Real-Time and Hybrid Systems

Patricia Bouyer

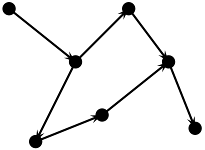
LSV – CNRS & ENS de Cachan

Winter School MOVEP'04

Model-checking

Does the system satisfy the property?

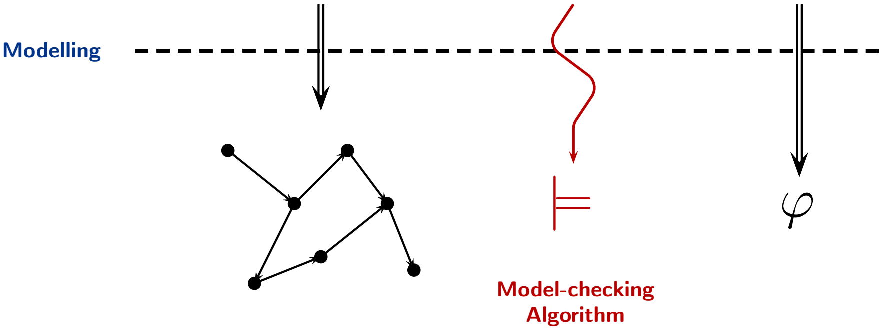
Modelling



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 & bibliography

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: $(a, 1.28).(b, 3.1).(c, 3.98)(a, 6.13)$

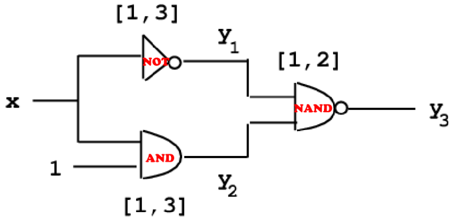
A case for dense-time

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

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

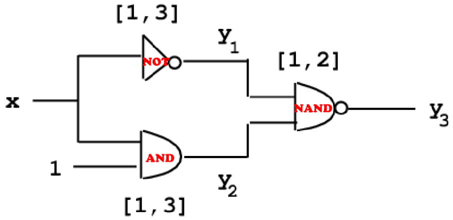
A digital circuit [Alur 91]

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



A digital circuit [Alur 91]

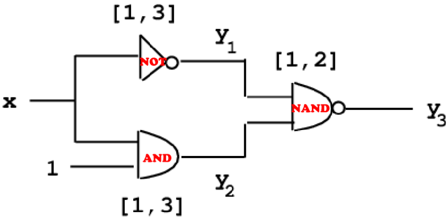
Discussion in the context of reachability problems for asynchronous digital circuits [Brzozowski, Seger 1991]



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]

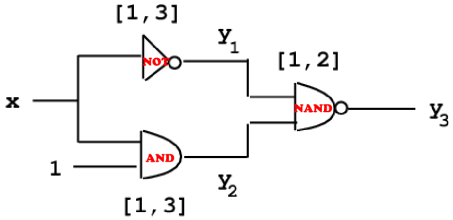


Start with x=0 and y=[101] (stable configuration)

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)

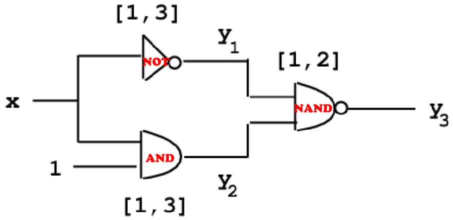
The input x changes to 1. The corresponding stable state is y=[011]

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)

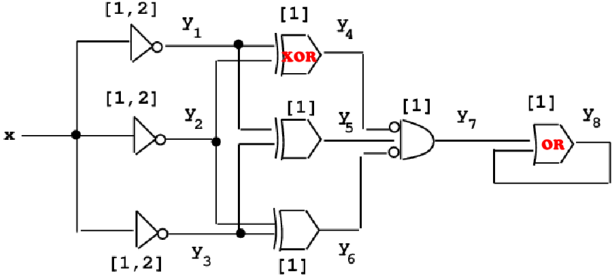
The input x changes to 1. The corresponding stable state is y=[011]

However, many possible behaviours, e.g.

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

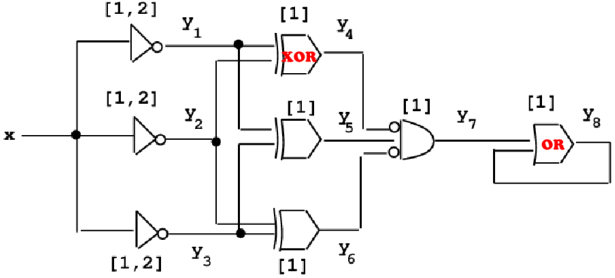
Reachable configurations: {[101], [111], [110], [010], [011], [001]}

Is discretizing sufficient? An example [Alur 91]



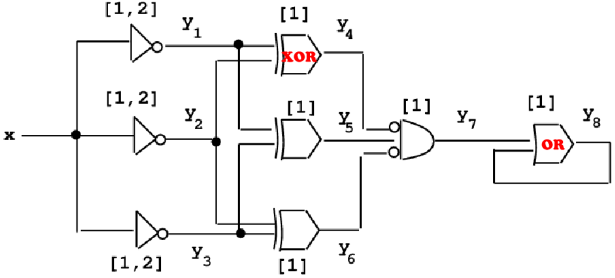
- This digital circuit **is not** 1-discretizable.

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)

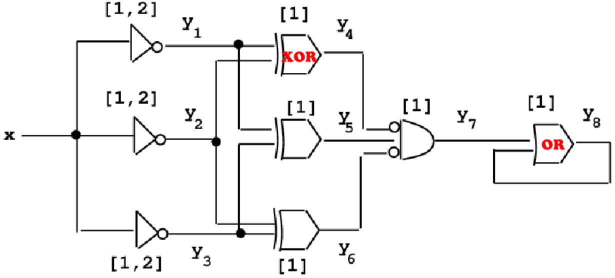
Is discretizing sufficient? An example [Alur 91]



- 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]

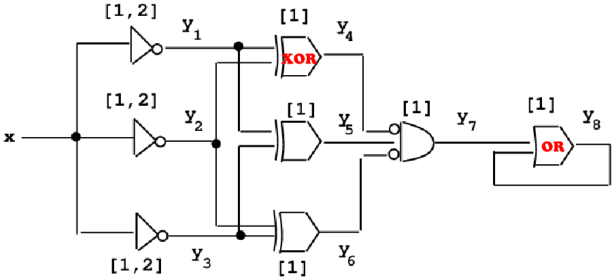


- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, x is set to 1)

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

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

Is discretizing sufficient? An example [Alur 91]



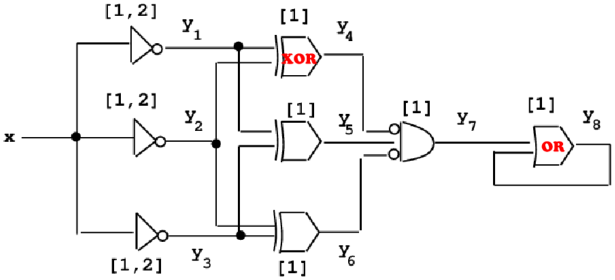
- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, x is set to 1)

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

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

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

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)

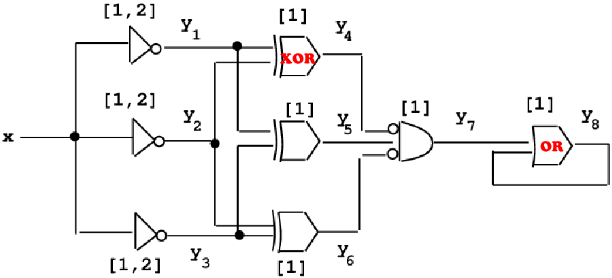
$[11100000] \xrightarrow[1]{y_1} [01100000] \xrightarrow[1.5]{y_2} [00100000] \xrightarrow[2]{y_3, y_5} [00001000] \xrightarrow[3]{y_5, y_7} [00000010] \xrightarrow[4]{y_7, y_8} [00000001]$

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

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

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

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?

[Brzozowski Seger 1991]

Theorem: 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?

[Brzozowski Seger 1991]

Theorem: 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?

