Foundation for Timed Systems

Patricia Bouyer

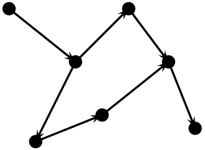
LSV – CNRS & ENS de Cachan – France

October 2, 2005

Model-checking

Does the system satisfy the property?

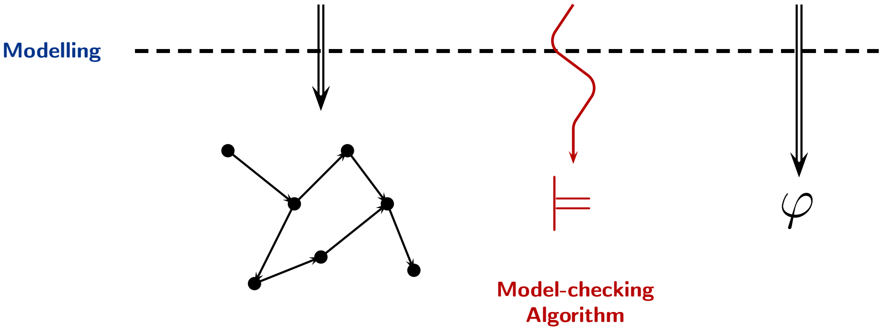
Modelling



φ

Model-checking

Does the system satisfy the property?



Time!

Context: verification of embedded critical systems

Time

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

➜ Need of models and specification languages integrating timing aspects

Outline

1. **About time semantics**
2. Timed automata, decidability issues
3. Some extensions of the model
4. Implementation of timed automata
5. Conclusion

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 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 N
    Ex: $(a, 1)(b, 3)(c, 4)(a, 6)$
  - **Dense-time semantics:** dates are *e.g.* taken in Q^+, or in R^+
    Ex: $(a, 1.28).(b, 3.1).(c, 3.98)(a, 6.13)$

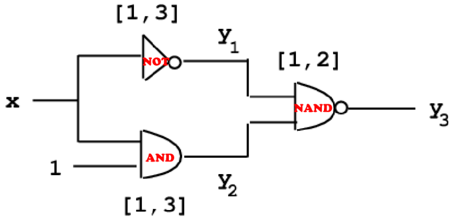
A case for dense-time

Time domain: discrete (*e.g.* N) or dense (*e.g.* Q^+)

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

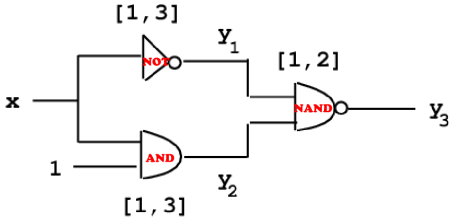
A digital circuit [Alur 91]

Discussion in the context of reachability problems for asynchronous digital circuits [Brzozowski, Seger 1991]



A digital circuit [Alur 91]

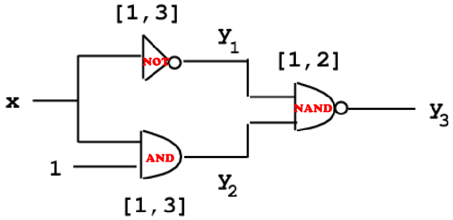
Discussion in the context of reachability problems for asynchronous digital circuits [Brzozowski, Seger 1991]



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

A digital circuit [Alur 91]

Discussion in the context of reachability problems for asynchronous digital circuits [Brzozowski, Seger 1991]

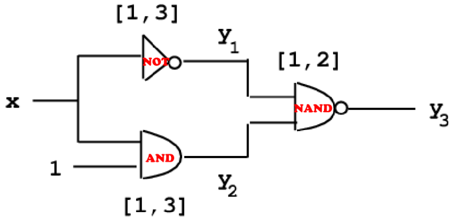


Start with x=0 and y=[101] (stable configuration)

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

A digital circuit [Alur 91]

Discussion in the context of reachability problems for asynchronous digital circuits [Brzozowski, Seger 1991]



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

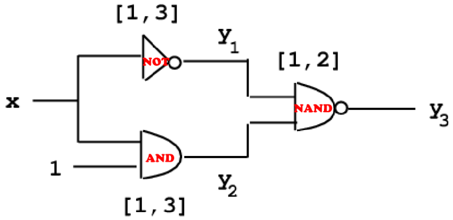
The input x changes to 1. The corresponding stable state is y=[011]

However, many possible behaviours, e.g.

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

A digital circuit [Alur 91]

Discussion in the context of reachability problems for asynchronous digital circuits [Brzozowski, Seger 1991]



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

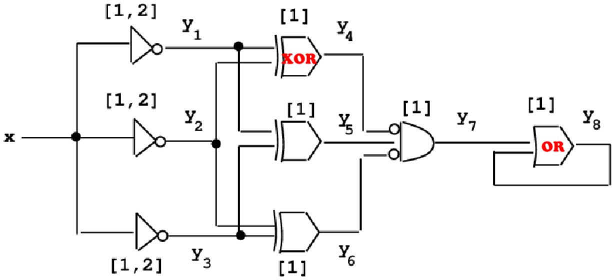
The input x changes to 1. The corresponding stable state is y=[011]

However, many possible behaviours, e.g.

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

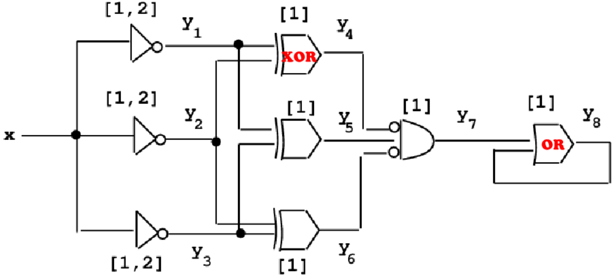
Reachable configurations: {[101], [111], [110], [010], [011], [001]}

Is discretizing sufficient? An example [Alur 91]



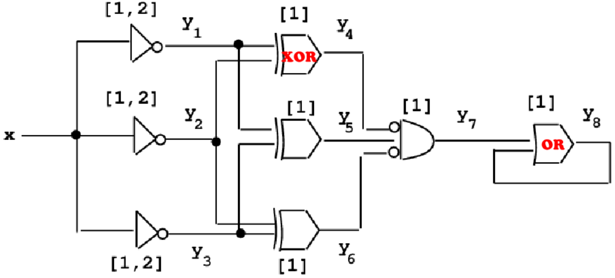
- This digital circuit **is not** 1-discretizable.

Is discretizing sufficient? An example [Alur 91]



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

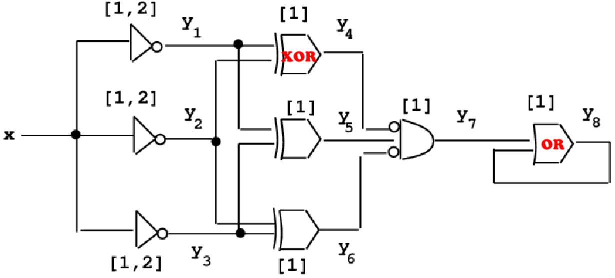
Is discretizing sufficient? An example [Alur 91]



- 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 [Alur 91]

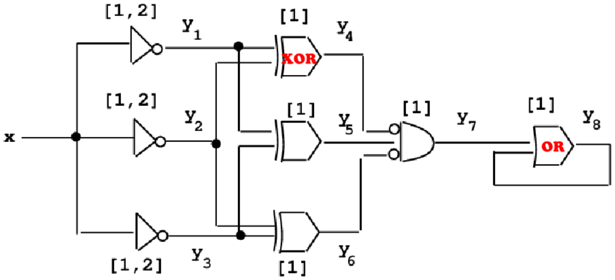


- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, x is set to 1)

[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 [Alur 91]



