An introduction to timed automata

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France

Outline

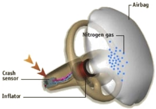
1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

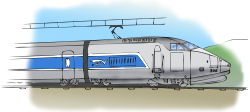
Time-dependent systems

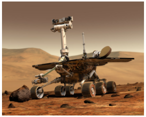
- We are interested in timed systems

Time-dependent systems

- We are interested in timed systems

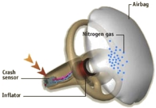


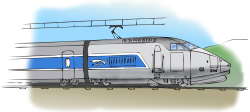


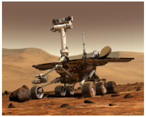


Time-dependent systems

- We are interested in timed systems



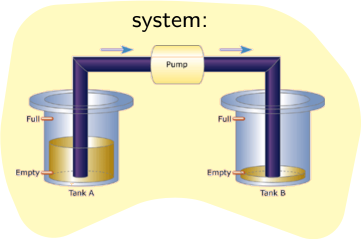


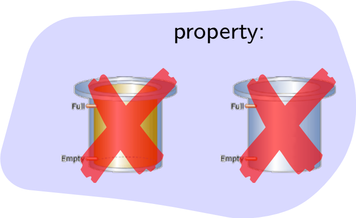


- and in their correctness

"Will the airbag open within 5ms after the car crashes?"
"Will the robot explore a given area without getting out of energy?"

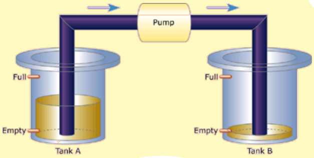
Model-checking and control



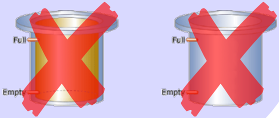


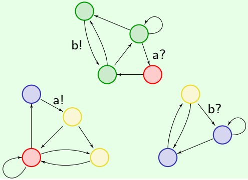
Model-checking and control

system:



property:

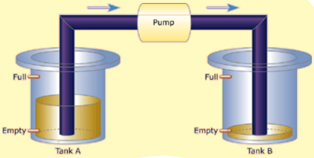




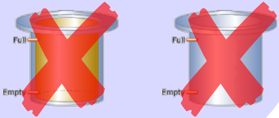
$\mathbf{AG}(\neg \texttt{B.overfull} \wedge \neg \texttt{B.dried\_up})$

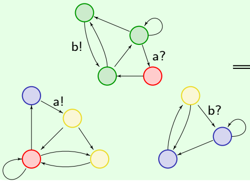
Model-checking and control

system:



property:



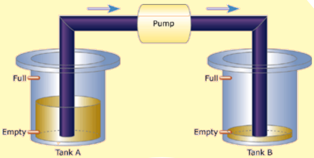


algorithm

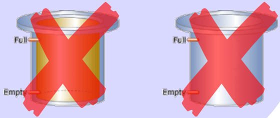
AG($\neg$`B.overfull`
$\wedge$ $\neg$`B.dried_up`)

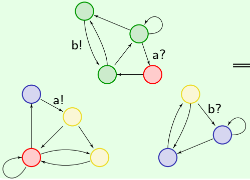
Model-checking and control

system:



property:





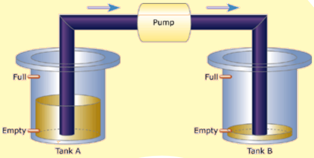
model-checking algorithm

$\mathbf{AG}(\neg\texttt{B.overfull} \wedge \neg\texttt{B.dried\_up})$

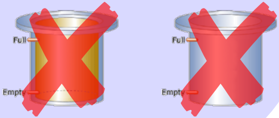
yes/no

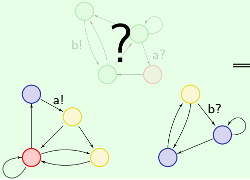
Model-checking and control

system:



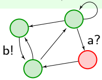
property:





control/synthesis algorithm

$\mathbf{AG}(\neg \texttt{B.overfull} \wedge \neg \texttt{B.dried\_up})$



Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

Reasoning about real-time systems

A plethora of models

- timed circuits,
- time(d) Petri nets,
- timed automata,
- timed process algebra,
- ...

Reasoning about real-time systems

A plethora of models

- timed circuits,
- time(d) Petri nets,
- **timed automata,**
- timed process algebra,
- . . .

[AD94] Alur, Dill. A Theory of Timed Automata. *Theor. Comp. Science*, 1994.

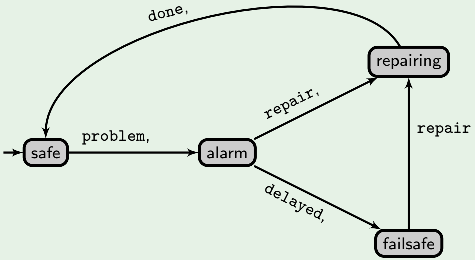
Reasoning about real-time systems

The model of timed automata [AD94]

A timed automaton is made of

- a finite automaton-based structure

Example



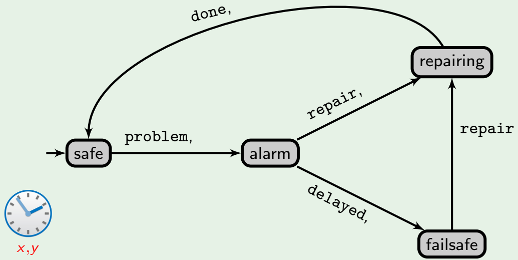
Reasoning about real-time systems

The model of timed automata [AD94]

A timed automaton is made of

- a finite automaton-based structure
- a set of clocks

Example



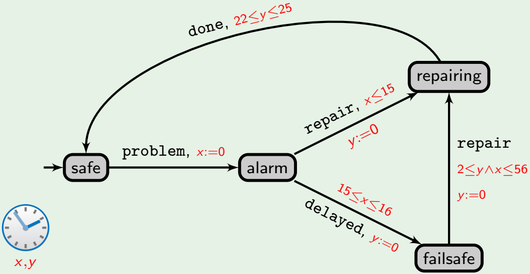
Reasoning about real-time systems

The model of timed automata [AD94]

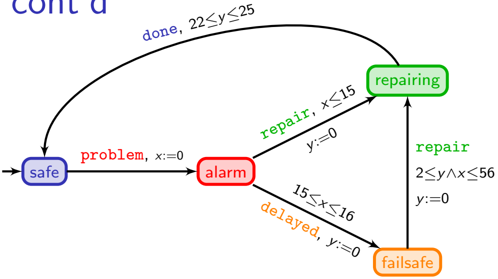
A timed automaton is made of

- a finite automaton-based structure
- a set of clocks
- timing constraints and clock resets on transitions

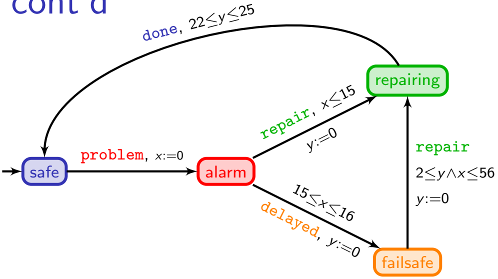
Example



Example – cont'd

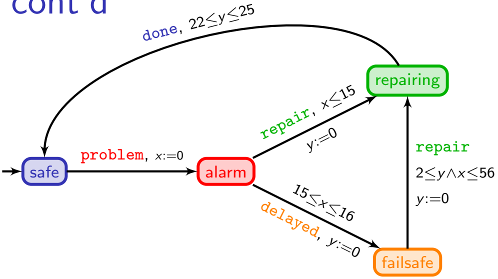


Example – cont'd



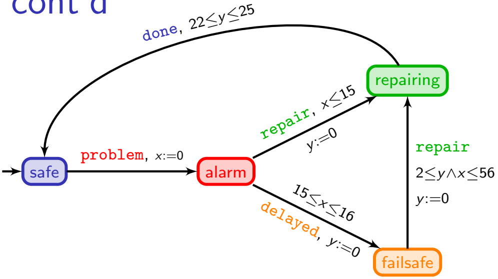
	safe
x	0
y	0

Example – cont'd



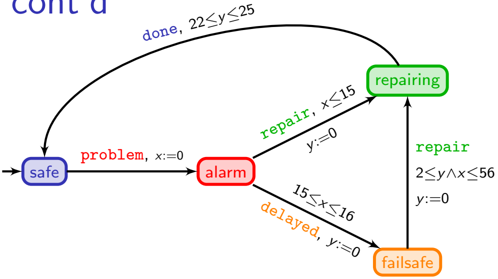
	safe	$\xrightarrow{23}$	safe
x	0		23
y	0		23

Example – cont'd



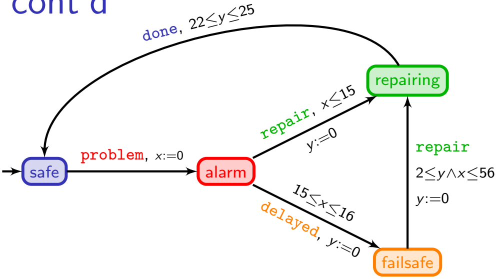
	safe	$\xrightarrow{23}$	safe	$\xrightarrow{\texttt{problem}}$	alarm
x	0		23		0
y	0		23		23

Example – cont'd



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

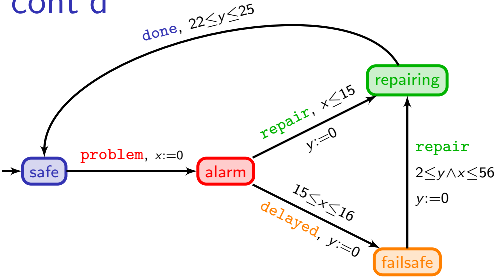
Example – cont'd



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

	failsafe
⋯	15.6
	0

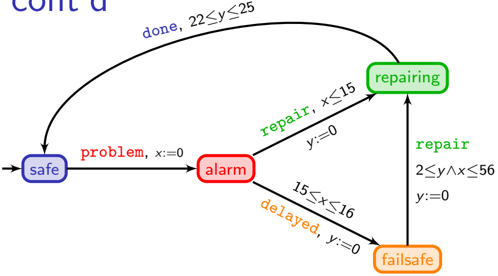
Example – cont'd



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

	failsafe	$\xrightarrow{2.3}$	failsafe
⋯	15.6		17.9
	0		2.3

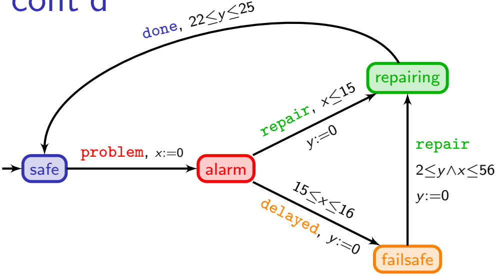
Example – cont'd



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

	failsafe	$\xrightarrow{2.3}$	failsafe	$\xrightarrow{\text{repair}}$	repairing
···	15.6		17.9		17.9
	0		2.3		0

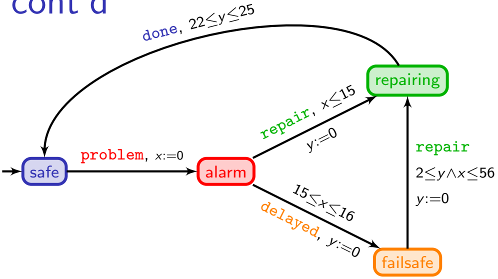
Example – cont'd



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

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

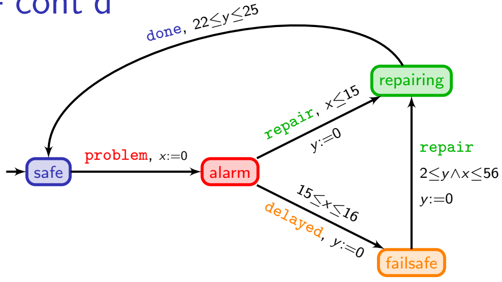
Example – cont'd



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

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

Example – cont'd

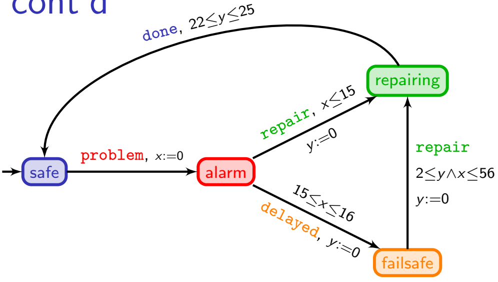


valuation

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

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

Example – cont'd



valuation

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

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

This run reads the timed word
(`problem`, 23)(`delayed`, 38.6)(`repair`, 40.9)(`done`, 63).

Discrete-time semantics

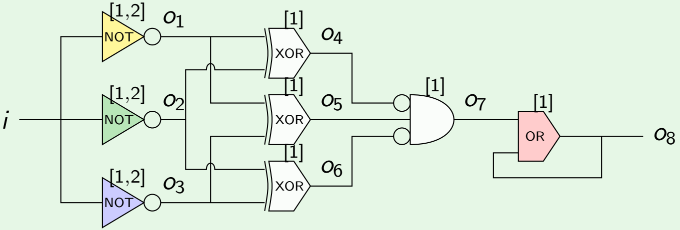
...because computers are digital!

[Alur91] Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

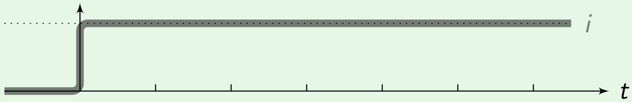
Discrete-time semantics

...because computers are digital!

Example [Alur91]



- under discrete-time, the output is always 0:

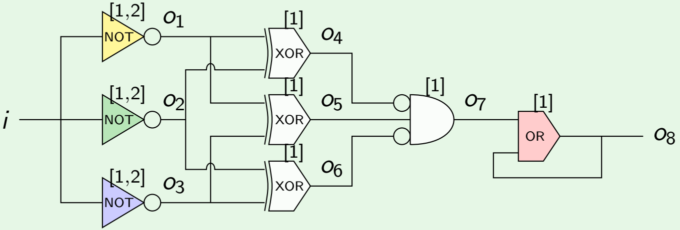


[Alur91] Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

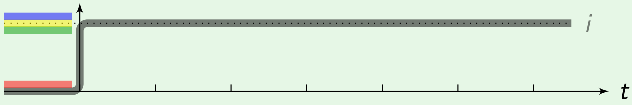
Discrete-time semantics

...because computers are digital!

Example [Alur91]



- under discrete-time, the output is always 0:

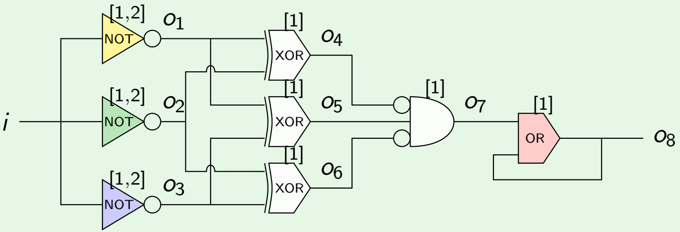


[Alur91] Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

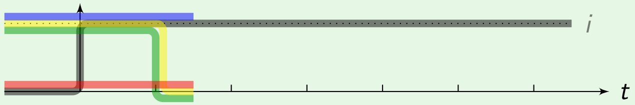
Discrete-time semantics

...because computers are digital!

Example [Alur91]



- under discrete-time, the output is always 0:

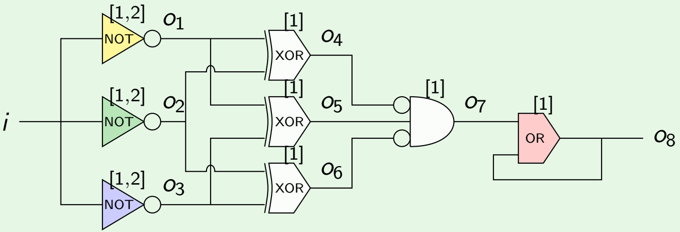


[Alur91] Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

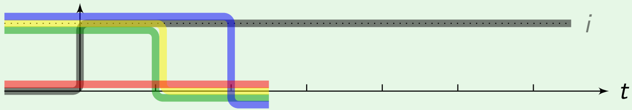
Discrete-time semantics

...because computers are digital!

Example [Alur91]



- under discrete-time, the output is always 0:

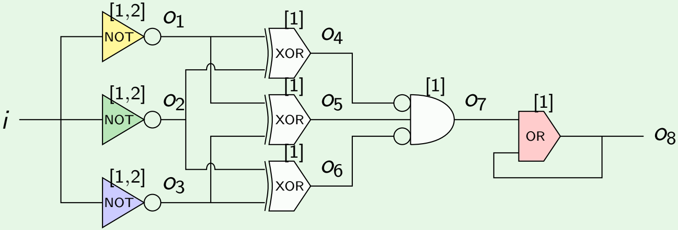


[Alur91] Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

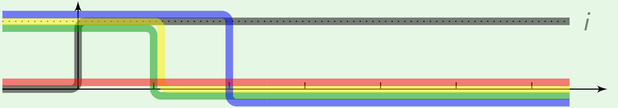
Discrete-time semantics

...because computers are digital!

Example [Alur91]



- under discrete-time, the output is always 0:

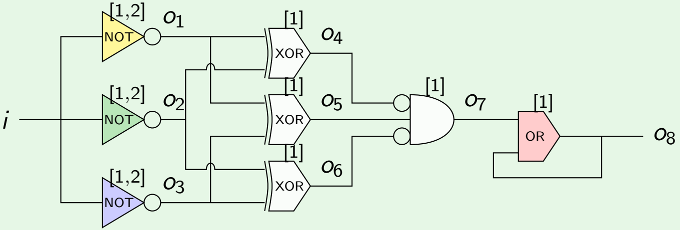


[Alur91] Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

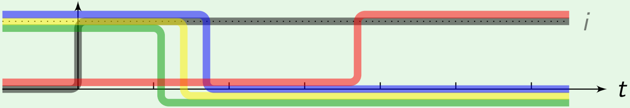
Discrete-time semantics

...because computers are digital!

Example [Alur91]



- under continuous-time, the output can be 1:

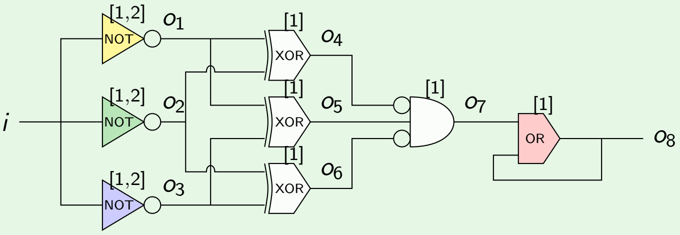


[Alur91] Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

Discrete-time semantics

...because computers are digital!

Example [Alur91]



Finding the correct granularity (if one exists) is hard!

[Alur91] Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

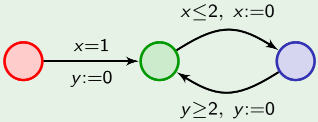
Continuous-time semantics

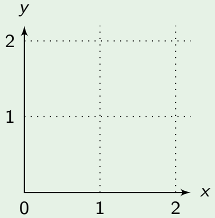
...real-time models for real-time systems!

Continuous-time semantics

...real-time models for real-time systems!

Example

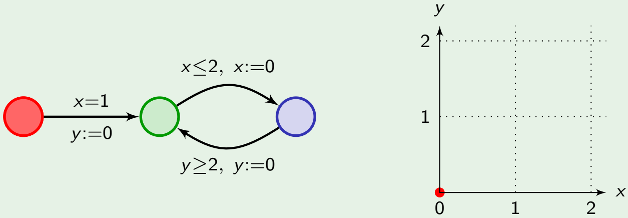




Continuous-time semantics

...real-time models for real-time systems!

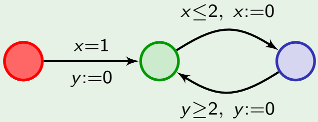
Example

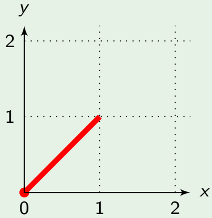


Continuous-time semantics

...real-time models for real-time systems!

Example

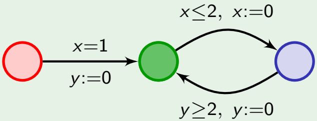


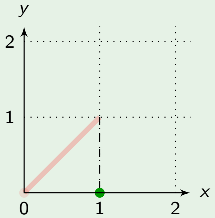


Continuous-time semantics

...real-time models for real-time systems!

Example

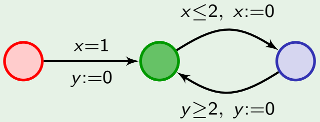


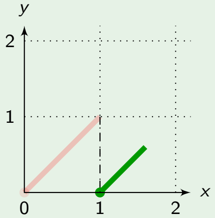


Continuous-time semantics

...real-time models for real-time systems!

Example

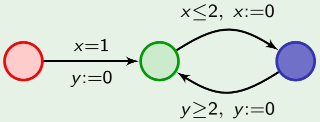


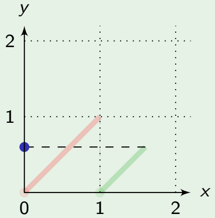


Continuous-time semantics

...real-time models for real-time systems!

Example

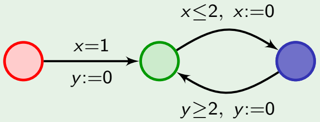


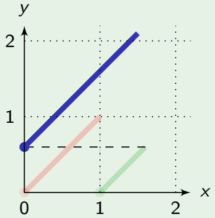


Continuous-time semantics

...real-time models for real-time systems!

Example

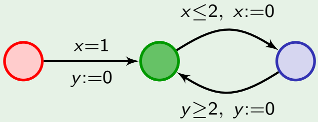


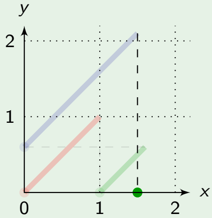


Continuous-time semantics

...real-time models for real-time systems!

Example

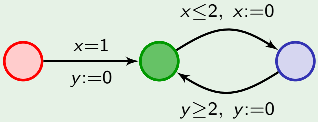


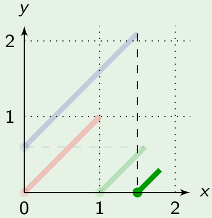


Continuous-time semantics

...real-time models for real-time systems!

Example

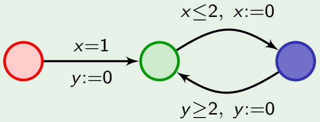


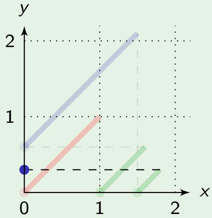


Continuous-time semantics

...real-time models for real-time systems!

Example

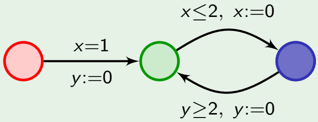


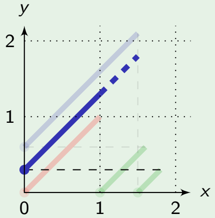


Continuous-time semantics

...real-time models for real-time systems!

Example

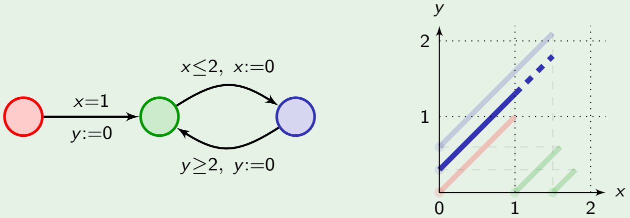




Continuous-time semantics

...real-time models for real-time systems!

Example



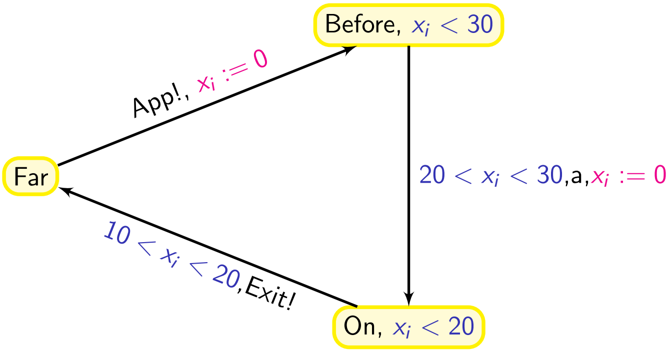
We will focus on the continuous-time semantics, and discuss further its relevance at the end of the tutorial

Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

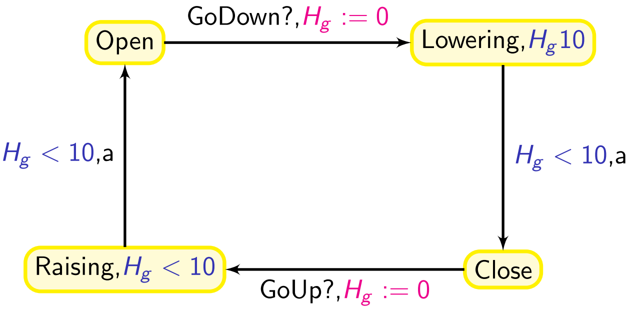
The train crossing example

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



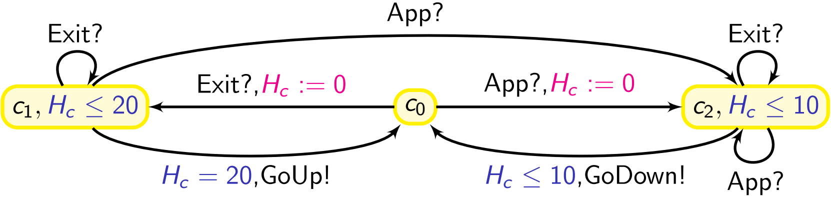
The train crossing example – cont'd

The gate:



The train crossing example – cont'd

The controller:



The train crossing example – cont'd

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 – cont'd

Some properties one could check:

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

The train crossing example – cont'd

Some properties one could check:

- Is the gate closed when a train crosses the road?
- Is the gate always closed for less than 5 minutes?