[Brzozowski Seger 1991]

Theorem: 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

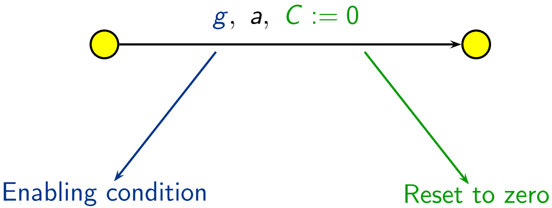
Further counter-example: there exist systems for which no granularity exists

(see later)

Outline

Timed automata [Alur & Dill 90's]

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



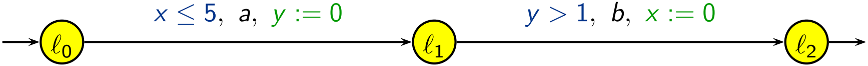
- An enabling condition (or guard) is:

$$g \ ::= \ x \sim c \ \mid \ g \wedge g$$

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

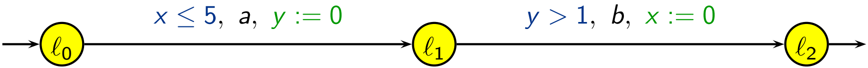
Timed automata (example)

x, y : clocks



Timed automata (example)

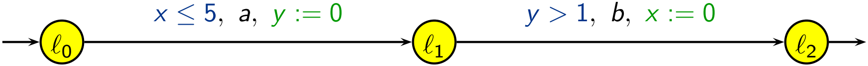
x, y : clocks



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

Timed automata (example)

x, y : clocks

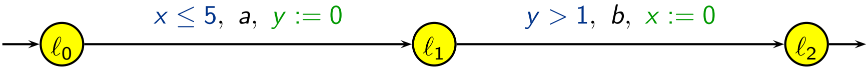


$$
\begin{array}{ccccccccc}
 & \ell_0 & \xrightarrow{\delta(4.1)} & \ell_0 & \xrightarrow{a} & \ell_1 & \xrightarrow{\delta(1.4)} & \ell_1 & \xrightarrow{b} & \ell_2 \\
x & 0 & & 4.1 & & 4.1 & & 5.5 & & 0 \\
y & 0 & & 4.1 & & 0 & & 1.4 & & 1.4
\end{array}
$$

(clock) valuation

Timed automata (example)

x, y : clocks



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

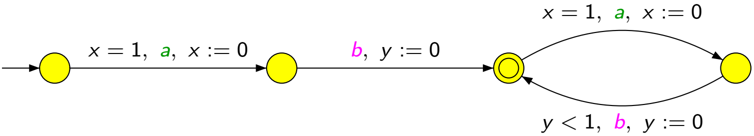
(clock) valuation

➜ timed word $(a, 4.1)(b, 5.5)$

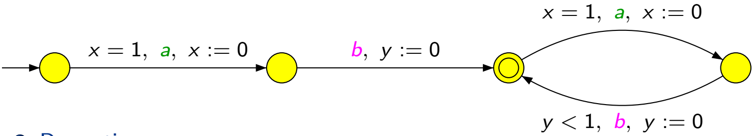
Timed automata semantics

- $\mathcal{A} = (\Sigma, L, X, \longrightarrow)$ 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

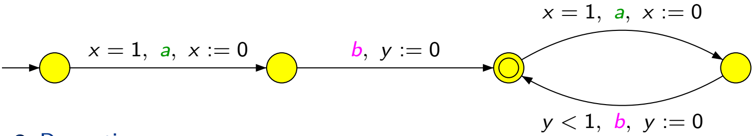


Discrete *vs* dense-time semantics



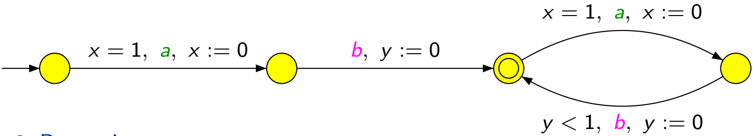
- Dense-time:
  $L_{dense} = \{((ab)^\omega, \tau) \mid \forall i,\ \tau_{2i-1} = i \text{ and } \tau_{2i} - \tau_{2i-1} > \tau_{2i+2} - \tau_{2i+1}\}$

Discrete *vs* dense-time semantics



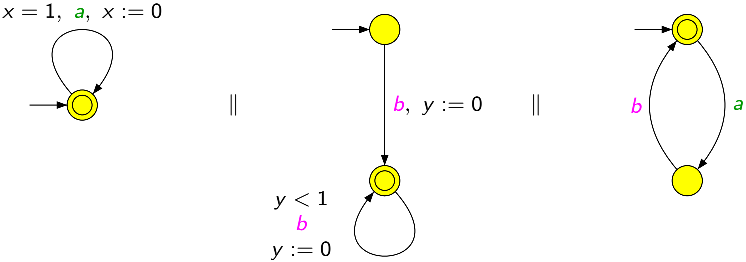
- Dense-time:
  $L_{dense} = \{((ab)^\omega, \tau) \mid \forall i,\ \tau_{2i-1} = i \text{ and } \tau_{2i} - \tau_{2i-1} > \tau_{2i+2} - \tau_{2i+1}\}$
- Discrete-time: $L_{discrete} = \emptyset$

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: $L_{discrete} = \emptyset$

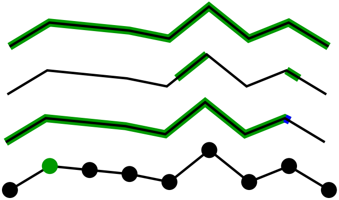


Classical verification problems

- 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 »

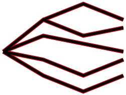
$F\phi$ « Eventually »

$\phi U \phi'$ « Until »

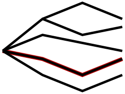
$X\phi$ « Next »

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:

$$AG(problem \Rightarrow AF\ alarm)$$

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

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

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:

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

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:

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

How can we express:

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

- Temporal logics with **subscripts**.

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

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$AG(problem \Rightarrow AF\ alarm)$$

How can we express:

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

- Temporal logics with **subscripts**.

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

- Temporal logics with **clocks**.

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

Adding time to temporal logics

Classical temporal logics allow us to express that

"any problem is followed by an alarm"

With CTL:

$$AG(problem \Rightarrow AF\ alarm)$$

How can we express:

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

- Temporal logics with **subscripts**.

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

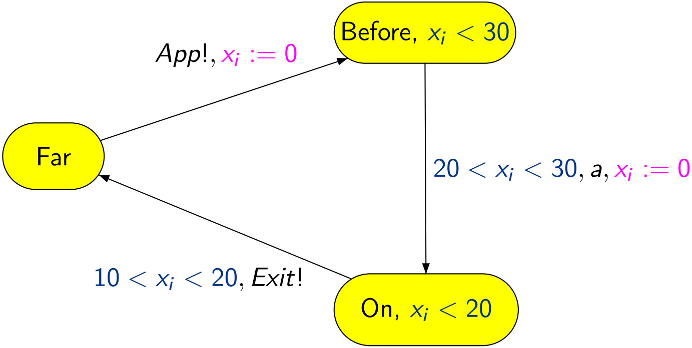
- Temporal logics with **clocks**.

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

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

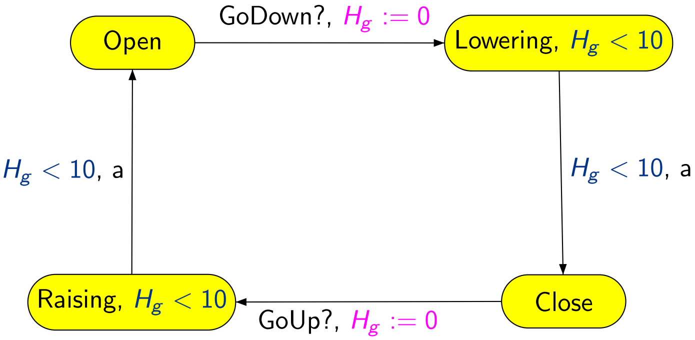
The train crossing example (1)

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



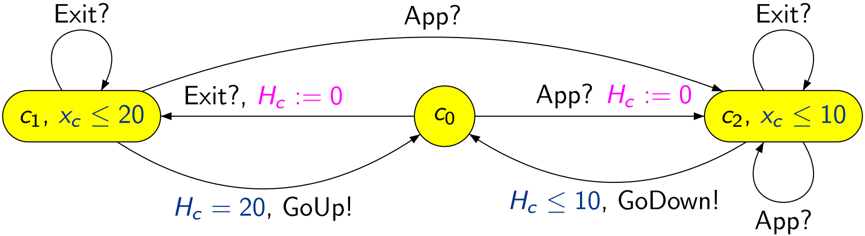
The train crossing example (2)

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?