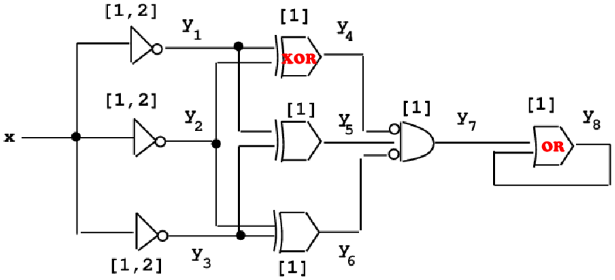
- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, x is set to 1)

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

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

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

Is discretizing sufficient? An example [Alur 91]



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

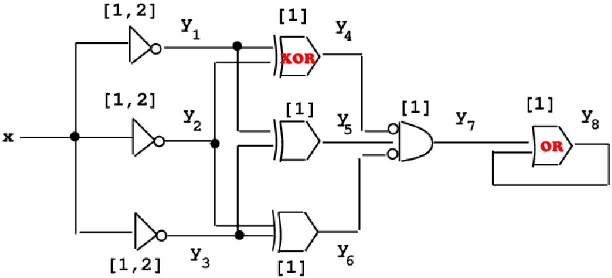
$[11100000] \xrightarrow[1]{y_1} [01100000] \xrightarrow[1.5]{y_2} [00100000] \xrightarrow[2]{y_3,y_5} [00001000] \xrightarrow[3]{y_5,y_7} [00000010] \xrightarrow[4]{y_7,y_8} [00000001]$

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

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

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

Is discretizing sufficient? An example [Alur 91]



- 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 [Brzozowski Seger 1991]

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 $\frac{1}{k}$).

Is discretizing sufficient?

Theorem [Brzozowski Seger 1991]

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.

Is discretizing sufficient?

Theorem [Brzozowski Seger 1991]

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.

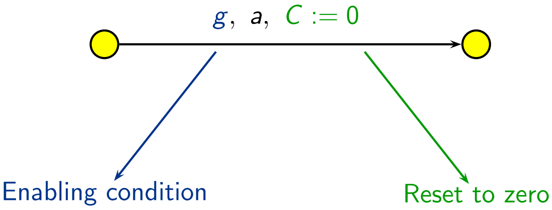
Going further... There exist systems for which no granularity exists.

(see later)

Outline

Timed automata [Alur & Dill 90's]

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



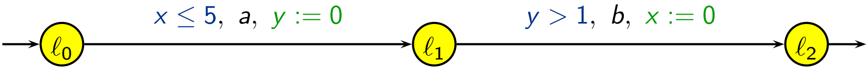
- An enabling condition (or guard) is:

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

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

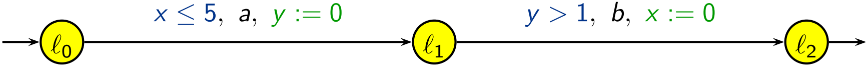
Timed automata (example)

x, y : clocks



Timed automata (example)

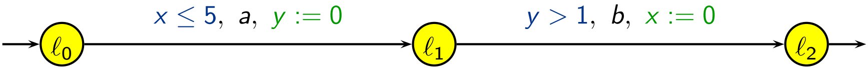
x, y : clocks



$$
\begin{array}{cccccccccc}
 & \ell_0 & \xrightarrow{\delta(4.1)} & \ell_0 & \xrightarrow{a} & \ell_1 & \xrightarrow{\delta(1.4)} & \ell_1 & \xrightarrow{b} & \ell_2 \\
x & 0 & & 4.1 & & 4.1 & & 5.5 & & 0 \\
y & 0 & & 4.1 & & 0 & & 1.4 & & 1.4
\end{array}
$$

Timed automata (example)

x, y : clocks

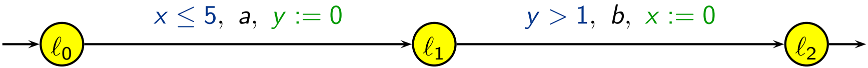


	ℓ_0	$\xrightarrow{\delta(4.1)}$	ℓ_0	$\xrightarrow{a}$	ℓ_1	$\xrightarrow{\delta(1.4)}$	ℓ_1	$\xrightarrow{b}$	ℓ_2
x	0		4.1		4.1		5.5		0
y	0		4.1		0		1.4		1.4

(clock) valuation

Timed automata (example)

x, y : clocks



$$\begin{array}{cccccccccc} & \ell_0 & \xrightarrow{\delta(4.1)} & \ell_0 & \xrightarrow{a} & \ell_1 & \xrightarrow{\delta(1.4)} & \ell_1 & \xrightarrow{b} & \ell_2 \\ x & 0 & & 4.1 & & 4.1 & & 5.5 & & 0 \\ y & 0 & & 4.1 & & 0 & & 1.4 & & 1.4 \end{array}$$

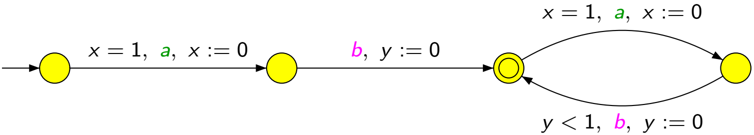
(clock) valuation

➜ timed word $(a, 4.1)(b, 5.5)$

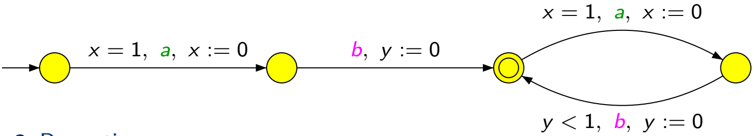
Timed automata semantics

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

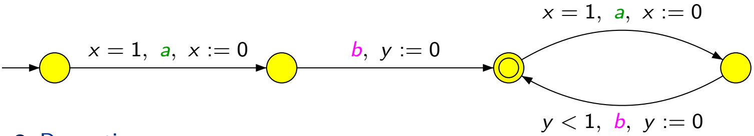


Discrete *vs* dense-time semantics



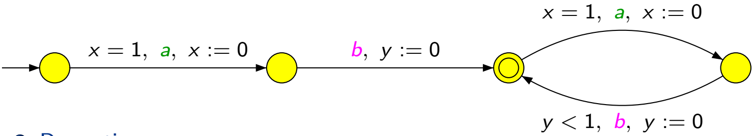
- Dense-time:
  $L_{dense} = \{((ab)^\omega, \tau) \mid \forall i,\ \tau_{2i-1} = i \text{ and } \tau_{2i} - \tau_{2i-1} > \tau_{2i+2} - \tau_{2i+1}\}$

Discrete *vs* dense-time semantics

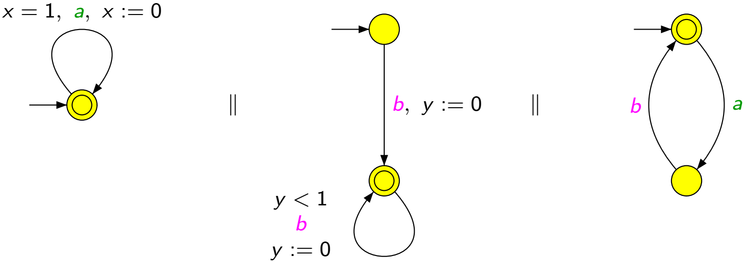


- Dense-time:
  $L_{dense} = \{((ab)^\omega, \tau) \mid \forall i,\ \tau_{2i-1} = i \text{ and } \tau_{2i} - \tau_{2i-1} > \tau_{2i+2} - \tau_{2i+1}\}$
- Discrete-time: $L_{discrete} = \emptyset$

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: $L_{discrete} = \emptyset$

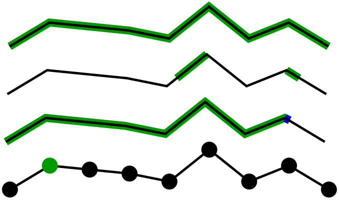


Classical verification problems

- 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:



$G\phi$ « Always »

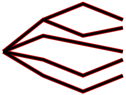
$F\phi$ « Eventually »

$\phi U \phi'$ « Until »

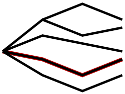
$X\phi$ « Next »

State formulas:

$A\psi$



$E\psi$



➜ LTL: Linear Temporal Logic **[Pnueli 1977]**,
CTL: Computation Tree Logic **[Emerson, Clarke 1982]**

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$AG(\text{problem} \Rightarrow AF\ \text{alarm})$$

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$AG(\text{problem} \Rightarrow AF\ \text{alarm})$$

How can we express:

"any problem is followed by an alarm **in at most** 20 **time units**"

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$AG(\text{problem} \Rightarrow AF\ \text{alarm})$$

How can we express:

"any problem is followed by an alarm **in at most** 20 **time units**"

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

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$AG(\text{problem} \Rightarrow AF\ \text{alarm})$$

How can we express:

"any problem is followed by an alarm **in at most** 20 **time units**"

- Temporal logics with **subscripts**.

$$AG(\text{problem} \Rightarrow AF_{\leq 20}\ \text{alarm})$$

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$AG(\text{problem} \Rightarrow AF\ \text{alarm})$$

How can we express:

"any problem is followed by an alarm **in at most** 20 **time units**"

- Temporal logics with **subscripts**.

$$AG(\text{problem} \Rightarrow AF_{\leq 20}\ \text{alarm})$$

- Temporal logics with **clocks**.

$$AG(\text{problem} \Rightarrow (x\ \mathtt{in}\ AF(x \leq 20 \wedge \text{alarm})))$$

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$AG(\text{problem} \Rightarrow AF \text{ alarm})$$

How can we express:

"any problem is followed by an alarm **in at most** 20 **time units**"

- Temporal logics with **subscripts**.

$$AG(\text{problem} \Rightarrow AF_{\leq 20} \text{ alarm})$$

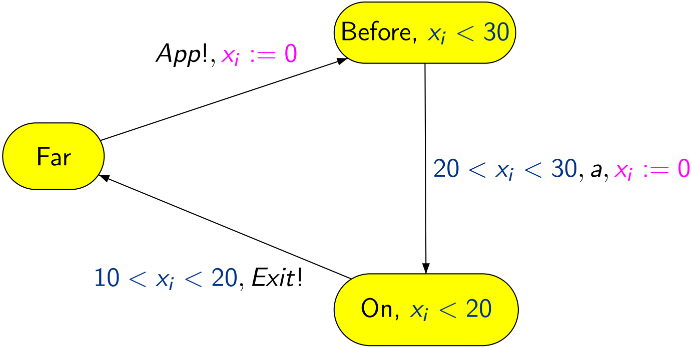
- Temporal logics with **clocks**.

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

➜ **TCTL**: Timed CTL **[ACD90,ACD93,HNSY94]**

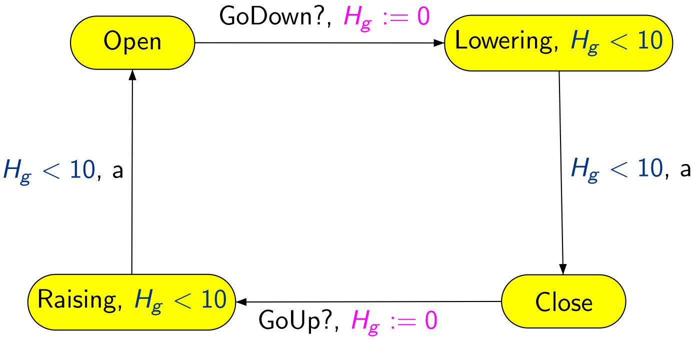
The train crossing example (1)

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



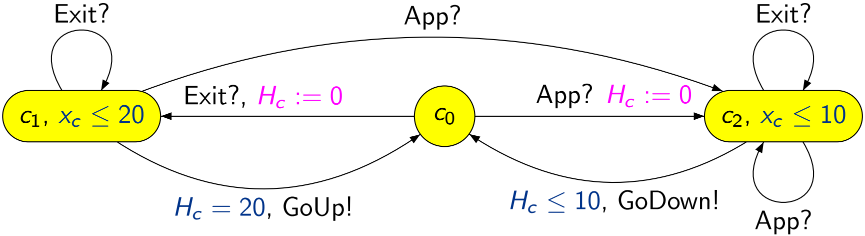
The train crossing example (2)

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?

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

The train crossing example (5)

Some properties one could check:

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

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

- 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?

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

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

$$AG\ AF_{<5\text{min}}(\neg\text{gate.Close})$$

Verification

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

- reachability properties (final states)
- basic liveness properties (Büchi (or other) conditions)

Verification

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

- **Problem:** the set of configurations is infinite
  ➜ classical methods can not be applied

Verification

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

- **Problem:** the set of configurations is infinite
  ➜ classical methods can not be applied

- **Positive key point:** variables (clocks) have the same speed

Verification

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

- **Problem:** the set of configurations is infinite
  ➜ classical methods can not be applied
- **Positive key point:** variables (clocks) have the same speed

Theorem [Alur & Dill 1990's]

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

Verification

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

- **Problem:** the set of configurations is infinite
  ➜ classical methods can not be applied

- **Positive key point:** variables (clocks) have the same speed

Theorem [Alur & Dill 1990's]

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

Note: This is also the case for the discrete semantics.

Verification

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

- **Problem:** the set of configurations is infinite
  ➜ classical methods can not be applied

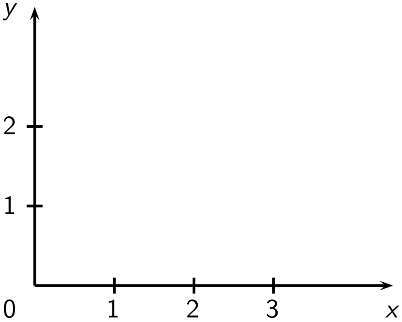
- **Positive key point:** variables (clocks) have the same speed

Theorem [Alur & Dill 1990's]

