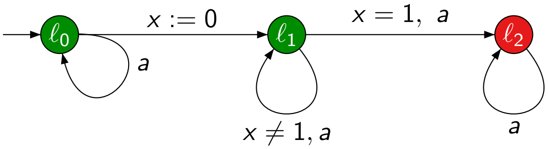
Example



Execution over timed word $(a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)$

$$\{\ \boxed{(\ell_0, 0)}\ \}$$
$$\downarrow$$
$$\{\ \boxed{(\ell_0, .3)}, \boxed{(\ell_1, 0)}\ \}$$

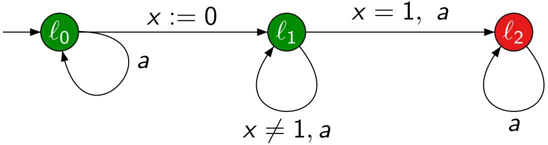
Example



Execution over timed word $(a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)$

$$\{\boxed{(\ell_0, 0)}\}$$
$$\downarrow$$
$$\{\boxed{(\ell_0, .3)}, \boxed{(\ell_1, 0)}\}$$
$$\downarrow$$
$$\{\boxed{(\ell_0, .8)}, \boxed{(\ell_1, 0)}, \boxed{(\ell_1, .5)}\}$$

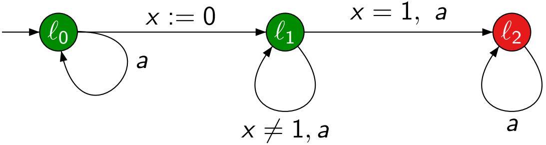
Example



Execution over timed word $(a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)$

$$\{ \boxed{(\ell_0, 0)} \}$$
$$\downarrow$$
$$\{ \boxed{(\ell_0, .3)}, \boxed{(\ell_1, 0)} \}$$
$$\downarrow$$
$$\{ \boxed{(\ell_0, .8)}, \boxed{(\ell_1, 0)}, \boxed{(\ell_1, .5)} \}$$
$$\downarrow$$
$$\{ \boxed{(\ell_0, 1.4)}, \boxed{(\ell_1, 0)}, \boxed{(\ell_1, .6)}, \boxed{(\ell_1, 1.1)} \}$$

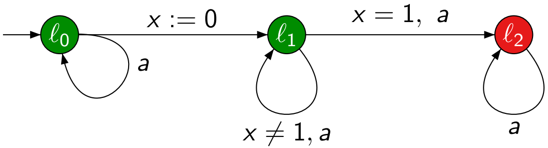
Example



Execution over timed word $(a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)$

$$\{(\ell_0, 0)\}$$
$$\downarrow$$
$$\{(\ell_0, .3), (\ell_1, 0)\}$$
$$\downarrow$$
$$\{(\ell_0, .8), (\ell_1, 0), (\ell_1, .5)\}$$
$$\downarrow$$
$$\{(\ell_0, 1.4), (\ell_1, 0), (\ell_1, .6), (\ell_1, 1.1)\}$$
$$\downarrow$$
$$\{(\ell_0, 1.8), (\ell_1, 0), (\ell_1, .4), (\ell_2, 1), (\ell_1, 1.5)\}$$

Example



Execution over timed word $(a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)$

$$\{(\ell_0, 0)\}$$
$$\downarrow$$
$$\{(\ell_0, .3), (\ell_1, 0)\}$$
$$\downarrow$$
$$\{(\ell_0, .8), (\ell_1, 0), (\ell_1, .5)\}$$
$$\downarrow$$
$$\{(\ell_0, 1.4), (\ell_1, 0), (\ell_1, .6), (\ell_1, 1.1)\}$$
$$\downarrow$$
$$\{(\ell_0, 1.8), (\ell_1, 0), (\ell_1, .4), (\ell_2, 1), (\ell_1, 1.5)\}$$
$$\downarrow$$
$$\{(\ell_0, 2), (\ell_1, 0), (\ell_1, .2), (\ell_1, .6), (\ell_2, 1.2), (\ell_1, 1.7)\}$$

An abstraction

A configuration = a finite set of pairs (ℓ, x)

$$(\ell, 0) \quad (\ell, 0.3) \quad (\ell, 1.2) \quad (\ell, 2.3) \quad (\ell', 0.4) \quad (\ell', 1) \quad (\ell', 0.8)$$

An abstraction

A configuration = a finite set of pairs (ℓ, x)

$(\ell, 0)$ $(\ell, 0.3)$ $(\ell, 1.2)$ $(\ell, 2.3)$ $(\ell', 0.4)$ $(\ell', 1)$ $(\ell', 0.8)$

$\{(\ell, 0), (\ell', 1)\}$

0.0

An abstraction

A configuration = a finite set of pairs (ℓ, x)

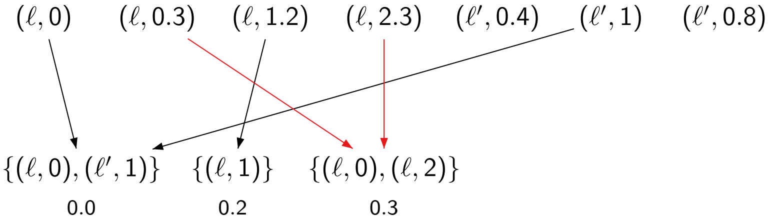
$(\ell, 0)$ $(\ell, 0.3)$ $(\ell, 1.2)$ $(\ell, 2.3)$ $(\ell', 0.4)$ $(\ell', 1)$ $(\ell', 0.8)$

$\{(\ell, 0), (\ell', 1)\}$ $\{(\ell, 1)\}$

0.0 0.2

An abstraction

A configuration = a finite set of pairs (ℓ, x)



An abstraction

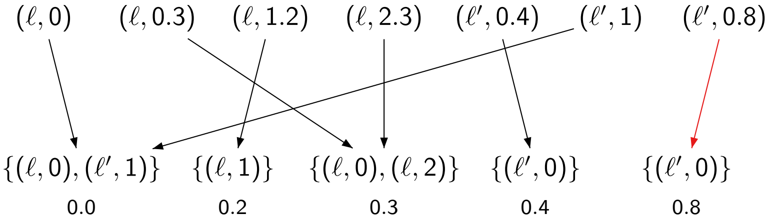
A configuration = a finite set of pairs (ℓ, x)

$$(\ell, 0) \quad (\ell, 0.3) \quad (\ell, 1.2) \quad (\ell, 2.3) \quad (\ell', 0.4) \quad (\ell', 1) \quad (\ell', 0.8)$$

$$\underset{0.0}{\{(\ell,0),(\ell',1)\}} \quad \underset{0.2}{\{(\ell,1)\}} \quad \underset{0.3}{\{(\ell,0),(\ell,2)\}} \quad \underset{0.4}{\{(\ell',0)\}}$$