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

The train crossing example (5)

Some properties one could check:

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

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

- 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?

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

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

$$\neg EF(\mathrm{gate.Close} \wedge (\mathrm{gate.Close}\ U_{>5\ min}\ \neg \mathrm{gate.Close}))$$

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: The emptiness problem for timed automata is decidable.
It is PSPACE-complete. **[Alur & Dill 1990's]**

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: The emptiness problem for timed automata is decidable.
It is PSPACE-complete. **[Alur & Dill 1990's]**

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

Verification

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

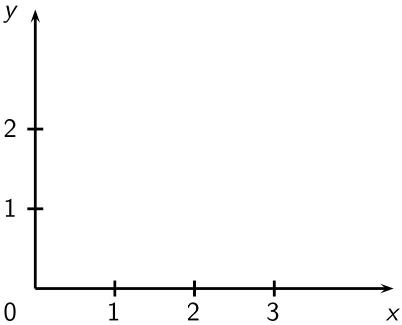
- **Problem:** the set of configurations is infinite
  ➜ classical methods can not be applied

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

Theorem: The emptiness problem for timed automata is decidable.
It is PSPACE-complete. **[Alur & Dill 1990's]**

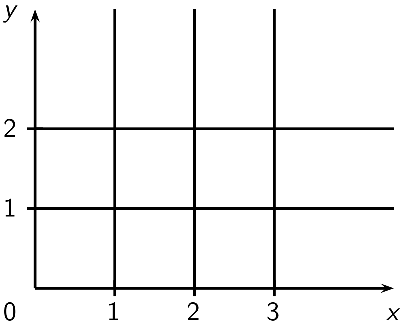
Method: construct a finite abstraction

The region abstraction



Equivalence of finite index

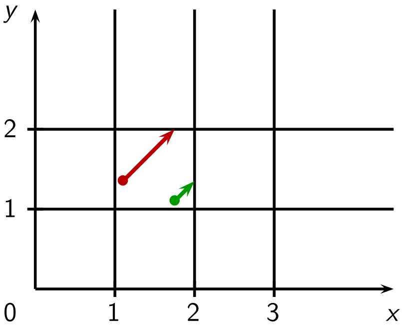
The region abstraction



Equivalence of finite index

- "compatibility" between regions and constraints

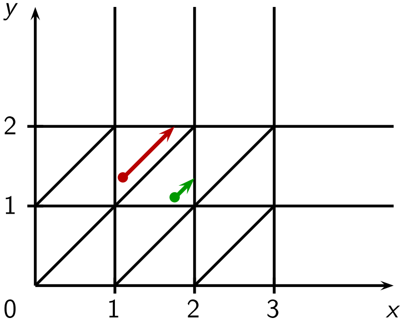
The region abstraction



Equivalence of finite index

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

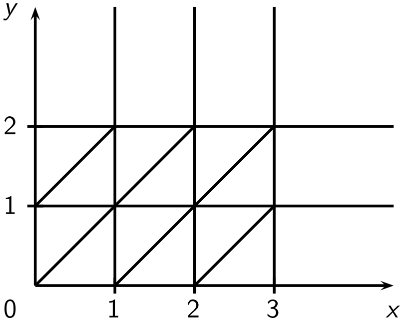
The region abstraction



Equivalence of finite index

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

The region abstraction

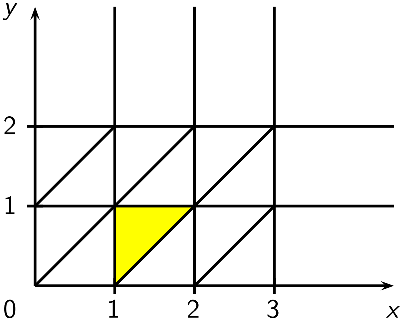


Equivalence of finite index

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

➜ a bisimulation property

The region abstraction



Equivalence of finite index

region defined by

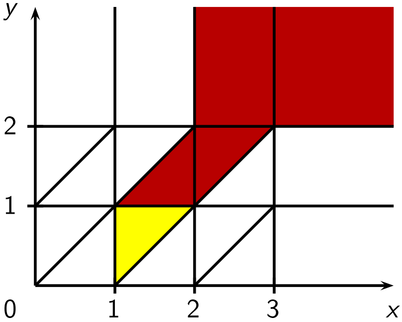
$I_x =]1;2[$, $I_y =]0;1[$

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

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

➜ a bisimulation property

The region abstraction



Equivalence of finite index

region defined by

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

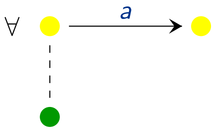
$\{x\} < \{y\}$

successor regions

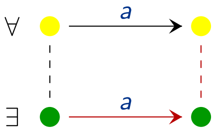
- "compatibility" between regions and constraints
- "compatibility" between regions and time elapsing

➜ a bisimulation property

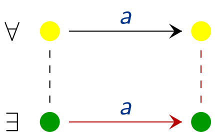
Time-abstract bisimulation

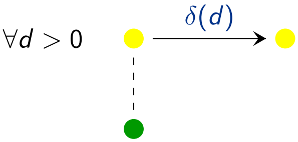


Time-abstract bisimulation

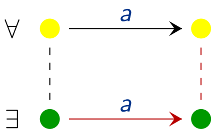


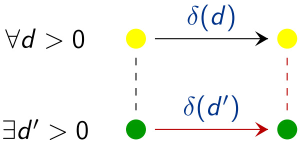
Time-abstract bisimulation



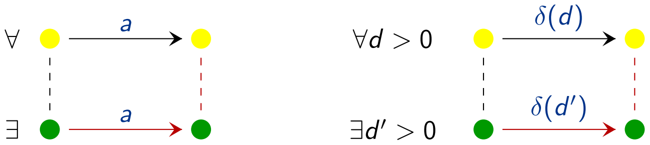


Time-abstract bisimulation



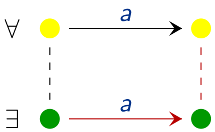


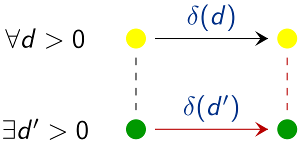
Time-abstract bisimulation



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

Time-abstract bisimulation

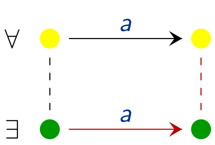


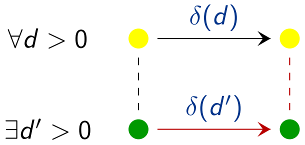


$$\begin{tikzcd}
(\ell_0, v_0) \arrow[r, "a_1, t_1"] \arrow[d] & (\ell_1, v_1) \arrow[r, "a_2, t_2"] \arrow[d] & (\ell_2, v_2) \arrow[r, "a_3, t_3"] \arrow[d] & \dots \\
(\ell_0, R_0) \arrow[r, "a_1"] & (\ell_1, R_1) \arrow[r, "a_2"] & (\ell_2, R_2) \arrow[r, "a_3"] & \dots
\end{tikzcd}$$

with $v_i \in R_i$ for all i.

Time-abstract bisimulation

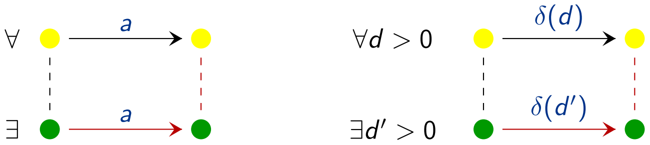




$$\begin{array}{ccccccc}
(\ell_0, v_0) & \xrightarrow{a_1, t_1} & (\ell_1, v_1) & \xrightarrow{a_2, t_2} & (\ell_2, v_2) & \xrightarrow{a_3, t_3} & \dots \\
\updownarrow & & \updownarrow & & \updownarrow & & \\
(\ell_0, R_0) & \xrightarrow{a_1} & (\ell_1, R_1) & \xrightarrow{a_2} & (\ell_2, R_2) & \xrightarrow{a_3}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

with $v_i \in R_i$ for all i.

Time-abstract bisimulation



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

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.

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.