Another example: A mutual exclusion protocol

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

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

Another example: A mutual exclusion protocol

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

Process i:

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

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

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

Another example: A mutual exclusion protocol

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

Process i:

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

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

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

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

Another example: The task graph scheduling problem

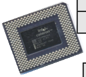
Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

P_1 (fast):

time	
$+$	2 picoseconds
$\times$	3 picoseconds

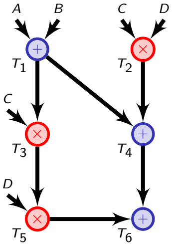
energy	
idle	10 Watt
in use	90 Watts

P_2 (slow):



time	
$+$	5 picoseconds
$\times$	7 picoseconds

energy	
idle	20 Watts
in use	30 Watts



Another example: The task graph scheduling problem

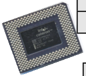
Compute $D \times (C \times (A + B)) + (A + B) + (C \times D)$ using two processors:

P_1 (fast):

time	
$+$	2 picoseconds
$\times$	3 picoseconds

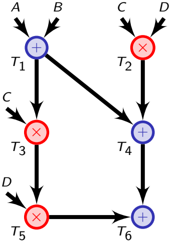
energy	
idle	10 Watt
in use	90 Watts

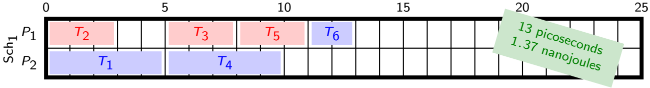
P_2 (slow):



time	
$+$	5 picoseconds
$\times$	7 picoseconds

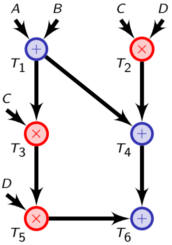
energy	
idle	20 Watts
in use	30 Watts





Another example: The task graph scheduling problem

Compute $D\times(C\times(A+B))+(A+B)+(C\times D)$ using two processors:

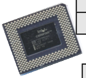


P_1 (fast):

time	
$+$	2 picoseconds
$\times$	3 picoseconds

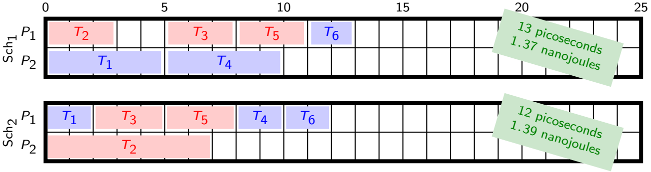
energy	
idle	10 Watt
in use	90 Watts

P_2 (slow):



time	
$+$	5 picoseconds
$\times$	7 picoseconds

energy	
idle	20 Watts
in use	30 Watts



Another example: The task graph scheduling problem

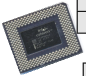
Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

P_1 (fast):

time	
$+$	2 picoseconds
$\times$	3 picoseconds

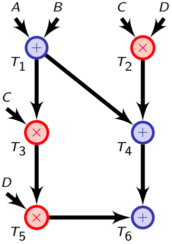
energy	
idle	10 Watt
in use	90 Watts

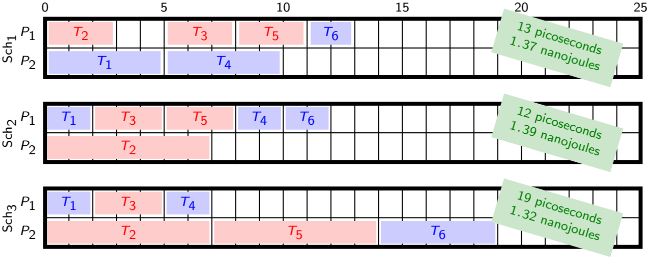
P_2 (slow):



time	
$+$	5 picoseconds
$\times$	7 picoseconds

energy	
idle	20 Watts
in use	30 Watts

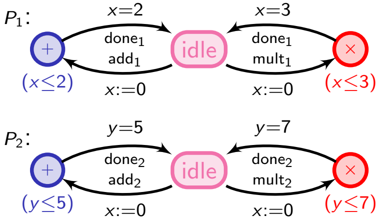




Modelling the task graph scheduling problem

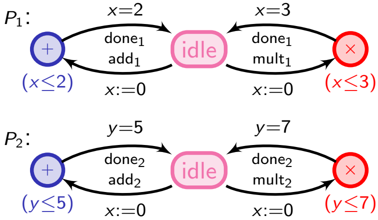
Modelling the task graph scheduling problem

- Processors

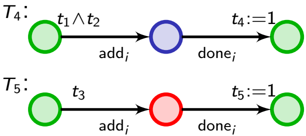


Modelling the task graph scheduling problem

- Processors

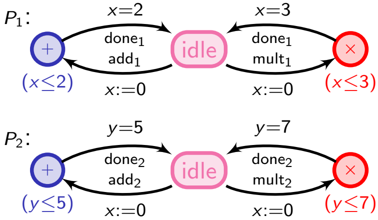


- Tasks

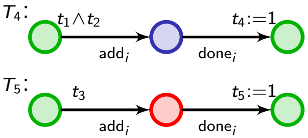


Modelling the task graph scheduling problem

- Processors



- Tasks

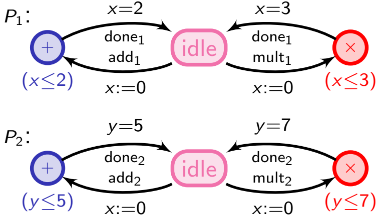


$\rightsquigarrow$ build the synchronized product of all these automata

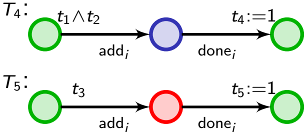
$$(P_1 \parallel P_2) \parallel_s (T_1 \parallel T_2 \parallel \cdots \parallel T_6)$$

Modelling the task graph scheduling problem

- Processors



- Tasks



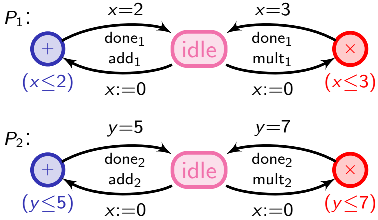
$\rightsquigarrow$ build the synchronized product of all these automata

$$(P_1 \parallel P_2) \parallel_s (T_1 \parallel T_2 \parallel \cdots \parallel T_6)$$

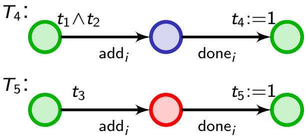
A schedule: a path in the global system which reaches $t_1 \wedge \cdots \wedge t_6$

Modelling the task graph scheduling problem

- Processors



- Tasks



$\rightsquigarrow$ build the synchronized product of all these automata

$$(P_1 \parallel P_2) \parallel_s (T_1 \parallel T_2 \parallel \cdots \parallel T_6)$$

A schedule: a path in the global system which reaches $t_1 \wedge \cdots \wedge t_6$

Questions one can ask

- Can the computation be made in no more than 10 time units?
- Is there a scheduling along which no processor is ever idle?
- ...

Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

Verification

Basic verification problems

- basic reachability/safety properties
- basic liveness properties

Verification

Basic verification problems

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

Is the language accepted by a timed automaton empty?

Verification

Basic verification problems

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

Verification

Basic verification problems

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

Verification

Basic verification problems

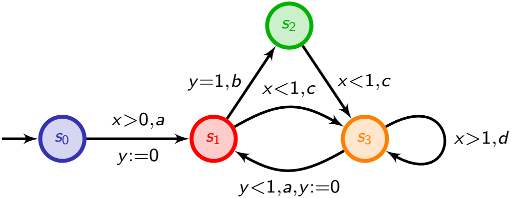
- Problem: the set of configurations is infinite
  $\rightsquigarrow$ classical methods for finite-state systems cannot be applied
- Positive key point: variables (clocks) increase at the same speed

Theorem [AD90,AD94]

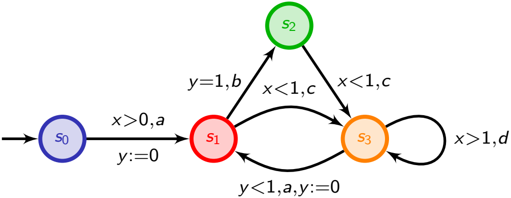
The emptiness problem for timed automata is decidable and PSPACE-complete.

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

An example [AD94]

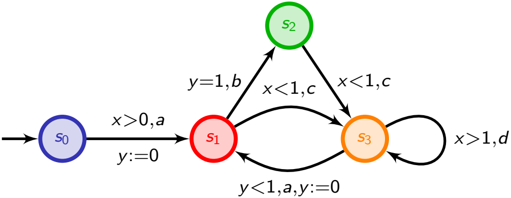


An example [AD94]



Starting at s_0, can we visit s_2 and then s_3?

An example [AD94]



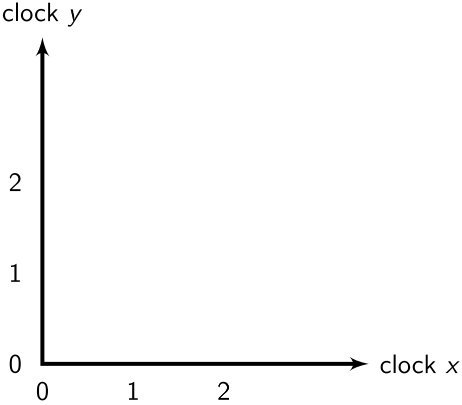
Starting at s_0, can we visit s_2 and then s_3?

Method: construct a finite abstraction

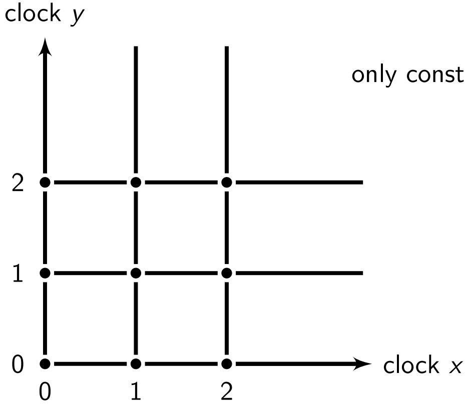
Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

The region abstraction



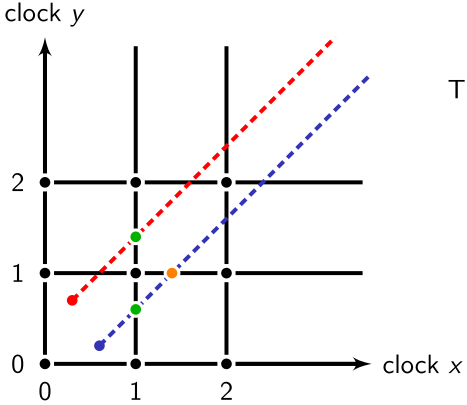
The region abstraction



only constraints: $x \sim c$ with $c \in \{0, 1, 2\}$
$y \sim c$ with $c \in \{0, 1, 2\}$

- "compatibility" between regions and constraints

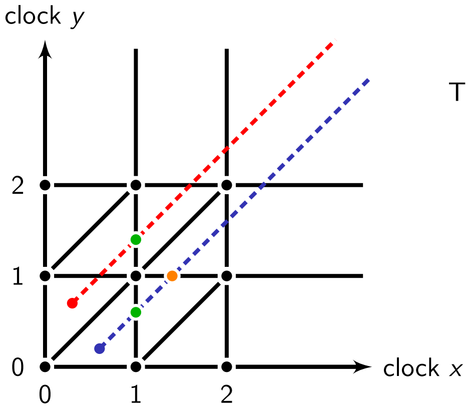
The region abstraction



The path
- can be fired from
- cannot be fired from

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

The region abstraction

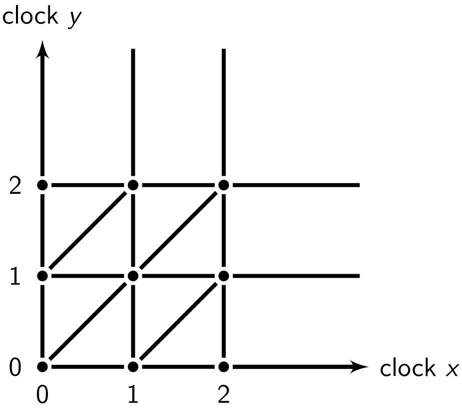


The path 

- can be fired from
- cannot be fired from

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

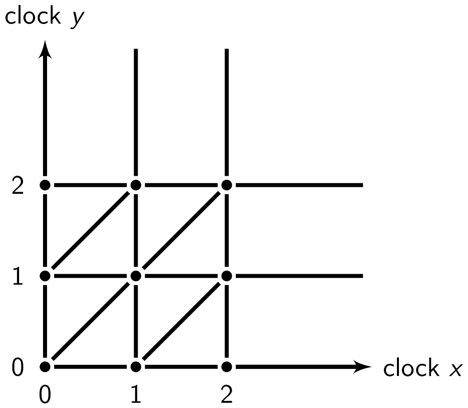
The region abstraction



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

$\rightsquigarrow$ an equivalence of finite index

The region abstraction



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

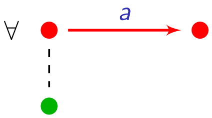
$\rightsquigarrow$ an equivalence of finite index
a time-abstract bisimulation

Time-abstract bisimulation

This is a relation between • and • such that:

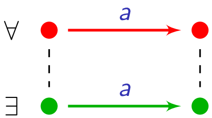
Time-abstract bisimulation

This is a relation between • and • such that:



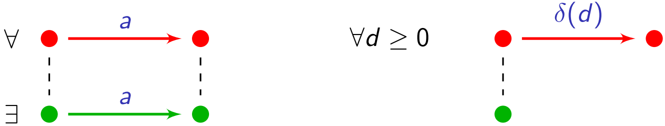
Time-abstract bisimulation

This is a relation between • and • such that:



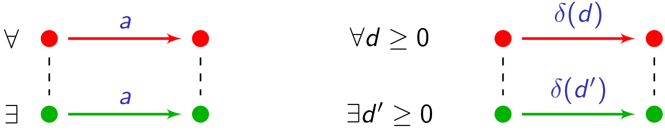
Time-abstract bisimulation

This is a relation between • and • such that:



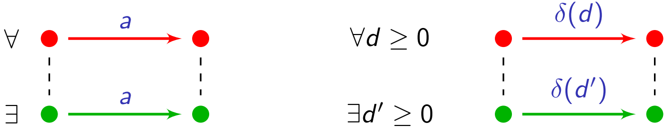
Time-abstract bisimulation

This is a relation between • and • such that:



Time-abstract bisimulation

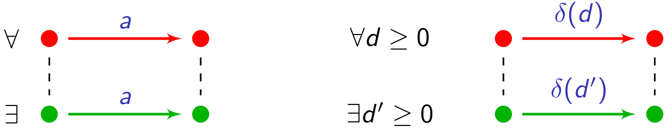
This is a relation between • and • such that:



... and vice-versa (swap • and •).

Time-abstract bisimulation

This is a relation between • and • such that:



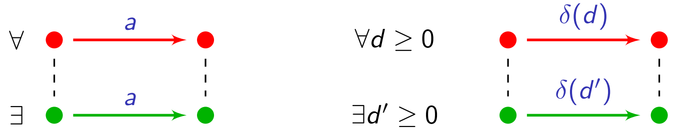
... and vice-versa (swap • and •).

Consequence

$$\forall \quad (\ell_1, v_1) \xrightarrow{d_1,a_1} (\ell_2, v_2) \xrightarrow{d_2,a_2} (\ell_3, v_3) \xrightarrow{d_3,a_3} \quad \cdots$$

Time-abstract bisimulation

This is a relation between • and • such that:



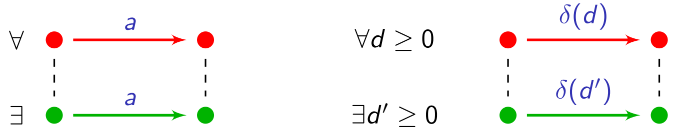
... and vice-versa (swap • and •).

Consequence

$$
\forall \quad
\begin{tikzcd}
(\ell_1, v_1) \arrow[r, "{d_1, a_1}"] \arrow[d] & (\ell_2, v_2) \arrow[r, "{d_2, a_2}"] \arrow[d] & (\ell_3, v_3) \arrow[r, "{d_3, a_3}"] \arrow[d] & \cdots \\
(\ell_1, R_1) \arrow[r, "a_1"] & (\ell_2, R_2) \arrow[r, "a_2"] & (\ell_3, R_3) \arrow[r, "a_3"] & \cdots
\end{tikzcd}
\quad \text{with } v_i \in R_i
$$

Time-abstract bisimulation

This is a relation between • and • such that:



... and vice-versa (swap • and •).

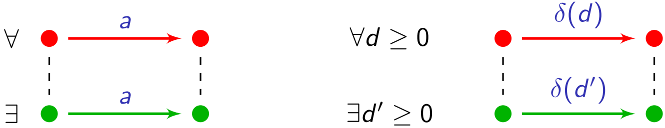
Consequence

$$
\begin{array}{lccccccccl}
\forall & (\ell_1, v_1) & \xrightarrow{d_1, a_1} & (\ell_2, v_2) & \xrightarrow{d_2, a_2} & (\ell_3, v_3) & \xrightarrow{d_3, a_3} & \cdots & & \\
 & \downarrow & & \downarrow & & \downarrow & & & & \\
 & (\ell_1, R_1) & \xrightarrow{a_1} & (\ell_2, R_2) & \xrightarrow{a_2} & (\ell_3, R_3) & \xrightarrow{a_3} & \cdots & & \text{with } v_i \in R_i
\end{array}
$$

$\forall v'_1 \in R_1$

Time-abstract bisimulation

This is a relation between • and • such that:

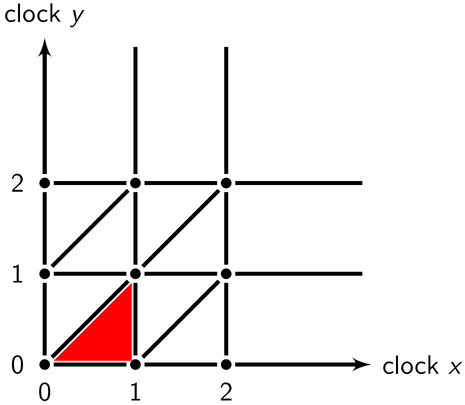


... and vice-versa (swap • and •).

Consequence

$$
\begin{array}{rccccccccl}
\forall & (\ell_1, v_1) & \xrightarrow{d_1, a_1} & (\ell_2, v_2) & \xrightarrow{d_2, a_2} & (\ell_3, v_3) & \xrightarrow{d_3, a_3} & \cdots & & \\
 & \downarrow & & \downarrow & & \downarrow & & & & \\
 & (\ell_1, R_1) & \xrightarrow{a_1} & (\ell_2, R_2) & \xrightarrow{a_2} & (\ell_3, R_3) & \xrightarrow{a_3} & \cdots & & \text{with } v_i \in R_i \\
 & \downarrow & & \downarrow & & \downarrow & & & & \\
\forall v'_1 \in R_1 \; \exists & (\ell_1, v'_1) & \xrightarrow{d'_1, a_1} & (\ell_2, v'_2) & \xrightarrow{d'_2, a_2} & (\ell_3, v'_3) & \xrightarrow{d'_3, a_3} & \cdots & & \text{with } v'_i \in R_i
\end{array}
$$

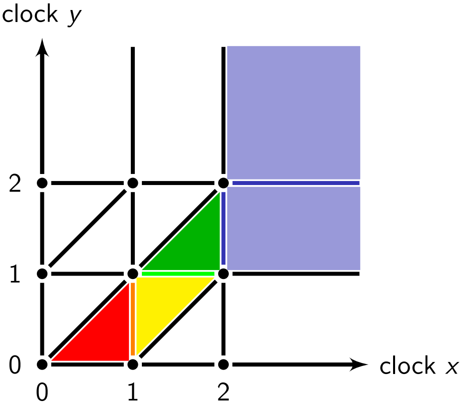
The region abstraction



- region R defined by:

$$\begin{cases} 0 < x < 1 \\ 0 < y < 1 \\ y < x \end{cases}$$

The region abstraction

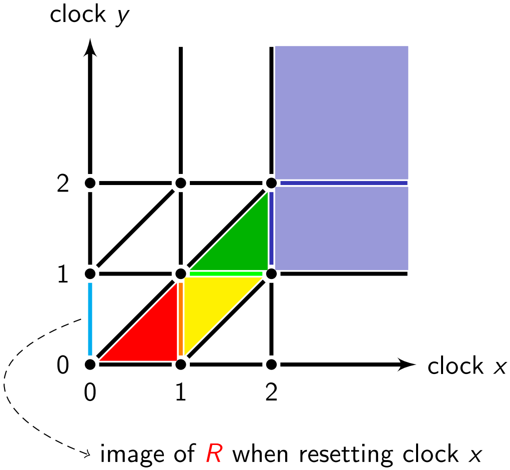


- region R defined by:

$$\begin{cases} 0 < x < 1 \\ 0 < y < 1 \\ y < x \end{cases}$$

- time successors of R

The region abstraction



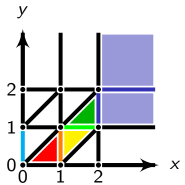
- region R defined by:

$$\begin{cases} 0 < x < 1 \\ 0 < y < 1 \\ y < x \end{cases}$$

- time successors of R

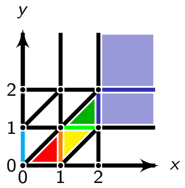
The construction of the region graph

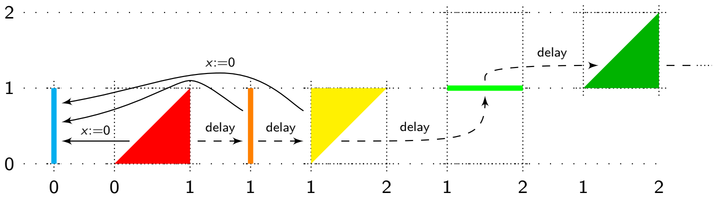
It "mimicks" the behaviours of the clocks.



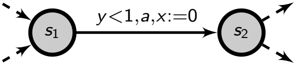
The construction of the region graph

It "mimicks" the behaviours of the clocks.



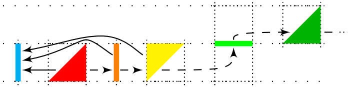


Region automaton $\equiv$ finite bisimulation quotient



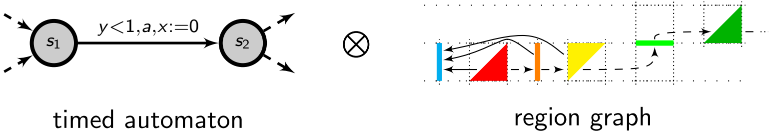
timed automaton

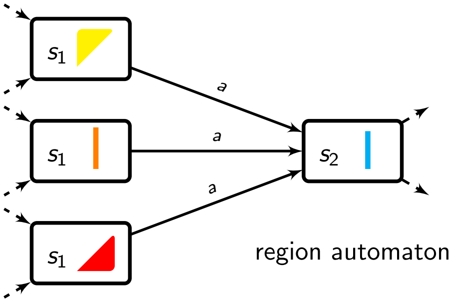
$\otimes$



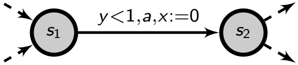
region graph

Region automaton $\equiv$ finite bisimulation quotient



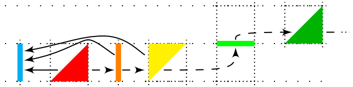


Region automaton ≡ finite bisimulation quotient

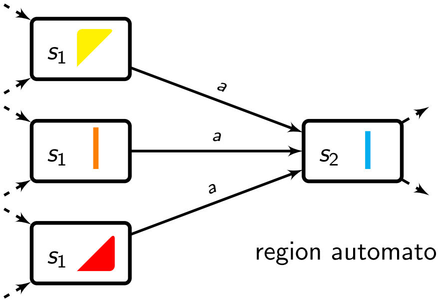


timed automaton

$\otimes$



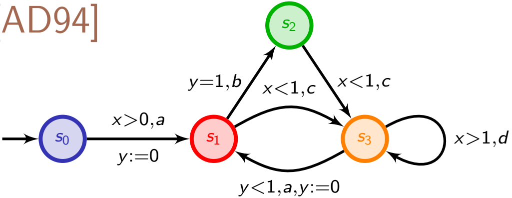
region graph



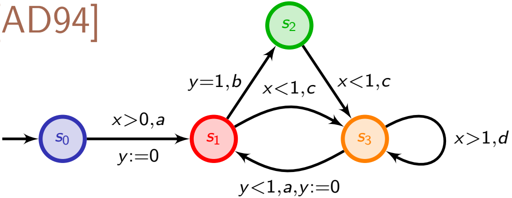
region automaton

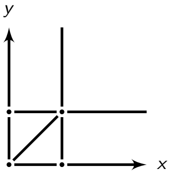
language(reg. aut.) = UNTIME(language(timed aut.))