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

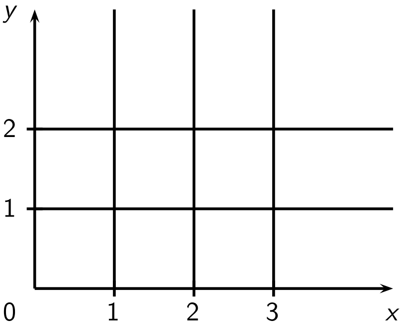
Method: construct a finite abstraction

The region abstraction



Equivalence of finite index

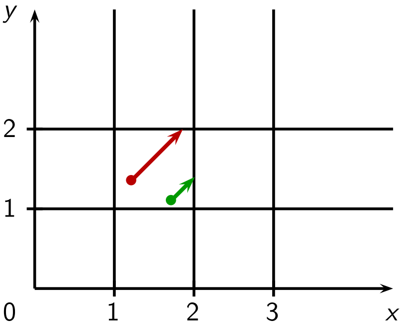
The region abstraction



Equivalence of finite index

- "compatibility" between regions and constraints

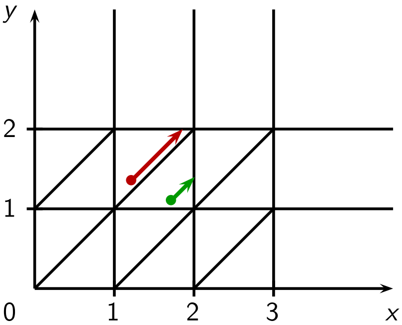
The region abstraction



Equivalence of finite index

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

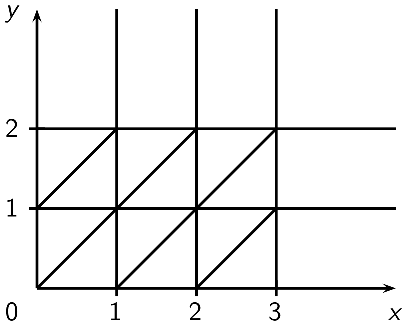
The region abstraction



Equivalence of finite index

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

The region abstraction

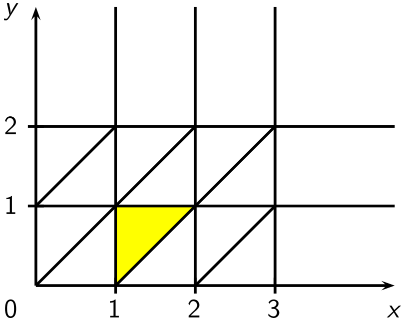


Equivalence of finite index

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

➜ a bisimulation property

The region abstraction



Equivalence of finite index

region defined by

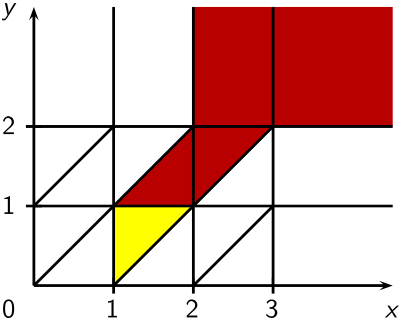
$I_x =]1;2[$, $I_y =]0;1[$

$\{x\} < \{y\}$

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

➜ a bisimulation property

The region abstraction



Equivalence of finite index

region defined by

$I_x =]1;2[,\ I_y =]0;1[$

$\{x\} < \{y\}$

successor regions

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

➜ a bisimulation property

The region automaton

timed automaton $\otimes$ region abstraction

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

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

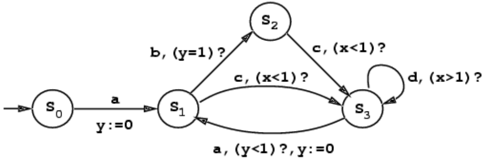
- $R'' \subseteq g$
- $[C \leftarrow 0]R'' \subseteq R'$

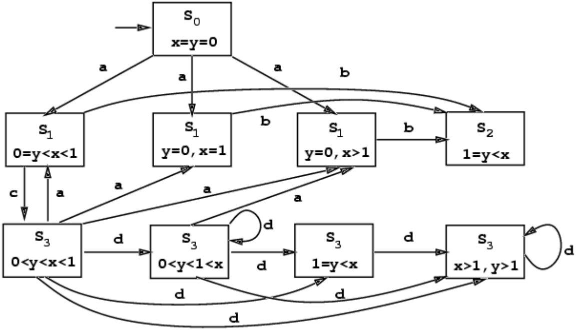
➜ **time-abstract bisimulation**

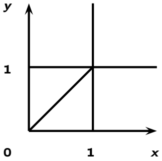
$\mathcal{L}$(reg. aut.) $=$ UNTIME($\mathcal{L}$(timed aut.))

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

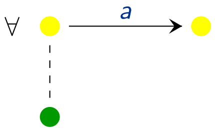
An example [AD 90's]



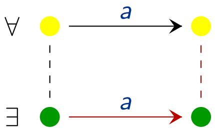




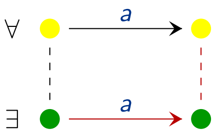
Time-abstract bisimulation

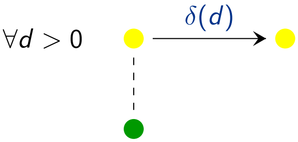


Time-abstract bisimulation

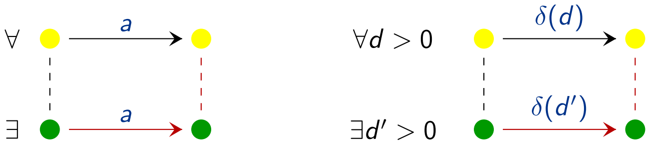


Time-abstract bisimulation

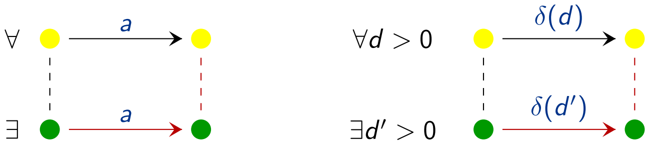




Time-abstract bisimulation

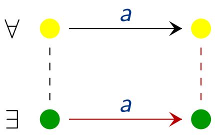


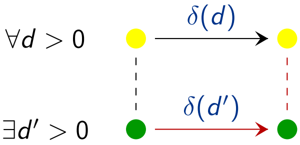
Time-abstract bisimulation



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

Time-abstract bisimulation

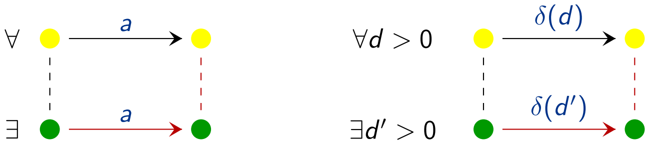




$$
\begin{tikzcd}
(\ell_0, v_0) \arrow[r, "a_1, t_1"] \arrow[d] & (\ell_1, v_1) \arrow[r, "a_2, t_2"] \arrow[d] & (\ell_2, v_2) \arrow[r, "a_3, t_3"] \arrow[d] & \dots \\
(\ell_0, R_0) \arrow[r, "a_1"] & (\ell_1, R_1) \arrow[r, "a_2"] & (\ell_2, R_2) \arrow[r, "a_3"] & \dots
\end{tikzcd}
$$

with $v_i \in R_i$ for all i.

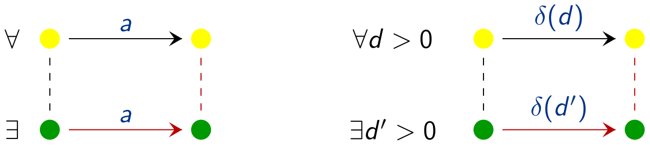
Time-abstract bisimulation



$$
\begin{array}{ccccccc}
(\ell_0, v_0) & \xrightarrow{a_1, t_1} & (\ell_1, v_1) & \xrightarrow{a_2, t_2} & (\ell_2, v_2) & \xrightarrow{a_3, t_3} & \dots \\
\updownarrow & & \updownarrow & & \updownarrow & & \\
(\ell_0, R_0) & \xrightarrow{a_1} & (\ell_1, R_1) & \xrightarrow{a_2} & (\ell_2, R_2) & \xrightarrow{a_3} & \dots
\end{array}
$$

with $v_i \in R_i$ for all i.

Time-abstract bisimulation



$$
\begin{array}{ccccccc}
(\ell_0, v_0) & \xrightarrow{a_1, t_1} & (\ell_1, v_1) & \xrightarrow{a_2, t_2} & (\ell_2, v_2) & \xrightarrow{a_3, t_3} & \ldots \\
\updownarrow & & \updownarrow & & \updownarrow & & \\
(\ell_0, R_0) & \xrightarrow{a_1} & (\ell_1, R_1) & \xrightarrow{a_2} & (\ell_2, R_2) & \xrightarrow{a_3} & \ldots
\end{array}
$$

with $v_i \in R_i$ for all i.

Remark: Real-time properties can not be checked with a time-abstract bisimulation. For TCTL, a clock associated with the formula needs to be added.

PSPACE-easiness

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

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

PSPACE-easiness

¡ 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-easiness

¡ 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-easiness

¡ 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

➜ needs polynomial space

PSPACE-easiness

¡ 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

➜ needs polynomial space

- By guessing a path: needs only to store two configurations

PSPACE-easiness

¡ 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

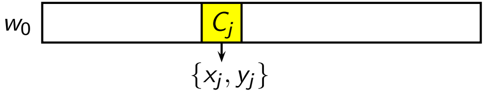
➜ needs polynomial space

- By guessing a path: needs only to store two configurations

➜ 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)

➜ proof taken in **[Aceto & Laroussinie 2002]**

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

Consequence of region automata construction

Region automata: correct finite abstraction for checking reachability/Büchi-like properties

Consequence of region automata construction

Region automata: correct finite abstraction for checking reachability/Büchi-like properties

However, everything can not be reduced to finite automata...

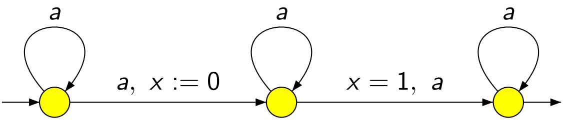
A model not far from undecidability

- Universality is undecidable **[Alur & Dill 90's]**
- Inclusion is undecidable **[Alur & Dill 90's]**
- Determinizability is undecidable **[Tripakis 2003]**
- Complementability is undecidable **[Tripakis 2003]**
- ...