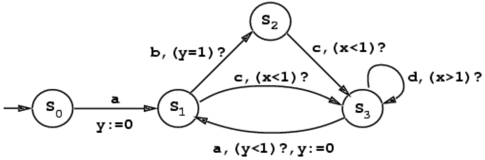
- $R'' \subseteq g$
- $[C \leftarrow 0]R'' \subseteq R'$

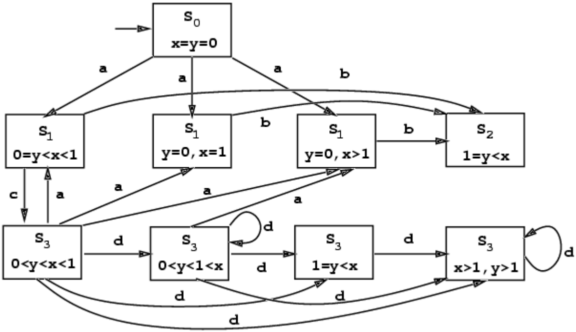
➜ **time-abstract bisimulation**

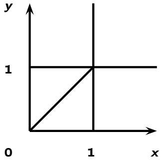
$\mathcal{L}$**(reg. aut.) = UNTIME(**$\mathcal{L}$**(timed aut.))**

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

An example [AD 90's]







PSPACE-easyness

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

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

PSPACE-easyness

¡ 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-easyness

¡ 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-easyness

¡ 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-easyness

¡ 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-easyness

¡ 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

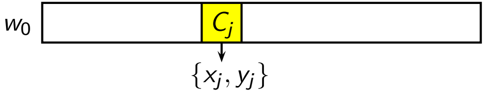
➜ needs polynomial space

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

➜ in NPSPACE, thus in PSPACE

PSPACE-hardness

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



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...

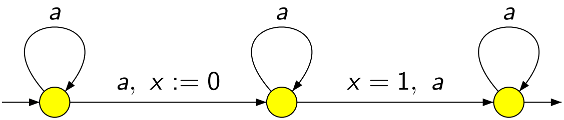
A model not far from undecidability

- 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]
- ...

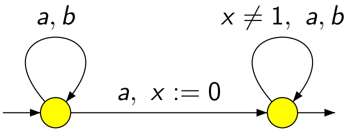
An example of non-determinizable/non-complementable timed aut.:



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:

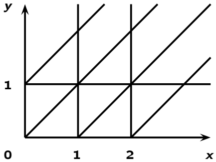
- the "theoretical" comprehension of timed automata (*cf* **[Asarin 2004]**)
- extensions of the model (to ease modelling)
  - expressiveness
  - analyzability
- algorithmic problems and implementation

Outline

Role of diagonal constraints

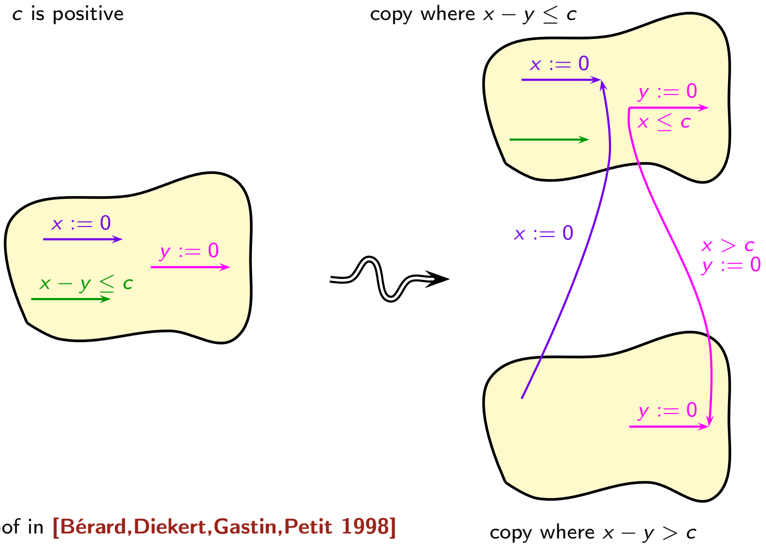
$$\boxed{x - y \sim c \quad \text{and} \quad x \sim c}$$

- **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.)

Open question: is this construction "optimal"?

In the sense that timed automata with diagonal constraints are exponentially more concise than diagonal-free timed automata.

Adding silent actions

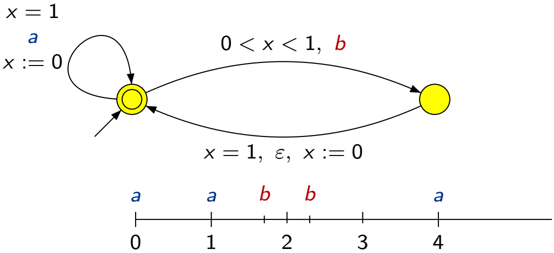
$$\xrightarrow{g, \varepsilon, C := 0}$$

[Bérard,Diekert,Gastin,Petit 1998]

- **Decidability:** yes

  (actions have no influence on region automaton construction)

- **Expressiveness:** strictly more expressive!



Adding silent actions

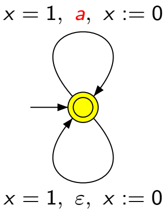
$$\xrightarrow{g,\,\varepsilon,\,C:=0}$$

[Bérard,Diekert,Gastin,Petit 1998]

- **Decidability:** yes

  (actions have no influence on region automaton construction)

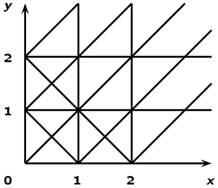
- **Expressiveness:** strictly more expressive!



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

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

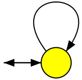
- **Decidability:** - for two clocks, decidable using the abstraction



  - for four clocks (or more), undecidable!

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

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



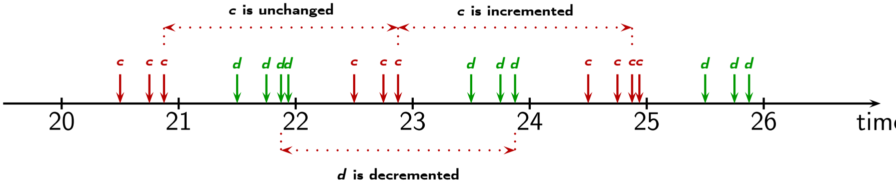
The two-counter machine

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:

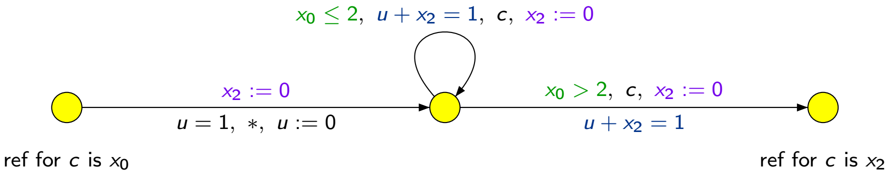
- u, "tic" clock (each time unit)
- x_0, x_1, x_2: reference clocks for the two counters

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

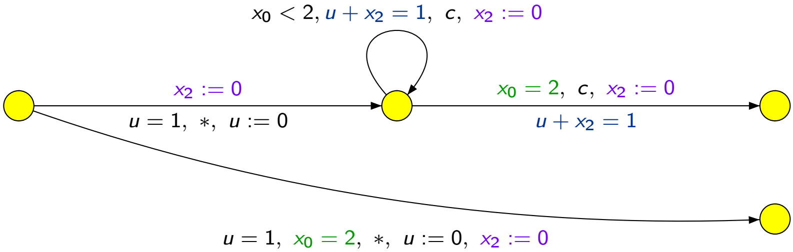
[Bérard,Dufourd 2000]

Undecidability proof (cont.)

- **Incrementation of counter c:**

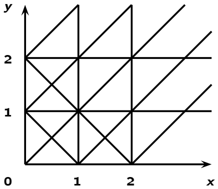


- **Decrementation of counter c:**



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

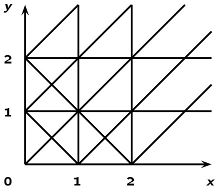
- Two clocks: decidable using the abstraction



- Four clocks (or more): undecidable!