An example [AD94]

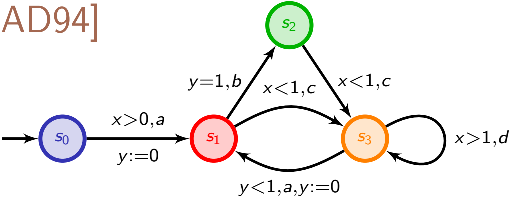


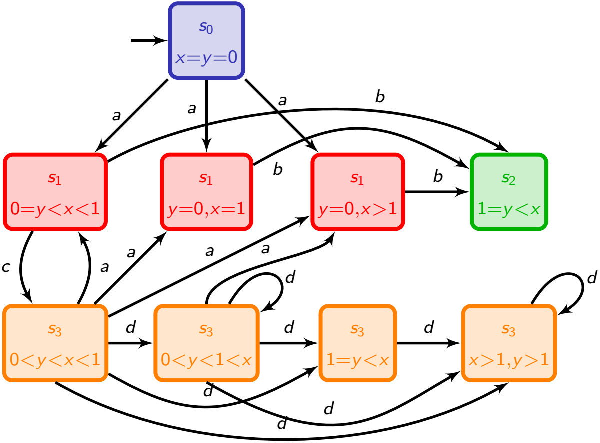
An example [AD94]

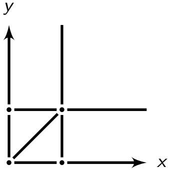


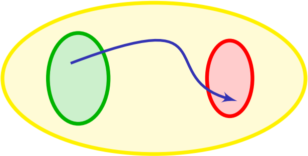


An example [AD94]

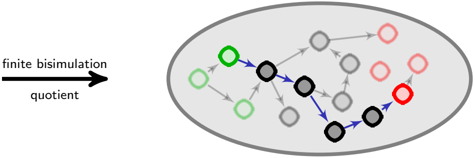




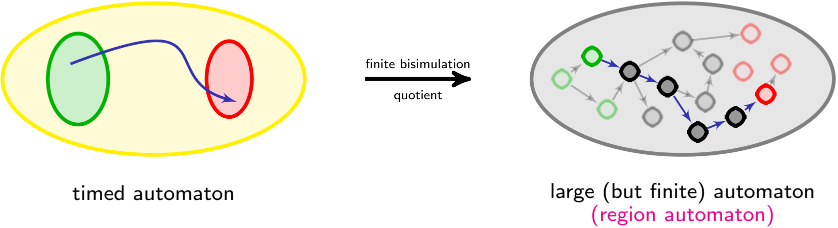




timed automaton

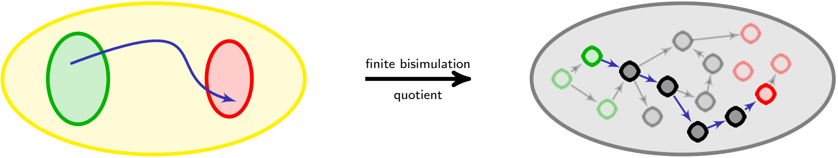


large (but finite) automaton
(region automaton)



- large: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

$$\prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|}$$



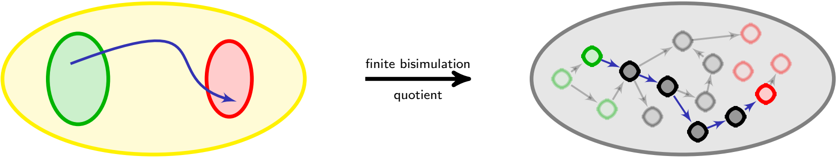
timed automaton

large (but finite) automaton
(region automaton)

- large: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

$$\prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|}$$

- It can be used to check for:
  - reachability/safety properties
  - liveness properties (Büchi/ω-regular properties)
  - LTL properties



timed automaton

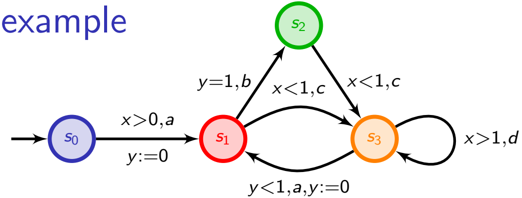
large (but finite) automaton
(region automaton)

- large: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

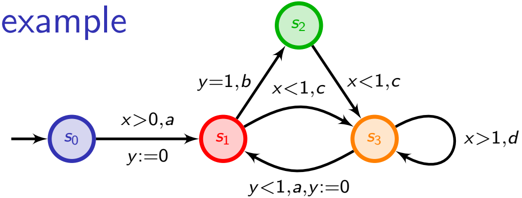
$$\prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|}$$

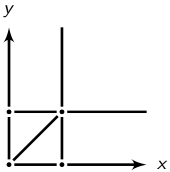
- It can be used to check for:
  - reachability/safety properties
  - liveness properties (Büchi/ω-regular properties)
  - LTL properties
- Problems with Zeno behaviours?
  (infinitely many actions in bounded time)

Back to the example

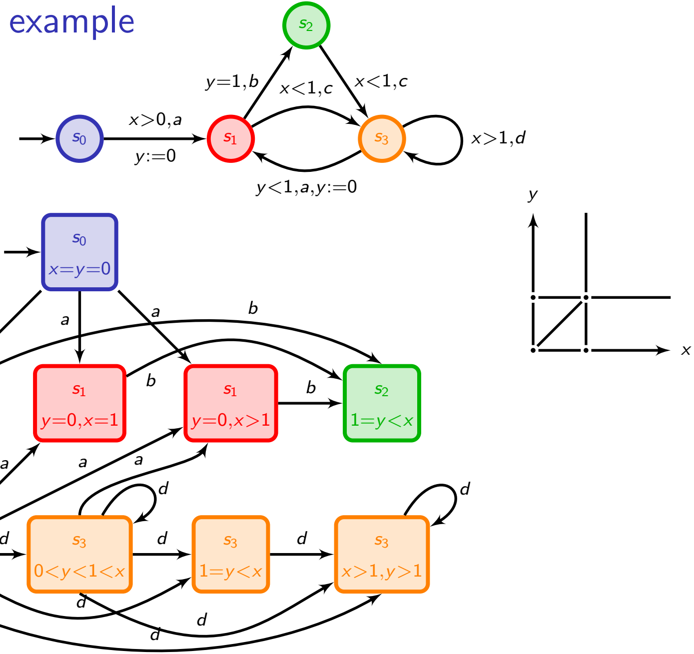


Back to the example

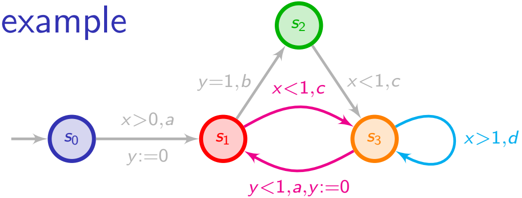


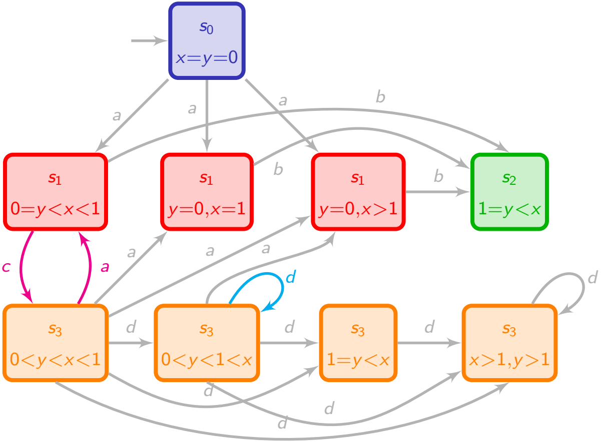


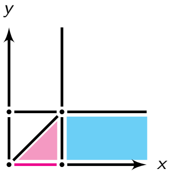
Back to the example



Back to the example

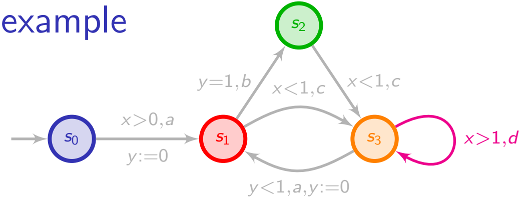


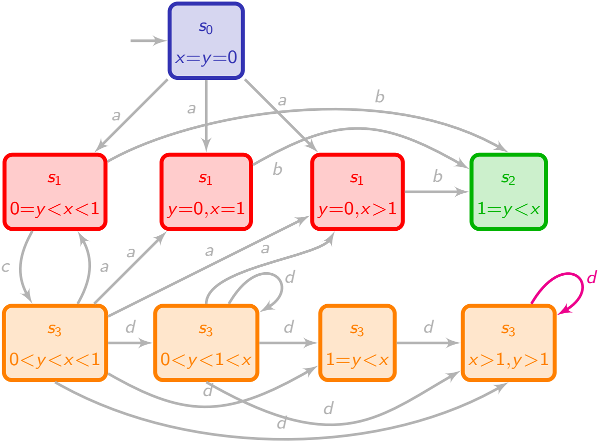


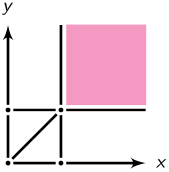


Zeno cycles

Back to the example







Cycles with
non-Zeno behaviours

Complexity issues

Theorem [AD90,AD94]

The emptiness problem for timed automata is decidable and PSPACE-complete. It even holds for two-clock timed automata [FJ13]. It is NLOGSPACE-complete for one-clock timed automata [LMS04].

[AD90] Alur, Dill. Automata for modeling real-time systems *(ICALP'90)*.
[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.
[LMS04] Laroussinie, Markey, Schnoebelen. Model checking timed automata with one or two clocks *(CONCUR'04)*.
[FJ13] Fearnley, Jurdziński. Reachability in two-clock timed automata is PSPACE-complete *(ICALP'13)*.

Complexity issues

Theorem [AD90,AD94]

The emptiness problem for timed automata is decidable and PSPACE-complete. It even holds for two-clock timed automata [FJ13]. It is NLOGSPACE-complete for one-clock timed automata [LMS04].

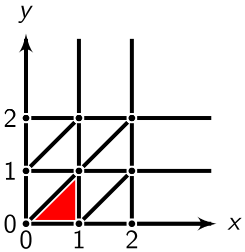
- PSPACE upper bound: guess a path in the region automaton

Complexity issues

Theorem [AD90,AD94]

The emptiness problem for timed automata is decidable and PSPACE-complete. It even holds for two-clock timed automata [FJ13]. It is NLOGSPACE-complete for one-clock timed automata [LMS04].

- PSPACE upper bound: guess a path in the region automaton



region R defined by:

$$\begin{cases} 0 < x < 1 \\ 0 < y < 1 \\ y < x \end{cases}$$

Complexity issues

Theorem [AD90,AD94]

The emptiness problem for timed automata is decidable and PSPACE-complete. It even holds for two-clock timed automata [FJ13]. It is NLOGSPACE-complete for one-clock timed automata [LMS04].

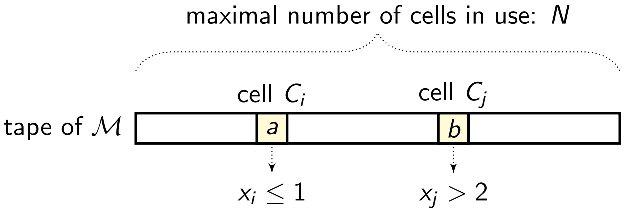
- PSPACE upper bound: guess a path in the region automaton
- PSPACE lower bound: by reduction from a linearly-bounded Turing machine $\mathcal{M}$

Complexity issues

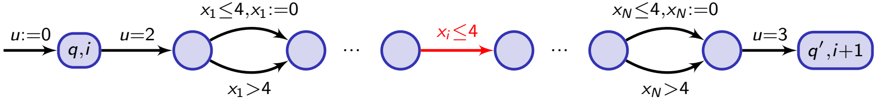
Theorem [AD90,AD94]

The emptiness problem for timed automata is decidable and PSPACE-complete. It even holds for two-clock timed automata [FJ13]. It is NLOGSPACE-complete for one-clock timed automata [LMS04].

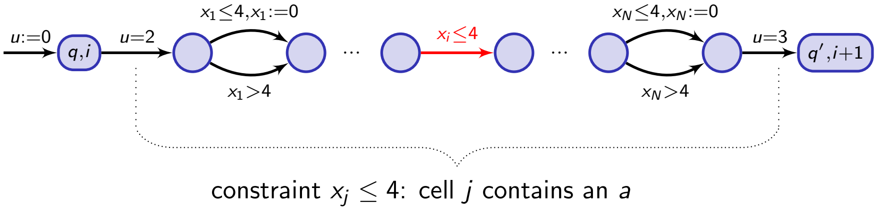
- PSPACE upper bound: guess a path in the region automaton
- PSPACE lower bound: by reduction from a linearly-bounded Turing machine $\mathcal{M}$



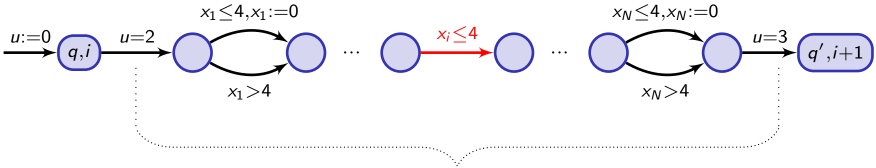
Example of the simulation of a rule $(q, a, b, q', \rightarrow)$:



Example of the simulation of a rule $(q, a, b, q', \rightarrow)$:

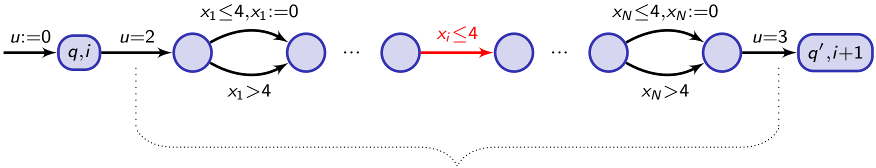


Example of the simulation of a rule $(q, a, b, q', \rightarrow)$:



constraint $x_j \leq 4$: cell j contains an a
constraint $x_j > 4$: cell j contains a b

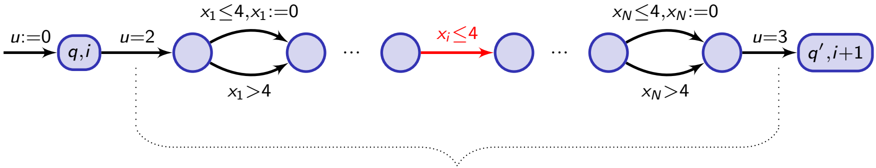
Example of the simulation of a rule $(q, a, b, q', \rightarrow)$:



constraint $x_j \leq 4$: cell j contains an a
constraint $x_j > 4$: cell j contains a b

reset of clock x_j: the new content is an a

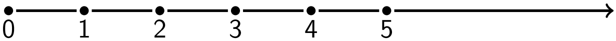
Example of the simulation of a rule $(q, a, b, q', \rightarrow)$:



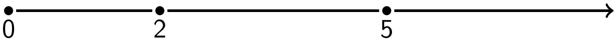
constraint $x_j \leq 4$: cell j contains an a
constraint $x_j > 4$: cell j contains a b

reset of clock x_j: the new content is an a
no reset of clock x_j: the new content is a b

The case of single-clock timed automata



The case of single-clock timed automata



if only constants 0, 2 and 5 are used

Discussion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata

Discussion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed μ-calculus)

Discussion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)

Discussion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
  - o-minimal hybrid systems

Discussion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
  - o-minimal hybrid systems
  - $\cdots$

Discussion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
  - o-minimal hybrid systems
  - $\cdots$
- Note however that it might be hard to prove there is a finite bisimulation quotient!

Outline

1 Introduction
- Timed automata
- Examples

2 Decidability of basic properties
- The region abstraction
- Extensions of timed automata
- Weighted timed automata

3 Implementation and tools

4 Other verification problems
- Equivalence (or preorder) checking
- Verification of timed temporal logics (short)

5 Timed control
- Timed games
- Weighted timed games

6 Conclusion

What if we extend the clock constraints?

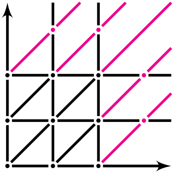
- Diagonal constraints (*i.e.* $x - y \leq 2$)

What if we extend the clock constraints?

- Diagonal constraints (*i.e.* $x - y \leq 2$)
  - decidable (with the same complexity)

What if we extend the clock constraints?

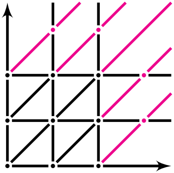
- Diagonal constraints (*i.e.* $x - y \leq 2$)
  - decidable (with the same complexity)



is also a time-abstract bisimulation!

What if we extend the clock constraints?

- Diagonal constraints (*i.e.* $x - y \leq 2$)
  - decidable (with the same complexity)



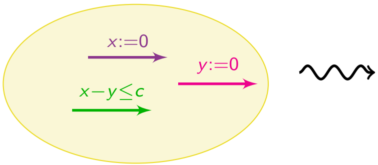
is also a time-abstract bisimulation!

  - they can be removed (at an exponential price)

Removing diagonal constraints

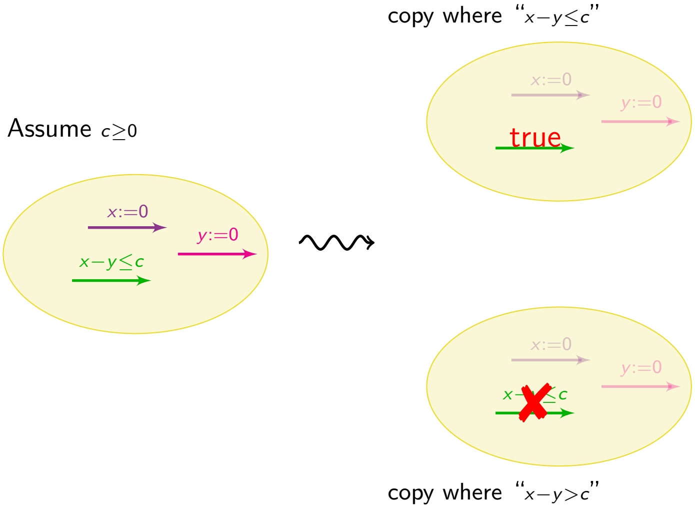
copy where "$x-y \leq c$"

Assume $c \geq 0$



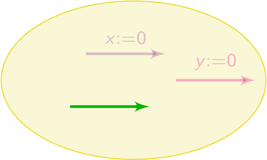
copy where "$x-y > c$"

Removing diagonal constraints

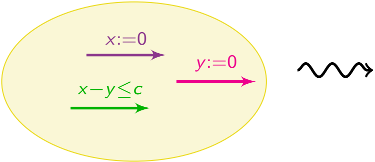


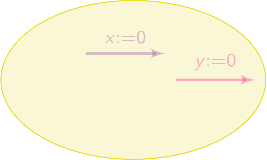
Removing diagonal constraints

copy where "$x - y \leq c$"



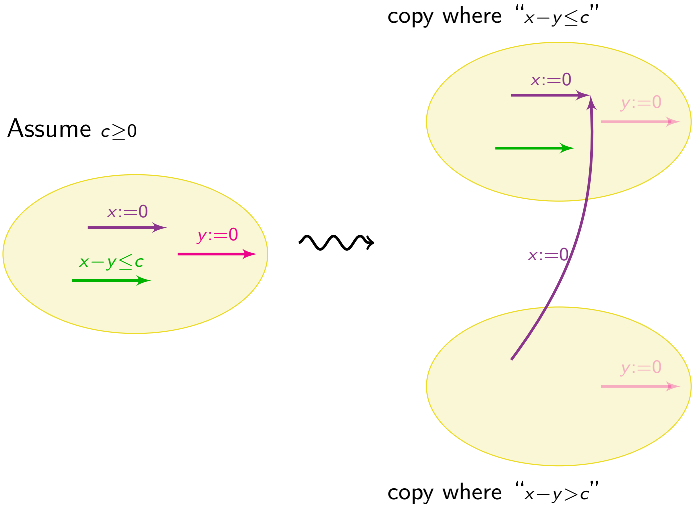
Assume $c \geq 0$





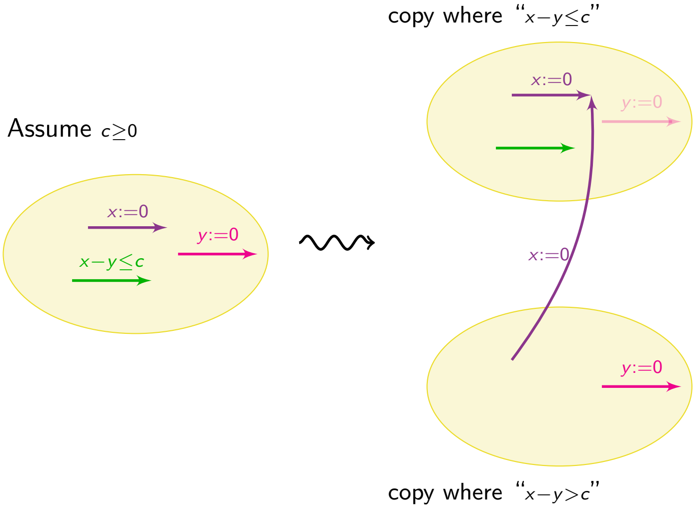
copy where "$x - y > c$"

Removing diagonal constraints

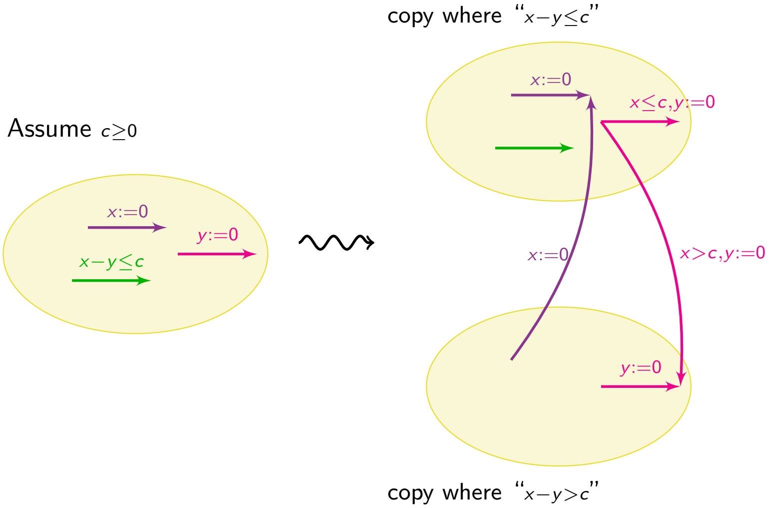


Removing diagonal constraints

Assume $c \geq 0$

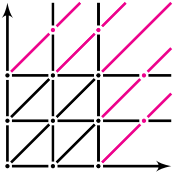


Removing diagonal constraints



What if we extend the clock constraints?

- Diagonal constraints (*i.e.* $x - y \leq 2$)
  - decidable (with the same complexity)

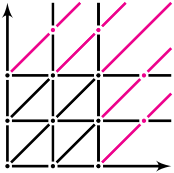


is also a time-abstract bisimulation!

  - they can be removed (at an exponential price)

What if we extend the clock constraints?

- Diagonal constraints (*i.e.* $x - y \leq 2$)
  - decidable (with the same complexity)



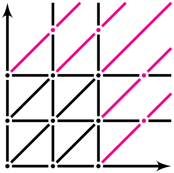
is also a time-abstract bisimulation!

  - they can be removed (at an exponential price)

- Linear constraints (*i.e.* $2x + 3y \sim 5$)

What if we extend the clock constraints?

- Diagonal constraints (*i.e.* $x - y \leq 2$)
  - decidable (with the same complexity)



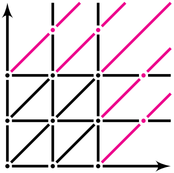
is also a time-abstract bisimulation!

  - they can be removed (at an exponential price)

- Linear constraints (*i.e.* $2x + 3y \sim 5$)
  - undecidable in general

What if we extend the clock constraints?

- Diagonal constraints ($i.e.$ $x - y \leq 2$)
  - decidable (with the same complexity)



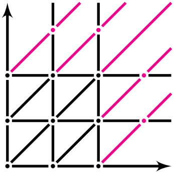
is also a time-abstract bisimulation!

  - they can be removed (at an exponential price)

- Linear constraints ($i.e.$ $2x + 3y \sim 5$)
  - undecidable in general
  - only decidable in few cases

What if we extend the clock constraints?

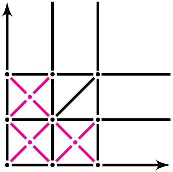
- Diagonal constraints (*i.e.* $x - y \leq 2$)
  - decidable (with the same complexity)



is also a time-abstract bisimulation!

  - they can be removed (at an exponential price)

- Linear constraints (*i.e.* $2x + 3y \sim 5$)
  - undecidable in general
  - only decidable in few cases



is a time-abstract bisimulation (when two clocks x and y and constraints $x + y \sim c$)!

What if we allow more operations on clocks?

- that can be transfer operations (*i.e.* $x := y$), or reinitialization operations (*i.e.* $x := 4$), or ... [BDFP04]

[BDFP04] Bouyer, Dufourd, Fleury, Petit. Updatable Timed Automata *(Theoretical Computer Science)*.

What if we allow more operations on clocks?

- that can be transfer operations (*i.e.* $x := y$), or reinitialization operations (*i.e.* $x := 4$), or ... [BDFP04]

	simple constraints	+ diagonal constraints
$x := c,\ x := y$		
$x := x + 1$		
$x := y + c$		
$x := x - 1$		
$x :< c$		
$x :> c$		
$x :\sim y + c$		
$y + c <: x :< y + d$		
$y + c <: x :< z + d$		

[BDFP04] Bouyer, Dufourd, Fleury, Petit. Updatable Timed Automata *(Theoretical Computer Science)*.

What if we allow more operations on clocks?

- that can be transfer operations (*i.e.* $x := y$), or reinitialization operations (*i.e.* $x := 4$), or ... [BDFP04]

<table>
<tr><th></th><th>simple constraints</th><th>+ diagonal constraints</th></tr>
<tr><td>$x := c$, $x := y$</td><td rowspan="3">decidable</td><td>decidable</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">decidable</td><td>decidable</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>

[BDFP04] Bouyer, Dufourd, Fleury, Petit. Updatable Timed Automata *(Theoretical Computer Science)*.

What if we allow more operations on clocks?

- that can be transfer operations (*i.e.* $x := y$), or reinitialization operations (*i.e.* $x := 4$), or ... [BDFP04]

<table>
<tr><th></th><th>simple constraints</th><th>+ diagonal constraints</th></tr>
<tr><td>$x := c$, $x := y$</td><td rowspan="3">decidable</td><td>decidable</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">decidable</td><td>decidable</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>

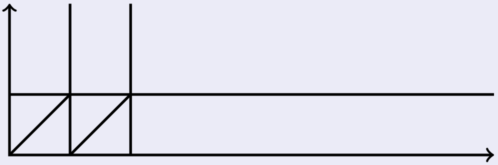
$\rightsquigarrow$ need of being very careful when using more operations on clocks!

[BDFP04] Bouyer, Dufourd, Fleury, Petit. Updatable Timed Automata *(Theoretical Computer Science)*.

Visually...

The example of decrement updates $x := x - 1$

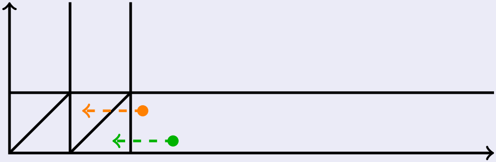
If we want a time-abstract bisimulation...



Visually...

The example of decrement updates $x := x - 1$

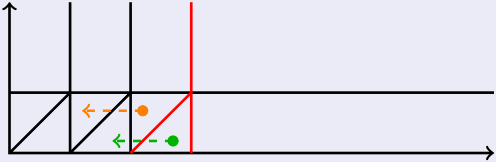
If we want a time-abstract bisimulation...



Visually...

The example of decrement updates $x := x - 1$

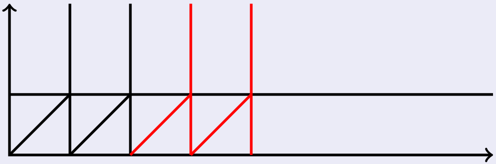
If we want a time-abstract bisimulation...



Visually...

The example of decrement updates $x := x - 1$

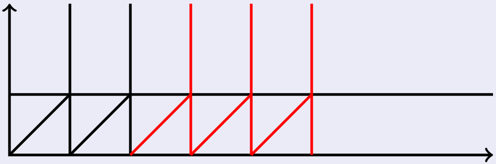
If we want a time-abstract bisimulation...



Visually...

The example of decrement updates $x := x - 1$

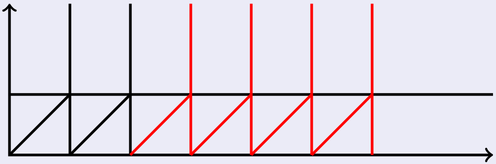
If we want a time-abstract bisimulation...



Visually...

The example of decrement updates $x := x - 1$

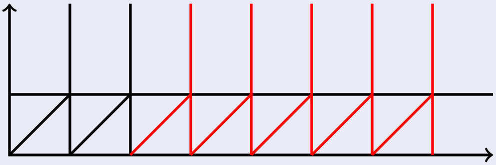
If we want a time-abstract bisimulation...



Visually...

The example of decrement updates $x := x - 1$

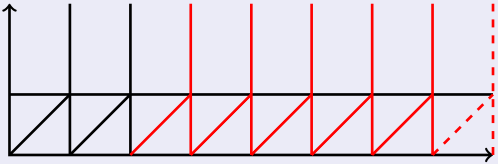
If we want a time-abstract bisimulation...



Visually...

The example of decrement updates $x := x - 1$

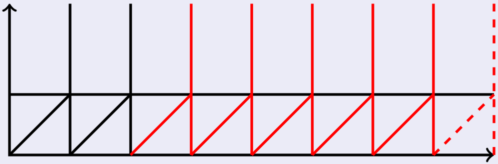
If we want a time-abstract bisimulation...



Visually...

The example of decrement updates $x := x - 1$

If we want a time-abstract bisimulation...



$\rightsquigarrow$ an infinite number of regions

And formally...

- We can simulate a two-counter machine!

And formally...

- We can simulate a two-counter machine!

Definition

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

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

Theorem [Minsky 67]

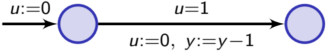
The halting and recurring problems for two counter machines are undecidable.

And formally...

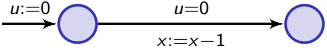
- We can simulate a two-counter machine!
- Clocks x and y store the two counters...

And formally...

- We can simulate a two-counter machine!
- Clocks x and y store the two counters...