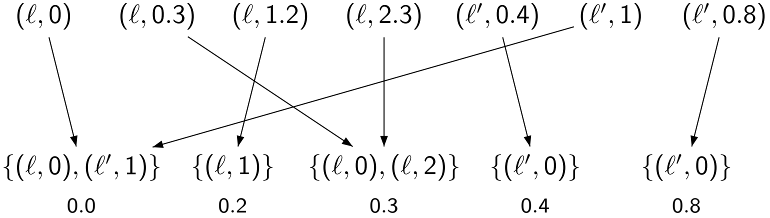
An abstraction

A configuration = a finite set of pairs (ℓ, x)



An abstraction

A configuration = a finite set of pairs (ℓ, x)



An abstraction

A configuration = a finite set of pairs (ℓ, x)

$(\ell, 0)$ $\quad (\ell, 0.3)$ $\quad (\ell, 1.2)$ $\quad (\ell, 2.3)$ $\quad (\ell', 0.4)$ $\quad (\ell', 1)$ $\quad (\ell', 0.8)$

$\{(\ell,0),(\ell',1)\}$	$\{(\ell,1)\}$	$\{(\ell,0),(\ell,2)\}$	$\{(\ell',0)\}$	$\{(\ell',0)\}$
0.0	0.2	0.3	0.4	0.8

Abstracted into: $\quad \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\} \cdot \{(\ell',0)\}$

Abstract transition system

$\{(\ell, 0), (\ell', 1)\} \cdot \{(\ell, 1)\} \cdot \{(\ell, 0), (\ell, 2)\} \cdot \{(\ell', 0)\} \cdot \{(\ell', 0)\}$

Abstract transition system

$\{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\} \cdot \{(\ell',0)\}$

Time successors:

Abstract transition system

$\{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\} \cdot \{(\ell',0)\}$

Time successors:

$\{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\}$

Abstract transition system

$\{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\} \cdot \{(\ell',0)\}$

Time successors:

$\{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\}$

$\{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\}$

Abstract transition system

$\{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\} \cdot \{(\ell',0)\}$

Time successors:

$\{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\}$

$\{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\}$

$\{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\}$

Abstract transition system

$\{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\} \cdot \{(\ell',0)\}$

Time successors:

$\{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\}$

$\{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\}$

$\{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\}$

$\{(\ell,2)\} \cdot \{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\}$

Abstract transition system

$\{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\} \cdot \{(\ell',0)\}$

Time successors:

$\{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\}$

$\{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\}$

$\{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\}$

$\{(\ell,2)\} \cdot \{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\}$

$\{(\ell,1),(\ell',2)\} \cdot \{(\ell,2)\} \cdot \{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\}$

Abstract transition system

$\{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\} \cdot \{(\ell',0)\}$

Time successors:

$\{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\}$

$\{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\}$

$\{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\}$

$\{(\ell,2)\} \cdot \{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\}$

$\{(\ell,1),(\ell',2)\} \cdot \{(\ell,2)\} \cdot \{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\}$

Transition $\ell \xrightarrow{x>2,x:=0} \ell''$**:**

Abstract transition system

$\{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\} \cdot \{(\ell',0)\}$

Time successors:

$\{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\} \cdot \{(\ell',0)\}$

$\{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\} \cdot \{(\ell,0),(\ell,2)\}$

$\{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\} \cdot \{(\ell,1)\}$

$\{(\ell,2)\} \cdot \{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\} \cdot \{(\ell,0),(\ell',1)\}$

$\{(\ell,1),(\ell',2)\} \cdot \{(\ell,2)\} \cdot \{(\ell,1),(\ell,3)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\}$

Transition $\ell \xrightarrow{x>2,x:=0} \ell''$**:**

$\{(\ell'',0)\} \cdot \{(\ell,1),(\ell',2)\} \cdot \{(\ell,1)\} \cdot \{(\ell',1)\} \cdot \{(\ell',1)\}$

What can we do with that abstract transition system?

Correctness?

What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.

What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

- ☹ possibly infinitely many abstract configurations

What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

- ☹ possibly infinitely many abstract configurations
- ☺ there is a well-quasi ordering on the set of abstract configurations! (subword relation $\sqsubseteq$)

What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

- ☹ possibly infinitely many abstract configurations
- ☺ there is a well-quasi ordering on the set of abstract configurations! (subword relation $\sqsubseteq$)
  $+$ downward compatibility:
  $$(\gamma_1 \sqsubseteq \gamma_1' \text{ and } \gamma_1' \rightsquigarrow \gamma_2') \Rightarrow (\gamma_1 \rightsquigarrow^* \gamma_2 \text{ and } \gamma_2 \sqsubseteq \gamma_2')$$

What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

- ☹ possibly infinitely many abstract configurations
- ☺ there is a well-quasi ordering on the set of abstract configurations! (subword relation $\sqsubseteq$)
  + downward compatibility:
  $$(\gamma_1 \sqsubseteq \gamma_1' \text{ and } \gamma_1' \leadsto \gamma_2') \Rightarrow (\gamma_1 \leadsto^* \gamma_2 \text{ and } \gamma_2 \sqsubseteq \gamma_2')$$
  + downward-closed objective (all states are accepting)

What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

- ☹ possibly infinitely many abstract configurations
- ☺ there is a well-quasi ordering on the set of abstract configurations! (subword relation $\sqsubseteq$)
  + downward compatibility:
  $$(\gamma_1 \sqsubseteq \gamma_1' \text{ and } \gamma_1' \rightsquigarrow \gamma_2') \Rightarrow (\gamma_1 \rightsquigarrow^* \gamma_2 \text{ and } \gamma_2 \sqsubseteq \gamma_2')$$
  + downward-closed objective (all states are accepting)

  A recipe:

What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

- ☹ possibly infinitely many abstract configurations
- ☺ there is a well-quasi ordering on the set of abstract configurations! (subword relation $\sqsubseteq$)
  + downward compatibility:
  $$(\gamma_1 \sqsubseteq \gamma_1' \text{ and } \gamma_1' \rightsquigarrow \gamma_2') \Rightarrow (\gamma_1 \rightsquigarrow^* \gamma_2 \text{ and } \gamma_2 \sqsubseteq \gamma_2')$$
  + downward-closed objective (all states are accepting)

  A recipe:

  $$(\text{Higman's lemma} + \text{Koenig's lemma}) \Rightarrow \text{termination}$$