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

- Two clocks: decidable using the abstraction



- Three clocks: **open question**

- 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)

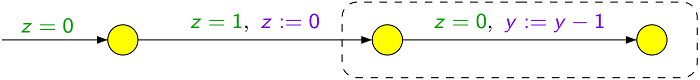
Adding new operations on clocks

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

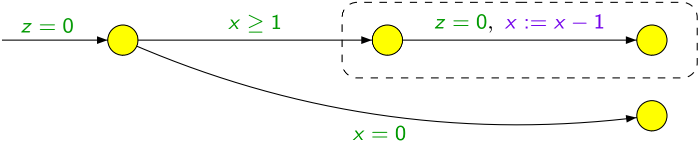
- The general model is undecidable.
  (simulation of a two-counter machine)

- Only decrementation also leads to undecidability

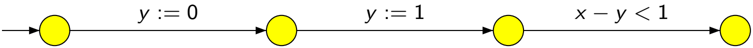
  - **Incrementation of counter x**



  - **Decrementation of counter x**



Decidability



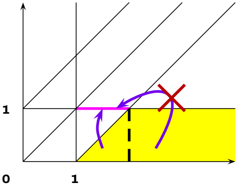


image by $y := 1$

➜ the bisimulation property is not met

The classical region automaton construction is not correct.

Decidability (cont.)

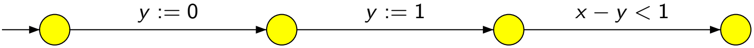
$\mathcal{A}$ $\rightsquigarrow$ Diophantine linear inequations system
$\rightsquigarrow$ is there a solution?
$\rightsquigarrow$ if yes, belongs to a decidable class

Examples:

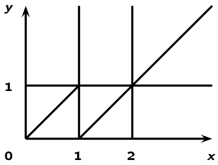
- constraint $x \sim c$ — $c \leq \max_x$
- constraint $x - y \sim c$ — $c \leq \max_{x,y}$
- update $x :\sim y + c$ — $\max_x \leq \max_y + c$
  and for each clock z, $\max_{x,z} \geq \max_{y,z} + c$, $\max_{z,x} \geq \max_{z,y} - c$
- update $x :< c$ — $c \leq \max_x$
  and for each clock z, $\max_z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

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

Decidability (cont.)



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

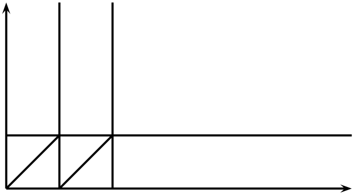


The bisimulation property is met.

What's wrong when undecidable?

Decrementation $x := x - 1$

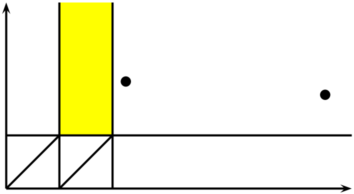
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

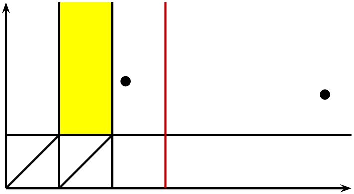
$\max_x \leq \max_x - 1$



What's wrong when undecidable?

Decrementation $x := x - 1$

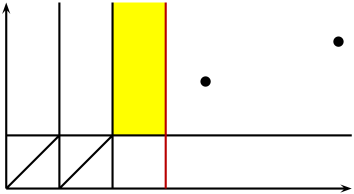
$\max_x \leq \max_x - 1$



What's wrong when undecidable?

Decrementation $x := x - 1$

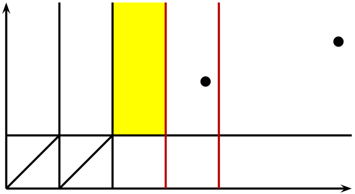
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

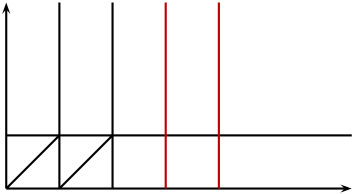
$\mathsf{max}_x \leq \mathsf{max}_x - 1$



What's wrong when undecidable?

Decrementation $x := x - 1$

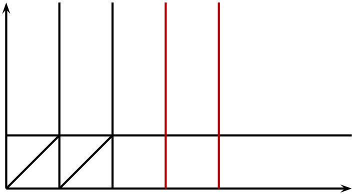
$$\max_x \leq \max_x - 1$$



What's wrong when undecidable?

Decrementation $x := x - 1$

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



etc...

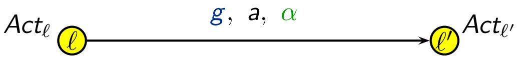
Decidability (cont.)

<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>

[Bouyer,Dufourd,Fleury,Petit 2000]

Linear hybrid automata [Henzinger 1996]

- A finite control structure + a set X of *dynamic variables*
- A transition is of the form:



- g is a linear constraint on variables
- α is a jump condition, *i.e.* an affine update of the form $X' = A.X + B$
- in each state, an activity function assigning a slope to each variable (for each $x \in X$, $Act(x) \in [\ell, u]$)

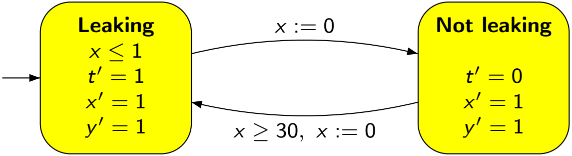
LHA semantics

- $\mathcal{H} = (\Sigma, L, X, Act \longrightarrow)$ is a LHA
- **Configurations:** $(\ell, v) \in L \times T^X$ where T is the domain
- **Timed Transition System:**
  - **action transition**: $(\ell, v) \xrightarrow{a} (\ell', v')$ if $\exists \ell \xrightarrow{g,a,J} \ell' \in \mathcal{A}$ s.t. $\begin{cases} v \models g \\ v' = \alpha(v) \end{cases}$
  - **delay transition**: $(\ell, v) \xrightarrow{\delta(d)} (\ell, v + d.Act(\ell))$ if $d \in T$

Linear hybrid automata (example)

The **gas burner** may leak. **[ACHH93]**

- each time a leakage is detected, it is repaired or stopped in less than 1s
- two leakages are separated by at least 30s



Is it possible that the gas burner leaks during a time greater than $\frac{1}{20}$ of the global time after the 60 first minutes?

$$\mathrm{AG}\,(y \geq 60 \implies 20t \leq y)$$

What about decidability?

➜ almost everything is undecidable
[Henzinger,Kopke,Puri,Varaiya 98]

Theorem. The class of LHA with clocks and only one variable having possibly two slopes $k_1 \neq k_2$ is undecidable.

Theorem. 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]

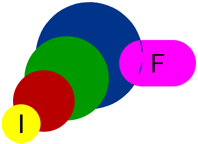
...but **on-the-fly technics** are prefered.

Reachability analysis

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

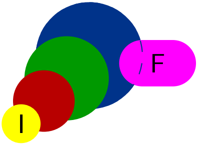
Reachability analysis

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

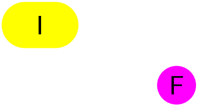


Reachability analysis

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

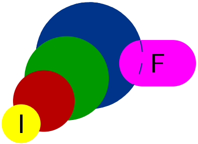


- **backward analysis algorithm:**
  compute the predecessors of final configurations

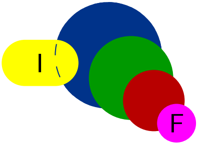


Reachability analysis

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



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



Symbolic representation, symbolic computation

- Need of a symbolic representation:

  Finite representation of infinite sets of configurations

Symbolic representation, symbolic computation

- Need of a symbolic representation:

  Finite representation of infinite sets of configurations

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

Symbolic representation, symbolic computation

- 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 $aa(aa)^*$

Symbolic representation, symbolic computation

- 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

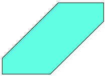
Symbolic representation, symbolic computation

- 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



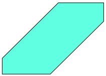
Symbolic representation, symbolic computation

- 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



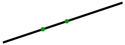
  - BDDs, DBMs (see later), CDDs, etc...

Symbolic representation, symbolic computation

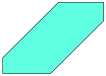
- Need of a symbolic representation:

  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...

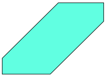
Symbolic representation, symbolic computation

- 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...



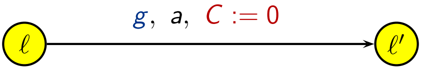
- Need of abstractions, heuristics, etc...