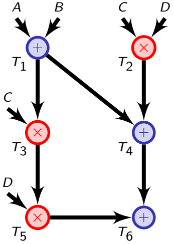
Increment x



Decrement x

Back to the task-graph scheduling problem

Compute $D\times(C\times(A+B))+(A+B)+(C\times D)$ using two processors:

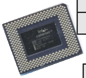


P_1 (fast):

time	
$+$	2 picoseconds
$\times$	3 picoseconds

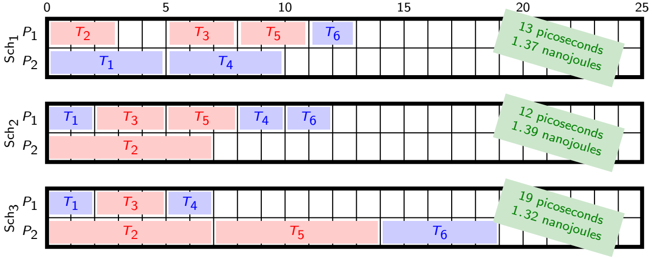
energy	
idle	10 Watt
in use	90 Watts

P_2 (slow):



time	
$+$	5 picoseconds
$\times$	7 picoseconds

energy	
idle	20 Watts
in use	30 Watts



Back to the task-graph scheduling problem

Compute $D\times(C\times(A+B))+(A+B)+(C\times D)$ using two processors:

P_1 (fast):

time	
+	2 picoseconds
×	3 picoseconds

energy	
idle	10 Watt

P_2 (slow):

time	
+	5 picoseconds
×	7 picoseconds

energy	
idle	20 Watts

How to model energy consumption?

A note on hybrid automata (see more on Thursday)

a discrete control (the mode of the system)

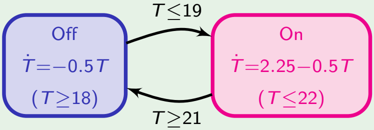
$+$ continuous evolution of the variables within a mode

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

A note on hybrid automata (see more on Thursday)

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

The thermostat example

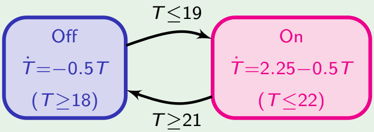


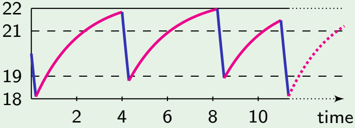
[HKPV95] Henzinger, Kopke, Puri, Varaiya. What's decidable wbout hybrid automata? *(SToC'95)*.

A note on hybrid automata (see more on Thursday)

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

The thermostat example





Theorem [HKPV95]

The reachability problem is undecidable in hybrid automata, even for stopwatch automata.

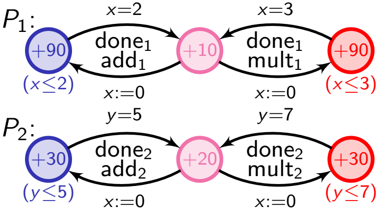
(stopwatch automata: timed automata in which clocks can be stopped)

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

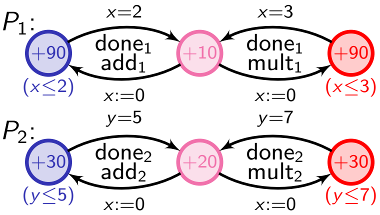
Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

Weighted timed automata



Weighted timed automata



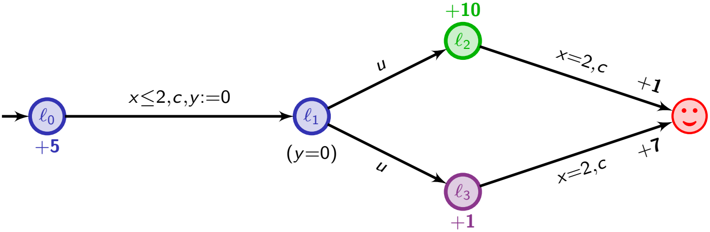
The model of weighted automata

hybrid variables are observer variables (they do not constrain a priori the system)

$\rightsquigarrow$ models energy consumption, bandwidth, price to pay, etc.

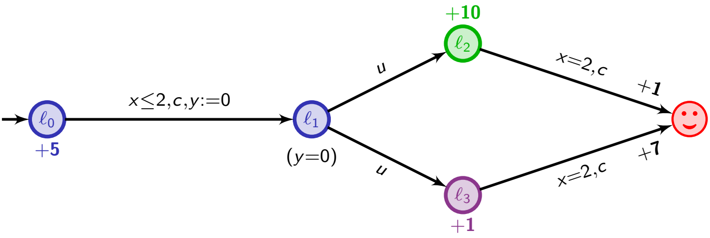
Skip

Weighted/priced timed automata [ALP01,BFH+01]



[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

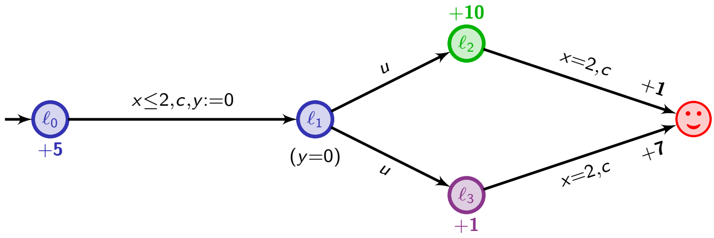
Weighted/priced timed automata [ALP01,BFH+01]



	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

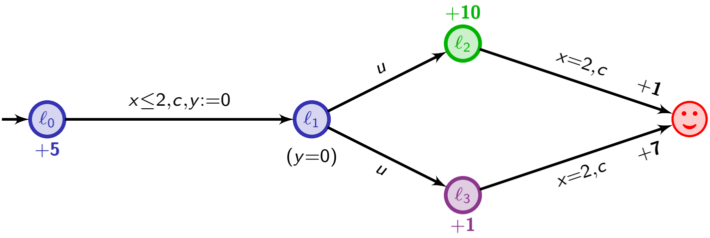


$$
\begin{array}{ccccccccccc}
 & \ell_0 & \xrightarrow{1.3} & \ell_0 & \xrightarrow{c} & \ell_1 & \xrightarrow{u} & \ell_3 & \xrightarrow{0.7} & \ell_3 & \xrightarrow{c} & ☺ \\
x & 0 & & 1.3 & & 1.3 & & 1.3 & & 2 & & \\
y & 0 & & 1.3 & & 0 & & 0 & & 0.7 & &
\end{array}
$$

cost :

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

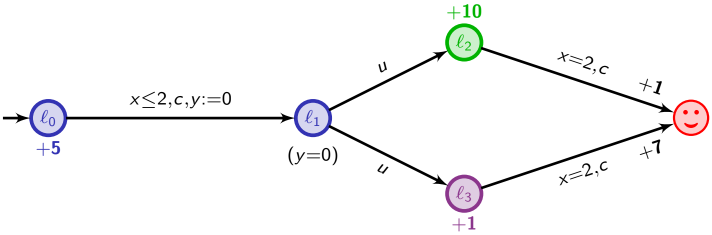


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : 6.5

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

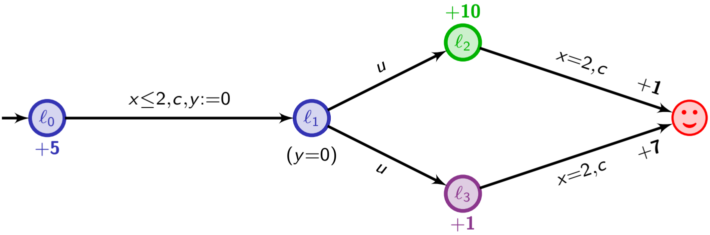


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : 6.5 + 0

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

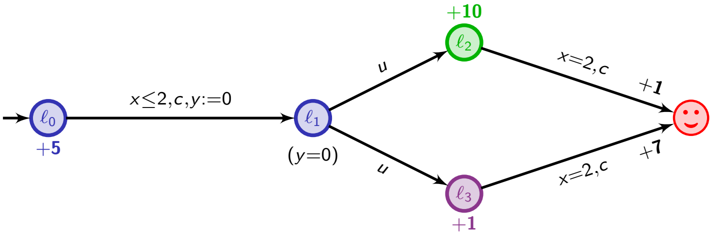


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : $6.5 + 0 + 0$

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

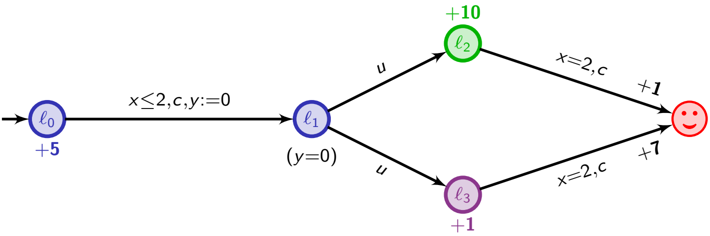


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : 6.5 + 0 + 0 + 0.7

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

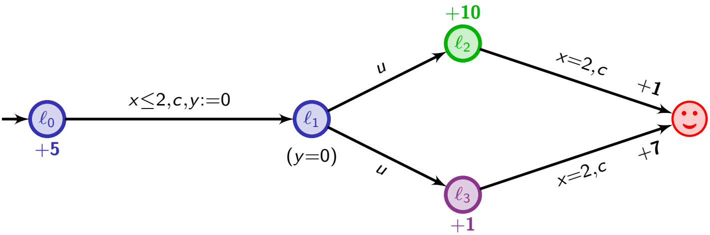


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : 6.5 + 0 + 0 + 0.7 + 7

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

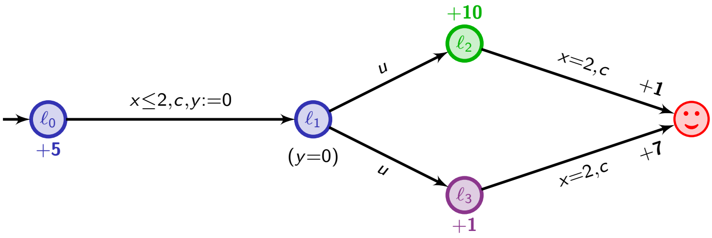


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : $6.5 + 0 + 0 + 0.7 + 7 = 14.2$

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

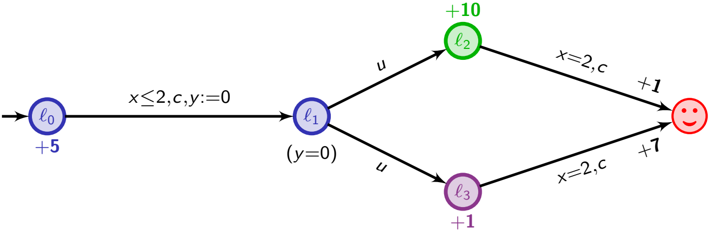
Weighted/priced timed automata [ALP01,BFH+01]



Question: what is the optimal cost for reaching 🙂 ?

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

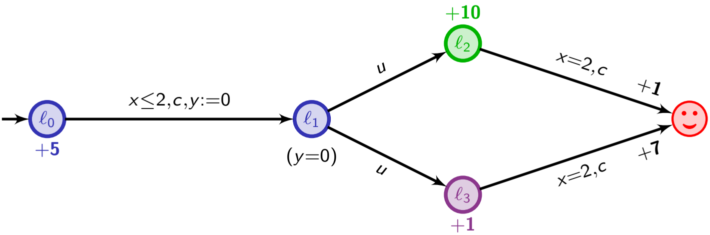


Question: what is the optimal cost for reaching 🙂?

$$5t + 10(2 - t) + 1$$

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

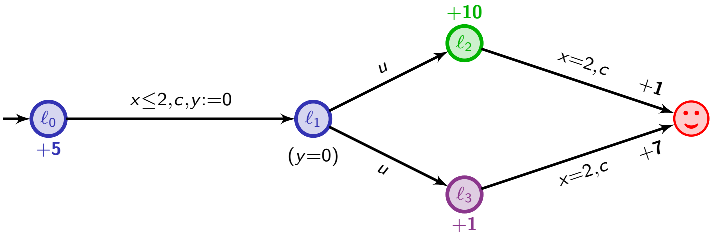


Question: what is the optimal cost for reaching 🙂 ?

$$5t + 10(2-t) + 1 \;,\; 5t + (2-t) + 7$$

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

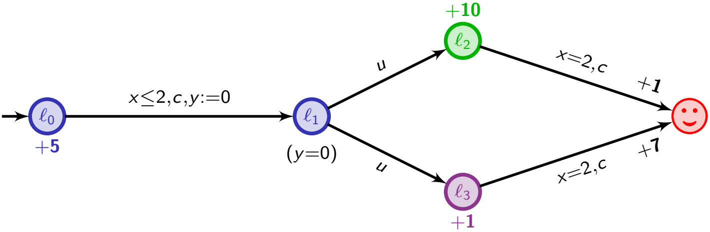


Question: what is the optimal cost for reaching ☺ ?

$$\min\left(\,5t + 10(2-t) + 1\,,\; 5t + (2-t) + 7\,\right)$$

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]

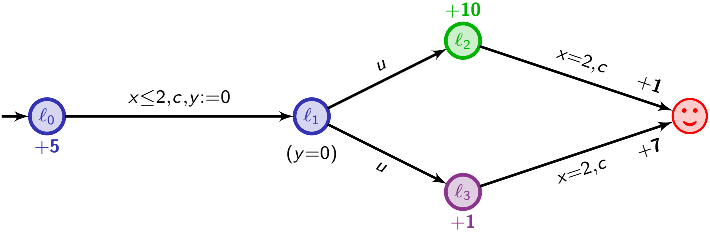


Question: what is the optimal cost for reaching ☺ ?

$$\inf_{0\leq t\leq 2} \min\left(\,5t+10(2-t)+1\,,\;5t+(2-t)+7\,\right)=9$$

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Weighted/priced timed automata [ALP01,BFH+01]



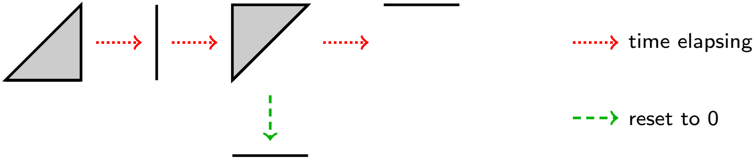
Question: what is the optimal cost for reaching 🙂?

$$\inf_{0 \leq t \leq 2} \min\left(5t + 10(2-t) + 1 \; , \; 5t + (2-t) + 7 \right) = 9$$

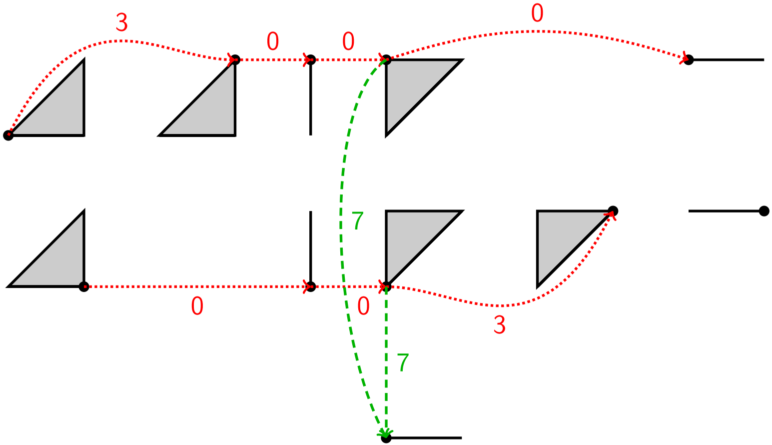
$\leadsto$ *strategy:* leave immediately ℓ_0, go to ℓ_3, and wait there 2 t.u.

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

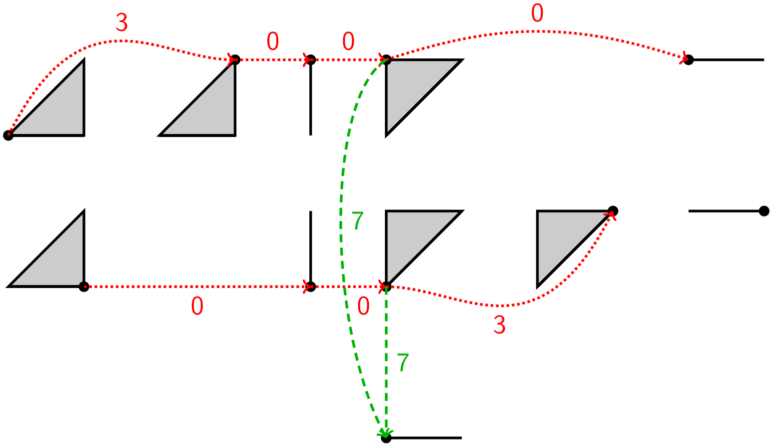
The region abstraction is not fine enough



The corner-point abstraction



The corner-point abstraction



We can somehow discretize the behaviours...

From timed to discrete behaviours

Optimal reachability as a linear programming problem

From timed to discrete behaviours

Optimal reachability as a linear programming problem

$$\circ \xrightarrow{t_1} \circ \xrightarrow{t_2} \circ \xrightarrow{t_3} \circ \xrightarrow{t_4} \circ \xrightarrow{t_5} \circ \quad \cdots$$

From timed to discrete behaviours

Optimal reachability as a linear programming problem

$$\circ \xrightarrow{t_1} \circ \xrightarrow[x \leq 2]{t_2} \circ \xrightarrow{t_3} \circ \xrightarrow{t_4} \circ \xrightarrow{t_5} \circ \quad \cdots \qquad \begin{cases} t_1 + t_2 \leq 2 \\ \end{cases}$$

From timed to discrete behaviours

Optimal reachability as a linear programming problem

$$\circ \xrightarrow[y:=0]{t_1} \circ \xrightarrow[x\leq 2]{t_2} \circ \xrightarrow{t_3} \circ \xrightarrow[y\geq 5]{t_4} \circ \xrightarrow{t_5} \circ \quad \cdots \qquad \begin{cases} t_1+t_2\leq 2 \\ t_2+t_3+t_4\geq 5 \end{cases}$$

From timed to discrete behaviours

Optimal reachability as a linear programming problem

$$\circ \xrightarrow[y:=0]{t_1} \circ \xrightarrow[x\leq 2]{t_2} \circ \xrightarrow{t_3} \circ \xrightarrow[y\geq 5]{t_4} \circ \xrightarrow{t_5} \circ \quad \cdots \qquad \begin{cases} t_1+t_2\leq 2 \\ t_2+t_3+t_4\geq 5 \end{cases}$$

Lemma

Let Z be a bounded constraint as above and f be a function

$$f : (t_1, ..., t_n) \mapsto \sum_{i=1}^{n} c_i t_i + c$$

well-defined on $\overline{Z}$. Then $inf_Z f$ is obtained on the border of $\overline{Z}$ with integer coordinates.

From timed to discrete behaviours

Optimal reachability as a linear programming problem

$$\circ \xrightarrow[y:=0]{t_1} \circ \xrightarrow[x\leq 2]{t_2} \circ \xrightarrow{t_3} \circ \xrightarrow[y\geq 5]{t_4} \circ \xrightarrow{t_5} \circ \quad \cdots \qquad \begin{cases} t_1+t_2\leq 2 \\ t_2+t_3+t_4\geq 5 \end{cases}$$

Lemma

Let Z be a bounded constraint as above and f be a function

$$f : (t_1, \ldots, t_n) \mapsto \sum_{i=1}^{n} c_i t_i + c$$

well-defined on $\overline{Z}$. Then $inf_Z f$ is obtained on the border of $\overline{Z}$ with integer coordinates.

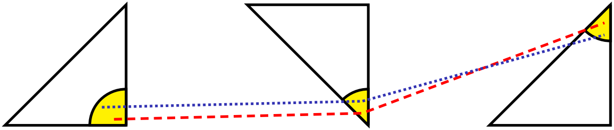
$\rightsquigarrow$ for every finite path π in $\mathcal{A}$, there exists a path Π in $\mathcal{A}_{\mathrm{cp}}$ such that

$$\mathrm{cost}(\Pi) \leq \mathrm{cost}(\pi)$$

[Π is a "corner-point projection" of π]

From discrete to timed behaviours

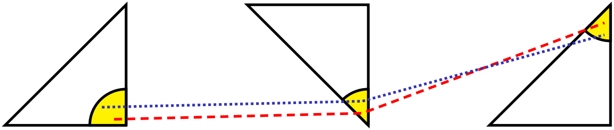
Approximation of abstract paths:



For any path Π of $\mathcal{A}_{\mathsf{cp}}$,

From discrete to timed behaviours

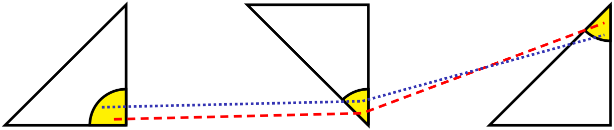
Approximation of abstract paths:



For any path Π of $\mathcal{A}_{\mathsf{cp}}$, for any $\varepsilon > 0$,

From discrete to timed behaviours

Approximation of abstract paths:

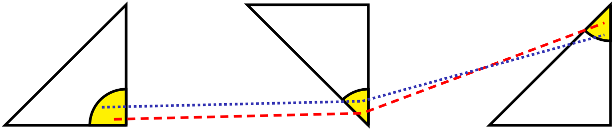


For any path Π of $\mathcal{A}_{\mathsf{cp}}$, for any $\varepsilon > 0$, there exists a path π_ε of $\mathcal{A}$ s.t.

$$\|\Pi - \pi_\varepsilon\|_\infty < \varepsilon$$

From discrete to timed behaviours

Approximation of abstract paths:



For any path Π of $\mathcal{A}_{\mathsf{cp}}$, for any $\varepsilon > 0$, there exists a path π_ε of $\mathcal{A}$ s.t.

$$\|\Pi - \pi_\varepsilon\|_\infty < \varepsilon$$

For every $\eta > 0$, there exists $\varepsilon > 0$ s.t.

$$\|\Pi - \pi_\varepsilon\|_\infty < \varepsilon \Rightarrow |\mathsf{cost}(\Pi) - \mathsf{cost}(\pi_\varepsilon)| < \eta$$

Optimal-cost reachability

Theorem [ALP01,BFH+01,BBBR07]

The optimal-cost reachability problem is decidable (and PSPACE-complete) in weighted timed automata.

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.
[BBBR07] Bouyer, Brihaye, Bruyère, Raskin. On the optimal reachability problem *(Formal Methods in System Design)*.

Further problems of interest

Relevant questions

- Optimization questions:
  - optimal reachability
  - optimal average consumption
  - ...
- Management of resources:
  - a lower bound global constraint (your bank account)
  - a lower and an upper bound global constraint (the tank of your car, the pressure in a pump)
  - ...

$\leadsto$ lots of developments, many open problems

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.
[BBBR07] Bouyer, Brihaye, Bruyère, Raskin. On the optimal reachability problem *(Formal Methods in System Design)*.
[BFLMS08] Bouyer, Fahrenberg, Larsen, Markey, Srba. Infinite runs in weighted timed automata with energy constraints *(FORMATS'08)*.
[BFLM10] Bouyer, Fahrenberg, Larsen, Markey. Timed automata with observers under energy constraints *(HSCC'10)*.
[BFM12] Bouyer, Larsen, Markey. Lower-bound constrained runs in weighted timed automata *(QEST'12)*.

Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

What about the practice?

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

What about the practice?

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

What do we need?

- Need of a symbolic representation:

Finite representation of infinite sets of configurations

What about the practice?

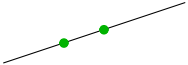
- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:

Finite representation of infinite sets of configurations

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



What about the practice?

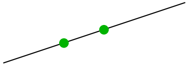
- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:

Finite representation of infinite sets of configurations

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



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

What about the practice?

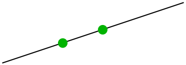
- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:

Finite representation of infinite sets of configurations

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



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

What about the practice?

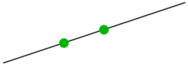
- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

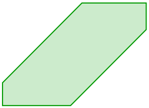
- Need of a symbolic representation:

Finite representation of infinite sets of configurations

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



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



What about the practice?

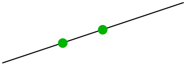
- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

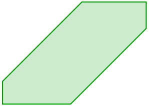
- Need of a symbolic representation:

Finite representation of infinite sets of configurations

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



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



What about the practice?

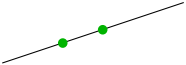
- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

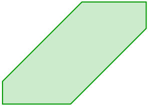
- Need of a symbolic representation:

  Finite representation of infinite sets of configurations

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



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

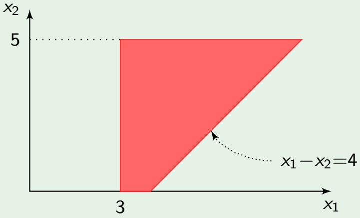


- Need of abstractions, heuristics, etc...

Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

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



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

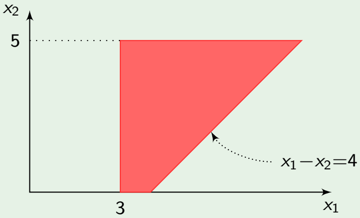
DBM: Difference Bound Matrice [BM83,Dill89]

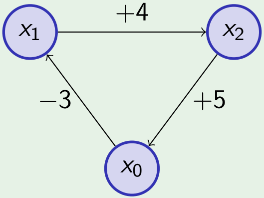
[BM83] Berthomieu, Menasche. An enumerative approach for analyzing time Petri nets *World Comupter Congress*.
[Dill89] Dill. Timing assumptions and verification of finite-state concurrent systems *(Automatic Verification Methods for Finite State Systems)*.

Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

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





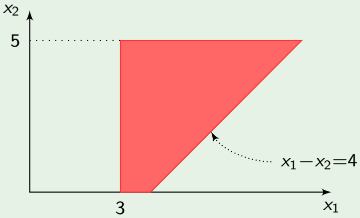
DBM: Difference Bound Matrice [BM83,Dill89]

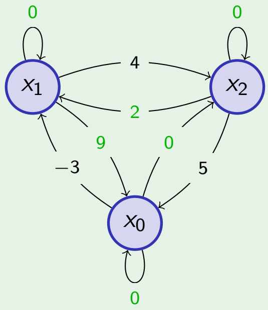
[BM83] Berthomieu, Menasche. An enumerative approach for analyzing time Petri nets *World Comupter Congress*.
[Dill89] Dill. Timing assumptions and verification of finite-state concurrent systems *(Automatic Verification Methods for Finite State Systems)*.

Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

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





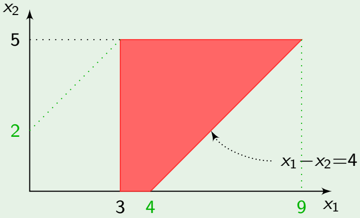
DBM: Difference Bound Matrice [BM83,Dill89]

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

Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

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



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

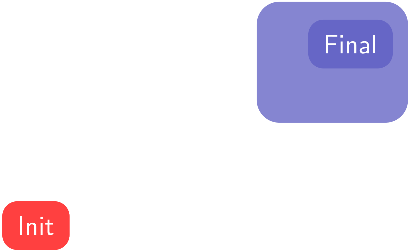
"normal form"

DBM: Difference Bound Matrice [BM83,Dill89]

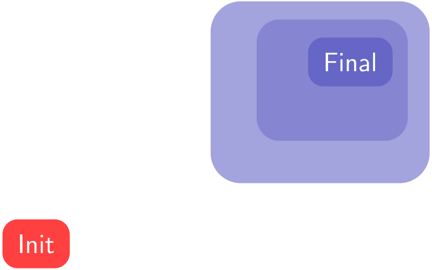
[BM83] Berthomieu, Menasche. An enumerative approach for analyzing time Petri nets *World Comupter Congress*.
[Dill89] Dill. Timing assumptions and verification of finite-state concurrent systems *(Automatic Verification Methods for Finite State Systems)*.