What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

- ☹ possibly infinitely many abstract configurations
- ☺ there is a well-quasi ordering on the set of abstract configurations!
  (subword relation $\sqsubseteq$)
  + downward compatibility:
  $$(\gamma_1 \sqsubseteq \gamma_1' \text{ and } \gamma_1' \leadsto \gamma_2') \Rightarrow (\gamma_1 \leadsto^* \gamma_2 \text{ and } \gamma_2 \sqsubseteq \gamma_2')$$
  + downward-closed objective (all states are accepting)

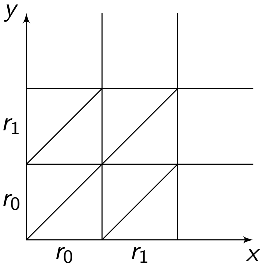
  A recipe:

  $$(\text{Higman's lemma} + \text{Koenig's lemma}) \Rightarrow \text{termination}$$

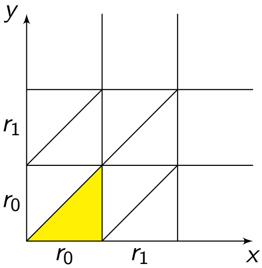
Alternative

The abstract transition system can be simulated by a kind of FIFO channel machine.

A digression on timed automata



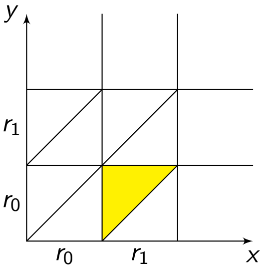
A digression on timed automata



$x, y \in r_0$, $\{y\} < \{x\}$

$\boxed{(y, r_0)} \cdot \boxed{(x, r_0)}$

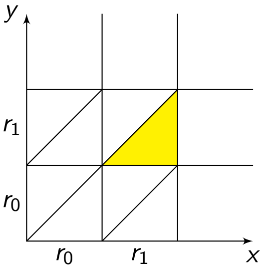
A digression on timed automata



$x \in r_1,\ y \in r_0,\ \{x\} < \{y\}$

$\boxed{(x, r_1)} \cdot \boxed{(y, r_0)}$

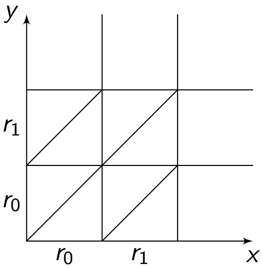
A digression on timed automata



$x, y \in r_1$, $\{y\} < \{x\}$

$\boxed{(y, r_1)} \cdot \boxed{(x, r_1)}$

A digression on timed automata



The classical region automaton can be simulated by a channel machine (with a single bounded channel).

Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata **[Abdulla,Jonsson 1998]**

Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata **[Abdulla,Jonsson 1998]**
- timed Petri nets **[Abdulla,Nylén 2001]**

Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata **[Abdulla,Jonsson 1998]**
- timed Petri nets **[Abdulla,Nylén 2001]**
- MTL model checking **[Ouaknine,Worrell 2005,2007]**

Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata **[Abdulla,Jonsson 1998]**
- timed Petri nets **[Abdulla,Nylén 2001]**
- MTL model checking **[Ouaknine,Worrell 2005,2007]**
- coFlatMTL model checking **[Bouyer,Markey,Ouaknine,Worrell 2007]**
  (using channel machines with a bounded number of cycles)

Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata **[Abdulla,Jonsson 1998]**
- timed Petri nets **[Abdulla,Nylén 2001]**
- MTL model checking **[Ouaknine,Worrell 2005,2007]**
- coFlatMTL model checking **[Bouyer,Markey,Ouaknine,Worrell 2007]**
  (using channel machines with a bounded number of cycles)
- single-clock automata inclusion checking **[Ouaknine,Worrell 2004]**

Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata **[Abdulla,Jonsson 1998]**
- timed Petri nets **[Abdulla,Nylén 2001]**
- MTL model checking **[Ouaknine,Worrell 2005,2007]**
- coFlatMTL model checking **[Bouyer,Markey,Ouaknine,Worrell 2007]**
  (using channel machines with a bounded number of cycles)
- single-clock automata inclusion checking **[Ouaknine,Worrell 2004]**
- ...

Outline

1. Introduction
2. The timed automaton model
3. Timed automata, decidability issues
4. How far can we extend the model and preserve decidability?
   - Hybrid systems
   - Smaller extensions of timed automata
   - An alternative way of proving decidability
5. Timed automata in practice
6. Conclusion

What about the practice?

- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:

Finite representation of infinite sets of configurations

What about the practice?

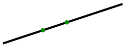
- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:

  Finite representation of infinite sets of configurations

  - in the plane, a line
    represented by two points.



What about the practice?

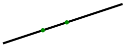
- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:

  Finite representation of infinite sets of configurations

  - in the plane, a line
    represented by two points.



  - set of words *aa*, *aaaa*, *aaaaaa*...
    represented by a rational expression $aa(aa)^*$

What about the practice?

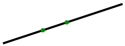
- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:

  Finite representation of infinite sets of configurations

  - in the plane, a line
    represented by two points.

  - set of words *aa*, *aaaa*, *aaaaaa*...
    represented by a rational expression $aa(aa)^*$
  - set of integers, represented using semi-linear sets

What about the practice?

- the region automaton is never computed
- instead, symbolic computations are performed

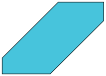
What do we need?

- Need of a symbolic representation:

Finite representation of infinite sets of configurations

- in the plane, a line
  represented by two points.

- set of words *aa*, *aaaa*, *aaaaaa*...
  represented by a rational expression $aa(aa)^*$
- set of integers, represented using semi-linear sets
- sets of constraints, polyhedra, zones, regions



What about the practice?

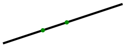
- the region automaton is never computed
- instead, symbolic computations are performed

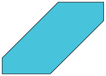
What do we need?

- Need of a symbolic representation:

  Finite representation of infinite sets of configurations

  - in the plane, a line
    represented by two points.
  - set of words *aa*, *aaaa*, *aaaaaa*...
    represented by a rational expression $aa(aa)^*$
  - set of integers, represented using semi-linear sets
  - sets of constraints, polyhedra, zones, regions
  - BDDs, DBMs (see later), CDDs, etc...





What about the practice?

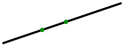
- the region automaton is never computed
- instead, symbolic computations are performed

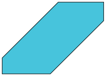
What do we need?

- Need of a symbolic representation:

  Finite representation of infinite sets of configurations

  - in the plane, a line
    represented by two points.
  - set of words *aa*, *aaaa*, *aaaaaa*...
    represented by a rational expression $aa(aa)^*$
  - set of integers, represented using semi-linear sets
  - sets of constraints, polyhedra, zones, regions
  - BDDs, DBMs (see later), CDDs, etc...





