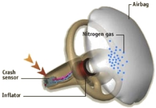
On the verification of timed systems...
... and beyond

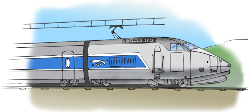
Patricia Bouyer-Decitre

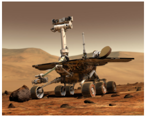
LSV, CNRS & ENS Paris-Saclay, France

Time-dependent systems

- We are interested in timed systems

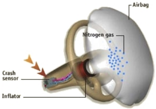


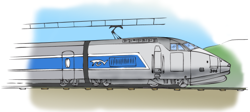


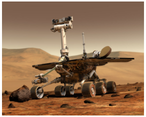


Time-dependent systems

- We are interested in timed systems



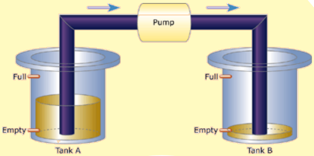




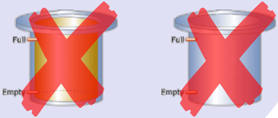
- ... and in their analysis and control

Model-checking and control

system:

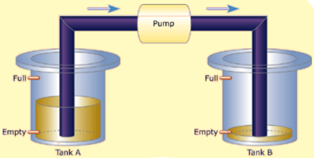


property:

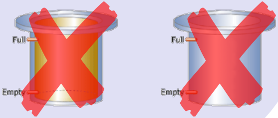


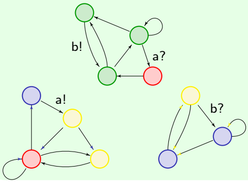
Model-checking and control

system:



property:

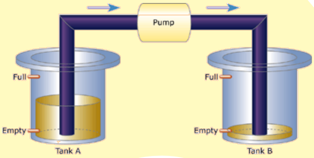




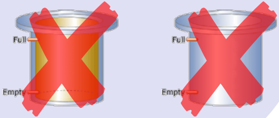
$$\mathbf{AG}(\neg \texttt{B.overfull} \wedge \neg \texttt{B.dried\_up})$$

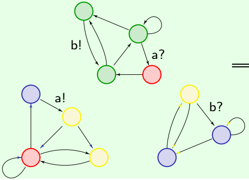
Model-checking and control

system:



property:



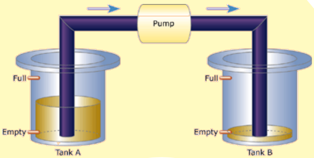


$\Longrightarrow$ algorithm $\Longleftarrow$

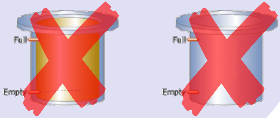
AG($\neg$`B.overfull` $\wedge$ $\neg$`B.dried_up`)

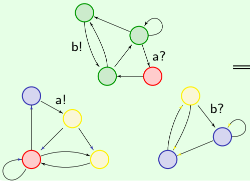
Model-checking and control

system:



property:



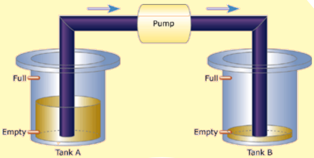


$\Longrightarrow$ model-checking algorithm $\Longleftarrow$ **AG**($\neg$`B.overfull` $\wedge$ $\neg$`B.dried_up`)

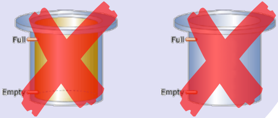
yes/no

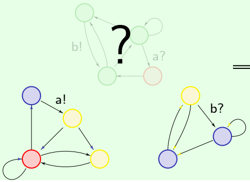
Model-checking and control

system:



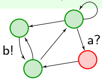
property:





control/synthesis algorithm

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



An example: The task graph scheduling problem

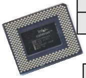
Compute $D\times(C\times(A+B))+(A+B)+(C\times D)$ using two processors:

P_1 (fast):

time	
+	2 picoseconds
×	3 picoseconds

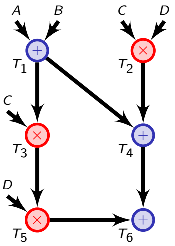
energy	
idle	10 Watt
in use	90 Watts

P_2 (slow):



time	
+	5 picoseconds
×	7 picoseconds

energy	
idle	20 Watts
in use	30 Watts



[BFLM10] Bouyer, Fahrenberg, Larsen, Markey. Quantitative Analysis of Real-Time Systems using Priced Timed Automata *(Communication of the ACM).*

An example: The task graph scheduling problem

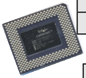
Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

P_1 (fast):

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

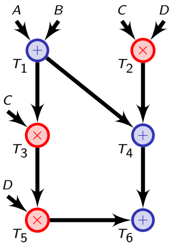
energy	
idle	10 Watt
in use	90 Watts

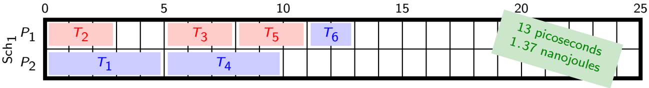
P_2 (slow):



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

energy	
idle	20 Watts
in use	30 Watts

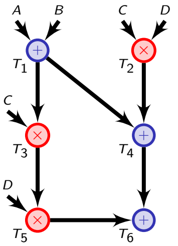




[BFLM10] Bouyer, Fahrenberg, Larsen, Markey. Quantitative Analysis of Real-Time Systems using Priced Timed Automata *(Communication of the ACM)*.