A model not far from undecidability

- Universality is undecidable [Alur & Dill 90's]
- Inclusion is undecidable [Alur & Dill 90's]
- Determinizability is undecidable [Tripakis 2003]
- Complementability is undecidable [Tripakis 2003]
- ...

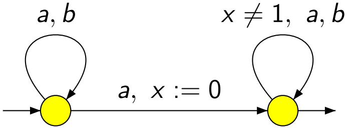
An example of non-determinizable/non-complementable timed aut.:



A model not far from undecidability

- Universality is undecidable [Alur & Dill 90's]
- Inclusion is undecidable [Alur & Dill 90's]
- Determinizability is undecidable [Tripakis 2003]
- Complementability is undecidable [Tripakis 2003]
- ...

An example of non-determinizable/non-complementable timed aut.: [Alur,Madhusudan 2004]



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!)**

Partial conclusion

➜ a timed model interesting for verification purposes

Numerous works have been (and are) devoted to:

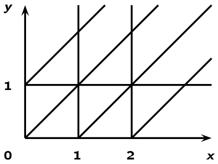
- the "theoretical" comprehension of timed automata (*cf* **[Asarin 2004]**)
- extensions of the model (to ease modelling)
  - expressiveness
  - analyzability
- algorithmic problems and implementation

Outline

Role of diagonal constraints

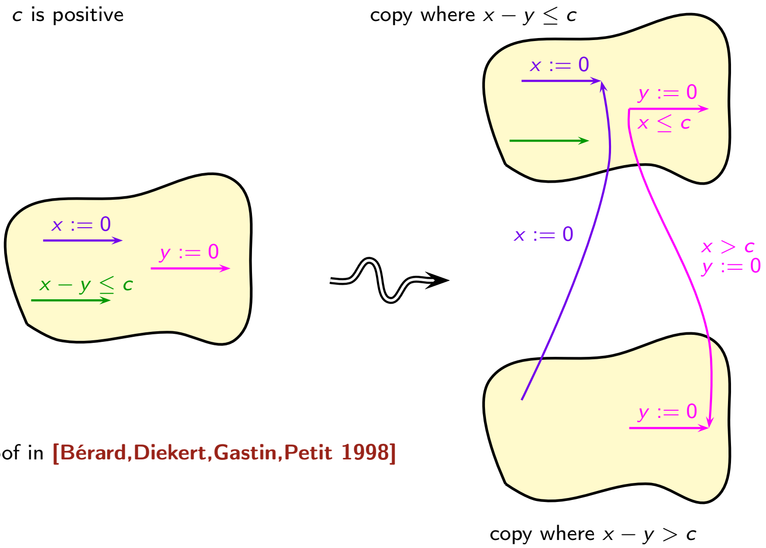
$$\boxed{x - y \sim c \quad \text{and} \quad x \sim c}$$

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



- **Expressiveness:** no additional expressive power

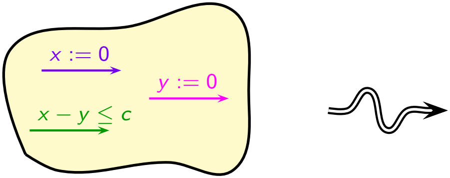
Role of diagonal constraints (cont.)



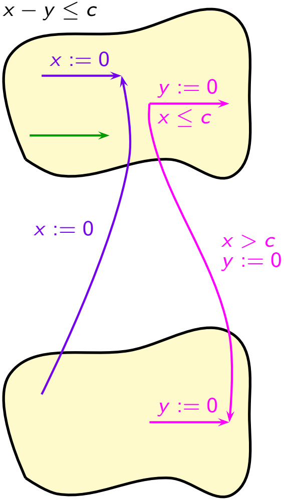
➜ proof in **[Bérard,Diekert,Gastin,Petit 1998]**

Role of diagonal constraints (cont.)

c is positive



copy where $x - y \leq c$



copy where $x - y > c$

- ➜ proof in **[Bérard,Diekert,Gastin,Petit 1998]**
- ➜ exponential blowup unavoidable in general **[Bouyer,Chevalier 2005]**

Adding silent actions

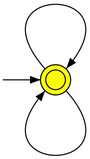
$$\xrightarrow{g,\ \varepsilon,\ C := 0}$$

[Bérard,Diekert,Gastin,Petit 1998]

- **Decidability:** yes

  (actions have no influence on region automaton construction)

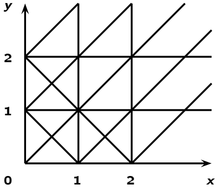
- **Expressiveness:** strictly more expressive!



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

$\boxed{x + y \sim c \quad \text{and} \quad x \sim c}$ **[Bérard,Dufourd 2000]**

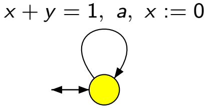
- **Decidability:** - for two clocks, decidable using the abstraction



  - for four clocks (or more), undecidable!

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

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



The two-counter machine

Definition

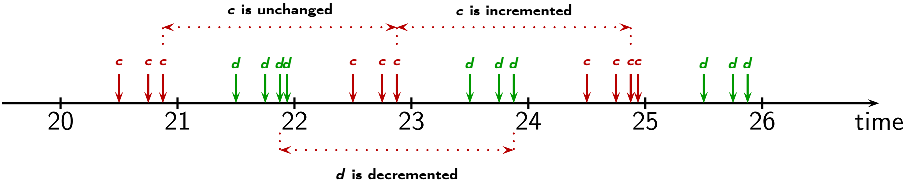
A two-counter machine is a finite set of instructions over two counters (x and y):

- Incrementation:
  (p): $x := x + 1$; goto (q)
- Decrementation:
  (p): if $x > 0$ then $x := x - 1$; goto (q) else goto (r)

Theorem [Minsky 67]

The halting problem for two counter machines is undecidable.

Undecidability proof



➜ simulation of
- decrementation of a counter
- incrementation of a counter

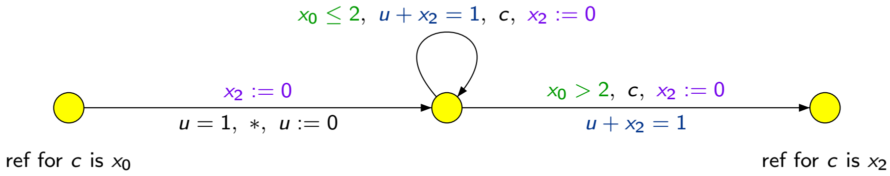
We will use 4 clocks:
- u, "tic" clock (each time unit)
- x_0, x_1, x_2: reference clocks for the two counters

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

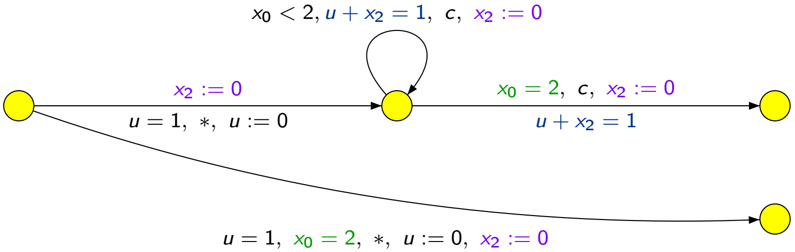
[Bérard,Dufourd 2000]

Undecidability proof (cont.)

- **Incrementation of counter c:**

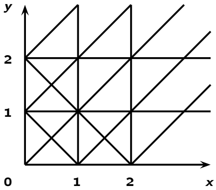


- **Decrementation of counter c:**



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

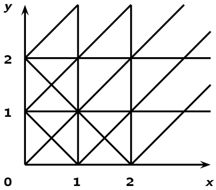
- Two clocks: decidable using the abstraction



- Four clocks (or more): undecidable!