- Need of abstractions, heuristics, etc...

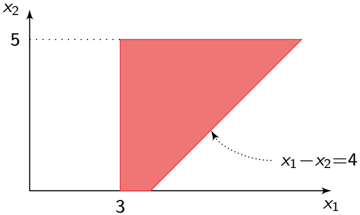
An example of computation with HyTech

```
command:  /usr/local/bin/hytech gas_burner
=================================================================
HyTech:  symbolic model checker for embedded systems
Version 1.04f (last modified 1/24/02) from v1.04a of 12/6/96
For more info:
    email:  hytech@eecs.berkeley.edu
    http://www.eecs.berkeley.edu/~tah/HyTech
Warning:  Input has changed from version 1.00(a).  Use -i for more info
=================================================================

Backward computation
Number of iterations required for reachability:  6
System satisfies non-leaking duration property

Location:  not_leaking
x >= 0 & t >= 3 & y <= 20t & y >= 0
| x + 20t >= y + 11 & y <= 20t + 19 & t >= 2 & x >= 0 & y >= 0
| y >= 0 & t >= 1 & x + 20t >= y + 22 & y <= 20t + 8 & x >= 0
| y >= 0 & x + 20t >= y + 33 & 20t >= y + 3 & x >= 0
Location:  leaking
19x + y <= 20t + 19 & y >= x + 59 & x <= 1 & x >= 0
| t >= x + 2 & x <= 1 & y >= 0 & 19x + y <= 20t + 19 & x >= 0
| t >= x + 1 & x <= 1 & y >= 0 & 19x + y <= 20t + 8 & x >= 0
| 20t >= 19x + y + 3 & y >= 0 & x <= 1 & x >= 0

=================================================================
Max memory used = 0 pages = 0 bytes = 0.00 MB
Time spent = 0.02u + 0.00s = 0.02 sec total

=================================================================
```

Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

$$Z \;=\; (x_1 \geq 3) \;\wedge\; (x_2 \leq 5) \;\wedge\; (x_1 - x_2 \leq 4)$$



$$\begin{array}{cc} & \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \\ x_2 \end{array} & \begin{pmatrix} \infty & -3 & \infty \\ \infty & \infty & 4 \\ 5 & \infty & \infty \end{pmatrix} \end{array}$$

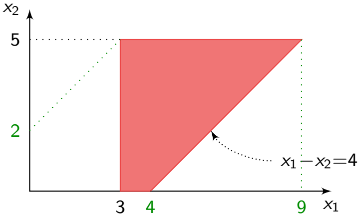
DBM: Difference Bound Matrice [BM83,Dill89]

[BM83] Berthomieu, Menasche. An enumerative approach for analyzing time Petri nets *World Comupter Congress*.
[Dill89] Dill. Timing assumptions and verification of finite-state concurrent systems *(Automatic Verification Methods for Finite State Systems)*.

Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

$$Z \;=\; (x_1 \geq 3) \;\wedge\; (x_2 \leq 5) \;\wedge\; (x_1 - x_2 \leq 4)$$



$$\begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix} \end{array}$$

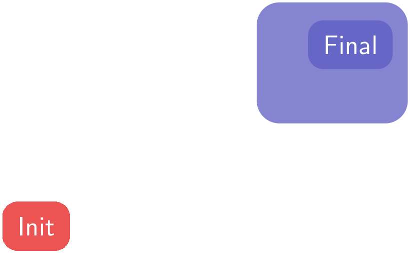
"normal form"

DBM: Difference Bound Matrice [BM83,Dill89]

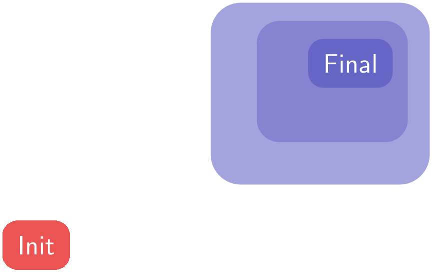
[BM83] Berthomieu, Menasche. An enumerative approach for analyzing time Petri nets *World Comupter Congress*.
[Dill89] Dill. Timing assumptions and verification of finite-state concurrent systems *(Automatic Verification Methods for Finite State Systems)*.

Backward computation

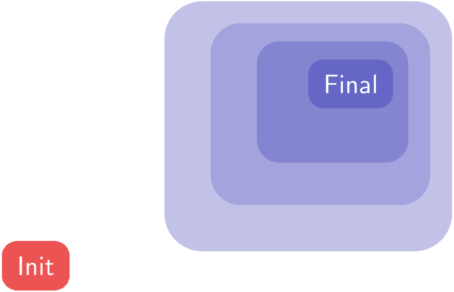
Backward computation



Backward computation

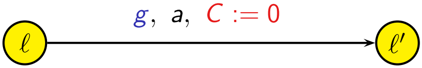


Backward computation



Backward computation

Note on the backward analysis of TA

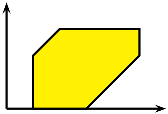


$$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g} \qquad Z$$

Note on the backward analysis of TA

$$\ell \xrightarrow{g,\ a,\ C := 0} \ell'$$

$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$ $\qquad$ Z

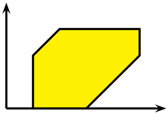


Z

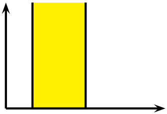
Note on the backward analysis of TA

$$\ell \xrightarrow{g,\ a,\ C := 0} \ell'$$

$$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g} \qquad\qquad Z$$



Z

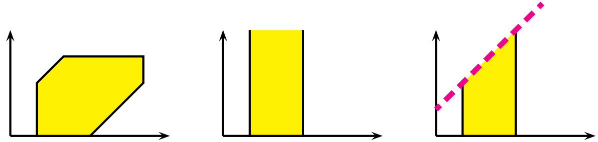


$[C \leftarrow 0]^{-1}(Z \cap (C = 0))$

Note on the backward analysis of TA

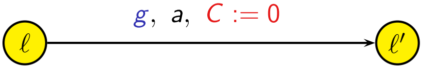
$$\ell \xrightarrow{g,\ a,\ C:=0} \ell'$$

$$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C=0)) \cap g} \qquad\qquad Z$$