An example: The task graph scheduling problem

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

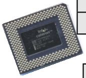


P_1 (fast):

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

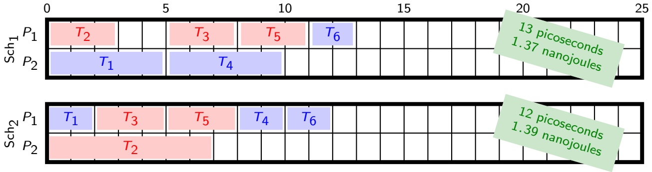
energy	
idle	10 Watt
in use	90 Watts

P_2 (slow):



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

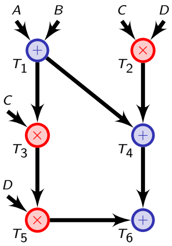
energy	
idle	20 Watts
in use	30 Watts



[BFLM10] Bouyer, Fahrenberg, Larsen, Markey. Quantitative Analysis of Real-Time Systems using Priced Timed Automata *(Communication of the ACM)*.

An example: The task graph scheduling problem

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

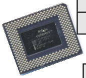


P_1 (fast):

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

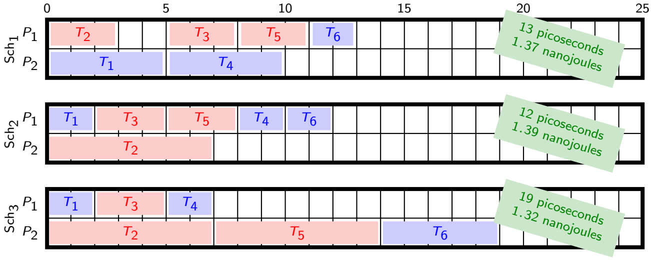
energy	
idle	10 Watt
in use	90 Watts

P_2 (slow):



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

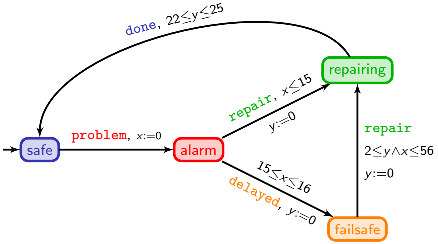
energy	
idle	20 Watts
in use	30 Watts



[BFLM10] Bouyer, Fahrenberg, Larsen, Markey. Quantitative Analysis of Real-Time Systems using Priced Timed Automata *(Communication of the ACM)*.

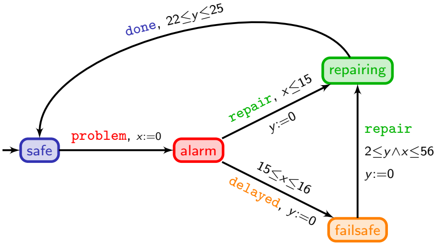
Outline

The model of timed automata



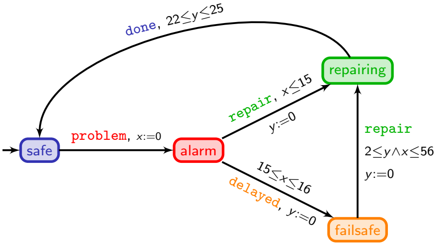
[AD94] Alur, Dill. A Theory of Timed Automata *(Theoretical Computer Science)*.

The model of timed automata



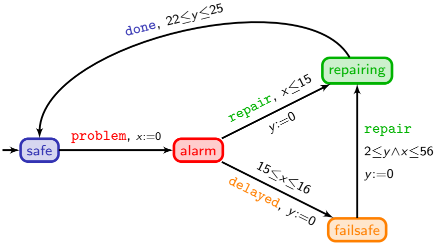
	safe
x	0
y	0

The model of timed automata



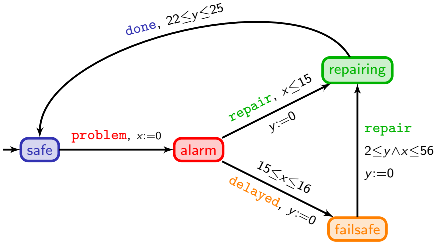
	safe	$\xrightarrow{23}$	safe
x	0		23
y	0		23

The model of timed automata



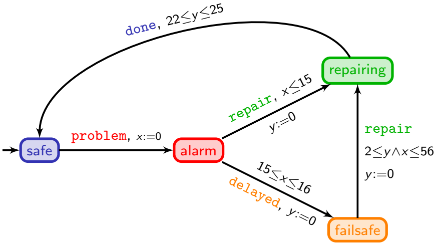
	safe	$\xrightarrow{23}$	safe	$\xrightarrow{\texttt{problem}}$	alarm
x	0		23		0
y	0		23		23