Backward computation

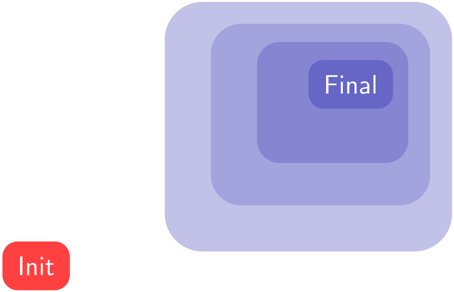
Backward computation



Backward computation

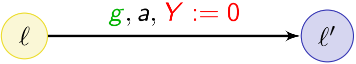


Backward computation



Backward computation

Note on the backward analysis of timed automata

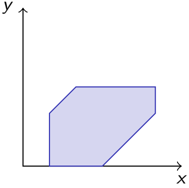


$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g}$ $\qquad Z$

Note on the backward analysis of timed automata

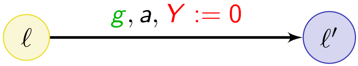
$$\ell \xrightarrow{g,\, a,\, Y := 0} \ell'$$

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

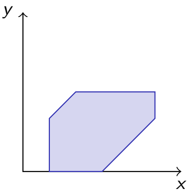


Z

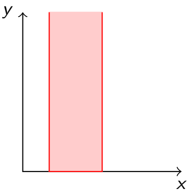
Note on the backward analysis of timed automata



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

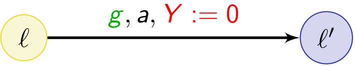


Z

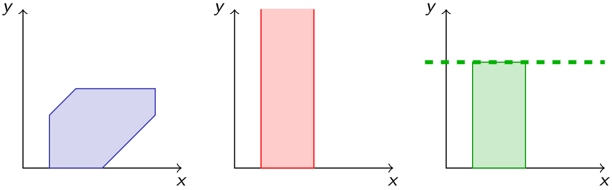


$[y{\leftarrow}0]^{-1}(Z{\cap}(y{=}0))$

Note on the backward analysis of timed automata

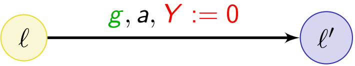


$$\overleftarrow{[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g} \qquad Z$$

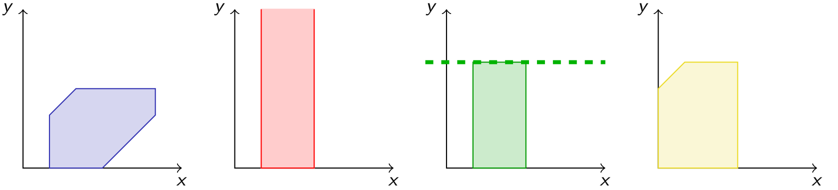


Z $\qquad$ $[y{\leftarrow}0]^{-1}(Z{\cap}(y{=}0))$

Note on the backward analysis of timed automata



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



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

Note on the backward analysis (cont.)

☺ All previous operations can be computed using DBMs!

Note on the backward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints

Note on the backward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- inverse reset w.r.t y: relax constraints on y (on a DBM on normal form)

Note on the backward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- inverse reset w.r.t y: relax constraints on y (on a DBM on normal form)
- past: relax lower bounds (on a DBM on normal form)

Note on the backward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- inverse reset w.r.t y: relax constraints on y (on a DBM on normal form)
- past: relax lower bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

Note on the backward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- inverse reset w.r.t y: relax constraints on y (on a DBM on normal form)
- past: relax lower bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

The backward computation terminates

Because of the bisimulation property of the region abstraction:

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

Note on the backward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- inverse reset w.r.t y: relax constraints on y (on a DBM on normal form)
- past: relax lower bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

The backward computation terminates

Because of the bisimulation property of the region abstraction:

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

Let R be a region. Assume:

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

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

Note on the backward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- inverse reset w.r.t y: relax constraints on y (on a DBM on normal form)
- past: relax lower bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

The backward computation terminates

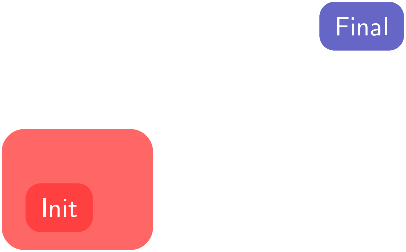
Because of the bisimulation property of the region abstraction:

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

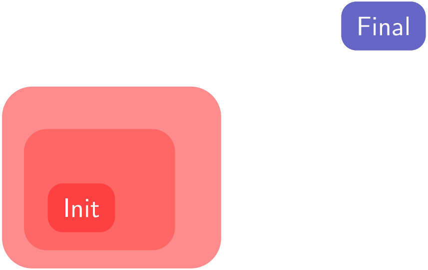
However the backward computation is not appropriate to manipulate other variables (think for instance of assignment $i := j.k + l$)

Forward computation

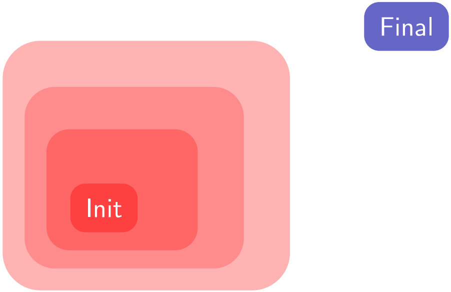
Forward computation



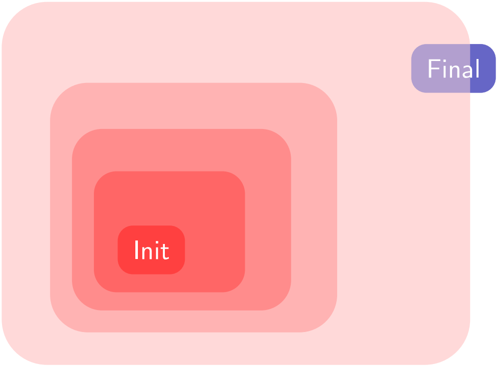
Forward computation



Forward computation



Forward computation

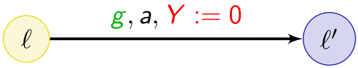


Forward analysis of timed automata

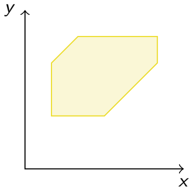
$$\ell \xrightarrow{g,\, a,\, Y := 0} \ell'$$

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

Forward analysis of timed automata

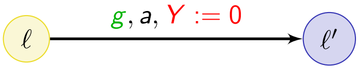


Z $\qquad$ $[Y \leftarrow 0](\overrightarrow{Z} \cap g)$

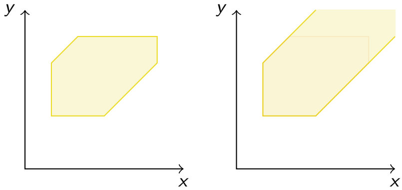


Z

Forward analysis of timed automata

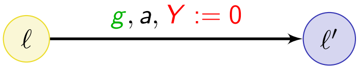


Z $[Y \leftarrow 0](\overrightarrow{Z} \cap g)$

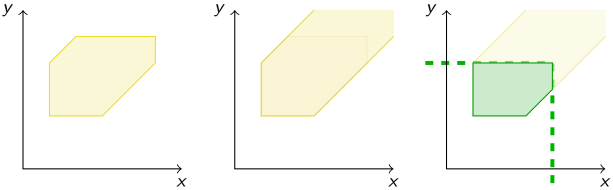


Z $\overrightarrow{Z}$

Forward analysis of timed automata



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

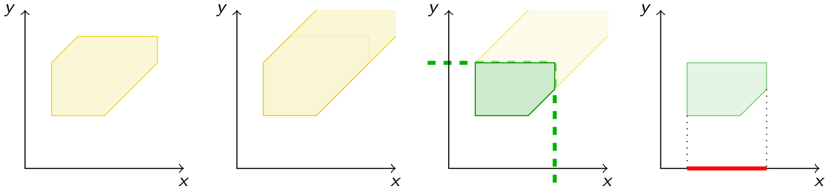


Z $\qquad$ $\overrightarrow{Z}$ $\qquad$ $\overrightarrow{Z} \cap g$

Forward analysis of timed automata

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

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



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

Note on the forward analysis (cont.)

☺ All previous operations can be computed using DBMs!

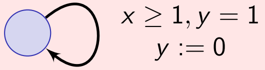
- intersection: take the minimum of the two constraints
- reset w.r.t y: set constraint if y to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

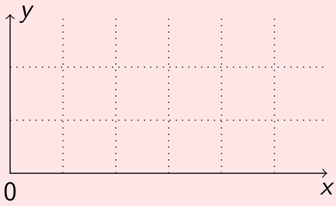
Note on the forward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- reset w.r.t y: set constraint if y to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

☹ The forward computation may not terminate...



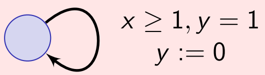


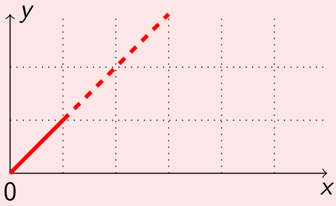
Note on the forward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- reset w.r.t y: set constraint if y to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

☹ The forward computation may not terminate...



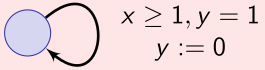


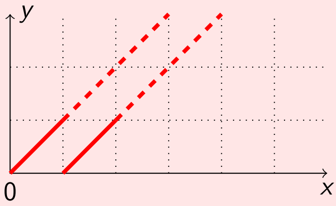
Note on the forward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- reset w.r.t y: set constraint if y to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

☹ The forward computation may not terminate...



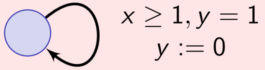


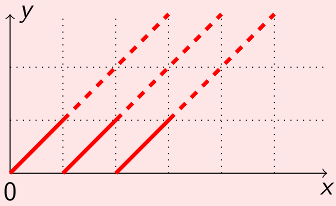
Note on the forward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- reset w.r.t y: set constraint if y to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

☹ The forward computation may not terminate...



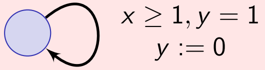


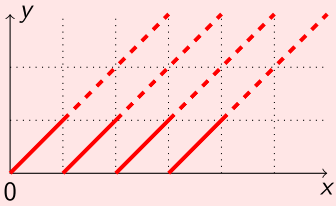
Note on the forward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- reset w.r.t y: set constraint if y to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

☹ The forward computation may not terminate...



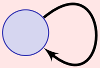


Note on the forward analysis (cont.)

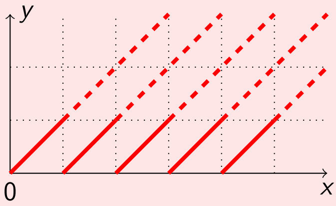
☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- reset w.r.t y: set constraint if y to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

☹ The forward computation may not terminate...



$$x \geq 1, y = 1$$
$$y := 0$$

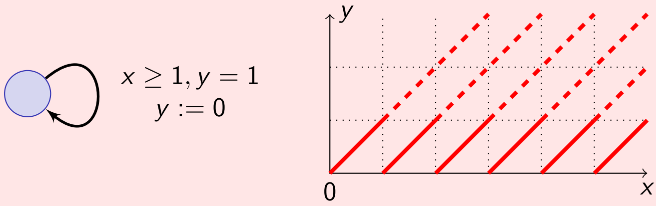


Note on the forward analysis (cont.)

☺ All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- reset w.r.t y: set constraint if y to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

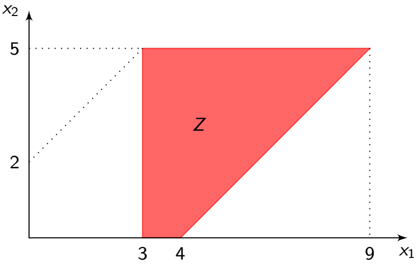
☹ The forward computation may not terminate...



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

An abstraction: the extrapolation operator

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

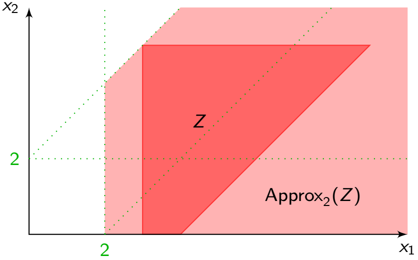


$$\begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix}$$

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

An abstraction: the extrapolation operator

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



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

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

The extrapolation: correctness

Theorem

The algorithm using the extrapolation w.r.t. the maximal constant is correct for timed automata with only rectangular constraints.
Note: the hypothesis on the constraints is crucial.

[Bou04] Bouyer. Forward analysis of updatable timed automata *(Formal Methods in System Design)*.

The extrapolation: correctness

Theorem

The algorithm using the extrapolation w.r.t. the maximal constant is correct for timed automata with only rectangular constraints.
Note: the hypothesis on the constraints is crucial.

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

[Bou04] Bouyer. Forward analysis of updatable timed automata *(Formal Methods in System Design)*.

Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded contraints [BBLP03,BBLP06,HSW12,HSW13]

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification *(TACAS'03)*.
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata *(TACAS'04)*.
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata *(International Journal on Software Tools for Technology Transfer)*.
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal *(FORMATS'03)*.
[DHLP06] David, Håkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction *(FORMATS'06)*.
[HSW12] Herbreteau, Srivathsan, Walukiewicz. Better abstractions for timed automata *(LICS'12)*.
[HSW13] Herbreteau, Srivathsan, Walukiewicz. Lazy abstractions for timed automata *(CAV'13)*.

Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded contraints [BBLP03,BBLP06,HSW12,HSW13]
- over-approximations can be used
  - convex-hull

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification *(TACAS'03)*.
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata *(TACAS'04)*.
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata *(International Journal on Software Tools for Technology Transfer)*.
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal *(FORMATS'03)*.
[DHLP06] David, Håkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction *(FORMATS'06)*.
[HSW12] Herbreteau, Srivathsan, Walukiewicz. Better abstractions for timed automata *(LICS'12)*.
[HSW13] Herbreteau, Srivathsan, Walukiewicz. Lazy abstractions for timed automata *(CAV'13)*.

Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded contraints [BBLP03,BBLP06,HSW12,HSW13]
- over-approximations can be used
  - convex-hull
- heuristics can be added
  - order for exploration
  - symmetry reduction [HBL+03]

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification *(TACAS'03)*.
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata *(TACAS'04)*.
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata *(International Journal on Software Tools for Technology Transfer)*.
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal *(FORMATS'03)*.
[DHLP06] David, Håkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction *(FORMATS'06)*.
[HSW12] Herbreteau, Srivathsan, Walukiewicz. Better abstractions for timed automata *(LICS'12)*.
[HSW13] Herbreteau, Srivathsan, Walukiewicz. Lazy abstractions for timed automata *(CAV'13)*.

Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded contraints [BBLP03,BBLP06,HSW12,HSW13]
- over-approximations can be used
  - convex-hull
- heuristics can be added
  - order for exploration
  - symmetry reduction [HBL+03]
- the representation of zones can be improved [DHLP06]

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification *(TACAS'03)*.
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata *(TACAS'04)*.
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata *(International Journal on Software Tools for Technology Transfer)*.
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal *(FORMATS'03)*.
[DHLP06] David, Håkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction *(FORMATS'06)*.
[HSW12] Herbreteau, Srivathsan, Walukiewicz. Better abstractions for timed automata *(LICS'12)*.
[HSW13] Herbreteau, Srivathsan, Walukiewicz. Lazy abstractions for timed automata *(CAV'13)*.

Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

Outline

1 Introduction
- Timed automata
- Examples

2 Decidability of basic properties
- The region abstraction
- Extensions of timed automata
- Weighted timed automata

3 Implementation and tools

4 Other verification problems
- Equivalence (or preorder) checking
- Verification of timed temporal logics (short)

5 Timed control
- Timed games
- Weighted timed games

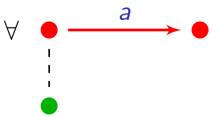
6 Conclusion

Strong timed (bi)simulation

This is a relation between • and • such that:

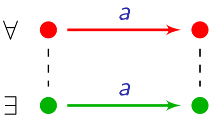
Strong timed (bi)simulation

This is a relation between • and • such that:



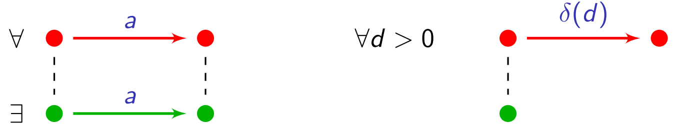
Strong timed (bi)simulation

This is a relation between • and • such that:



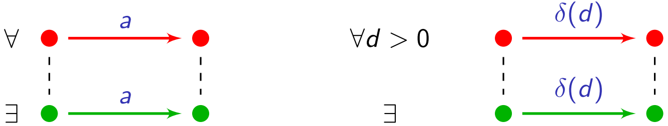
Strong timed (bi)simulation

This is a relation between • and • such that:



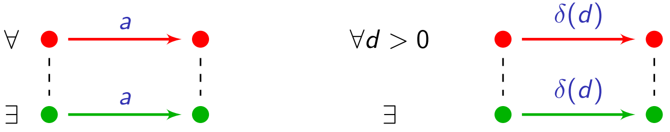
Strong timed (bi)simulation

This is a relation between • and • such that:



Strong timed (bi)simulation

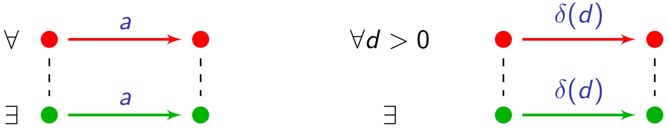
This is a relation between • and • such that:



... and vice-versa (swap • and •) for the bisimulation relation.

Strong timed (bi)simulation

This is a relation between • and • such that:



... and vice-versa (swap • and •) for the bisimulation relation.

Theorem

Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.

(see later for a simple proof of the upper bound)

Language (or trace) equivalence and inclusion

Question

Given two timed automata $\mathcal{A}$ and $\mathcal{B}$, is $L(\mathcal{A}) = L(\mathcal{B})$ (resp. $L(\mathcal{A}) \subseteq L(\mathcal{B})$)?

Language (or trace) equivalence and inclusion

Question

Given two timed automata $\mathcal{A}$ and $\mathcal{B}$, is $L(\mathcal{A}) = L(\mathcal{B})$ (resp. $L(\mathcal{A}) \subseteq L(\mathcal{B})$)?

Theorem [AD90,AD94]

Language equivalence and language inclusion are undecidable in timed automata.

... as a special case of the universality problem (are all timed words accepted by the automaton?).

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

Language (or trace) equivalence and inclusion

Question

Given two timed automata $\mathcal{A}$ and $\mathcal{B}$, is $L(\mathcal{A}) = L(\mathcal{B})$ (resp. $L(\mathcal{A}) \subseteq L(\mathcal{B})$)?

Theorem [AD90,AD94]

Language equivalence and language inclusion are undecidable in timed automata.

... as a special case of the universality problem (are all timed words accepted by the automaton?).

$\rightsquigarrow$ Proof by reduction from the recurring problem of a two-counter machine

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

Undecidability of universality

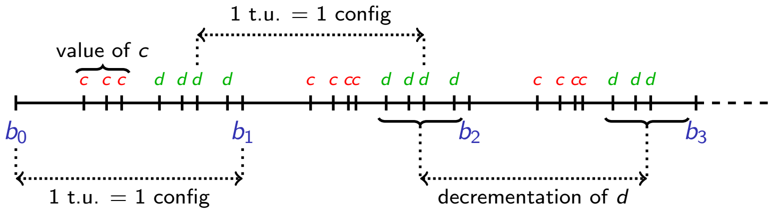
Theorem [AD90,AD94]

Universality of timed automata is undecidable.

Undecidability of universality

Theorem [AD90,AD94]

Universality of timed automata is undecidable.

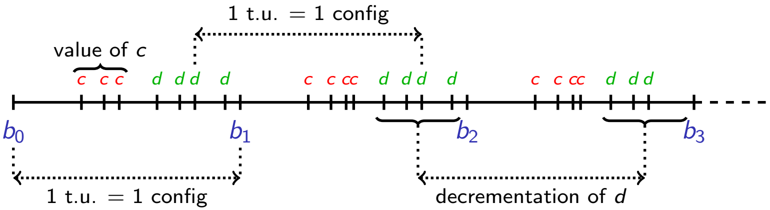


- one configuration is encoded in one time unit
- number of c's: value of counter c
- number of d's: value of counter d
- one time unit between two corresponding c's (resp. d's)

Undecidability of universality

Theorem [AD90,AD94]

Universality of timed automata is undecidable.

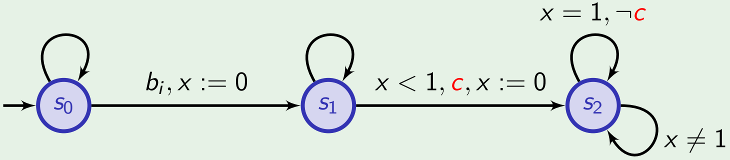


- one configuration is encoded in one time unit
- number of c's: value of counter c
- number of d's: value of counter d
- one time unit between two corresponding c's (resp. d's)

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

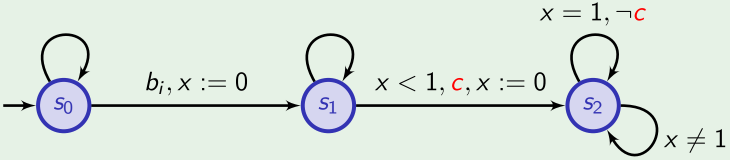
Example

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



Example

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



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

Bad consequences

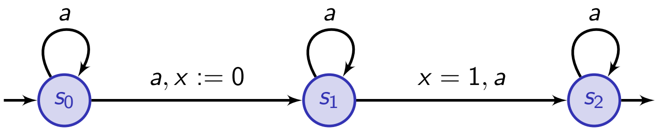
- Language inclusion is undecidable [AD90,AD94]
  (Bad news for the application to verification)
- Complementability is undecidable [Tri03,Fin06]
- ...

[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata *(FORMATS'03)*.
[Fin06] Finkel. Undecidable problems about timed automata *(FORMATS'06)*.

Bad consequences

- Language inclusion is undecidable [AD90,AD94]
  (Bad news for the application to verification)
- Complementability is undecidable [Tri03,Fin06]
- ...

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

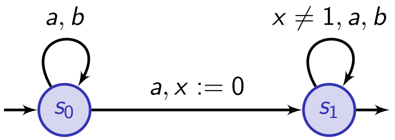


[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata *(FORMATS'03)*.
[Fin06] Finkel. Undecidable problems about timed automata *(FORMATS'06)*.

Bad consequences

- Language inclusion is undecidable [AD90,AD94]
  (Bad news for the application to verification)
- Complementability is undecidable [Tri03,Fin06]
- ...

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

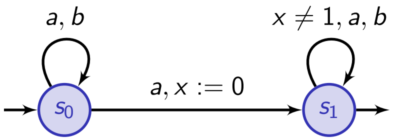


[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata *(FORMATS'03)*.
[Fin06] Finkel. Undecidable problems about timed automata *(FORMATS'06)*.
[AM04] Alur, Madhusudan. Decision problems for timed automata: A survey *(SFM-04:RT))*.

Bad consequences

- Language inclusion is undecidable [AD90,AD94]
  (Bad news for the application to verification)
- Complementability is undecidable [Tri03,Fin06]
- ...

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



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

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

Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

Timed temporal logics

- Branching-time: TCTL

$$\text{TCTL} \ni \varphi \ ::= \ a \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{E}\varphi\, \mathbf{U}_I\, \varphi$$

where I is an interval with integral bounds.

- Linear-time: MTL [Koy90]

$$\text{MTL} \ni \varphi \ ::= \ a \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi\, \mathbf{U}_I\, \varphi$$

where I is an interval with integral bounds.

- Alternative: add variables (clocks) to the logics, e.g. TPTL

[Koy90] Koymans. Specifying real-time properties with metric temporal logic (Real-time systems, 1990).

Timed temporal logics

- Branching-time: TCTL

$$\mathsf{TCTL} \ni \varphi ::= a \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{E}\varphi\, \mathbf{U}_I\, \varphi$$

where I is an interval with integral bounds.

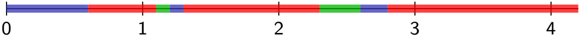
- Linear-time: MTL [Koy90]

$$\mathsf{MTL} \ni \varphi ::= a \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi\, \mathbf{U}_I\, \varphi$$

where I is an interval with integral bounds.

- Alternative: add variables (clocks) to the logics, e.g. TPTL

$\leadsto$ interpreted over **signals**



[Koy90] Koymans. Specifying real-time properties with metric temporal logic (Real-time systems, 1990).

Timed temporal logics

- Branching-time: TCTL

$$\text{TCTL} \ni \varphi ::= a \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{E}\varphi\,\mathbf{U}_I\,\varphi$$

where I is an interval with integral bounds.

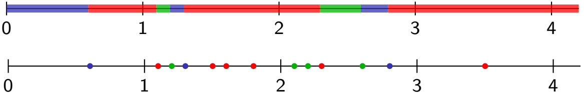
- Linear-time: MTL [Koy90]

$$\text{MTL} \ni \varphi ::= a \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi\,\mathbf{U}_I\,\varphi$$

where I is an interval with integral bounds.

- Alternative: add variables (clocks) to the logics, e.g. TPTL

$\rightsquigarrow$ interpreted over **signals** (or over timed words)



$$(\bullet, .6)(\bullet, 1.1)(\bullet, 1.2)(\bullet, 1.3)\ldots$$

[Koy90] Koymans. Specifying real-time properties with metric temporal logic (Real-time systems, 1990).

Timed temporal logics

- Branching-time: TCTL

$$\mathsf{TCTL} \ni \varphi \ ::= \ a \ | \ \neg\varphi \ | \ \varphi \vee \varphi \ | \ \varphi \wedge \varphi \ | \ \mathbf{E}\varphi \, \mathbf{U}_I \, \varphi$$

  where I is an interval with integral bounds.

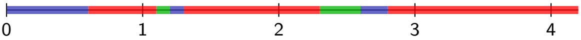
- Linear-time: MTL [Koy90]

$$\mathsf{MTL} \ni \varphi \ ::= \ a \ | \ \neg\varphi \ | \ \varphi \vee \varphi \ | \ \varphi \wedge \varphi \ | \ \varphi \, \mathbf{U}_I \, \varphi$$

  where I is an interval with integral bounds.

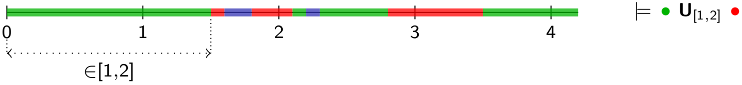
- Alternative: add variables (clocks) to the logics, e.g. TPTL

$\rightsquigarrow$ interpreted over **signals** (or over timed words)

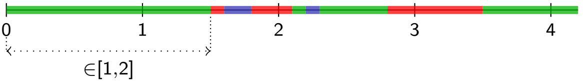


[Koy90] Koymans. Specifying real-time properties with metric temporal logic (Real-time systems, 1990).