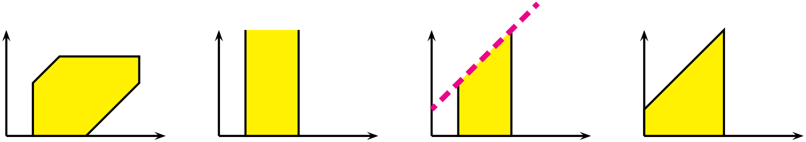


Z $\qquad$ $[C \leftarrow 0]^{-1}(Z \cap (C=0))$

Note on the backward analysis of TA

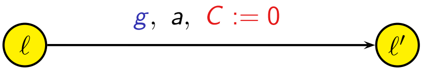


$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$ $\qquad\qquad$ Z

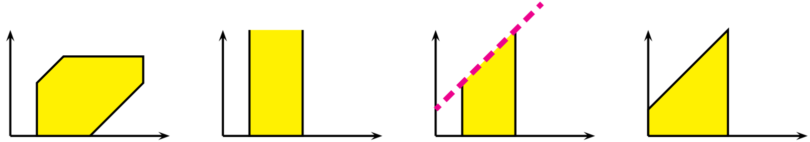


Z $\qquad$ $[C \leftarrow 0]^{-1}(Z \cap (C = 0))$ $\qquad$ $\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$

Note on the backward analysis of TA



$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$ $\qquad\qquad$ Z



Z $\qquad$ $[C \leftarrow 0]^{-1}(Z \cap (C = 0))$ $\qquad$ $\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$

☺ the backward computation always terminates!

☺☺ ... and it is correct!!!

Note on the backward analysis (cont.)

If $\mathcal{A}$ is a timed automaton, we construct its corresponding set of regions.

Because of the bisimulation property, we get that:

"Every set of valuations which is computed along the backward computation is a finite union of regions"

Note on the backward analysis (cont.)

If $\mathcal{A}$ is a timed automaton, we construct its corresponding set of regions.

Because of the bisimulation property, we get that:

"Every set of valuations which is computed along the backward computation is a finite union of regions"

Let R be a region. Assume:

- $v \in \overleftarrow{R}$ (for ex. $v + t \in R$)
- $v' \equiv_{\text{reg.}} v$

There exists t' s.t. $v' + t' \equiv_{\text{reg.}} v + t$, which implies that $v' + t' \in R$ and thus $v' \in \overleftarrow{R}$.

Note on the backward analysis (cont.)

If $\mathcal{A}$ is a timed automaton, we construct its corresponding set of regions.

Because of the bisimulation property, we get that:

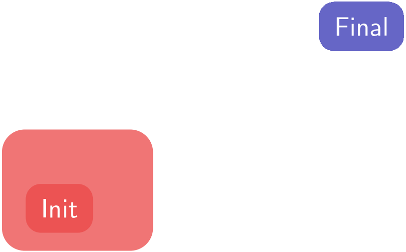
"Every set of valuations which is computed along the backward computation is a finite union of regions"

But, the backward computation is not so nice, when also dealing with integer variables...

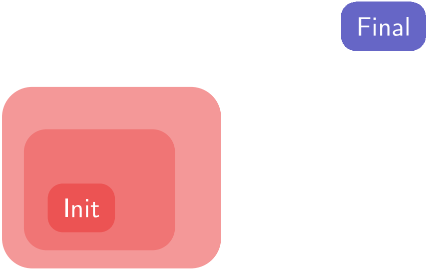
$$i := j.k + \ell.m$$

Forward computation

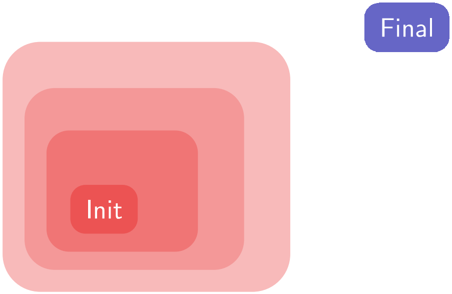
Forward computation



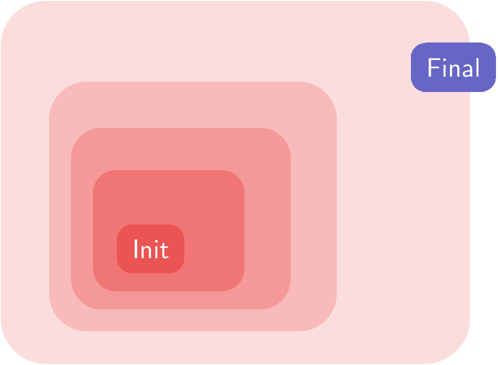
Forward computation



Forward computation



Forward computation



Forward analysis of timed automata

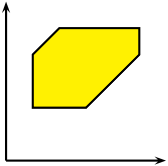
$$\ell \xrightarrow{\;g,\ a,\ C := 0\;} \ell'$$

zones $\quad Z \quad\quad [C \leftarrow 0](\overrightarrow{Z} \cap g)$

Forward analysis of timed automata

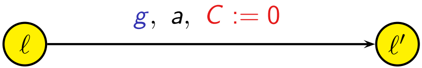
$$\ell \xrightarrow{g,\ a,\ C := 0} \ell'$$

zones $\quad Z \quad [C \leftarrow 0](\overrightarrow{Z} \cap g)$

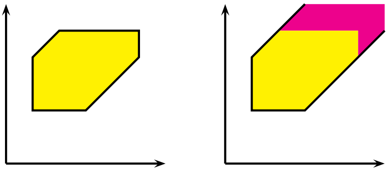


Z

Forward analysis of timed automata

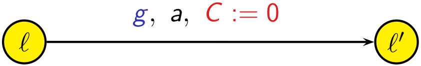


zones $\quad Z \quad [C \leftarrow 0](\overrightarrow{Z} \cap g)$

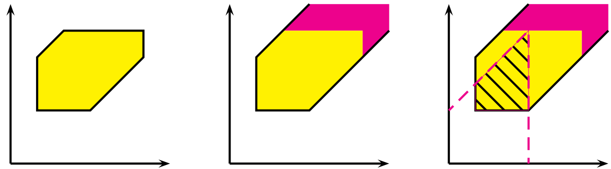


$Z \qquad \overrightarrow{Z}$

Forward analysis of timed automata

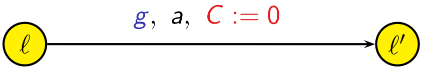


zones $\qquad Z \qquad [C \leftarrow 0](\overrightarrow{Z} \cap g)$

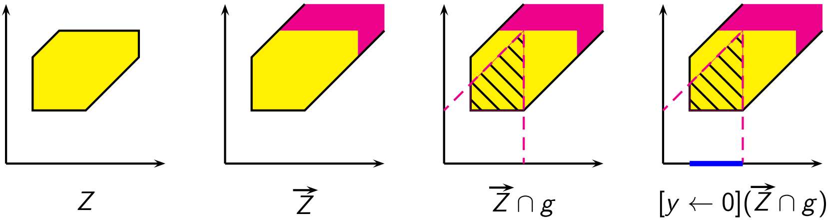


$Z \qquad \overrightarrow{Z} \qquad \overrightarrow{Z} \cap g$

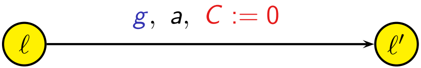
Forward analysis of timed automata



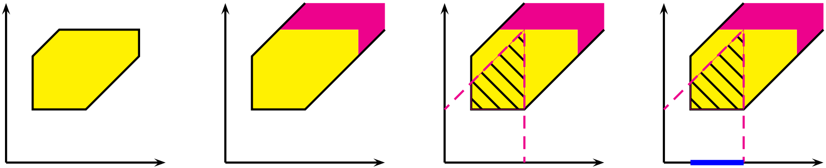
zones $\quad Z \quad\quad [C \leftarrow 0](\overrightarrow{Z} \cap g)$