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

- Two clocks: decidable using the abstraction



- **Three clocks: open question**

  We only know that the coarsest time-abstract bisimulation respecting these constraints is infinite. **[Robin 2004]**

- 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)

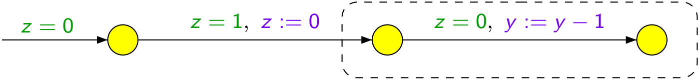
Adding new operations on clocks

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

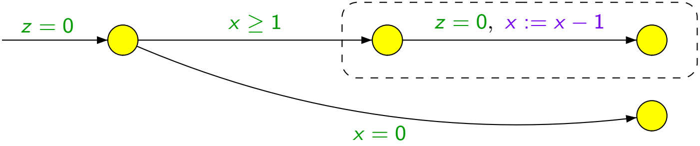
- The general model is undecidable.
  (simulation of a two-counter machine)

- Only decrementation also leads to undecidability

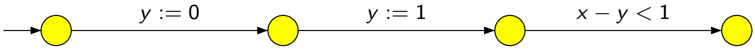
  - **Incrementation of counter x**



  - **Decrementation of counter x**



Decidability



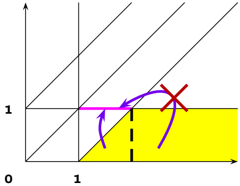


image by $y := 1$

➜ the bisimulation property is not met

The classical region automaton construction is not correct.

Decidability (cont.)

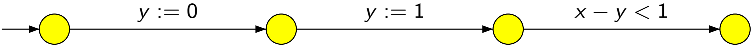
$$\begin{aligned} \mathcal{A} &\leadsto \text{Diophantine linear inequations system} \\ &\leadsto \text{is there a solution?} \\ &\leadsto \text{if yes, belongs to a decidable class} \end{aligned}$$

Examples:

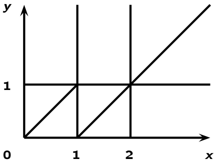
- constraint $x \sim c$ — $c \leq \max_x$
- constraint $x - y \sim c$ — $c \leq \max_{x,y}$
- update $x :\sim y + c$ — $\max_x \leq \max_y + c$
  and for each clock z, $\max_{x,z} \geq \max_{y,z} + c$, $\max_{z,x} \geq \max_{z,y} - c$
- update $x :< c$ — $c \leq \max_x$
  and for each clock z, $\max_z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

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

Decidability (cont.)



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

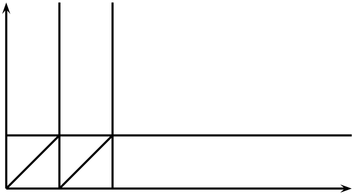


The bisimulation property is met.

What's wrong when undecidable?

Decrementation $x := x - 1$

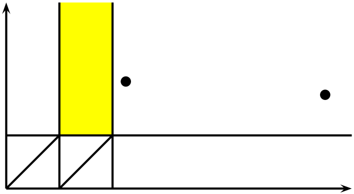
$\max_x \leq \max_x - 1$



What's wrong when undecidable?

Decrementation $x := x - 1$

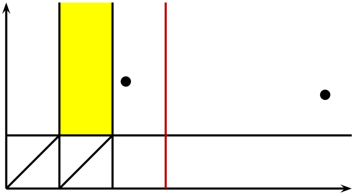
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

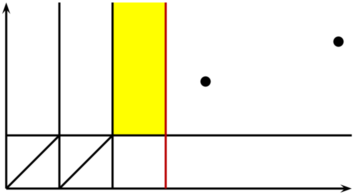
$\max_x \leq \max_x - 1$



What's wrong when undecidable?

Decrementation $x := x - 1$

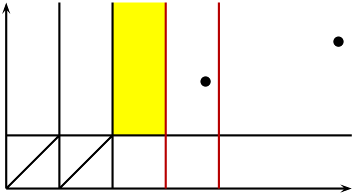
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

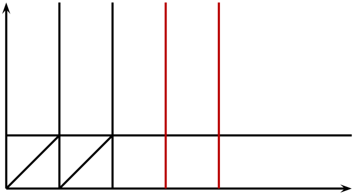
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

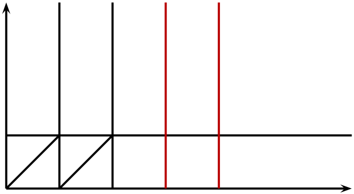
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

$\mathsf{max}_x \leq \mathsf{max}_x - 1$



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>

[Bouyer,Dufourd,Fleury,Petit 2000]

Other extensions which have been considered

- New operations on clocks **[Bouyer, Dufourd, Fleury, Petit 2004]**

$$x := y + c,\ x :< c,\ x :> c,\ \text{etc...}$$

- Alternation **[Lasota, Walukiewicz 2005]** **[Ouaknine, Worrell 2005]**
  - One-clock alternating timed automata are decidable.
  - n-clock alternating timed automata are undecidable ($n \geq 2$).

- Slopes of variables: "Linear hybrid automata" **[Henzinger 1996]** **[Henzinger,Kopke,Puri,Varaiya 98]**
  - Almost everything is undecidable.
  - The class of LHA with clocks and only one variable having possibly two slopes $k_1 \neq k_2$ is undecidable.
  - The class of *stopwatch* automata is undecidable.
  - One of the "largest" classes of LHA which are decidable is the class of initialized rectangular automata

Outline

Notice

The region automaton is not used for implementation:

- suffers from a combinatorics explosion
  (the number of regions is exponential in the number of clocks)
- no really adapted data structure

Notice

The region automaton is not used for implementation:

- suffers from a combinatorics explosion
  (the number of regions is exponential in the number of clocks)
- no really adapted data structure

Algorithms for "minimizing" the region automaton have been proposed...
[Alur & Co 1992] [Tripakis,Yovine 2001]

Notice

The region automaton is not used for implementation:

- suffers from a combinatorics explosion
  (the number of regions is exponential in the number of clocks)
- no really adapted data structure

Algorithms for "minimizing" the region automaton have been proposed...
[Alur & Co 1992] [Tripakis,Yovine 2001]

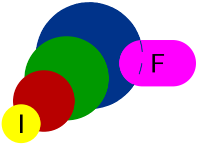
...but **on-the-fly technics** are prefered.

Reachability analysis

- **forward analysis algorithm:**
  compute the successors of initial configurations

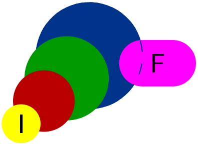
Reachability analysis

- **forward analysis algorithm:**
  compute the successors of initial configurations

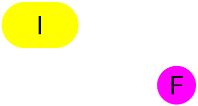


Reachability analysis

- **forward analysis algorithm:**
  compute the successors of initial configurations

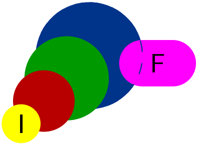


- **backward analysis algorithm:**
  compute the predecessors of final configurations

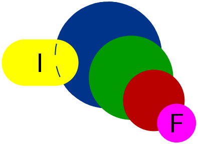


Reachability analysis

- **forward analysis algorithm:**
  compute the successors of initial configurations



- **backward analysis algorithm:**
  compute the predecessors of final configurations



Note on the backward analysis of TA

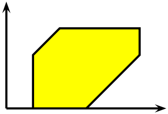
$$\ell \xrightarrow{g,\ a,\ C := 0} \ell'$$

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

Note on the backward analysis of TA

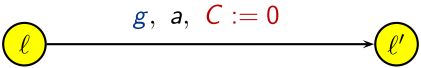
$$\ell \xrightarrow{\;g,\; a,\; C := 0\;} \ell'$$