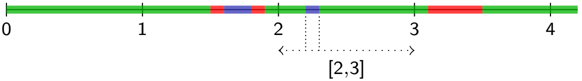
Examples



Examples

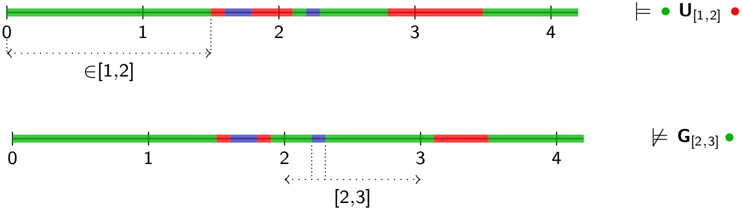


$\models \bullet\ \mathbf{U}_{[1,2]}\ \bullet$



$\not\models\ \mathbf{G}_{[2,3]}\ \bullet$

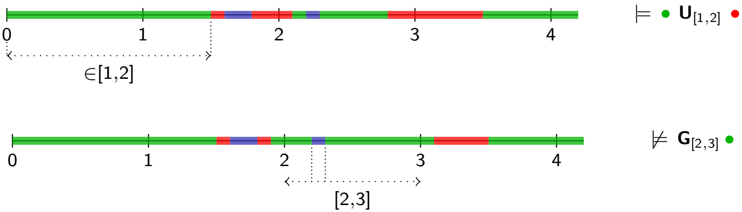
Examples



- "Every problem is followed within 56 time units by an alarm"

$$\mathbf{G}(\texttt{problem} \rightarrow \mathbf{F}_{\leq 56}\,\texttt{alarm})$$

Examples



- "Every problem is followed within 56 time units by an alarm"

$$\mathbf{G}(\mathtt{problem} \rightarrow \mathbf{F}_{\leq 56}\,\mathtt{alarm})$$

- "Each time there is a problem, it is either repaired within the next 15 time units, or an alarm rings during 3 time units 12 time units later"

$$\mathbf{G}(\mathtt{problem} \rightarrow (\mathbf{F}_{\leq 15}\,\mathtt{repair} \vee \mathbf{G}_{[12,15)}\,\mathtt{alarm}))$$

Model-checking timed temporal logics

Branching-time logic TCTL [ACD93]

The model-checking of TCTL is PSPACE-complete!
(The region abstraction can be used, with an extra clock for the formula)

[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time (I&C, 1993).

Model-checking timed temporal logics

Branching-time logic TCTL [ACD93]

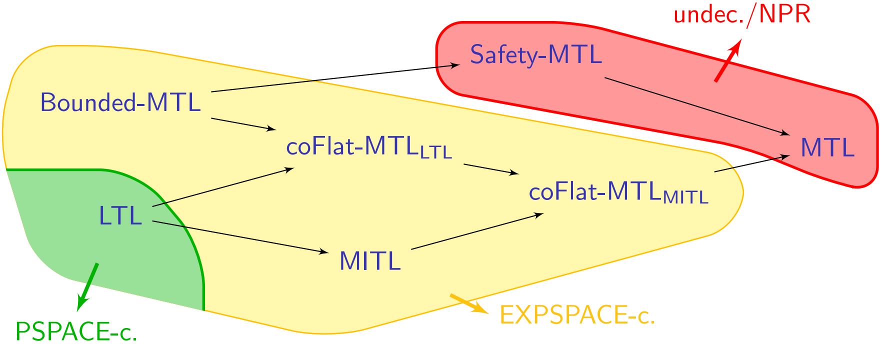
The model-checking of TCTL is PSPACE-complete!
(The region abstraction can be used, with an extra clock for the formula)

Linear-time logic MTL [AFH96,OW05]

The model-checking of MTL is undecidable/NPR.
Some fragments with decidable model-checking have been designed.

[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time (I&C, 1993).
[AFH96] Alur, Feder, Henzinger. The benefits of relaxing punctuality (Journal of the ACM, 1996).
[OW05] Ouaknine, Worrell. On the decidability of metric temporal logic *(LICS'05)*.

Model-checking timed temporal logics



Technics: alternating timed automata, channel machines, small-model properties

[AFH96] Alur, Feder, Henzinger. The benefits of relaxing punctuality (Journal of the ACM, 1996).
[BMOW08] Bouyer, Markey, Ouaknine, Worrell. On Expressiveness and Complexity in Real-time Model Checking *(ICALP'08)*.
[BMOW07] Bouyer, Markey, Ouaknine, Worrell. The cost of punctuality *(LICS'07)*.
[OW06] Ouaknine, Worrell. Safety Metric Temporal Logic is Fully Decidable *(TACAS'06)*.

Model-checking timed temporal logics

Branching-time logic TCTL [ACD93]

The model-checking of TCTL is PSPACE-complete!
(The region abstraction can be used, with an extra clock for the formula)

Linear-time logic MTL [AFH96,OW05]

The model-checking of MTL is undecidable/NPR.
Some fragments with decidable model-checking have been designed.

[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time (I&C, 1993).
[AFH96] Alur, Feder, Henzinger. The benefits of relaxing punctuality (Journal of the ACM, 1996).
[OW05] Ouaknine, Worrell. On the decidability of metric temporal logic *(LICS'05)*.

A focus on MITL

The nightmare of timed temporal logics

Requiring too much precision, and hence too many clocks!!

A focus on MITL

The nightmare of timed temporal logics

Requiring too much precision, and hence too many clocks!!

Example

$$\mathbf{G}(\bullet \to \mathbf{F}_{=1} \bullet)$$

- each time an • occurs, start a new clock, and check that a • occurs 1 time unit later
- this requires an unbounded number of clocks

A focus on MITL

The nightmare of timed temporal logics

Requiring too much precision, and hence too many clocks!!

Example

$$\mathbf{G}(\bullet \rightarrow \mathbf{F}_{=1} \bullet)$$

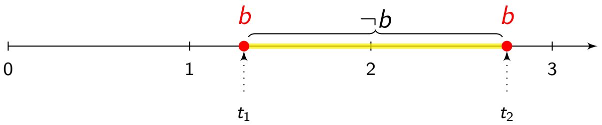
- each time an • occurs, start a new clock, and check that a • occurs 1 time unit later
- this requires an unbounded number of clocks

The logic MITL

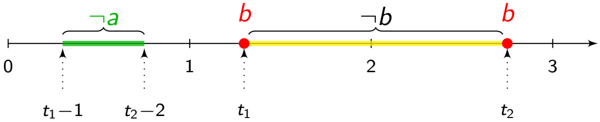
- Bans "punctual" constraints
- Consequences:
  - ☞ we can bound the variability of signals
  - ☞ an MITL formula defines a timed regular language

Formula $\mathbf{G}_{(0,1)}(a \rightarrow \mathbf{F}_{[1,2]}\, b)$

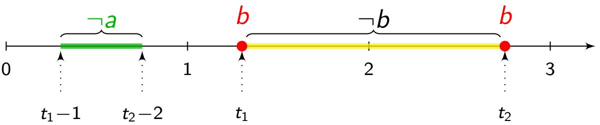
Formula $\mathbf{G}_{(0,1)}(a \to \mathbf{F}_{[1,2]}\, b)$

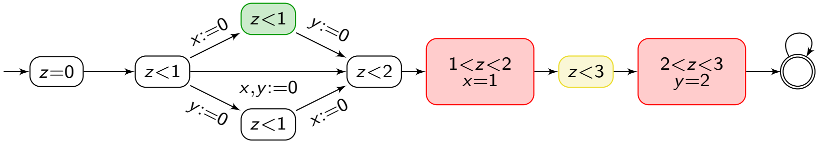


Formula $\mathbf{G}_{(0,1)}(a \rightarrow \mathbf{F}_{[1,2]}\, b)$

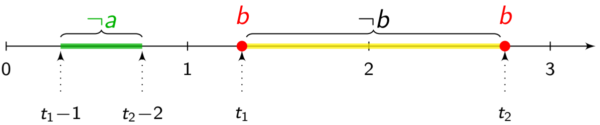


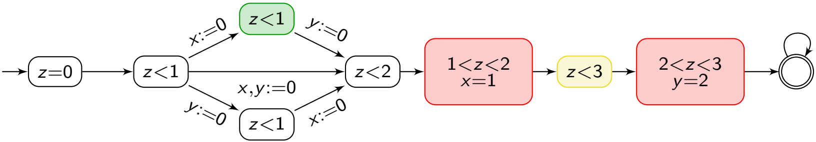
Formula $\mathbf{G}_{(0,1)}(a \rightarrow \mathbf{F}_{[1,2]}\, b)$





Formula $\mathbf{G}_{(0,1)}(a \rightarrow \mathbf{F}_{[1,2]}\, b)$





☞ This idea can be extended to any formula in MITL

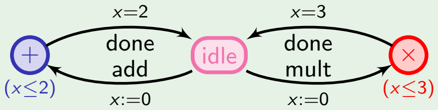
Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

Why (timed) games?

- to model uncertainty

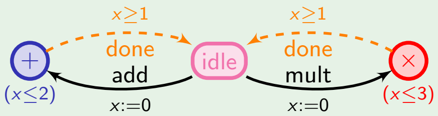
Example of a processor in the taskgraph example



Why (timed) games?

- to model uncertainty

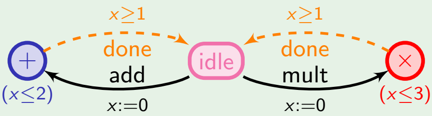
Example of a processor in the taskgraph example



Why (timed) games?

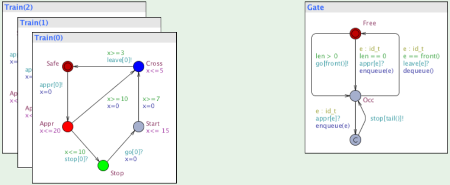
- to model uncertainty

Example of a processor in the taskgraph example



- to model an interaction with the environment

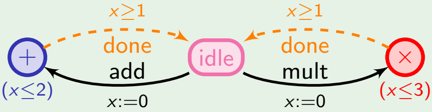
Example of the gate in the train/gate example



Why (timed) games?

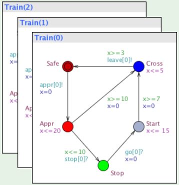
- to model uncertainty

Example of a processor in the taskgraph example



- to model an interaction with the environment

Example of the gate in the train/gate example

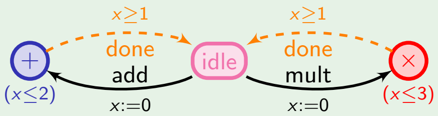


?

Why (timed) games?

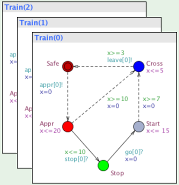
- to model uncertainty

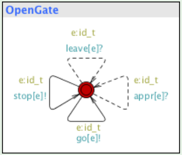
Example of a processor in the taskgraph example



- to model an interaction with the environment

Example of the gate in the train/gate example





Outline

1 Introduction
- Timed automata
- Examples

2 Decidability of basic properties
- The region abstraction
- Extensions of timed automata
- Weighted timed automata

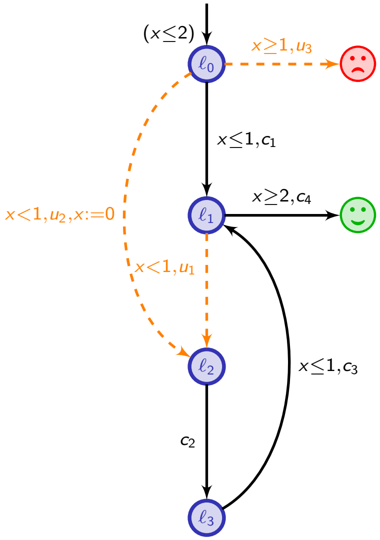
3 Implementation and tools

4 Other verification problems
- Equivalence (or preorder) checking
- Verification of timed temporal logics (short)

5 Timed control
- Timed games
- Weighted timed games

6 Conclusion

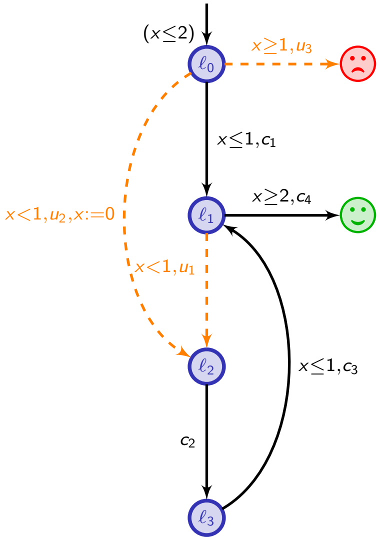
An example of a timed game



Rule of the game

- Aim: avoid ☹ and reach ☺

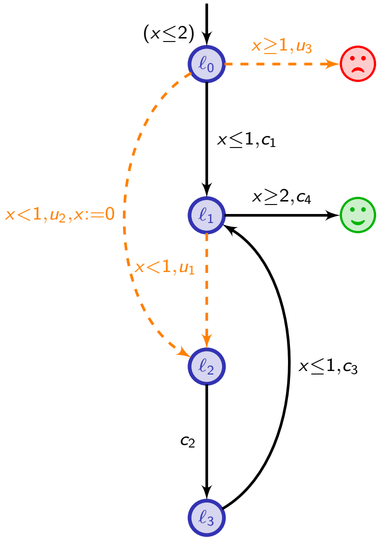
An example of a timed game



Rule of the game

- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

An example of a timed game

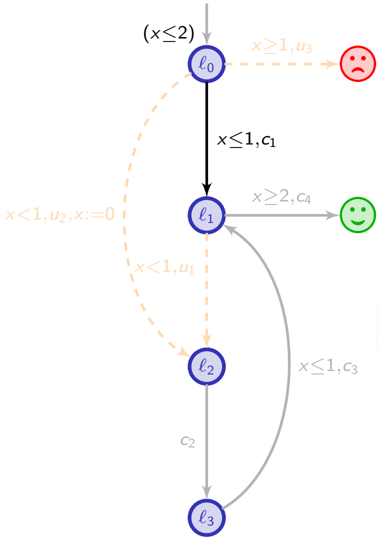


Rule of the game

- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

$$f : \text{history} \mapsto (\text{delay}, \text{cont. transition})$$

An example of a timed game



Rule of the game

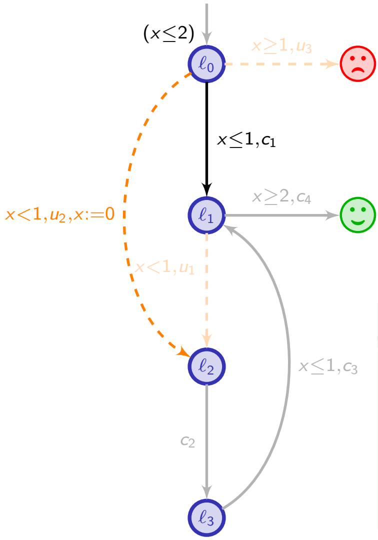
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

$$f : \text{history} \mapsto (\text{delay}, \text{cont. transition})$$

A (memoryless) winning strategy

- from $(\ell_0, 0)$, play $(0.5, c_1)$

An example of a timed game



Rule of the game

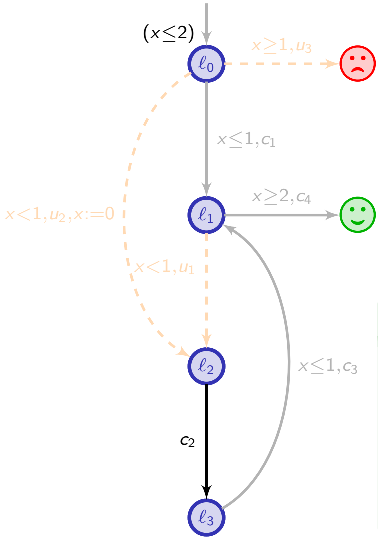
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

$$f : \text{history} \mapsto (\text{delay}, \text{cont. transition})$$

A (memoryless) winning strategy

- from $(\ell_0, 0)$, play $(0.5, c_1)$
  $\rightsquigarrow$ can be preempted by u_2

An example of a timed game



Rule of the game

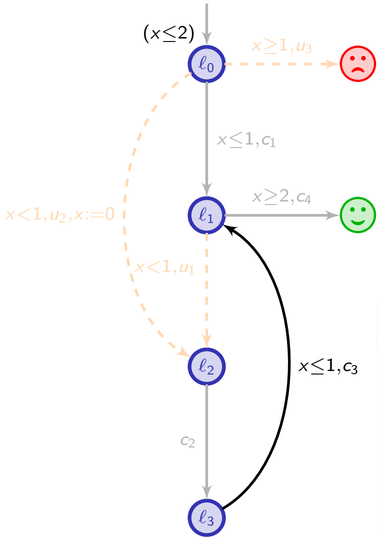
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

$$f : \text{history} \mapsto (\text{delay}, \text{cont. transition})$$

A (memoryless) winning strategy

- from $(\ell_0, 0)$, play $(0.5, c_1)$
  $\rightsquigarrow$ can be preempted by u_2
- from $(\ell_2, \star)$, play $(1 - \star, c_2)$

An example of a timed game



Rule of the game

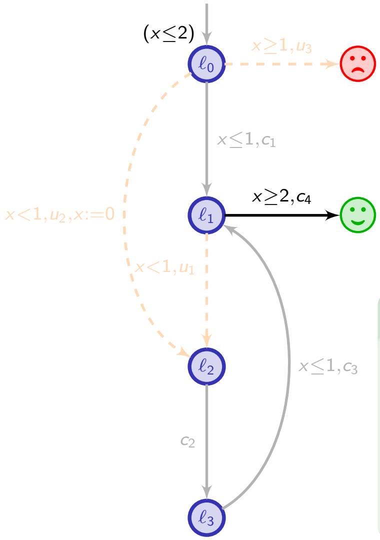
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

$$f : \text{history} \mapsto (\text{delay}, \text{cont. transition})$$

A (memoryless) winning strategy

- from $(\ell_0, 0)$, play $(0.5, c_1)$
  $\rightsquigarrow$ can be preempted by u_2
- from $(\ell_2, \star)$, play $(1 - \star, c_2)$
- from $(\ell_3, 1)$, play $(0, c_3)$

An example of a timed game



Rule of the game

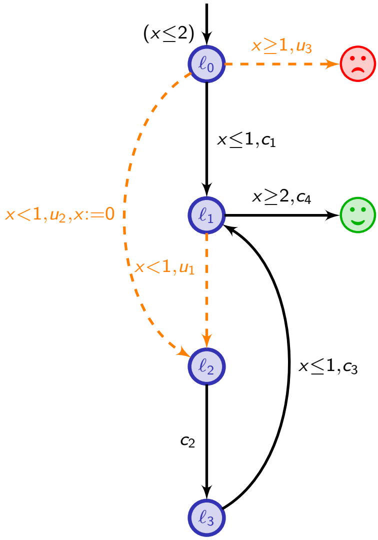
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

$$f : \text{history} \mapsto (\text{delay}, \text{cont. transition})$$

A (memoryless) winning strategy

- from $(\ell_0, 0)$, play $(0.5, c_1)$
  $\rightsquigarrow$ can be preempted by u_2
- from $(\ell_2, \star)$, play $(1 - \star, c_2)$
- from $(\ell_3, 1)$, play $(0, c_3)$
- from $(\ell_1, 1)$, play $(1, c_4)$

An example of a timed game



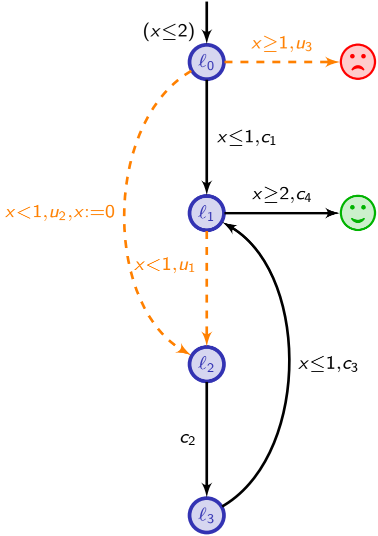
Rule of the game

- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

$$f : \text{history} \mapsto (\text{delay}, \text{cont. transition})$$

Problems to be considered

An example of a timed game



Rule of the game

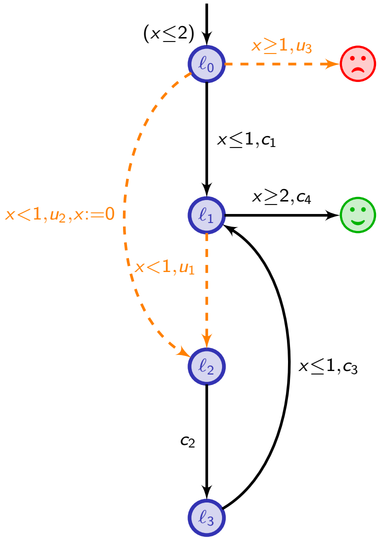
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

$$f : \text{history} \mapsto (\text{delay}, \text{cont. transition})$$

Problems to be considered

- Does there exist a winning strategy?

An example of a timed game



Rule of the game

- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

$$f : \text{history} \mapsto (\text{delay}, \text{cont. transition})$$

Problems to be considered

- Does there exist a winning strategy?
- If yes, compute one (as simple as possible).

Decidability of timed games

Theorem [AMPS98,HK99]

Reachability and safety timed games are decidable and EXPTIME-complete. Furthermore memoryless and “region-based” strategies are sufficient.

[AMPS98] Asarin, Maler, Pnueli, Sifakis. Controller synthesis for timed automata *(SSC'98)*.
[HK99] Henzinger, Kopke. Discrete-time control for rectangular hybrid automata *(Theoretical Computer Science)*.

Decidability of timed games

Theorem [AMPS98,HK99]

Reachability and safety timed games are decidable and EXPTIME-complete. Furthermore memoryless and "region-based" strategies are sufficient.

$\rightsquigarrow$ classical regions are sufficient for solving such problems

[AMPS98] Asarin, Maler, Pnueli, Sifakis. Controller synthesis for timed automata *(SSC'98)*.
[HK99] Henzinger, Kopke. Discrete-time control for rectangular hybrid automata *(Theoretical Computer Science)*.

Decidability of timed games

Theorem [AMPS98,HK99]

Reachability and safety timed games are decidable and EXPTIME-complete. Furthermore memoryless and “region-based” strategies are sufficient.

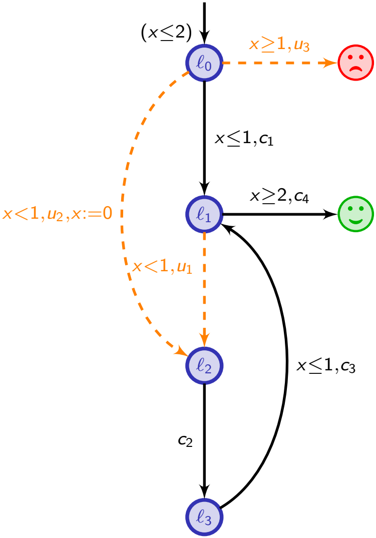
$\rightsquigarrow$ classical regions are sufficient for solving such problems

Theorem [AM99,BHPR07,JT07]

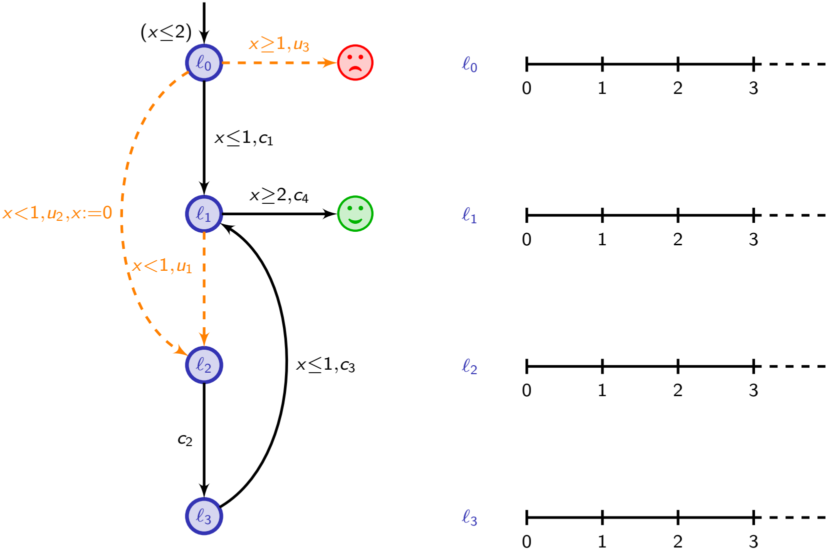
Optimal-time reachability timed games are decidable and EXPTIME-complete.

[AM99] Asarin, Maler. As soon as possible: time optimal control for timed automata *(HSCC'99)*.
[BHPR07] Brihaye, Henzinger, Prabhu, Raskin. Minimum-time reachability in timed games *(ICALP'07)*.
[JT07] Jurdziński, Trivedi. Reachability-time games on timed automata *(ICALP'07)*.