Forward analysis of timed automata



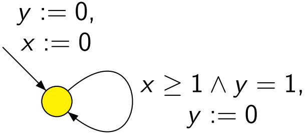
zones $\quad Z \quad\quad [C \leftarrow 0](\overrightarrow{Z} \cap g)$

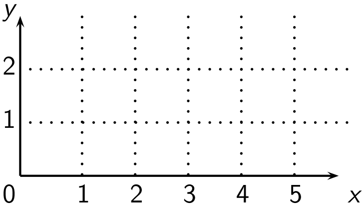


$Z \qquad \overrightarrow{Z} \qquad \overrightarrow{Z} \cap g \qquad [y \leftarrow 0](\overrightarrow{Z} \cap g)$

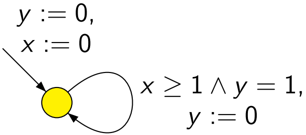
☹ the forward computation may not terminate...

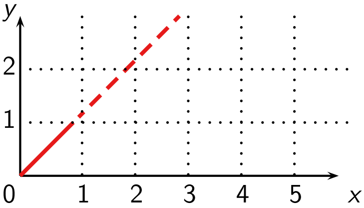
Non termination of the forward analysis



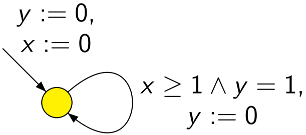


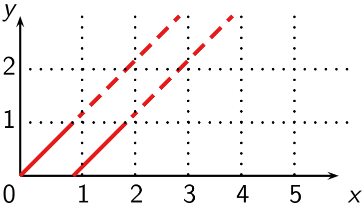
Non termination of the forward analysis



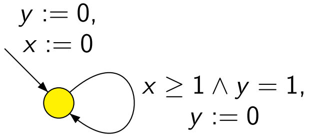


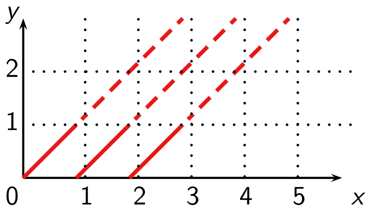
Non termination of the forward analysis



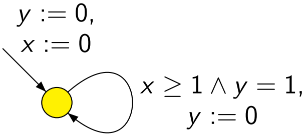


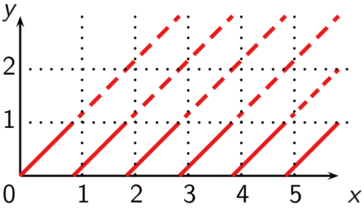
Non termination of the forward analysis





Non termination of the forward analysis





$\rightsquigarrow$ an infinite number of steps...

"Solutions" to this problem

(f.ex. in **[Larsen,Pettersson,Yi 1997]** or in **[Daws,Tripakis 1998]**)

- **inclusion checking**: if $Z \subseteq Z'$ and Z' already considered, then we don't need to consider Z

$\rightsquigarrow$ correct w.r.t. reachability

...

"Solutions" to this problem

(f.ex. in **[Larsen,Pettersson,Yi 1997]** or in **[Daws,Tripakis 1998]**)

- **inclusion checking**: if $Z \subseteq Z'$ and Z' already considered, then we don't need to consider Z

  $\rightsquigarrow$ correct w.r.t. reachability

- **activity**: eliminate redundant clocks **[Daws,Yovine 1996]**

  $\rightsquigarrow$ correct w.r.t. reachability

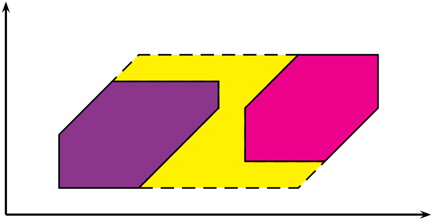
$$q \xrightarrow{g,a,C:=0} q' \quad \text{implies} \quad \mathsf{Act}(q) = \mathsf{clocks}(g) \cup (\mathsf{Act}(q') \setminus C)$$

...

"Solutions" to this problem (cont.)

- **convex-hull approximation**: if Z and Z' are computed then we overapproximate using "$Z \sqcup Z'$".

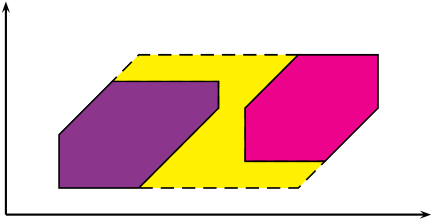
  $\rightsquigarrow$ "semi-correct" w.r.t. reachability



"Solutions" to this problem (cont.)

- **convex-hull approximation**: if Z and Z' are computed then we overapproximate using "$Z \sqcup Z'$".

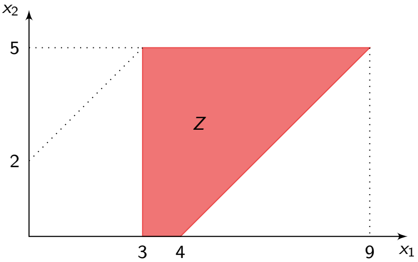
  $\rightsquigarrow$ "semi-correct" w.r.t. reachability



- **extrapolation**, an abstraction operator on zones

An abstraction: the extrapolation operator

$\mathsf{Approx}_2(Z)$: "the smallest zone containing Z that is defined only with constants no more than 2"

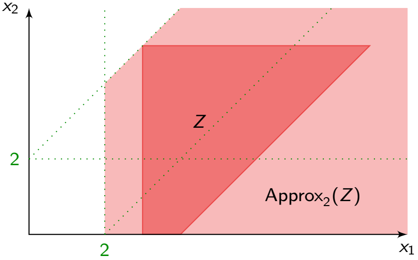


$$\begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix}$$

$\leadsto$ The extrapolation operator ensures termination of the computation!