The model of timed automata



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

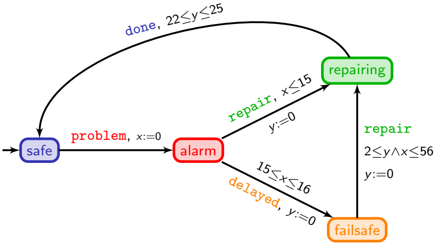
The model of timed automata



	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
⋯	15.6
	0

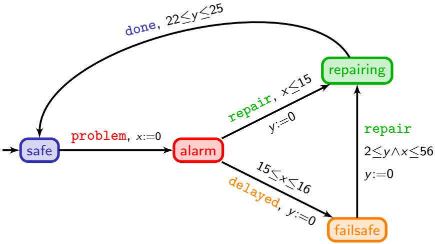
The model of timed automata



	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

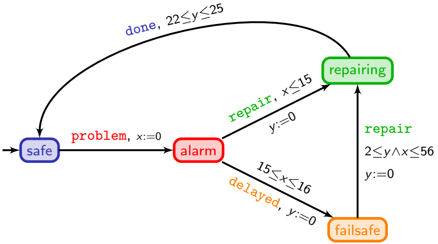
The model of timed automata



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

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

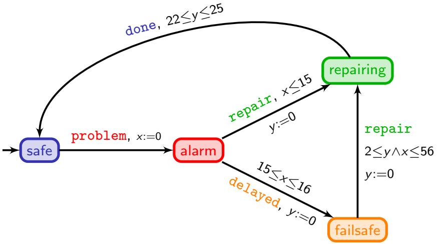
The model of timed automata



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

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

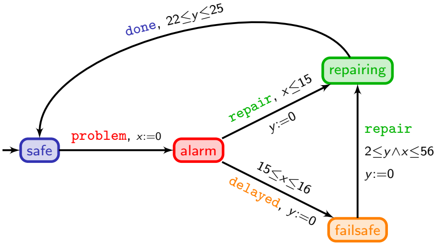
The model of timed automata



	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

The model of timed automata

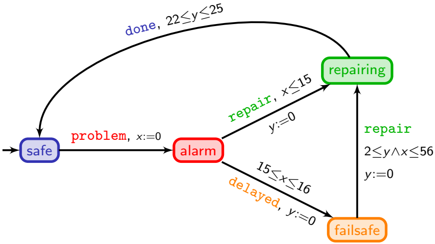


(clock) valuation

	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

The model of timed automata



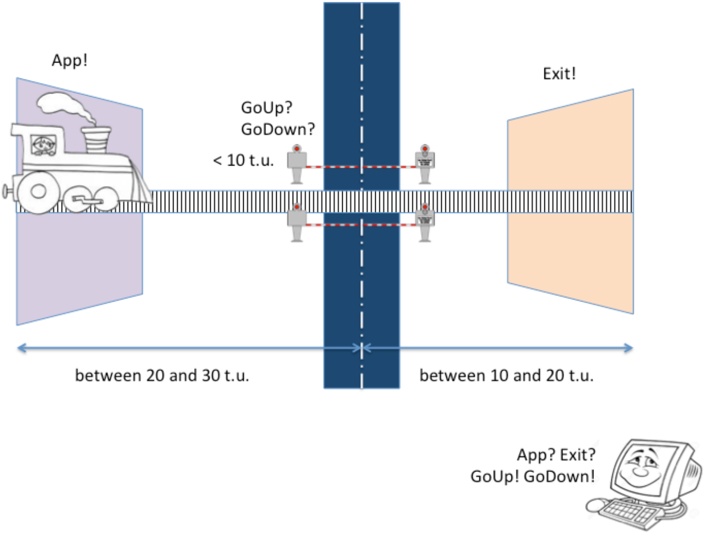
(clock) valuation

	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

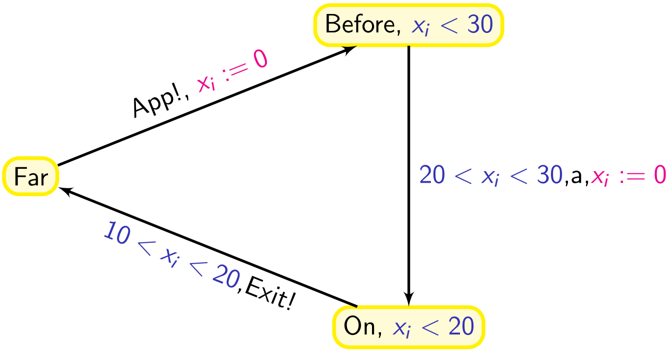
This run reads the timed word
$(\texttt{problem}, 23)(\texttt{delayed}, 38.6)(\texttt{repair}, 40.9)(\texttt{done}, 63)$