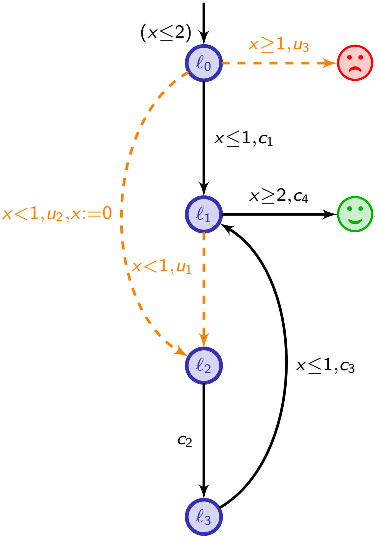
Back to the example: computing winning states

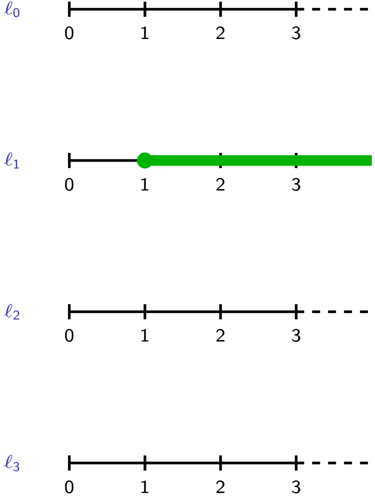


Back to the example: computing winning states

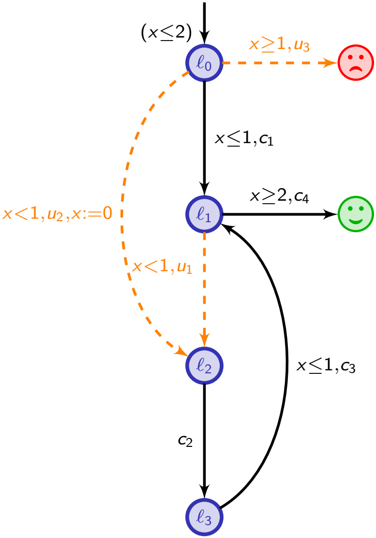


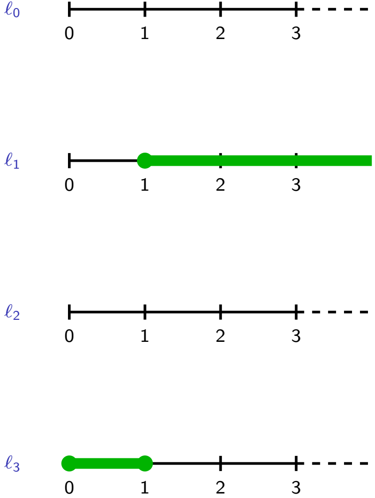
Back to the example: computing winning states



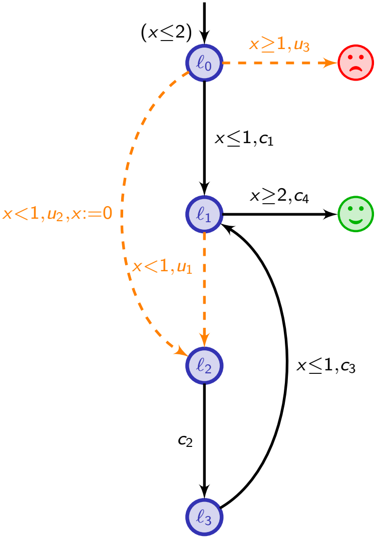


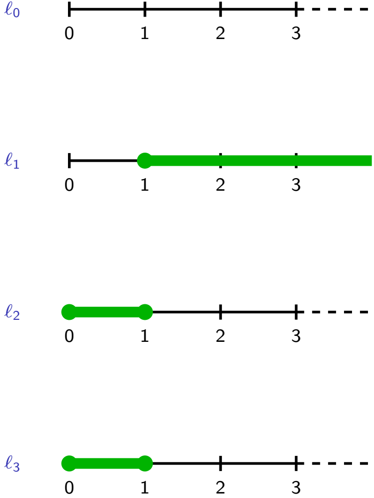
Back to the example: computing winning states



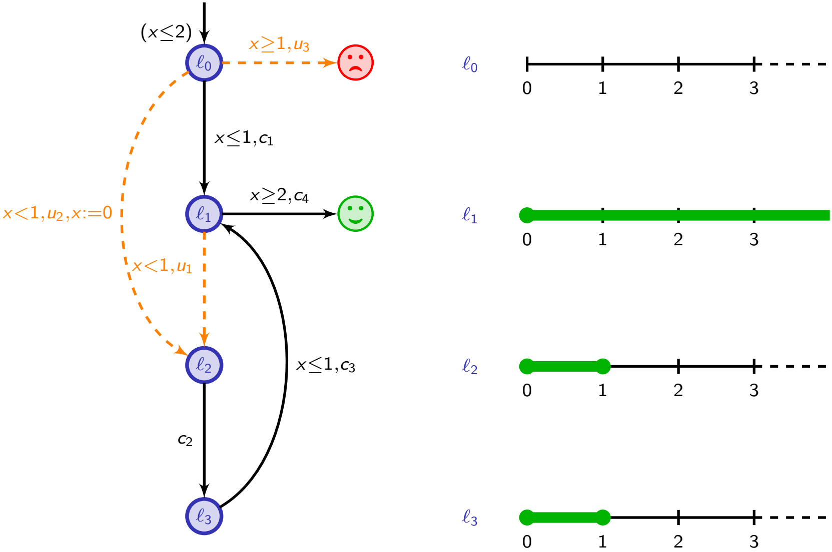


Back to the example: computing winning states

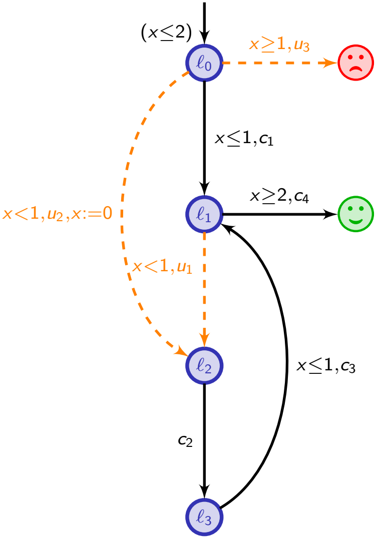


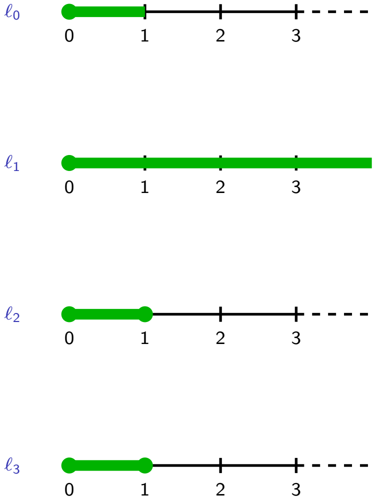


Back to the example: computing winning states

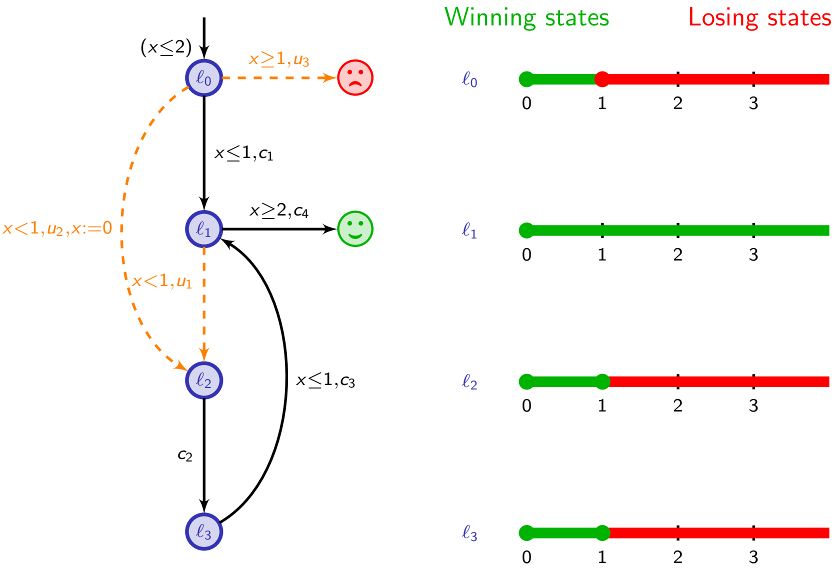


Back to the example: computing winning states





Back to the example: computing winning states



Decidability *via* attractors

Decidability *via* attractors

- $\mathsf{Pred}^a(X) = \{\bullet \mid \bullet \xrightarrow{a} \bullet \in X\}$

Decidability *via* attractors

- $\mathsf{Pred}^a(X) = \{\bullet \mid \bullet \xrightarrow{a} \bullet \in X\}$

- controllable and uncontrollable discrete predecessors:

$$\mathsf{cPred}(X) = \bigcup_{a \text{ cont.}} \mathsf{Pred}^a(X) \qquad \mathsf{uPred}(X) = \bigcup_{a \text{ uncont.}} \mathsf{Pred}^a(X)$$

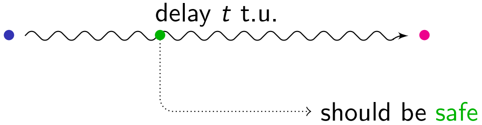
Decidability *via* attractors

- $\mathsf{Pred}^a(X) = \{\bullet \mid \bullet \xrightarrow{a} \bullet \in X\}$

- controllable and uncontrollable discrete predecessors:

$$\mathsf{cPred}(X) = \bigcup_{a \text{ cont.}} \mathsf{Pred}^a(X) \qquad \mathsf{uPred}(X) = \bigcup_{a \text{ uncont.}} \mathsf{Pred}^a(X)$$

- time controllable predecessors:



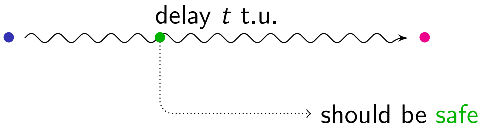
Decidability *via* attractors

- $\mathrm{Pred}^a(X) = \{\bullet \mid \bullet \xrightarrow{a} \bullet \in X\}$

- controllable and uncontrollable discrete predecessors:

$$\mathrm{cPred}(X) = \bigcup_{a \text{ cont.}} \mathrm{Pred}^a(X) \qquad \mathrm{uPred}(X) = \bigcup_{a \text{ uncont.}} \mathrm{Pred}^a(X)$$

- time controllable predecessors:



$$\mathrm{Pred}_\delta(X, \mathrm{Safe}) = \{\bullet \mid \exists t \geq 0,\ \bullet \xrightarrow{\delta(t)} \bullet$$
$$\text{and } \forall 0 \leq t' \leq t,\ \bullet \xrightarrow{\delta(t')} \bullet \in \mathrm{Safe}\}$$

Timed games with a reachability objective

We write:

$$\pi(X) = X \cup \mathsf{Pred}_\delta(\mathsf{cPred}(X), \neg\mathsf{uPred}(\neg X))$$

Timed games with a reachability objective

We write:

$$\pi(X) = X \cup \mathrm{Pred}_\delta(\mathrm{cPred}(X), \neg\mathrm{uPred}(\neg X))$$

- The states from which one can ensure ☺ in no more than 1 step is:

$$\mathrm{Attr}_1(☺) = \pi(☺)$$

Timed games with a reachability objective

We write:

$$\pi(X) = X \cup \mathsf{Pred}_\delta(\mathsf{cPred}(X), \neg\mathsf{uPred}(\neg X))$$

- The states from which one can ensure ☺ in no more than 1 step is:

$$\mathsf{Attr}_1(☺) = \pi(☺)$$

- The states from which one can ensure ☺ in no more than 2 steps is:

$$\mathsf{Attr}_2(☺) = \pi(\mathsf{Attr}_1(☺))$$

Timed games with a reachability objective

We write:

$$\pi(X) = X \cup \mathsf{Pred}_\delta(\mathsf{cPred}(X), \neg\mathsf{uPred}(\neg X))$$

- The states from which one can ensure ☺ in no more than 1 step is:

$$\mathsf{Attr}_1(☺) = \pi(☺)$$

- The states from which one can ensure ☺ in no more than 2 steps is:

$$\mathsf{Attr}_2(☺) = \pi(\mathsf{Attr}_1(☺))$$

- ...

Timed games with a reachability objective

We write:

$$\pi(X) = X \cup \mathsf{Pred}_\delta(\mathsf{cPred}(X), \neg\mathsf{uPred}(\neg X))$$

- The states from which one can ensure ☺ in no more than 1 step is:

$$\mathsf{Attr}_1(☺) = \pi(☺)$$

- The states from which one can ensure ☺ in no more than 2 steps is:

$$\mathsf{Attr}_2(☺) = \pi(\mathsf{Attr}_1(☺))$$

- ...
- The states from which one can ensure ☺ in no more than n steps is:

$$\mathsf{Attr}_n(☺) \quad = \quad \pi(\mathsf{Attr}_{n-1}(☺))$$

Timed games with a reachability objective

We write:

$$\pi(X) = X \cup \mathsf{Pred}_\delta(\mathsf{cPred}(X), \neg\mathsf{uPred}(\neg X))$$

- The states from which one can ensure ☺ in no more than 1 step is:

$$\mathsf{Attr}_1(☺) = \pi(☺)$$

- The states from which one can ensure ☺ in no more than 2 steps is:

$$\mathsf{Attr}_2(☺) = \pi(\mathsf{Attr}_1(☺))$$

- ...
- The states from which one can ensure ☺ in no more than n steps is:

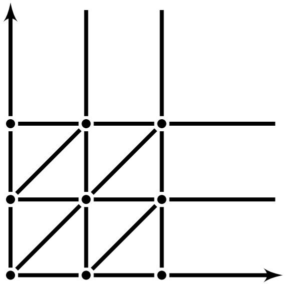
$$\begin{array}{rcl}\mathsf{Attr}_n(☺) & = & \pi(\mathsf{Attr}_{n-1}(☺)) \\ & = & \pi^n(☺)\end{array}$$

Stability w.r.t. regions

- if X is a union of regions, then:
  - $\mathsf{Pred}_a(X)$ is a union of regions,
  - and so are $\mathsf{cPred}(X)$ and $\mathsf{uPred}(X)$.

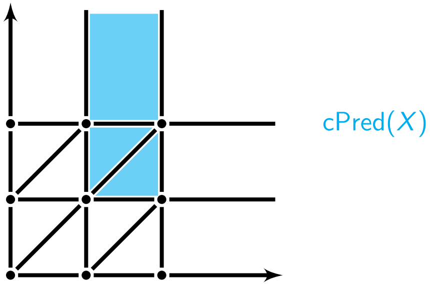
Stability w.r.t. regions

- if X is a union of regions, then:
  - $\text{Pred}_a(X)$ is a union of regions,
  - and so are $\text{cPred}(X)$ and $\text{uPred}(X)$.
- Does π also preserve unions of regions?



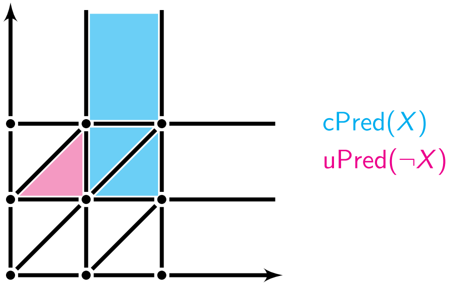
Stability w.r.t. regions

- if X is a union of regions, then:
  - $\mathrm{Pred}_a(X)$ is a union of regions,
  - and so are $\mathrm{cPred}(X)$ and $\mathrm{uPred}(X)$.
- Does π also preserve unions of regions?



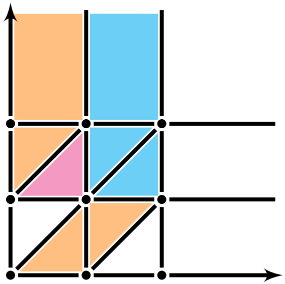
Stability w.r.t. regions

- if X is a union of regions, then:
  - $\mathrm{Pred}_a(X)$ is a union of regions,
  - and so are $\mathrm{cPred}(X)$ and $\mathrm{uPred}(X)$.
- Does π also preserve unions of regions?



Stability w.r.t. regions

- if X is a union of regions, then:
  - $\mathrm{Pred}_a(X)$ is a union of regions,
  - and so are $\mathrm{cPred}(X)$ and $\mathrm{uPred}(X)$.
- Does π also preserve unions of regions?



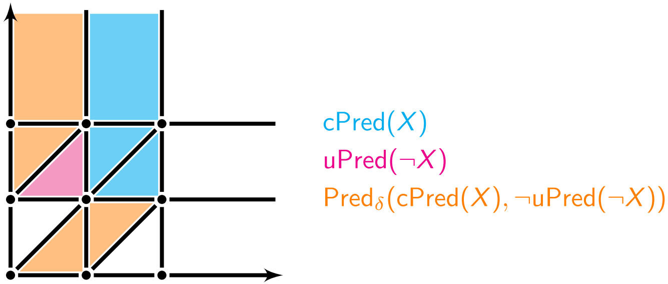
$\mathrm{cPred}(X)$

$\mathrm{uPred}(\neg X)$

$\mathrm{Pred}_\delta(\mathrm{cPred}(X), \neg\mathrm{uPred}(\neg X))$

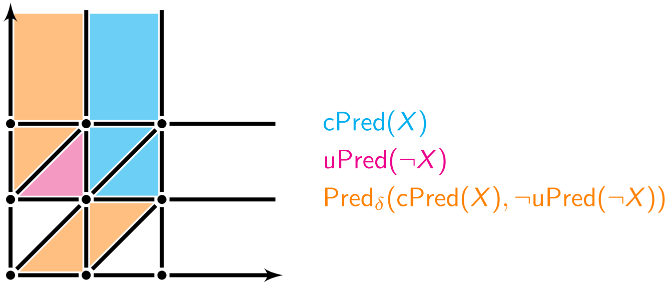
Stability w.r.t. regions

- if X is a union of regions, then:
  - $\mathrm{Pred}_a(X)$ is a union of regions,
  - and so are $\mathrm{cPred}(X)$ and $\mathrm{uPred}(X)$.
- Does π also preserve unions of regions? Yes!



Stability w.r.t. regions

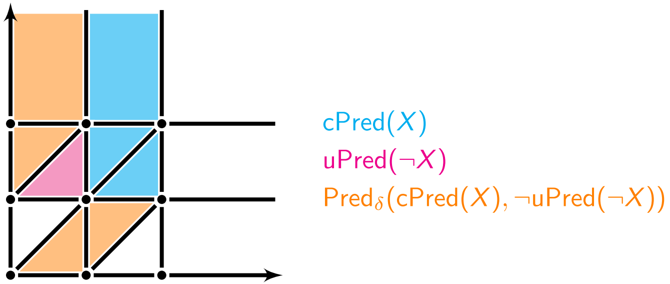
- if X is a union of regions, then:
  - $\mathsf{Pred}_a(X)$ is a union of regions,
  - and so are $\mathsf{cPred}(X)$ and $\mathsf{uPred}(X)$.
- Does π also preserve unions of regions? Yes!



(but it generates non-convex unions of regions...)

Stability w.r.t. regions

- if X is a union of regions, then:
  - $\text{Pred}_a(X)$ is a union of regions,
  - and so are $\text{cPred}(X)$ and $\text{uPred}(X)$.
- Does π also preserve unions of regions? Yes!

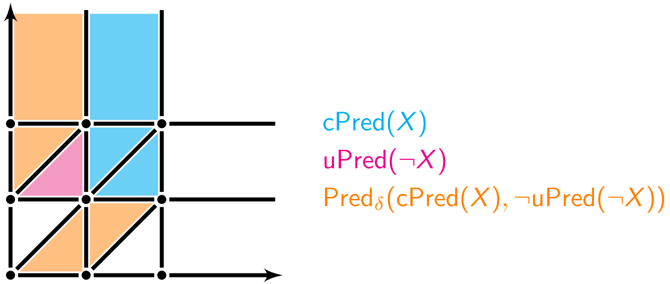


(but it generates non-convex unions of regions...)

$\rightsquigarrow$ the computation of π^*(☺) terminates!

Stability w.r.t. regions

- if X is a union of regions, then:
  - $\mathrm{Pred}_a(X)$ is a union of regions,
  - and so are $\mathrm{cPred}(X)$ and $\mathrm{uPred}(X)$.
- Does π also preserve unions of regions? Yes!



(but it generates non-convex unions of regions...)

$\rightsquigarrow$ the computation of π^*(☺) terminates!
... and is correct

Timed games with a safety objective

- We can use operator $\widetilde{\pi}$ defined by

$$\widetilde{\pi}(X) = \mathsf{Pred}_\delta(X \cap \mathsf{cPred}(X), \neg \mathsf{uPred}(\neg X))$$

instead of π, and compute $\widetilde{\pi}^*(\neg ☹)$

Timed games with a safety objective

- We can use operator $\widetilde{\pi}$ defined by

$$\widetilde{\pi}(X) = \mathsf{Pred}_\delta(X \cap \mathsf{cPred}(X), \neg \mathsf{uPred}(\neg X))$$

instead of π, and compute $\widetilde{\pi}^*(\neg ☹)$

- It is also stable w.r.t. regions.

Some remarks

The model

Our games are control games,

Some remarks

The model

Our games are control games, and in particular they:

- are asymmetric
  - the environment can preempt any decision of the controller
  - we take the point-of-view of the controller
- are neither concurrent nor turn-based
- do not take into account Zenoness considerations

$\rightsquigarrow$ can be done adding a Büchi winning condition

Some remarks

The model

Our games are control games, and in particular they:

- are asymmetric
  - the environment can preempt any decision of the controller
  - we take the point-of-view of the controller
- are neither concurrent nor turn-based
- do not take into account Zenoness considerations

$\rightsquigarrow$ can be done adding a Büchi winning condition

Implementation

Uppaal-Tiga implements a forward algorithm to compute winning states and winning strategies [CDF+05,BCD+07]

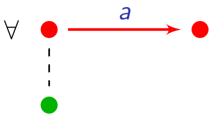
[CDF+05] Cassez, David, Fleury, Larsen, Lime. Efficient on-the-fly algorithms for the analysis of timed games *(CONCUR'05)*.
[BCD+07] Berhmann, Cougnard, David, Fleury, Larsen, Lime. Uppaal-Tiga: Time for playing games! *(CAV'07)*.

Application of timed games to strong timed bisimulation

This is a relation between • and • such that:

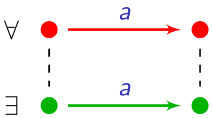
Application of timed games to strong timed bisimulation

This is a relation between • and • such that:



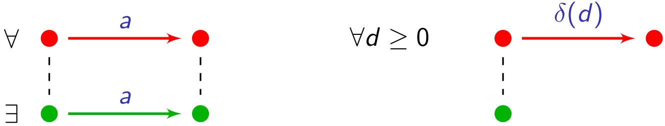
Application of timed games to strong timed bisimulation

This is a relation between • and • such that:



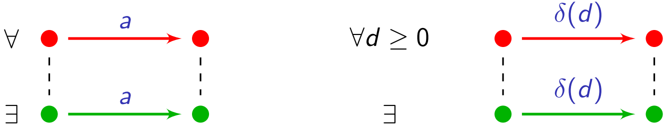
Application of timed games to strong timed bisimulation

This is a relation between • and • such that:



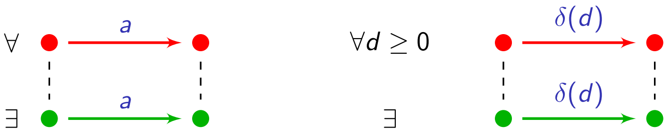
Application of timed games to strong timed bisimulation

This is a relation between • and • such that:



Application of timed games to strong timed bisimulation

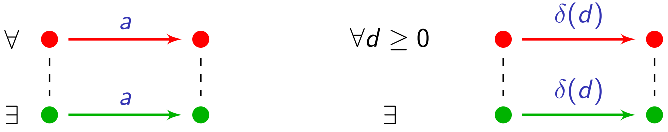
This is a relation between • and • such that:



... and vice-versa (swap • and •) for the bisimulation relation.

Application of timed games to strong timed bisimulation

This is a relation between • and • such that:

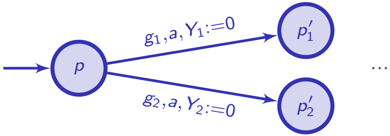


... and vice-versa (swap • and •) for the bisimulation relation.

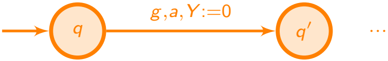
Theorem

Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.

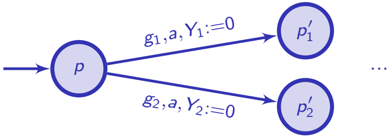
timed automaton $\mathcal{A}$



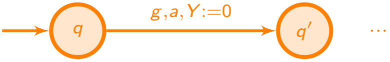
timed automaton $\mathcal{B}$

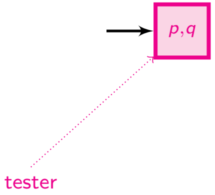


timed automaton $\mathcal{A}$

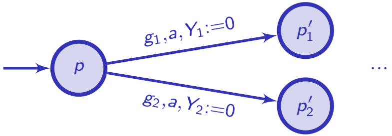


timed automaton $\mathcal{B}$

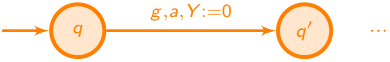


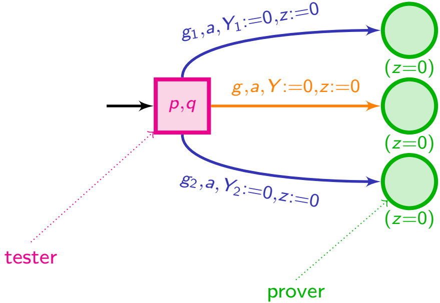


timed automaton $\mathcal{A}$

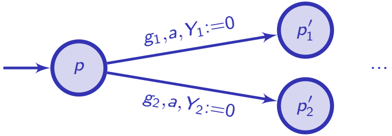


timed automaton $\mathcal{B}$

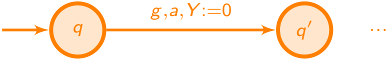


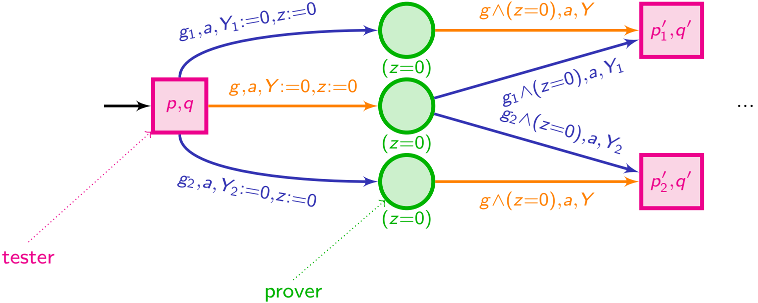


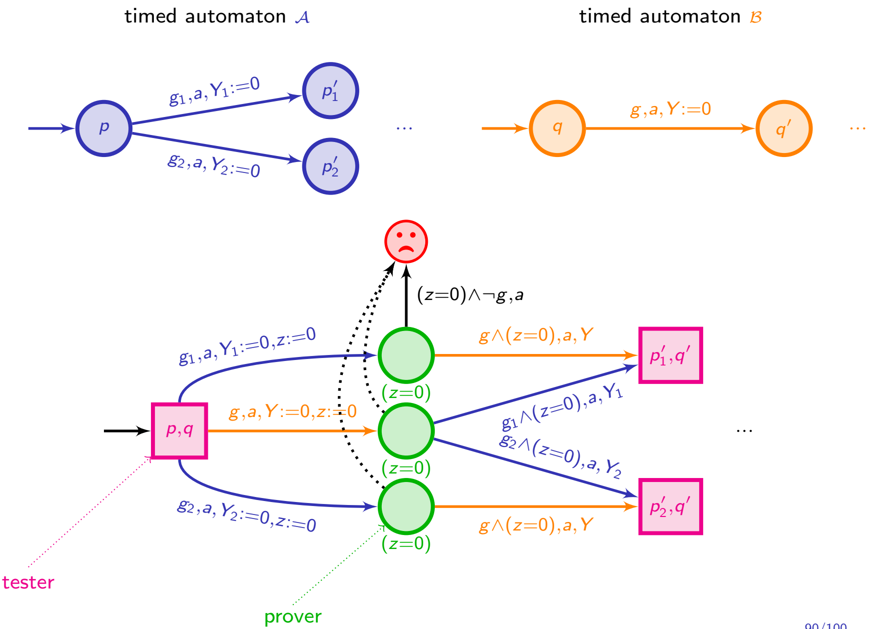
timed automaton $\mathcal{A}$

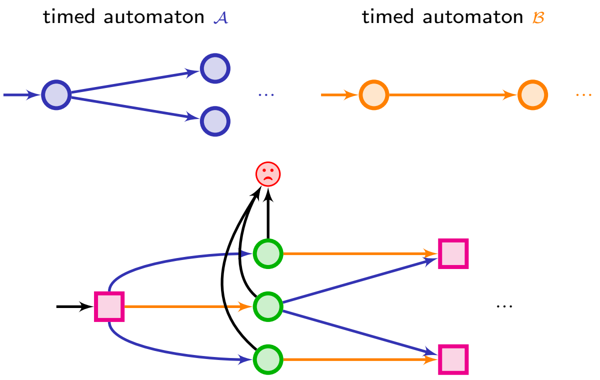


timed automaton $\mathcal{B}$






timed automaton $\mathcal{A}$
timed automaton $\mathcal{B}$
p
$g_1, a, Y_1{:=}0$
p'_1
$g_2, a, Y_2{:=}0$
p'_2
q
$g, a, Y{:=}0$
q'
$(z{=}0) \wedge \neg g, a$
$g_1, a, Y_1{:=}0, z{:=}0$
$g \wedge (z{=}0), a, Y$
p'_1, q'
$(z{=}0)$
p, q
$g, a, Y{:=}0, z{:=}0$
$g_1 \wedge (z{=}0), a, Y_1$
$g_2 \wedge (z{=}0), a, Y_2$
$g_2, a, Y_2{:=}0, z{:=}0$
p'_2, q'
tester
prover



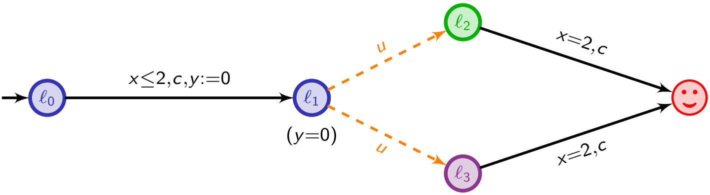
$\mathcal{A}$ and $\mathcal{B}$ are strongly timed bisimilar
iff
the prover has a winning strategy to avoid ☹

Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

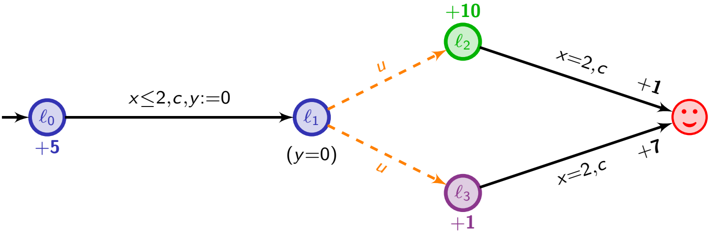
A simple weighted timed game

Skip



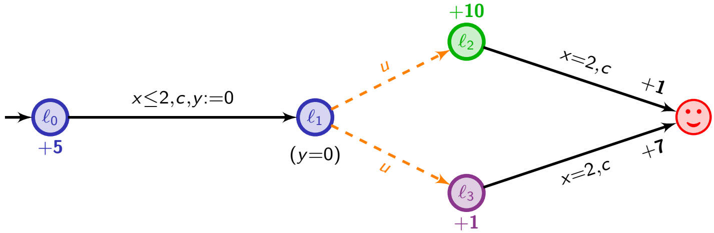
A simple weighted timed game

Skip



A simple weighted timed game

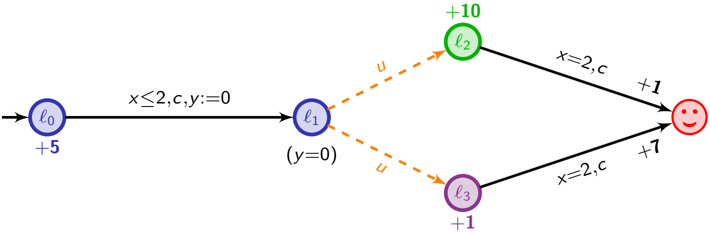
Skip



Question: what is the optimal cost we can ensure while reaching ?