An abstraction: the extrapolation operator

$\mathsf{Approx}_2(Z)$: "the smallest zone containing Z that is defined only with constants no more than 2"



$$\begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix} \xRightarrow{\mathsf{Approx}_2} \begin{pmatrix} 0 & -2 & 0 \\ \infty & 0 & \infty \\ \infty & 2 & 0 \end{pmatrix}$$

$\rightsquigarrow$ The extrapolation operator ensures termination of the computation!

Classical algorithm, focus on correctness

Challenge

Choose a **good** constant for the extrapolation so that the forward computation is correct. Classical algorithm, the choice goes to the maximal constant.

[Bou03] Bouyer. Untameable timed automata! *(STACS'03)*.
[Bou04] Bouyer. Forward analysis of updatable timed automata *(Formal Methods in System Design)*.

Classical algorithm, focus on correctness

Challenge

Choose a **good** constant for the extrapolation so that the forward computation is correct. Classical algorithm, the choice goes to the maximal constant.

- Implemented in tools like Uppaal, Kronos, RT-Spin...
- Successfully used on many real-life examples

[Bou03] Bouyer. Untameable timed automata! *(STACS'03)*.
[Bou04] Bouyer. Forward analysis of updatable timed automata *(Formal Methods in System Design)*.

Classical algorithm, focus on correctness

Challenge

Choose a **good** constant for the extrapolation so that the forward computation is correct. Classical algorithm, the choice goes to the maximal constant.

- Implemented in tools like Uppaal, Kronos, RT-Spin...
- Successfully used on many real-life examples

Theorem

The classical algorithm is correct for diagonal-free timed automata.

[Bou03] Bouyer. Untameable timed automata! *(STACS'03)*.
[Bou04] Bouyer. Forward analysis of updatable timed automata *(Formal Methods in System Design)*.

Classical algorithm, focus on correctness

Challenge

Choose a **good** constant for the extrapolation so that the forward computation is correct. Classical algorithm, the choice goes to the maximal constant.

- Implemented in tools like Uppaal, Kronos, RT-Spin...
- Successfully used on many real-life examples

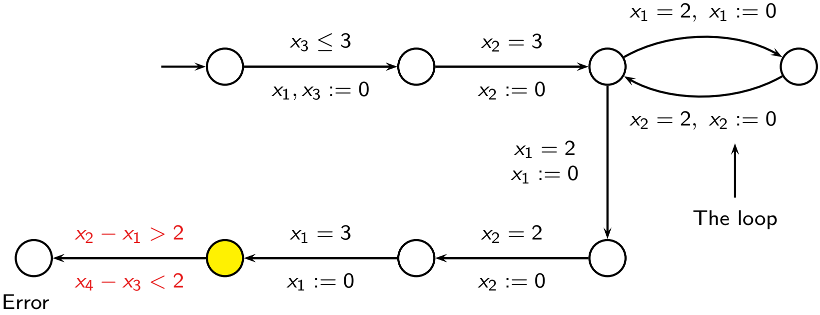
Theorem

The classical algorithm is correct for diagonal-free timed automata.

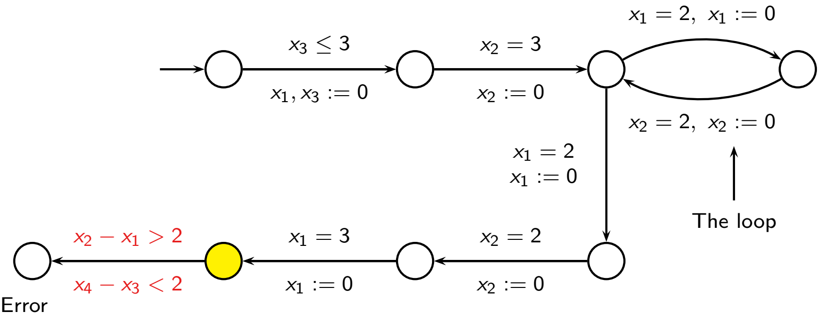
This theorem does not extend to timed automata using diagonal clock constraints... [Bou03,Bou04]

[Bou03] Bouyer. Untameable timed automata! *(STACS'03)*.
[Bou04] Bouyer. Forward analysis of updatable timed automata *(Formal Methods in System Design)*.

A problematic automaton

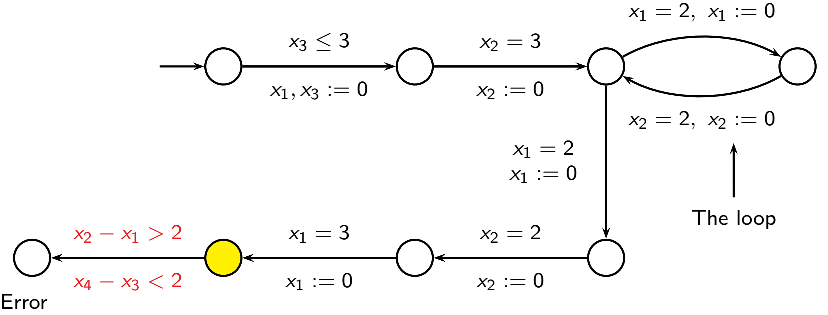


A problematic automaton

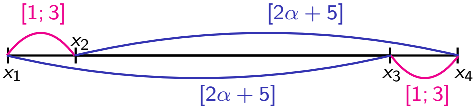


$$\begin{cases} v(x_1) = 0 \\ v(x_2) = d \\ v(x_3) = 2\alpha + 5 \\ v(x_4) = 2\alpha + 5 + d \end{cases}$$

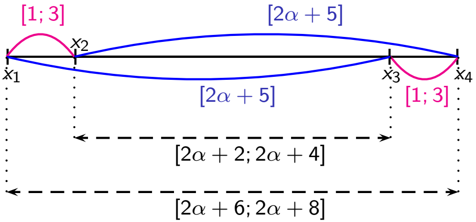
A problematic automaton



$$\begin{cases} v(x_1) = 0 \\ v(x_2) = d \\ v(x_3) = 2\alpha + 5 \\ v(x_4) = 2\alpha + 5 + d \end{cases}$$

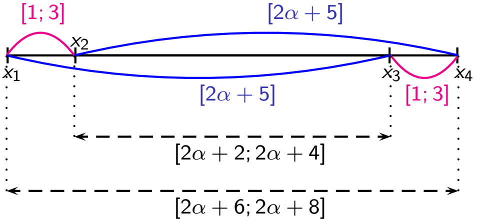


The problematic zone



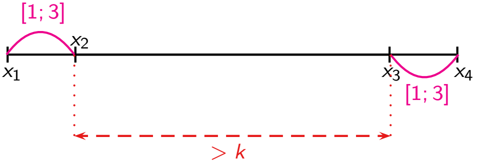
implies $x_1 - x_2 = x_3 - x_4$.