The train crossing example



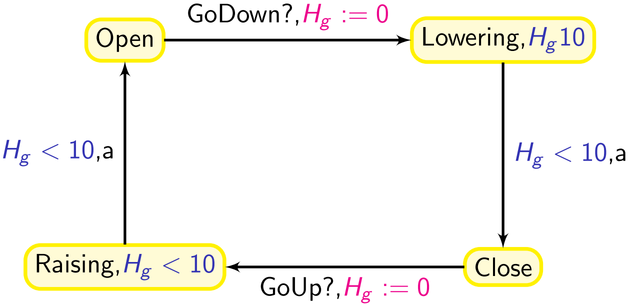
Modelling the train crossing example

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



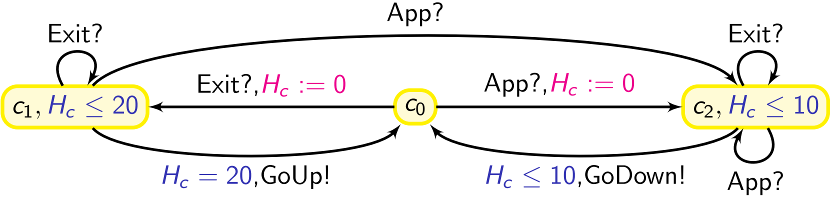
The train crossing example – cont'd

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?

Back to the task graph scheduling problem

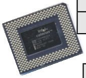
Compute $D\times(C\times(A+B))+(A+B)+(C\times D)$ using two processors:

P_1 (fast):

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

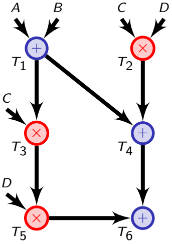
energy	
idle	10 Watt
in use	90 Watts

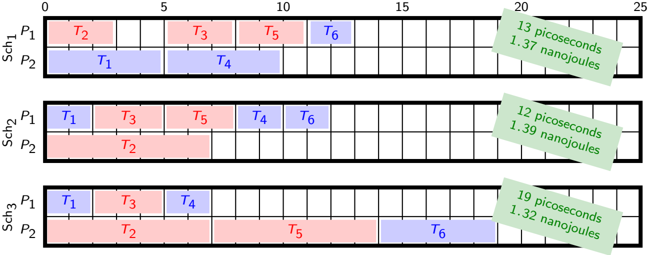
P_2 (slow):



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

energy	
idle	20 Watts
in use	30 Watts

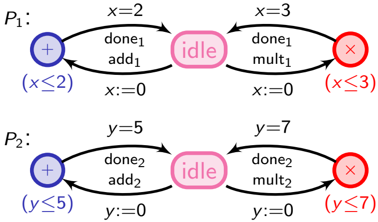




Modelling the task graph scheduling problem

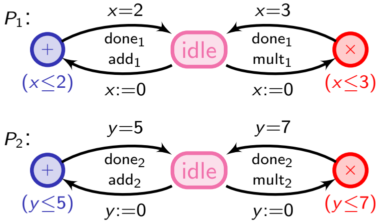
Modelling the task graph scheduling problem

- Processors

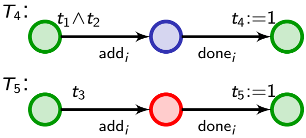


Modelling the task graph scheduling problem

- Processors

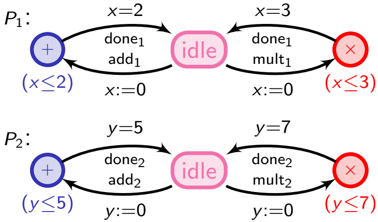


- Tasks

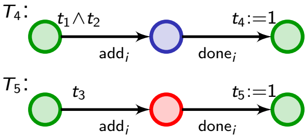


Modelling the task graph scheduling problem

- Processors



- Tasks

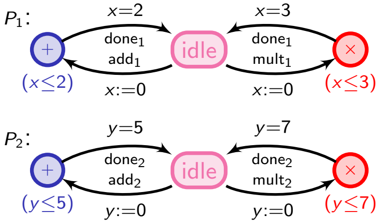


$\rightsquigarrow$ build the synchronized product of all these automata

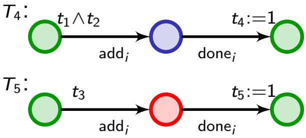
$$(P_1 \parallel P_2) \parallel_s (T_1 \parallel T_2 \parallel \cdots \parallel T_6)$$

Modelling the task graph scheduling problem

- Processors



- Tasks



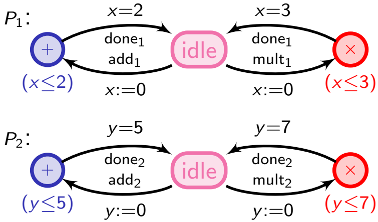
$\rightsquigarrow$ build the synchronized product of all these automata

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

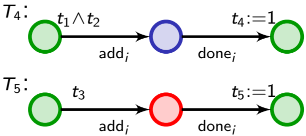
A schedule: a path in the global system which reaches $t_1 \wedge \cdots \wedge t_6$

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?
- ...