Examples of systems: counter automata, pushdown systems, linear hybrid automata, timed automata, 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

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

Note on the backward analysis of TA

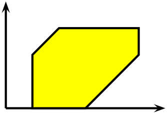


$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$ $\qquad\qquad$ Z

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

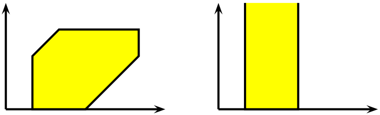


Z

Note on the backward analysis of TA

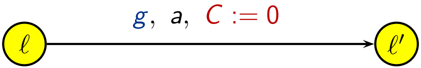
$$\ell \xrightarrow{g,\ a,\ C := 0} \ell'$$

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

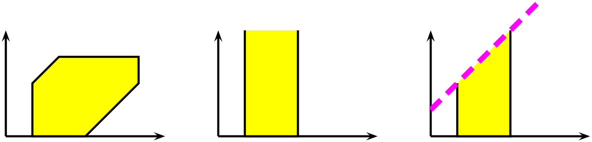


Z $\qquad$ $[C \leftarrow 0]^{-1}(Z \cap (C = 0))$

Note on the backward analysis of TA

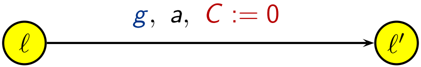


$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$ $\quad\quad$ Z



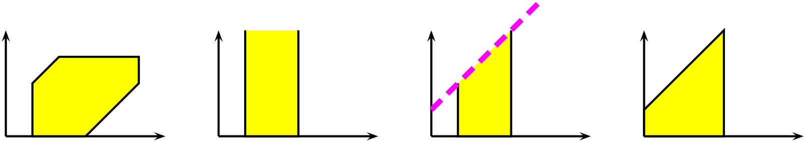
Z $\quad\quad$ $[C \leftarrow 0]^{-1}(Z \cap (C = 0))$

Note on the backward analysis of TA



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

Z

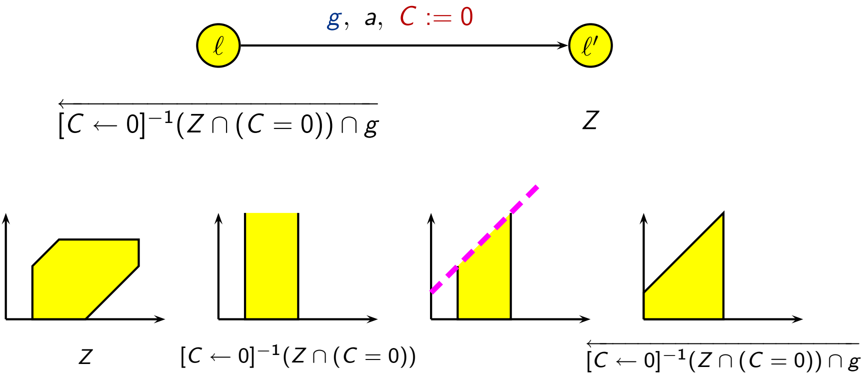


Z

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

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

Note on the backward analysis of TA



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

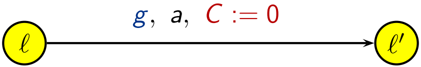
$$\ell \xrightarrow{\quad g,\ a,\ C := 0 \quad} \ell'$$

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

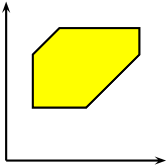
A zone is a set of valuations defined by a clock constraint

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

Forward analysis of timed automata



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

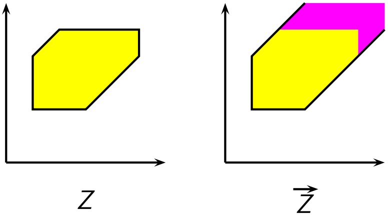


Z

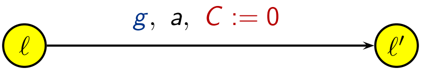
Forward analysis of timed automata

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

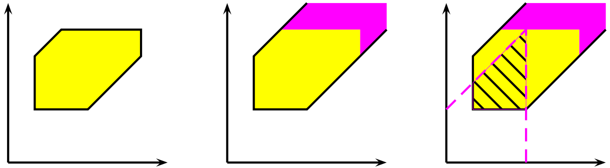
zones $\quad Z \quad [C \leftarrow 0](\overrightarrow{Z} \cap g)$



Forward analysis of timed automata

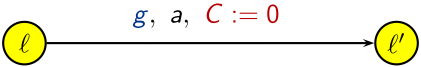


zones $\quad Z \quad\quad [C \leftarrow 0](\overrightarrow{Z} \cap g)$

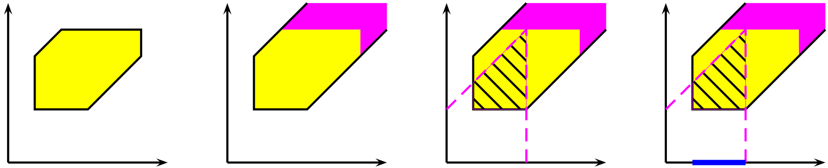


$Z \quad\quad \overrightarrow{Z} \quad\quad \overrightarrow{Z} \cap g$

Forward analysis of timed automata

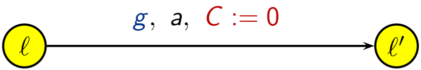


zones Z $[C \leftarrow 0](\overrightarrow{Z} \cap g)$

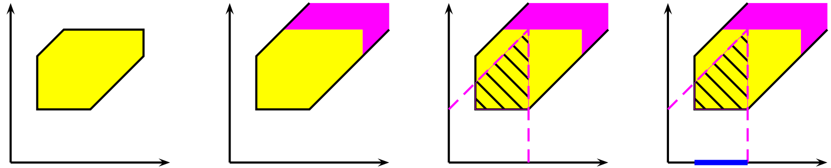


Z $\overrightarrow{Z}$ $\overrightarrow{Z} \cap g$ $[y \leftarrow 0](\overrightarrow{Z} \cap g)$

Forward analysis of timed automata



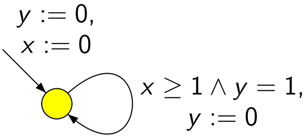
zones $\quad Z \quad [C \leftarrow 0](\overrightarrow{Z} \cap g)$

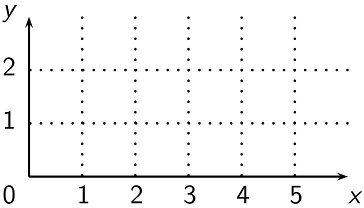


$Z \qquad \overrightarrow{Z} \qquad \overrightarrow{Z} \cap g \qquad [y \leftarrow 0](\overrightarrow{Z} \cap g)$

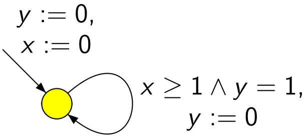
➜ a termination problem

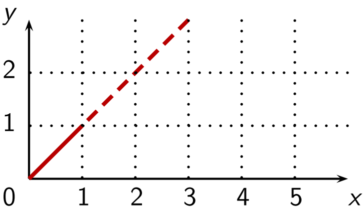
Non termination of the forward analysis



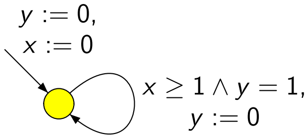


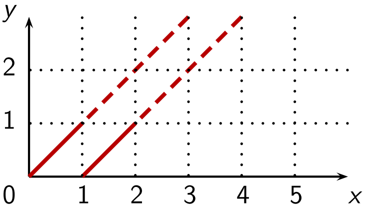
Non termination of the forward analysis



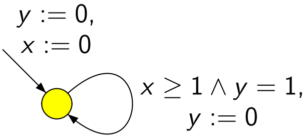


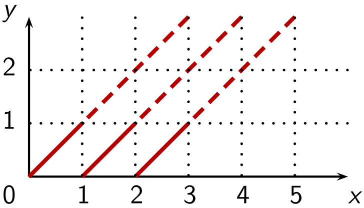
Non termination of the forward analysis



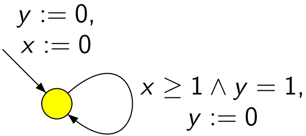


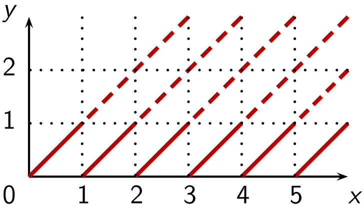
Non termination of the forward analysis





Non termination of the forward analysis





➜ an infinite number of steps...