A simple weighted timed game

Skip

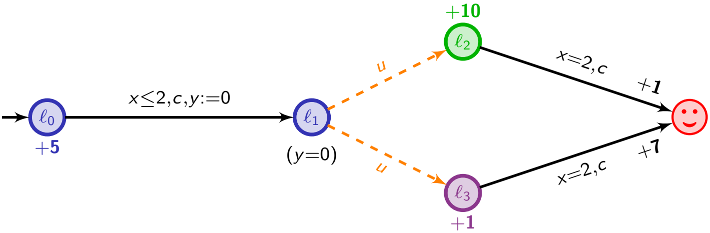


Question: what is the optimal cost we can ensure while reaching 🙂?

$$5t + 10(2 - t) + 1$$

A simple weighted timed game

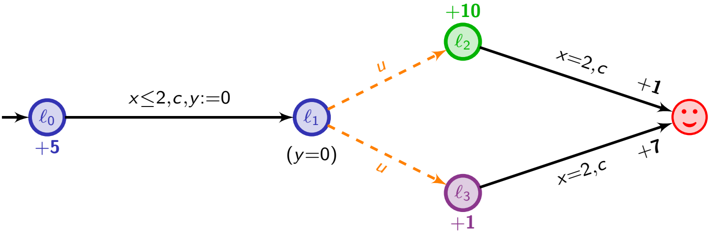
Skip



Question: what is the optimal cost we can ensure while reaching 🙂?

$$5t + 10(2-t) + 1\ ,\ 5t + (2-t) + 7$$

A simple weighted timed game

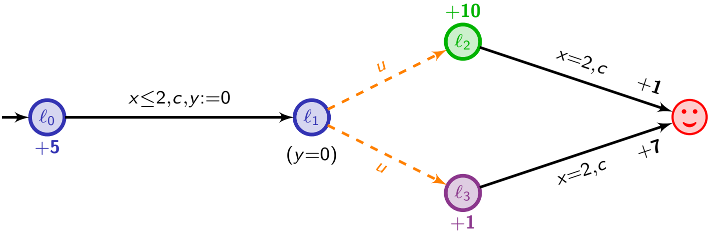


Question: what is the optimal cost we can ensure while reaching ☺?

$$\max\ (\ 5t + 10(2-t) + 1\ ,\ 5t + (2-t) + 7\ )$$

A simple weighted timed game

Skip

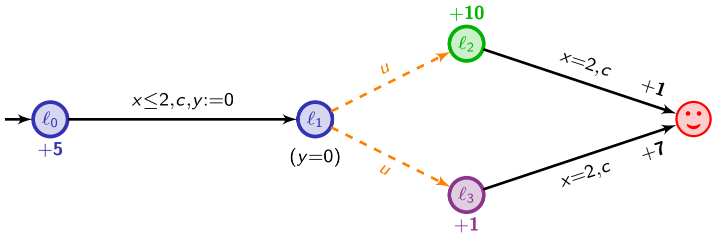


Question: what is the optimal cost we can ensure while reaching ☺?

$$\inf_{0 \leq t \leq 2} \max\left(5t + 10(2-t) + 1 \,,\, 5t + (2-t) + 7 \right) = 14 + \frac{1}{3}$$

A simple weighted timed game

Skip



Question: what is the optimal cost we can ensure while reaching ?

$$\inf_{0\le t\le 2} \max\left(\, 5t + 10(2-t) + 1 \;,\; 5t + (2-t) + 7 \,\right) = 14 + \frac{1}{3}$$

$\rightsquigarrow$ *strategy:* wait in ℓ_0, and when $t = \frac{4}{3}$, go to ℓ_1

Optimal reachability in weighted timed games

This topic has been fairly hot these last ten years...
[LMM02,ABM04,BCFL04,BBR05,BBM06,BLMR06,Rut11,HIM13,BGK+14]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.
[ABM04] Alur, Bernardsky, Madhusudan. Optimal reachability in weighted timed games *(ICALP'04)*.
[BCFL04] Bouyer, Cassez, Fleury, Larsen. Optimal strategies in priced timed game automata *(FSTTCS'04)*.
[BBR05] Brihaye, Bruyère, Raskin. On optimal timed strategies *(FORMATS'05)*.
[BBM06] Bouyer, Brihaye, Markey. Improved undecidability results on weighted timed automata *(Information Processing Letters)*.
[BLMR06] Bouyer, Larsen, Markey, Rasmussen. Almost-optimal strategies in one-clock priced timed automata *(FSTTCS'06)*.
[Rut11] Rutkowski. Two-player reachability-price games on single-clock timed automata *(QAPL'11)*.
[HIM13] Hansen, Ibsen-Jensen, Miltersen. A faster algorithm for solving one-clock priced timed games *(CONCUR'13)*.
[BGK+14] Brihaye, Geeraerts, Krishna, Manasa, Monmege, Trivedi. Adding Negative Prices to Priced Timed Games *(CONCUR'14)*.

Optimal reachability in weighted timed games

This topic has been fairly hot these last ten years...
[LMM02,ABM04,BCFL04,BBR05,BBM06,BLMR06,Rut11,HIM13,BGK+14]

Theorem [BBR05,BBM06,recent_work]

Optimal timed games are undecidable, as soon as automata have three clocks or more.

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.
[ABM04] Alur, Bernardsky, Madhusudan. Optimal reachability in weighted timed games *(ICALP'04)*.
[BCFL04] Bouyer, Cassez, Fleury, Larsen. Optimal strategies in priced timed game automata *(FSTTCS'04)*.
[BBR05] Brihaye, Bruyère, Raskin. On optimal timed strategies *(FORMATS'05)*.
[BBM06] Bouyer, Brihaye, Markey. Improved undecidability results on weighted timed automata *(Information Processing Letters)*.
[BLMR06] Bouyer, Larsen, Markey, Rasmussen. Almost-optimal strategies in one-clock priced timed automata *(FSTTCS'06)*.
[Rut11] Rutkowski. Two-player reachability-price games on single-clock timed automata *(QAPL'11)*.
[HIM13] Hansen, Ibsen-Jensen, Miltersen. A faster algorithm for solving one-clock priced timed games *(CONCUR'13)*.
[BGK+14] Brihaye, Geeraerts, Krishna, Manasa, Monmege, Trivedi. Adding Negative Prices to Priced Timed Games *(CONCUR'14)*.

Optimal reachability in weighted timed games

This topic has been fairly hot these last ten years...
[LMM02,ABM04,BCFL04,BBR05,BBM06,BLMR06,Rut11,HIM13,BGK+14]

Theorem [BBR05,BBM06,recent_work]

Optimal timed games are undecidable, as soon as automata have three clocks or more.

Theorem [BLMR06,Rut11,HIM13,BGK+14]

Turn-based optimal timed games are decidable in EXPTIME (resp. PTIME) when automata have a single clock (with two rates). They are PTIME-hard.

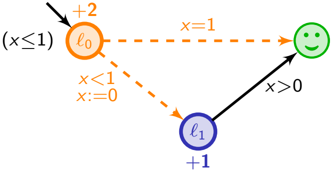
[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.
[ABM04] Alur, Bernardsky, Madhusudan. Optimal reachability in weighted timed games *(ICALP'04)*.
[BCFL04] Bouyer, Cassez, Fleury, Larsen. Optimal strategies in priced timed game automata *(FSTTCS'04)*.
[BBR05] Brihaye, Bruyère, Raskin. On optimal timed strategies *(FORMATS'05)*.
[BBM06] Bouyer, Brihaye, Markey. Improved undecidability results on weighted timed automata *(Information Processing Letters)*.
[BLMR06] Bouyer, Larsen, Markey, Rasmussen. Almost-optimal strategies in one-clock priced timed automata *(FSTTCS'06)*.
[Rut11] Rutkowski. Two-player reachability-price games on single-clock timed automata *(QAPL'11)*.
[HIM13] Hansen, Ibsen-Jensen, Miltersen. A faster algorithm for solving one-clock priced timed games *(CONCUR'13)*.
[BGK+14] Brihaye, Geeraerts, Krishna, Manasa, Monmege, Trivedi. Adding Negative Prices to Priced Timed Games *(CONCUR'14)*.

The positive side (one-clock case)

- Key: resetting the clock somehow resets the history...

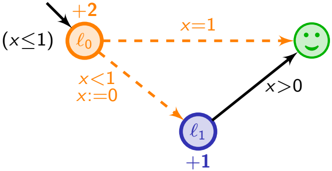
The positive side (one-clock case)

- Key: resetting the clock somehow resets the history...
- Memoryless strategies can be non-optimal...



The positive side (one-clock case)

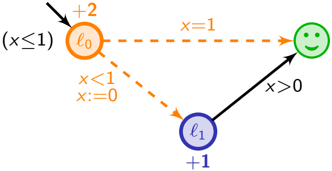
- Key: resetting the clock somehow resets the history...
- Memoryless strategies can be non-optimal...



- However, by unfolding and removing one by one the locations,we can synthesize memoryless almost-optimal winning strategies.

The positive side (one-clock case)

- Key: resetting the clock somehow resets the history...
- Memoryless strategies can be non-optimal...



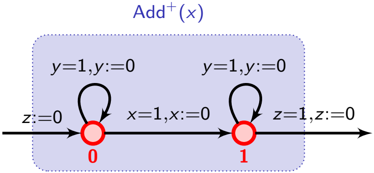
- However, by unfolding and removing one by one the locations,we can synthesize memoryless almost-optimal winning strategies.
- Rather involved proof of correctness for a simple algorithm.

The negative side: why is that hard?

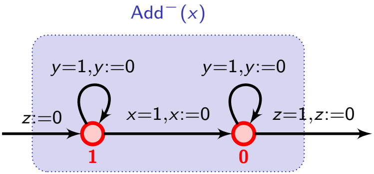
Given two clocks x and y, we can check whether $y = 2x$.

The negative side: why is that hard?

Given two clocks x and y, we can check whether $y = 2x$.



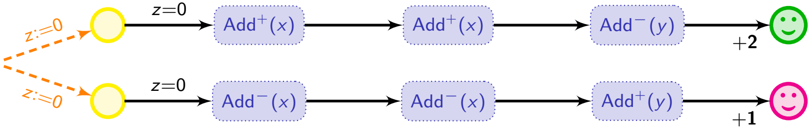
The cost is increased by x_0



The cost is increased by $1-x_0$

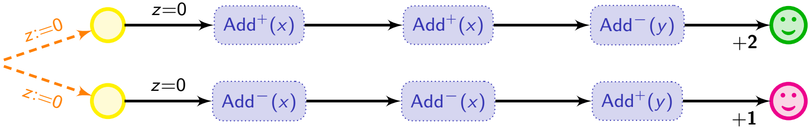
The negative side: why is that hard?

Given two clocks x and y, we can check whether $y = 2x$.



The negative side: why is that hard?

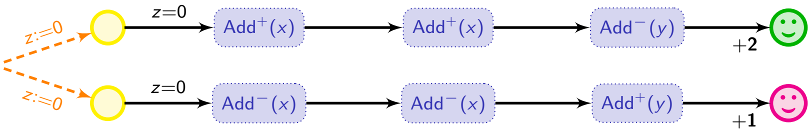
Given two clocks x and y, we can check whether $y = 2x$.



- In , cost $= 2x_0 + (1 - y_0) + 2$

The negative side: why is that hard?

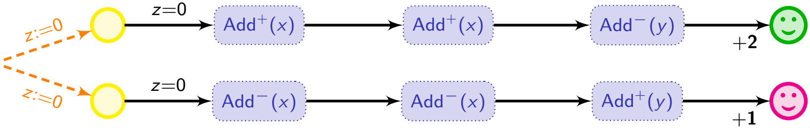
Given two clocks x and y, we can check whether $y = 2x$.



- In , cost $= 2x_0 + (1 - y_0) + 2$
  In , cost $= 2(1 - x_0) + y_0 + 1$

The negative side: why is that hard?

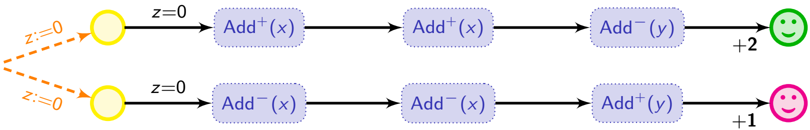
Given two clocks x and y, we can check whether $y = 2x$.



- In , cost $= 2x_0 + (1 - y_0) + 2$
  In , cost $= 2(1 - x_0) + y_0 + 1$
- if $y_0 < 2x_0$, player 2 chooses the first branch: cost > 3

The negative side: why is that hard?

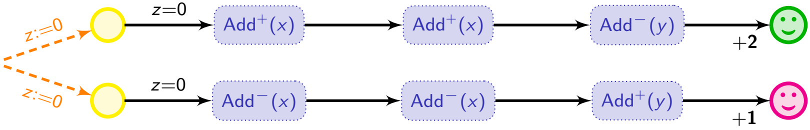
Given two clocks x and y, we can check whether $y = 2x$.



- In , cost $= 2x_0 + (1 - y_0) + 2$
  In , cost $= 2(1 - x_0) + y_0 + 1$
- if $y_0 < 2x_0$, player 2 chooses the first branch: cost > 3
  if $y_0 > 2x_0$, player 2 chooses the second branch: cost > 3

The negative side: why is that hard?

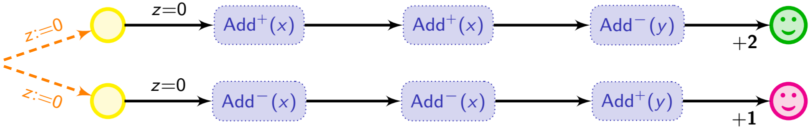
Given two clocks x and y, we can check whether $y = 2x$.



- In , cost $= 2x_0 + (1 - y_0) + 2$
  In , cost $= 2(1 - x_0) + y_0 + 1$
- if $y_0 < 2x_0$, player 2 chooses the first branch: cost > 3
  if $y_0 > 2x_0$, player 2 chooses the second branch: cost > 3
  if $y_0 = 2x_0$, in both branches, cost $= 3$

The negative side: why is that hard?

Given two clocks x and y, we can check whether $y = 2x$.



- In , cost $= 2x_0 + (1 - y_0) + 2$
  In , cost $= 2(1 - x_0) + y_0 + 1$
- if $y_0 < 2x_0$, player 2 chooses the first branch: cost > 3
  if $y_0 > 2x_0$, player 2 chooses the second branch: cost > 3
  if $y_0 = 2x_0$, in both branches, cost $= 3$
- Player 1 has a winning strategy with cost ≤ 3 iff $y_0 = 2x_0$

The negative side: why is that hard?

Player 1 will simulate a two-counter machine:

- each instruction is encoded as a module;
- the counter values c_1 and c_2 are encoded by two clocks:

$$x = \frac{1}{2^{c_1}} \qquad \text{and} \qquad y = \frac{1}{3^{c_2}}$$

The negative side: why is that hard?

Player 1 will simulate a two-counter machine:

- each instruction is encoded as a module;
- the counter values c_1 and c_2 are encoded by two clocks:

$$x = \frac{1}{2^{c_1}} \qquad \text{and} \qquad y = \frac{1}{3^{c_2}}$$

The two-counter machine has an halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.

The negative side: why is that hard?

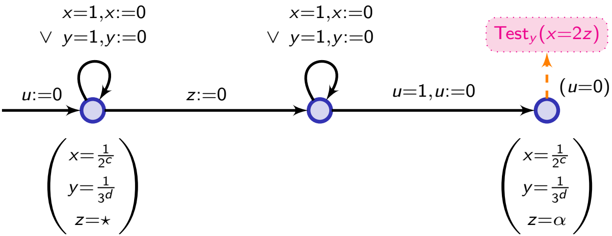
Player 1 will simulate a two-counter machine:

- each instruction is encoded as a module;
- the counter values c_1 and c_2 are encoded by two clocks:

$$x = \frac{1}{2^{c_1}} \qquad \text{and} \qquad y = \frac{1}{3^{c_2}}$$

The two-counter machine has an halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.

Globally, $(x\leq 1, y\leq 1, u\leq 1)$



Outline

1. Introduction
   - Timed automata
   - Examples
2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata
3. Implementation and tools
4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)
5. Timed control
   - Timed games
   - Weighted timed games
6. Conclusion

Conclusion

- The model of timed automata:

Conclusion

- The model of timed automata:
  - ☺ Some nice properties (decidability of many structural properties, symbolic algorithms, ...)

Conclusion

- The model of timed automata:
  - ☺ Some nice properties (decidability of many structural properties, symbolic algorithms, ...)
  - ☹ Not all good properties though...
    (e.g. inclusion undecidable)

Conclusion

- The model of timed automata:
  - ☺ Some nice properties (decidability of many structural properties, symbolic algorithms, ...)
  - ☹ Not all good properties though...
    (e.g. inclusion undecidable)
  - ☺ Sucessfully used!!

Conclusion

- The model of timed automata:
  - ☺ Some nice properties (decidability of many structural properties, symbolic algorithms, ...)
  - ☹ Not all good properties though...
    (e.g. inclusion undecidable)
  - ☺ Sucessfully used!!
- Many extensions have been studied, which allows more accurate modelling of real systems:
  - Weighted timed automata
  - Timed games
  - Probabilistic/stochastic timed automata
  - Alternating timed automata
  - Hybrid automata
  - ...

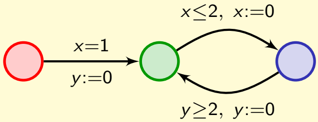
Conclusion

- The model of timed automata:
  - ☺ Some nice properties (decidability of many structural properties, symbolic algorithms, ...)
  - ☹ Not all good properties though...
    (e.g. inclusion undecidable)
  - ☺ Sucessfully used!!
- Many extensions have been studied, which allows more accurate modelling of real systems:
  - Weighted timed automata
  - Timed games
  - Probabilistic/stochastic timed automata
  - Alternating timed automata
  - Hybrid automata
  - ...
- Going further in the use of timed automata in verification...

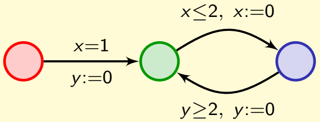
Conclusion

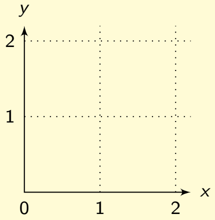
- The model of timed automata:
  - ☺ Some nice properties (decidability of many structural properties, symbolic algorithms, ...)
  - ☹ Not all good properties though...
    (e.g. inclusion undecidable)
  - ☺ Sucessfully used!!
- Many extensions have been studied, which allows more accurate modelling of real systems:
  - Weighted timed automata
  - Timed games
  - Probabilistic/stochastic timed automata
  - Alternating timed automata
  - Hybrid automata
  - ...
- Going further in the use of timed automata in verification...
  ... requires to think about the accurateness of the (mathematical) model we analyze w.r.t. the real-world system

Just a taste...

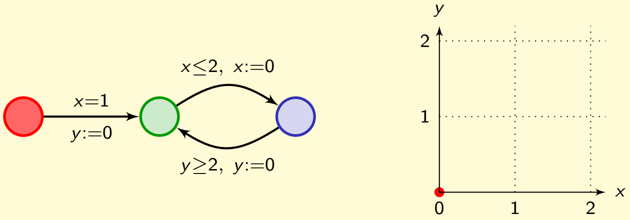


Just a taste...

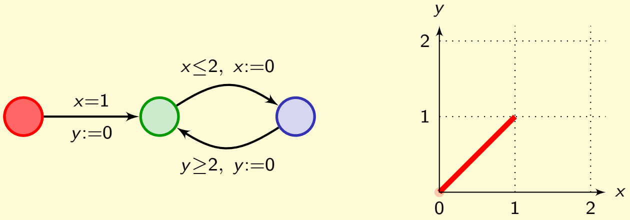




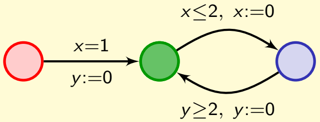
Just a taste...

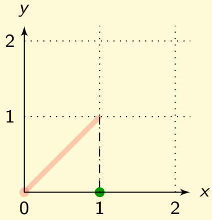


Just a taste...

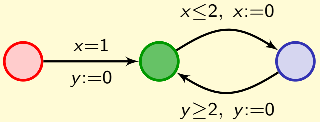


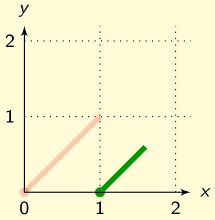
Just a taste...



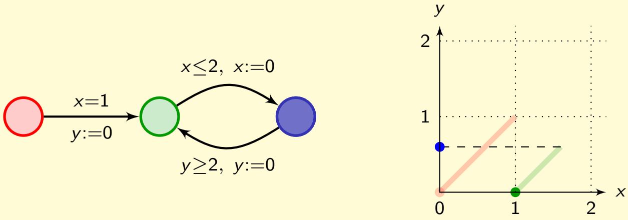


Just a taste...

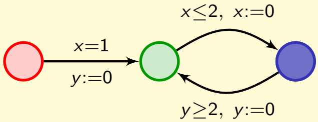


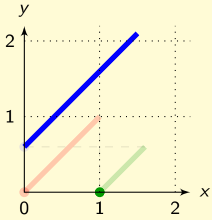


Just a taste...

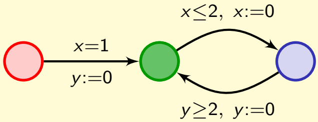


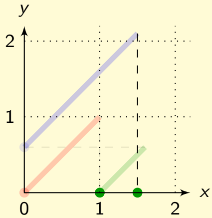
Just a taste...



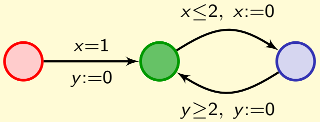


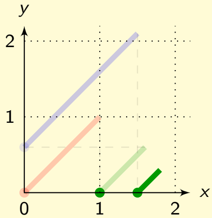
Just a taste...



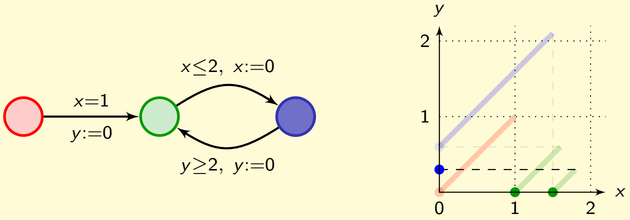


Just a taste...

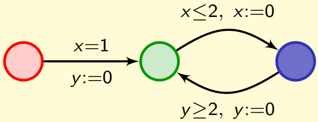


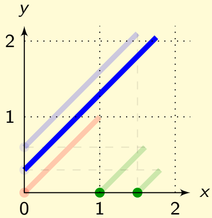


Just a taste...

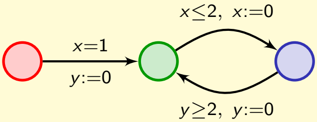


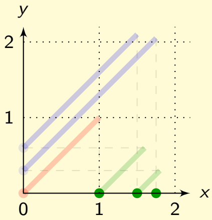
Just a taste...



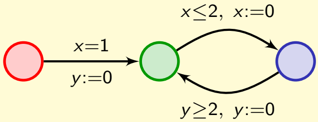


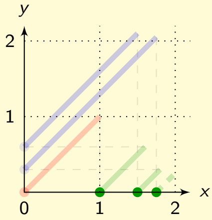
Just a taste...



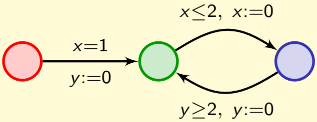


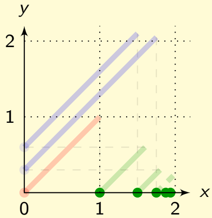
Just a taste...





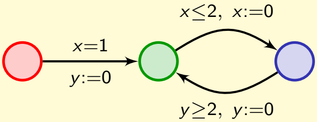
Just a taste...

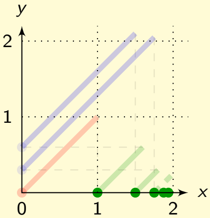




$\rightsquigarrow$ Value of clock x when hitting ◯ is converging, even though global time diverges

Just a taste...

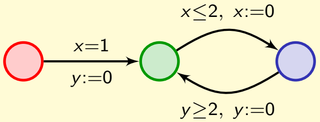


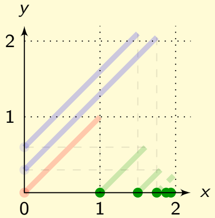


$\rightsquigarrow$ Value of clock x when hitting ◯ is converging, even though global time diverges

Can we implement such a strategy??

Just a taste...





$\rightsquigarrow$ Value of clock x when hitting ◯ is converging, even though global time diverges

Can we implement such a strategy??

☞ lecture of Pierre-Alain tomorrow afternoon!