What we have so far

- A model which can adequately represent systems with real-time constraint...
- ... on which we can ask relevant questions

What we have so far

- A model which can adequately represent systems with real-time constraint...
- ... on which we can ask relevant questions

Interesting problems

- Which semantics?
  (and be aware of the limits of the choice)
- Algorithms for automatic verification

Discrete-time semantics

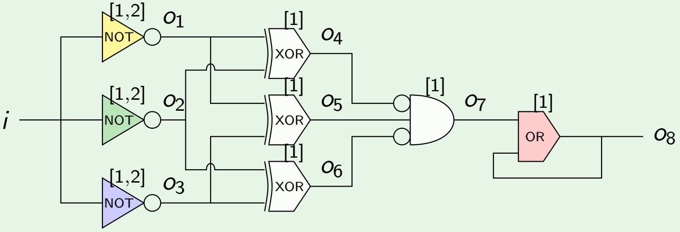
...because computers are digital!

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

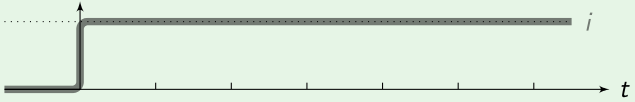
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

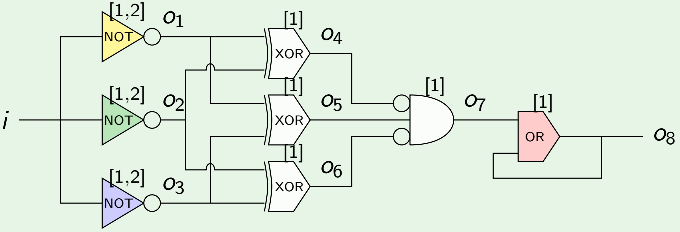


[Alur91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

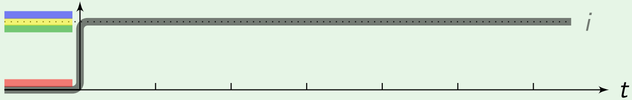
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

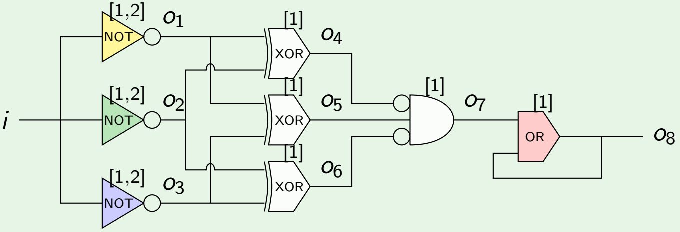


[Alur91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

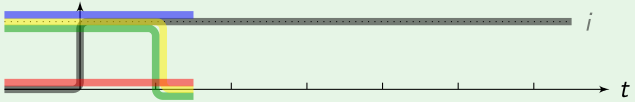
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

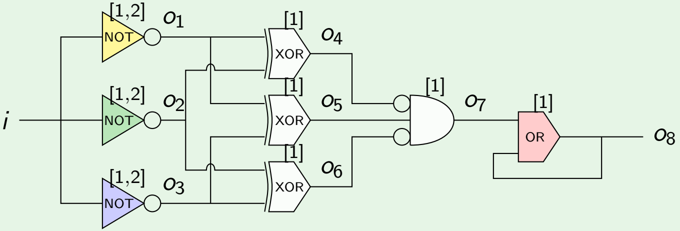


[Alur91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

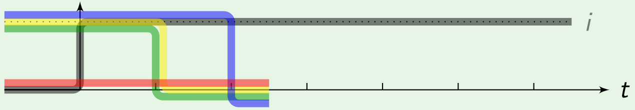
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

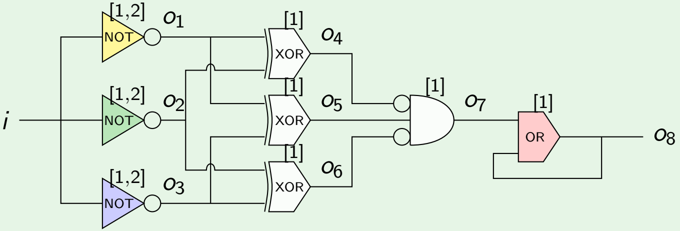


[Alur91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

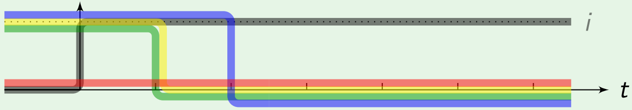
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

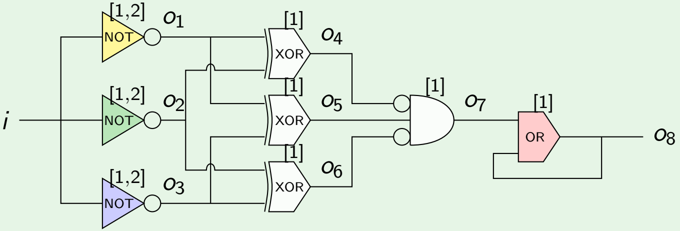


[Alur91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

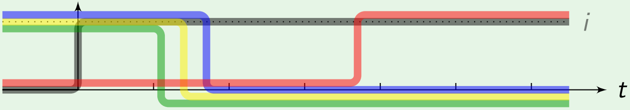
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

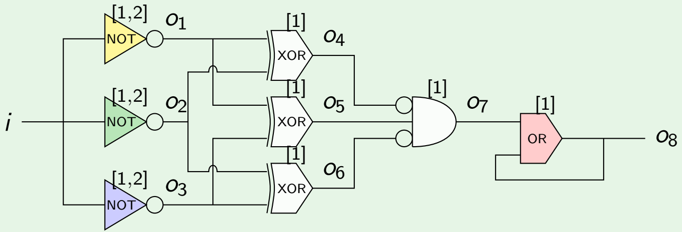


[Alur91] Alur. Techniques for automatic verification of real-time systems. *PhD thesis*, 1991.

Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

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

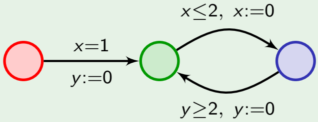
Continuous-time semantics

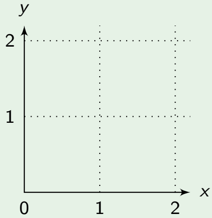
...real-time models for real-time systems!

Continuous-time semantics

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

Example

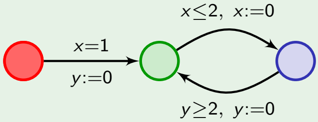


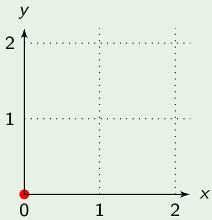


Continuous-time semantics

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

Example

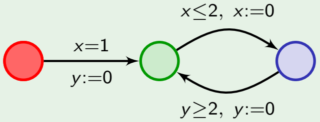


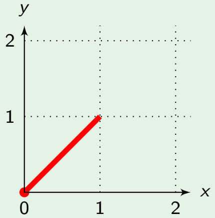


Continuous-time semantics

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

Example

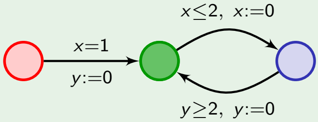


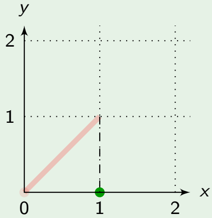


Continuous-time semantics

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

Example

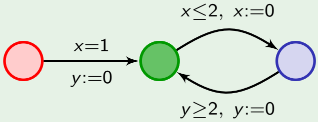


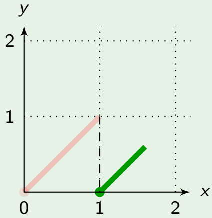


Continuous-time semantics

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

Example

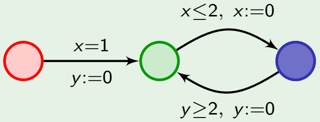


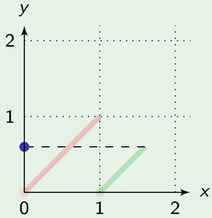


Continuous-time semantics

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

Example

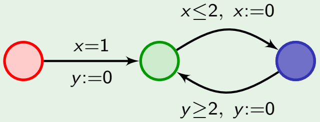


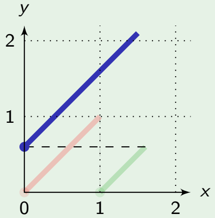


Continuous-time semantics

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

Example

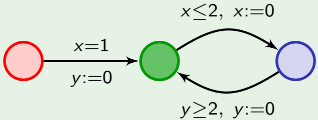


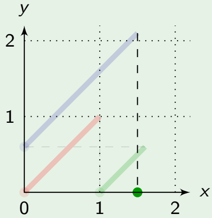


Continuous-time semantics

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

Example

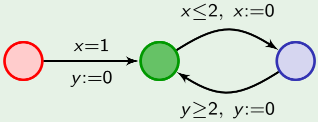


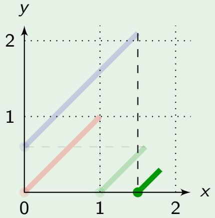


Continuous-time semantics

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

Example

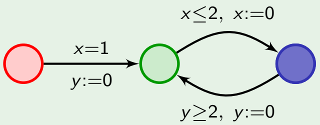


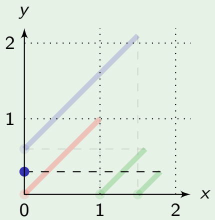


Continuous-time semantics

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

Example

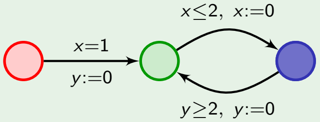


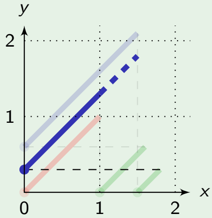


Continuous-time semantics

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

Example

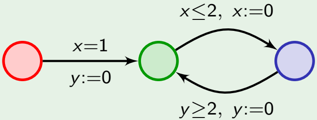


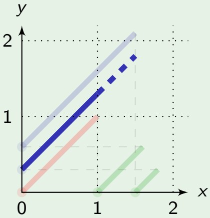


Continuous-time semantics

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

Example



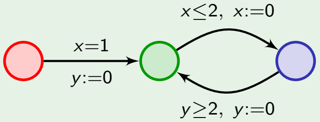


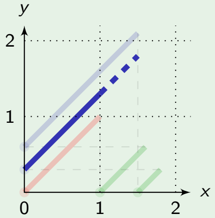
We will focus on the continuous-time semantics, since this is an adequate abstraction of real-time systems