"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

➜ 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

  ➜ correct w.r.t. reachability

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

  ➜ correct w.r.t. reachability

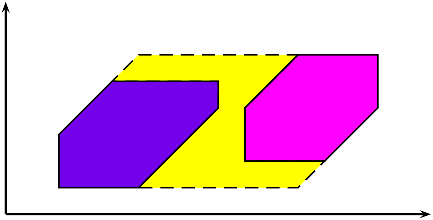
$$q \xrightarrow{g,a,C:=0} q' \quad \text{implies} \quad \mathrm{Act}(q) = \mathrm{clocks}(g) \cup (\mathrm{Act}(q') \setminus C)$$

...

"Solutions" to this problem (cont.)

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

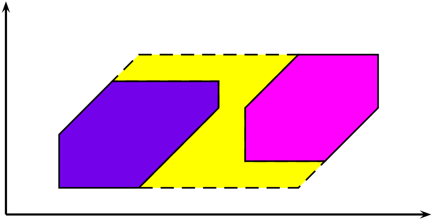
➜ "semi-correct" w.r.t. reachability



"Solutions" to this problem (cont.)

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

  ➜ "semi-correct" w.r.t. reachability



- **extrapolation**, a widening operator on zones

The DBM data structure

DBM (Difference Bounded Matrice) data structure

[Berthomieu, Menasche 1983] [Dill 1989]

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

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

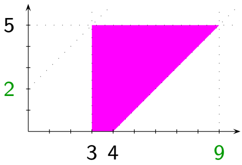
The DBM data structure

DBM (Difference Bounded Matrice) data structure

[Berthomieu, Menasche 1983] [Dill 1989]

$$(x_1 \geq 3) \;\wedge\; (x_2 \leq 5) \;\wedge\; (x_1 - x_2 \leq 4) \qquad \begin{array}{c c} & \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \\ x_2 \end{array} & \begin{pmatrix} +\infty & -3 & +\infty \\ +\infty & +\infty & 4 \\ 5 & +\infty & +\infty \end{pmatrix} \end{array}$$

- Existence of a normal form



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

The DBM data structure

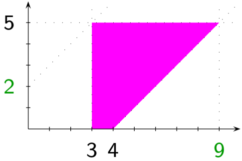
DBM (Difference Bounded Matrice) data structure

[Berthomieu, Menasche 1983] [Dill 1989]

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

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

- 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 $+k$)

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

- "intuitively", erase non-relevant constraints

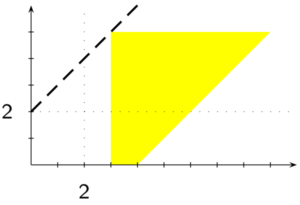
➜ ensures termination

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



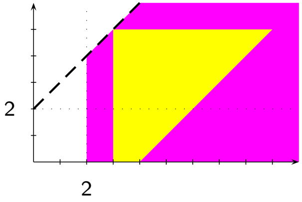
➜ ensures termination

The extrapolation operator

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

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

- "intuitively", erase non-relevant constraints



➜ ensures termination

Classical algorithm, focus on correctness

Challenge: choose a **good** constant for the extrapolation so that the forward computation is correct.

Classical algorithm, focus on correctness

Challenge: choose a **good** constant for the extrapolation so that the forward computation is correct.

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

Classical algorithm, focus on correctness

Challenge: choose a **good** constant for the extrapolation so that the forward computation is correct.

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

Theorem: this algorithm is correct for diagonal-free timed automata.

Classical algorithm, focus on correctness

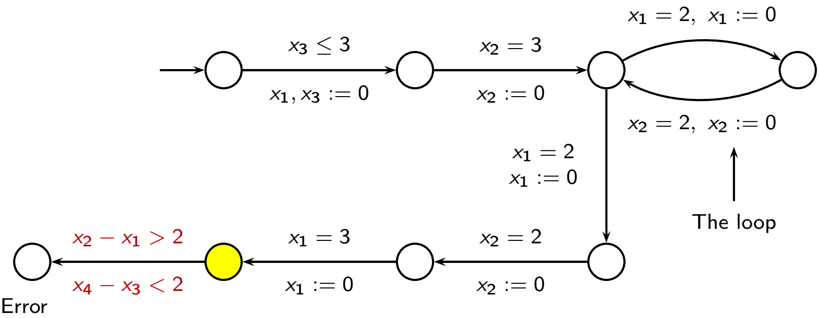
Challenge: choose a **good** constant for the extrapolation so that the forward computation is correct.

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

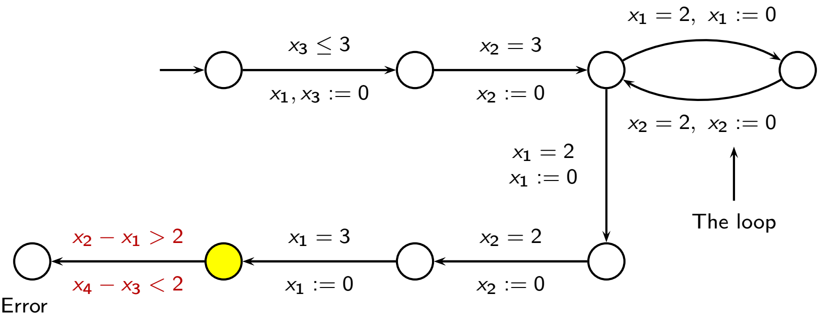
Theorem: this algorithm is correct for diagonal-free timed automata.

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

A problematic automaton

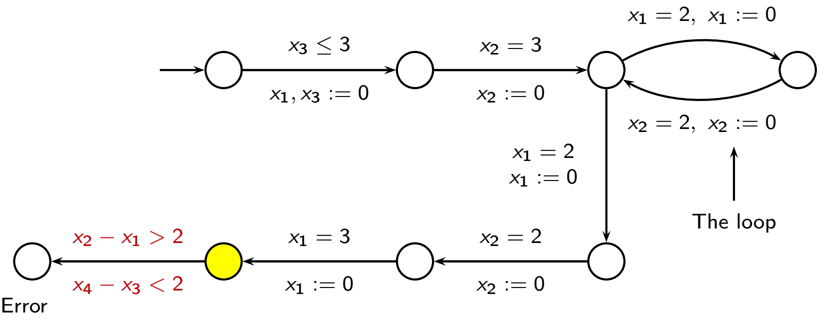


A problematic automaton

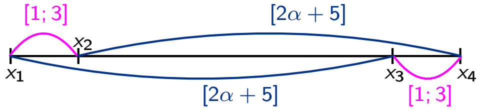


$$\begin{cases} v(x_1) = 0 \\ v(x_2) = d \\ v(x_3) = 2\alpha + 5 \\ v(x_4) = 2\alpha + 5 + d \end{cases}$$

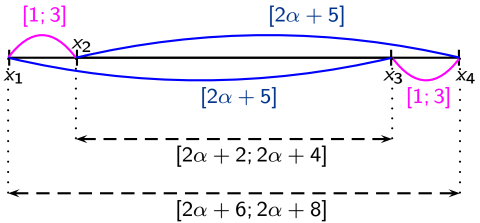
A problematic automaton



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

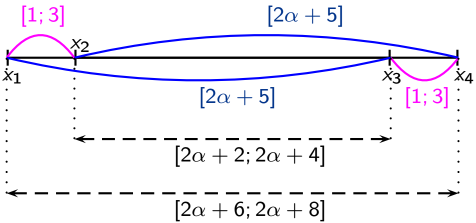


The problematic zone



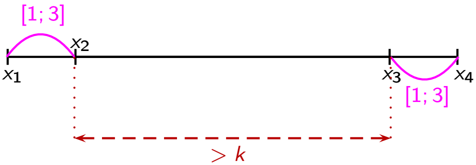
implies $x_1 - x_2 = x_3 - x_4$.

The problematic zone



implies $x_1 - x_2 = x_3 - x_4$.

If α is sufficiently large, after extrapolation:



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

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.

➜ 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?