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

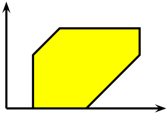


Z

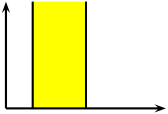
Note on the backward analysis of TA



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

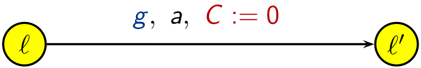


Z

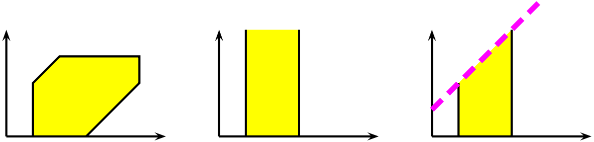


$[C \leftarrow 0]^{-1}(Z \cap (C = 0))$

Note on the backward analysis of TA

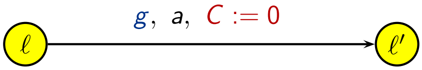


$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$ $\qquad\qquad$ Z

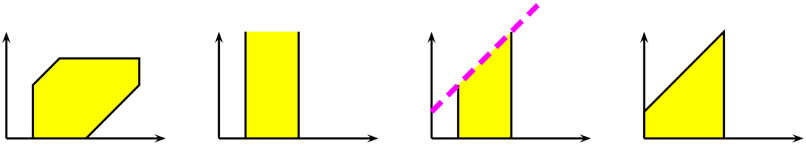


Z $\qquad$ $[C \leftarrow 0]^{-1}(Z \cap (C = 0))$

Note on the backward analysis of TA

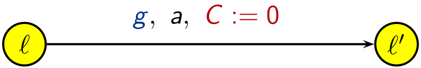


$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$ $\qquad Z$

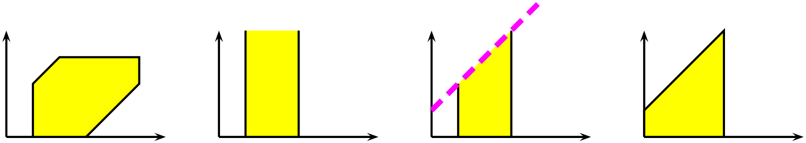


$Z \qquad [C \leftarrow 0]^{-1}(Z \cap (C = 0)) \qquad \overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$

Note on the backward analysis of TA



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



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

The exact backward computation terminates and 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. $v + t \in R$)
- $v' \equiv_{\mathbf{reg.}} v$

There exists t' s.t. $v' + t' \equiv_{\mathbf{reg.}} v + t$, 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.

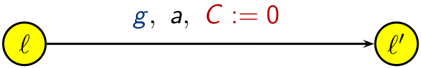
Because of the bisimulation property, we get that:

"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 analysis of timed automata



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

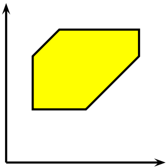
A zone is a set of valuations defined by a clock constraint

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

Forward analysis of timed automata

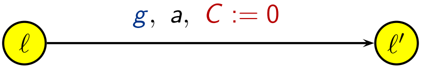
$$\ell \xrightarrow{\quad g,\ a,\ C := 0 \quad} \ell'$$

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

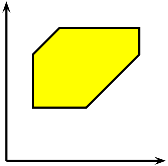


Z

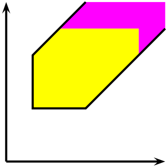
Forward analysis of timed automata



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

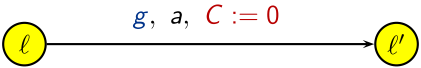


Z

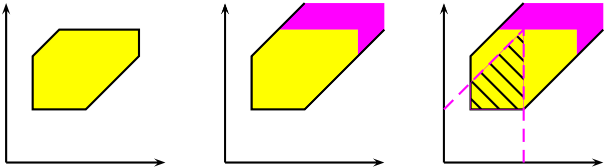


$\overrightarrow{Z}$

Forward analysis of timed automata

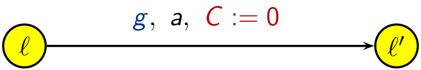


zones $\quad Z \quad [C \leftarrow 0](\overrightarrow{Z} \cap g)$

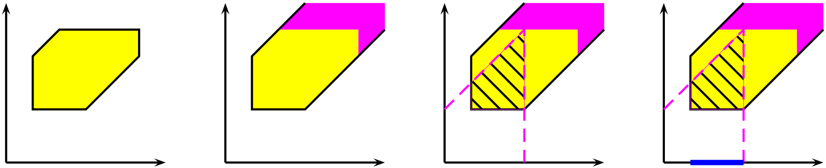


$Z \qquad \overrightarrow{Z} \qquad \overrightarrow{Z} \cap g$

Forward analysis of timed automata

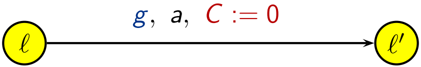


zones Z $[C \leftarrow 0](\overrightarrow{Z} \cap g)$

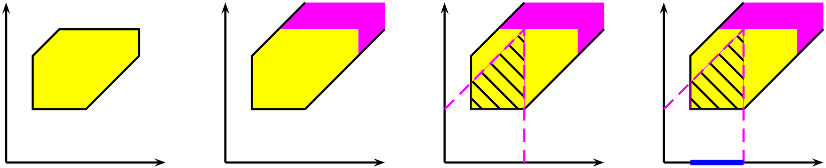


Z $\overrightarrow{Z}$ $\overrightarrow{Z} \cap g$ $[y \leftarrow 0](\overrightarrow{Z} \cap g)$

Forward analysis of timed automata



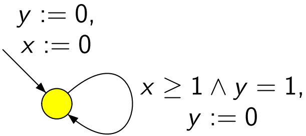
zones $\quad Z \quad [C \leftarrow 0](\overrightarrow{Z} \cap g)$

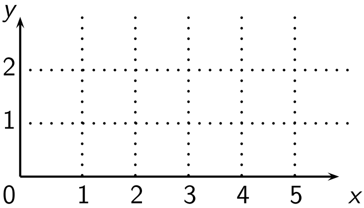


$Z \qquad \overrightarrow{Z} \qquad \overrightarrow{Z} \cap g \qquad [y \leftarrow 0](\overrightarrow{Z} \cap g)$

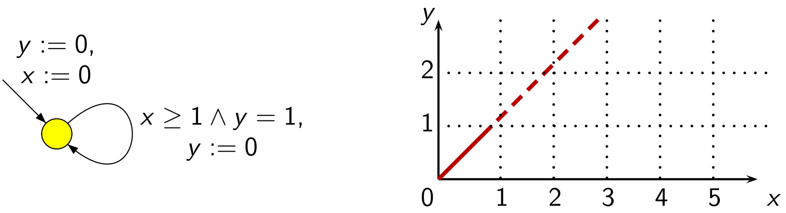
➜ a termination problem

Non termination of the forward analysis

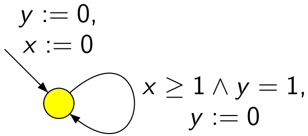


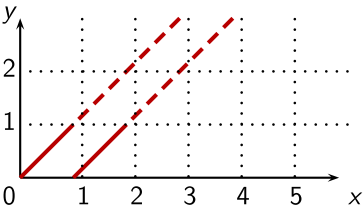


Non termination of the forward analysis

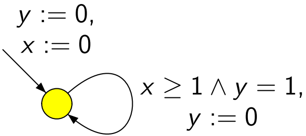


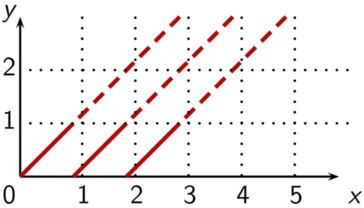
Non termination of the forward analysis



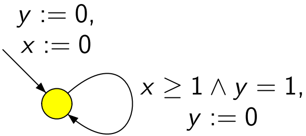


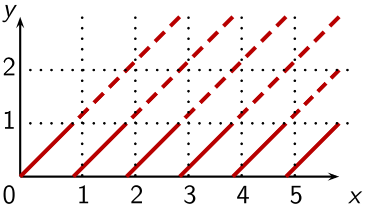
Non termination of the forward analysis





Non termination of the forward analysis





➜ an infinite number of steps...