Continuous-time semantics

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

Example

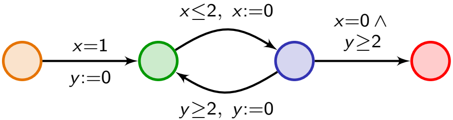


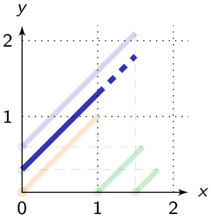


We will focus on the continuous-time semantics, since this is an adequate abstraction of real-time systems

Known limits: robustness issues (we will comment on that later)

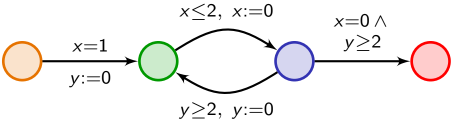
Analyzing timed automata

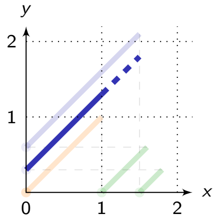




Can we reach state ?

Analyzing timed automata

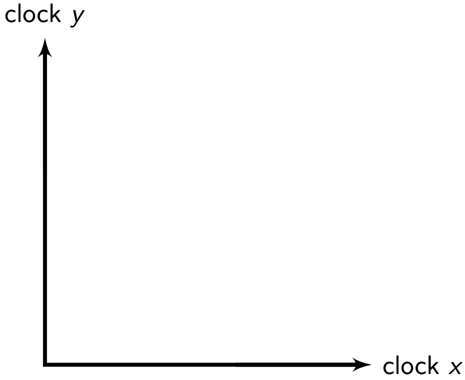




Can we reach state ?

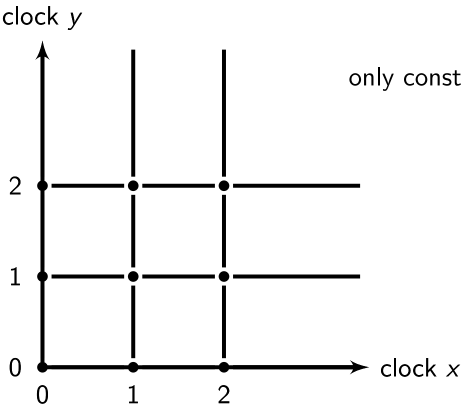
- Problem: the set of configurations is infinite
  $\rightsquigarrow$ classical methods for finite-state systems cannot be applied
- Positive key point: variables (clocks) increase at the same speed

Crux idea: Region abstraction



[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

Crux idea: Region abstraction

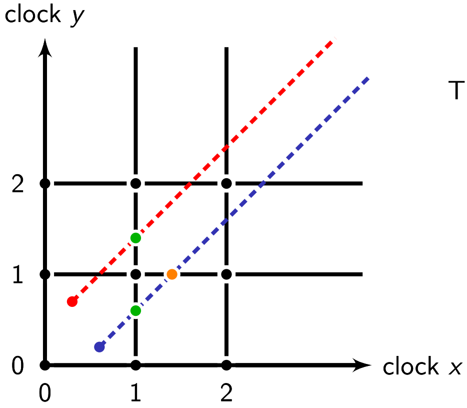


only constraints: $x \sim c$ with $c \in \{0,1,2\}$
$y \sim c$ with $c \in \{0,1,2\}$

- "compatibility" between regions and constraints

[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

Crux idea: Region abstraction



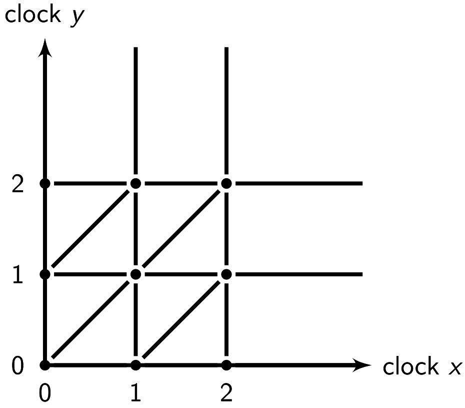
The path

- can be fired from
- cannot be fired from

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

[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

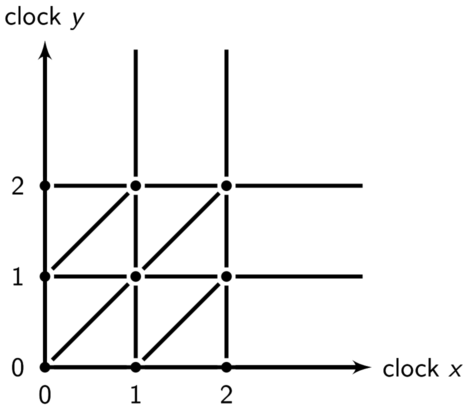
Crux idea: Region abstraction



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

[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

Crux idea: Region abstraction

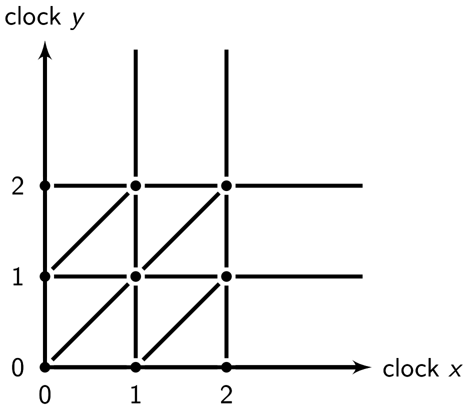


- "compatibility" between regions and constraints
- "compatibility" between regions and time elapsing