The problematic zone



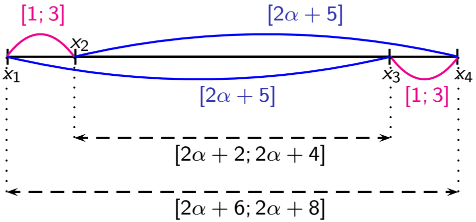
implies $x_1 - x_2 = x_3 - x_4$.

If α is sufficiently large, after extrapolation:



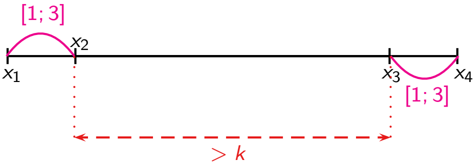
does not imply $x_1 - x_2 = x_3 - x_4$.

The problematic zone



implies $x_1 - x_2 = x_3 - x_4$.

If α is sufficiently large, after extrapolation:



does not imply $x_1 - x_2 = x_3 - x_4$.

Hence, any choice of constant is erroneous!

General abstractions

Criteria for a good abstraction operator Abs:

General abstractions

Criteria for a good abstraction operator Abs:

- easy computation **[Effectiveness]**

  $\mathrm{Abs}(Z)$ is a zone if Z is a zone

General abstractions

Criteria for a good abstraction operator Abs:

- easy computation **[Effectiveness]**
  $\mathrm{Abs}(Z)$ is a zone if Z is a zone
- finiteness of the abstraction **[Termination]**
  $\{\mathrm{Abs}(Z) \mid Z \text{ zone}\}$ is finite

General abstractions

Criteria for a good abstraction operator Abs:

- easy computation **[Effectiveness]**
  $\mathrm{Abs}(Z)$ is a zone if Z is a zone
- finiteness of the abstraction **[Termination]**
  $\{\mathrm{Abs}(Z) \mid Z \text{ zone}\}$ is finite
- completeness of the abstraction **[Completeness]**
  $Z \subseteq \mathrm{Abs}(Z)$

General abstractions

Criteria for a good abstraction operator Abs:

- easy computation **[Effectiveness]**
  $\mathrm{Abs}(Z)$ is a zone if Z is a zone
- finiteness of the abstraction **[Termination]**
  $\{\mathrm{Abs}(Z) \mid Z \text{ zone}\}$ is finite
- completeness of the abstraction **[Completeness]**
  $Z \subseteq \mathrm{Abs}(Z)$
- soundness of the abstraction **[Soundness]**
  the computation of $(\mathrm{Abs} \circ \mathrm{Post})^*$ is correct w.r.t. reachability

General abstractions

Criteria for a good abstraction operator Abs:

- easy computation **[Effectiveness]**
  $\mathrm{Abs}(Z)$ is a zone if Z is a zone
- finiteness of the abstraction **[Termination]**
  $\{\mathrm{Abs}(Z) \mid Z \text{ zone}\}$ is finite
- completeness of the abstraction **[Completeness]**
  $Z \subseteq \mathrm{Abs}(Z)$
- soundness of the abstraction **[Soundness]**
  the computation of $(\mathrm{Abs} \circ \mathrm{Post})^*$ is correct w.r.t. reachability

For the previous automaton,

no abstraction operator can satisfy all these criteria!

Why that?

Assume there is a "nice" operator Abs.

The set $\{M \text{ DBM representing a zone } \mathrm{Abs}(Z)\}$ is finite.

$\rightsquigarrow$ k the max. constant defining one of the previous DBMs

We get that, for every zone Z,

$$Z \subseteq \mathrm{Extra}_k(Z) \subseteq \mathrm{Abs}(Z)$$

Problem!

Open questions:
- which conditions can be made weaker?
- find a clever termination criterium?
- use an other data structure than zones/DBMs?

Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded contraints [BBLP03,BBLP06]

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification *(TACAS'03)*.
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata *(TACAS'04)*.
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata *(International Journal on Software Tools for Technology Transfer)*.
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal *(FORMATS'03)*.
[DHLP06] David, Håkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction *(FORMATS'06)*.

Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded contraints [BBLP03,BBLP06]
- heuristics can be added
  - order for exploration
  - symmetry reduction [HBL+03]

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification *(TACAS'03)*.
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata *(TACAS'04)*.
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata *(International Journal on Software Tools for Technology Transfer)*.
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal *(FORMATS'03)*.
[DHLP06] David, Håkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction *(FORMATS'06)*.

Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded contraints [BBLP03,BBLP06]
- heuristics can be added
  - order for exploration
  - symmetry reduction [HBL+03]
- the representation of zones can be improved [DHLP06]

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification *(TACAS'03)*.
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata *(TACAS'04)*.
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata *(International Journal on Software Tools for Technology Transfer)*.
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal *(FORMATS'03)*.
[DHLP06] David, Håkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction *(FORMATS'06)*.

Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded contraints [BBLP03,BBLP06]
- heuristics can be added
  - order for exploration
  - symmetry reduction [HBL+03]
- the representation of zones can be improved [DHLP06]

$\rightsquigarrow$ the tool Uppaal is under development since 1995...

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification *(TACAS'03)*.
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata *(TACAS'04)*.
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata *(International Journal on Software Tools for Technology Transfer)*.
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal *(FORMATS'03)*.
[DHLP06] David, Håkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction *(FORMATS'06)*.

Outline

Conclusion

- Justification of the dense-time paradigm
- Several technics for proving decidability of real-time systems
  - finite time-abstract bisimulation
  - well-quasi-order on the time-abstract transition system
- Timed automata are implemented in several model checking tools
  - Other timed models have been developed and have concurrent tools: for instance Romeo and Tina for time Petri nets

Conclusion

- Justification of the dense-time paradigm
- Several technics for proving decidability of real-time systems
  - finite time-abstract bisimulation
  - well-quasi-order on the time-abstract transition system
- Timed automata are implemented in several model checking tools
  - Other timed models have been developed and have concurrent tools: for instance Romeo and Tina for time Petri nets

Some current streams of research in timed systems:

- quantitative model-checking,
- real-time logics,
- robustness, implementability issues,
- timed games,
- modelling of resources,
- ...

Conclusion

- Justification of the dense-time paradigm
- Several technics for proving decidability of real-time systems
  - finite time-abstract bisimulation
  - well-quasi-order on the time-abstract transition system
- Timed automata are implemented in several model checking tools
  - Other timed models have been developed and have concurrent tools: for instance Romeo and Tina for time Petri nets

Some current streams of research in timed systems:

- quantitative model-checking,
- real-time logics,
- robustness, implementability issues,
- timed games,
- modelling of resources,
- ...