Outline

Discussion on complexity

[Alur 1991, Alur Henzinger 1994, Alur Courcoubetis Dill 1993, Aceto Laroussinie 2002]

	Kripke structures S	Timed automaton A
Reachability	NLOGSPACE-complete	
CTL/TCTL	P-complete	
AF-μ-calc./$L_{\mu,\nu}$	P-complete	
full μ-calc./$L^{+}_{\mu,\nu}$	NP $\cap$ co-NP	

Discussion on complexity

[Alur 1991, Alur Henzinger 1994, Alur Courcoubetis Dill 1993, Aceto Laroussinie 2002]

	Kripke structures S	Timed automaton A or $(S_1 \parallel \ldots \parallel S_n)$
Reachability	NLOGSPACE-complete	PSPACE-complete
CTL/TCTL	P-complete	PSPACE-complete
AF-μ-calc./$L_{\mu,\nu}$	P-complete	EXPTIME-complete
full μ-calc./$L^{+}_{\mu,\nu}$	NP $\cap$ co-NP	EXPTIME-complete

Timing constraints induce a complexity blowup!

Discussion on complexity

[Alur 1991, Alur Henzinger 1994, Alur Courcoubetis Dill 1993, Aceto Laroussinie 2002]

	Kripke structures S	Timed automaton A or $(S_1 \parallel \ldots \parallel S_n)$
Reachability	NLOGSPACE-complete	PSPACE-complete
CTL/TCTL	P-complete	PSPACE-complete
AF-μ-calc./$L_{\mu,\nu}$	P-complete	EXPTIME-complete
full μ-calc./$L^+_{\mu,\nu}$	NP $\cap$ co-NP	EXPTIME-complete

Timing constraints induce a complexity blowup!

From a complexity point of view, adding clocks = adding components!

Discussion on complexity

[Alur 1991, Alur Henzinger 1994, Alur Courcoubetis Dill 1993, Aceto Laroussinie 2002]

	Kripke structures S	Timed automaton A or $(S_1 \parallel \ldots \parallel S_n)$ or $(A_1 \parallel \ldots \parallel A_n)$
Reachability	NLOGSPACE-complete	PSPACE-complete
CTL/TCTL	P-complete	PSPACE-complete
AF-μ-calc./$L_{\mu,\nu}$	P-complete	EXPTIME-complete
full μ-calc./$L^{+}_{\mu,\nu}$	NP $\cap$ co-NP	EXPTIME-complete

Timing constraints induce a complexity blowup!

From a complexity point of view, adding clocks = adding components!

State explosion problem

- due to parallel composition
- due to timing constraints

State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view:

no double complexity gap!

State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view: ☺

no double complexity gap!

State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view: ☺

no double complexity gap!

In practice:

- BDD-like techniques try to avoid discrete state explosion problem in untimed systems ➜ SMV verifies very large systems

State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view: ☺

no double complexity gap!

In practice:

- BDD-like techniques try to avoid discrete state explosion problem in untimed systems ➜ SMV verifies very large systems
- **Timed systems:** problems to deal with both explosions. Much smaller systems can be analyzed in practice.

State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view: ☺

no double complexity gap!

In practice: ☹

- BDD-like techniques try to avoid discrete state explosion problem in untimed systems ➜ SMV verifies very large systems
- **Timed systems:** problems to deal with both explosions. Much smaller systems can be analyzed in practice.

State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view: ☺

no double complexity gap!

In practice: ☹

- BDD-like techniques try to avoid discrete state explosion problem in untimed systems ➜ SMV verifies very large systems
- **Timed systems:** problems to deal with both explosions. Much smaller systems can be analyzed in practice.

Tools for timed systems: Uppaal, HyTech, Kronos, etc...

Conclusion & Further Work

- Decidability is quite well understood.
- Needs to understand better the **geometry** of the reachable state space.
    - clever (and correct) implementation of timed automata
    - accelerate verification of timed automata
- Data structures for both **dense** and **discrete** parts

Conclusion & Further Work

- Decidability is quite well understood.
- Needs to understand better the **geometry** of the reachable state space.
  - clever (and correct) implementation of timed automata
  - accelerate verification of timed automata
- Data structures for both **dense** and **discrete** parts

To be continued...

Conclusion & Further Work

- Decidability is quite well understood.
- Needs to understand better the **geometry** of the reachable state space.
    - clever (and correct) implementation of timed automata
    - accelerate verification of timed automata
- Data structures for both **dense** and **discrete** parts

To be continued...

- Some other current challenges:
    - controller synthesis
    - implementability issues (program synthesis)
    - optimal computations (see Kim's talk)
    - ...

Bibliography I

[ACD+92] Alur, Courcoubetis, Dill, Halbwachs, Wong-Toi. **Minimization of Timed Transition Systems.** CONCUR'92 (LNCS 630).

[ACHH93] Alur, Courcoubetis, Henzinger, Ho. **Hybrid Automata: an Algorithmic Approach to Specification and Verification of Hybrid Systems.** Hybrid Systems I (LNCS 736).

[AD90] Alur, Dill. **Automata for Modeling Real-Time Systems.** ICALP'90 (LNCS 443).

[AD94] Alur, Dill. **A Theory of Timed Automata.** TCS 126(2), 1994.

[AH94] Alur, Henzinger. **A Really Temporal Logic.** JACM 41(1), 1994.

[Alur91] Alur. **Techniques for Automatic Verification of Real-Time Systems.** PhD Thesis, 1991.

[AL02] Aceto, Laroussinie. **Is your Model-Checker on Time? On the Complexity of Model-Checking for Timed Modal Logics.** JLAP 52-53, 2002.

[AM04] Alur, Madhusudan. **Decision Problems for Timed Automata.** SFM-04:RT (LNCS 3142).

[Asarin04] Asarin. **Challenges in Timed Languages: From Applied Theory to Basic Theory.** BEATCS 83, 2004.

Bibliography II

[BD00] Bérard, Dufourd. **Timed Automata and Additive Clock Constraints.** IPL 75(1–2), 2000.

[BDFP00a] Bouyer, Dufourd, Fleury, Petit. **Are Timed Automata Updatable?** CAV'00 (LNCS 1855).

[BDFP00b] Bouyer, Dufourd, Fleury, Petit. **Expressiveness of Updatable Timed Automata.** MFCS'00 (LNCS 1893).

[BDGP98] Bérard, Diekert, Gastin, Petit. **Characterization of the Expressive Power of Silent Transitions in Timed Automata.** Fundamenta Informaticae 36(2–3), 1998.

[BF99] Bérard, Fribourg. **Automatic Verification of a Parametric Real-Time Program: the ABR Conformance Protocol.** CAV'99 (LNCS 1633).

[BM83] Berthomieu, Menasche. **An Enumerative Approach for Analyzing Time Petri Nets.** World Comp. Congress, 1983.

[Bouyer03] Bouyer. **Untameable Timed Automata!** STACS'03 (LNCS 2607).

[Bouyer04] Bouyer. **Forward analysis of updatable timed automata**. Formal Methods in System Design 24(3),2004.

[BS91] Brzozowski, Seger. **Advances in Asynchronous Circuit Theory**. BEATCS, 1991.

Bibliography III

[Dill89] Dill. **Timing Assumptions and Verification of Finite-State Concurrent Systems.** Aut. Verif. Methods for Fin. State Sys. (LNCS 1989).

[DT98] Daws, Tripakis. **Model-Checking of Real-Time Reachability Properties using Abstractions.** TACAS'98 (LNCS 1384).

[DY96] Daws, Yovine. **Reducing the Number of Clock Variables of Timed Automata.** RTSS'96.

[Hen96] Henzinger. **The Theory of Hybrid Automata.** LICS'96.

[HKPV98] Henzinger, Kopke, Puri, Varaiya. **What's Decidable about Hybrid Automata?** J. Comp. and Sys. Sci 57, 1998.

[LPY97] Larsen, Pettersson, Yi. Uppaal **in a Nutshell**. Software Tools for Technology Transfer 1(1–2), 1997.

[LPWY99] Larsen, Pearson, Weise, Yi. **Clock Difference Diagrams.** Nordic Journal of Computation 6(3), 1999.

[Minsky67] Minsky. **Computation: Finite and Infinite Machines.** 1967.

[TY01] Tripakis, Yovine. **Analysis of Timed Systems using Time-Abstracting Bisimulations.** FMSD 18(1), 2001.

Hytech: `http://www-cad.eecs.berkeley.edu:80/~tah/HyTech/`

Kronos: `http://www-verimag.imag.fr/TEMPORISE/kronos/`

Uppaal: `http://www.uppaal.com/`