The DBM data structure

DBM (Difference Bound Matrice) data structure

[Berthomieu, Menasche 1983] [Dill 1989]

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

$$\begin{array}{c|ccc} & x_0 & x_1 & x_2 \\ \hline x_0 & +\infty & -3 & +\infty \\ x_1 & +\infty & +\infty & 4 \\ x_2 & 5 & +\infty & +\infty \end{array}$$

The DBM data structure

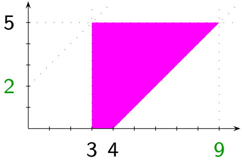
DBM (Difference Bound Matrice) data structure

[Berthomieu, Menasche 1983] [Dill 1989]

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

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

- Existence of a normal form



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

The DBM data structure

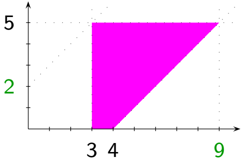
DBM (Difference Bound Matrice) data structure

[Berthomieu, Menasche 1983] [Dill 1989]

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

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

- Existence of a normal form



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

- All previous operations on zones can be computed using DBMs

The extrapolation operator

Fix an integer k ("$*$" represents an integer between $-k$ and $+k$)

$$\begin{pmatrix} * & \boxed{>k} & * \\ * & * & * \\ \boxed{<-k} & * & * \end{pmatrix} \rightsquigarrow \begin{pmatrix} * & \boxed{+\infty} & * \\ * & * & * \\ \boxed{-k} & * & * \end{pmatrix}$$

- "intuitively", erase non-relevant constraints

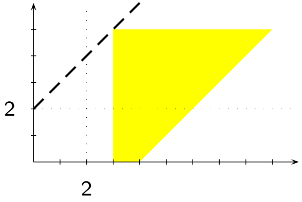
➜ ensures termination

The extrapolation operator

Fix an integer k ("$*$" represents an integer between $-k$ and $+k$)

$$\begin{pmatrix} * & \boxed{>k} & * \\ * & * & * \\ \boxed{<-k} & * & * \end{pmatrix} \rightsquigarrow \begin{pmatrix} * & \boxed{+\infty} & * \\ * & * & * \\ \boxed{-k} & * & * \end{pmatrix}$$

- "intuitively", erase non-relevant constraints



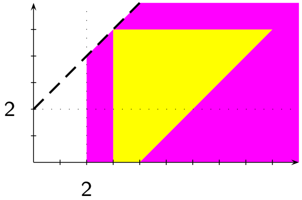
➜ ensures termination

The extrapolation operator

Fix an integer k ("$*$" represents an integer between $-k$ and $+k$)

$$\begin{pmatrix} * & \boxed{>k} & * \\ * & * & * \\ \boxed{<-k} & * & * \end{pmatrix} \rightsquigarrow \begin{pmatrix} * & \boxed{+\infty} & * \\ * & * & * \\ \boxed{-k} & * & * \end{pmatrix}$$

- "intuitively", erase non-relevant constraints



➜ ensures termination

Classical algorithm, focus on correctness

Take k the maximal constant appearing in the constraints of the automaton.

Classical algorithm, focus on correctness

Take k the maximal constant appearing in the constraints of the automaton.

Theorem

This algorithm is correct for diagonal-free timed automata.

Classical algorithm, focus on correctness

Take k the maximal constant appearing in the constraints of the automaton.

Theorem

This algorithm is correct for diagonal-free timed automata.

However, this theorem does not extend to timed automata using diagonal clock constraints...

A counter-example

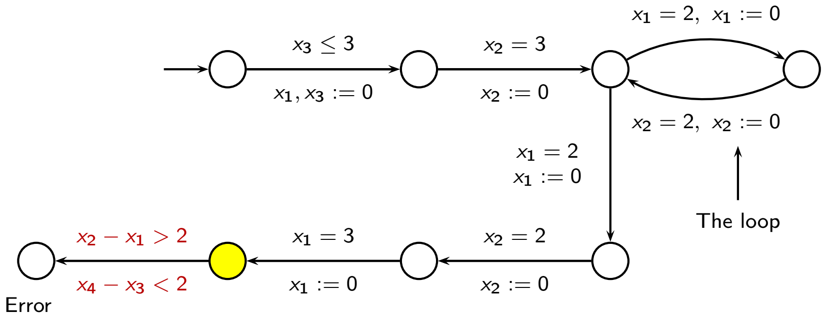
- Implemented in numerous tools:
  - Uppaal, `http://www.uppaal.com/`
  - Kronos, `http://www-verimag.imag.fr/TEMPORISE/kronos/`
  - ...
- Successfully used on many real-life examples since ten years.

Outline

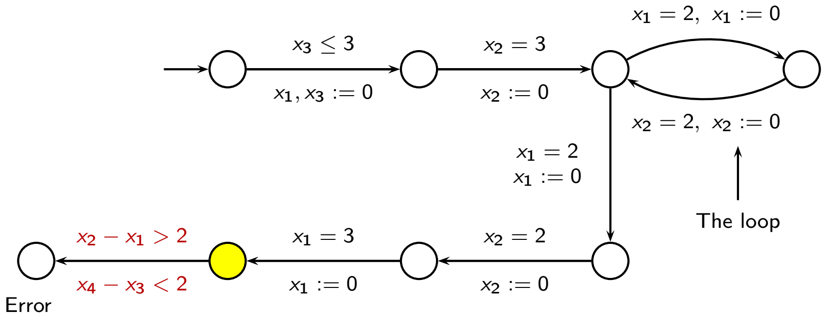
Conclusion & further work

- Decidability is quite well understood.
- There is still some progress which is done for the verification of timed automata. *(see Gerd's talk)*
- Some other current challenges:
  - controller synthesis
  - implementability issues (program synthesis) *(remember Jean-François' talk)*
  - optimal computations
  - ...

A problematic automaton

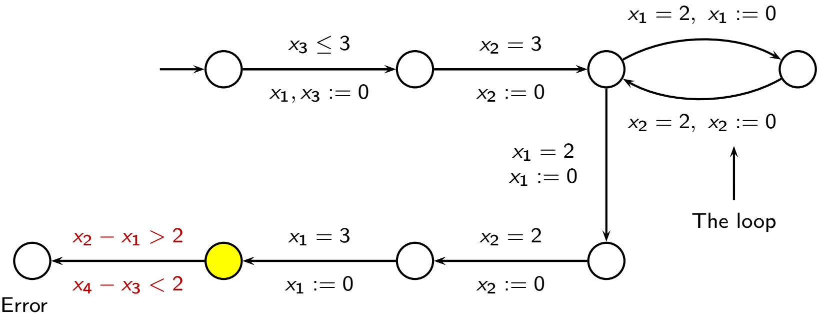


A problematic automaton

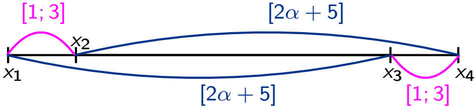


$$\begin{cases} v(x_1) = 0 \\ v(x_2) = d \\ v(x_3) = 2\alpha + 5 \\ v(x_4) = 2\alpha + 5 + d \end{cases}$$

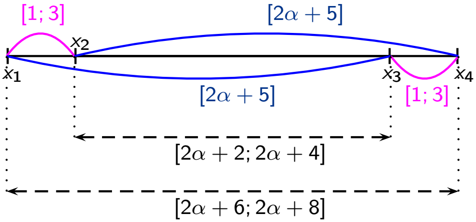
A problematic automaton



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

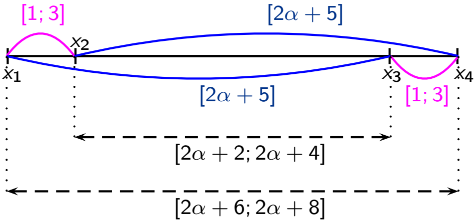


The problematic zone



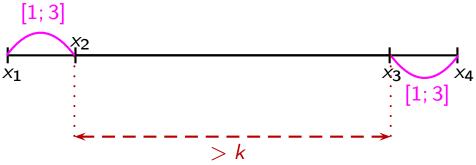
implies $x_1 - x_2 = x_3 - x_4$.

The problematic zone



implies $x_1 - x_2 = x_3 - x_4$.

If α is sufficiently large, after extrapolation:



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

Back