$\rightsquigarrow$ an equivalence of finite index

[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

Crux idea: Region abstraction



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

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

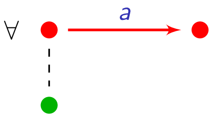
[AD94] Alur, Dill. A theory of timed automata *(Theoretical Computer Science)*.

Time-abstract bisimulation

This is a relation between • and • such that:

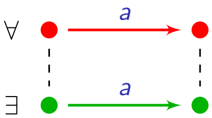
Time-abstract bisimulation

This is a relation between • and • such that:



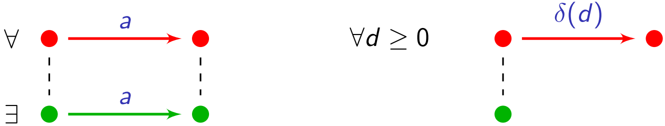
Time-abstract bisimulation

This is a relation between • and • such that:



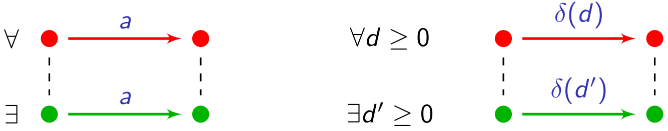
Time-abstract bisimulation

This is a relation between • and • such that:



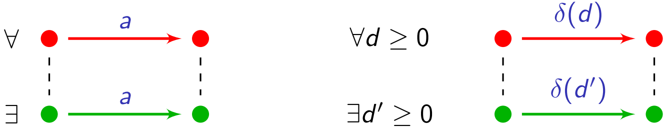
Time-abstract bisimulation

This is a relation between • and • such that:



Time-abstract bisimulation

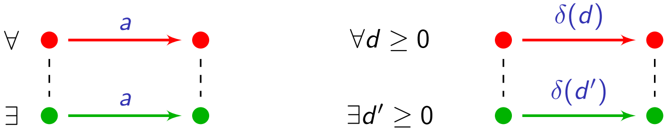
This is a relation between • and • such that:



... 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:

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

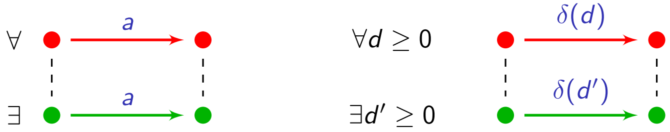
... and vice-versa (swap • and •).

Consequence

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

Time-abstract bisimulation

This is a relation between • and • such that:



... and vice-versa (swap • and •).

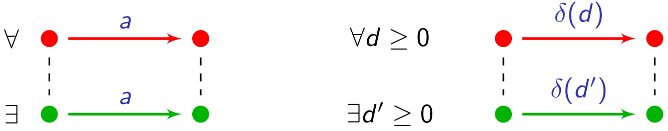
Consequence

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

$\forall v_1' \in R_1$

Time-abstract bisimulation

This is a relation between • and • such that:

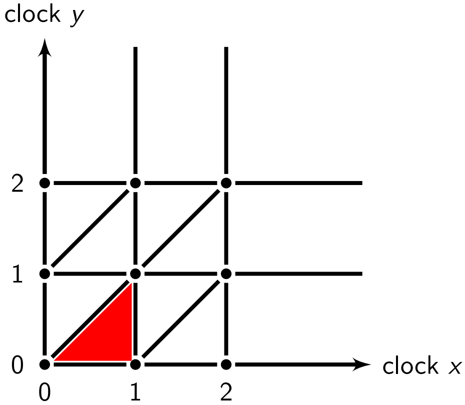


... and vice-versa (swap • and •).

Consequence

$$
\begin{tikzcd}
\forall & (\ell_1, v_1) \arrow[r, "d_1{,}a_1"] \arrow[d] & (\ell_2, v_2) \arrow[r, "d_2{,}a_2"] \arrow[d] & (\ell_3, v_3) \arrow[r, "d_3{,}a_3"] \arrow[d] & \cdots & \\
 & (\ell_1, R_1) \arrow[r, "a_1"] \arrow[d] & (\ell_2, R_2) \arrow[r, "a_2"] \arrow[d] & (\ell_3, R_3) \arrow[r, "a_3"] \arrow[d] & \cdots & \text{with } v_i \in R_i \\
\forall v'_1 \in R_1 \ \exists & (\ell_1, v'_1) \arrow[r, "d'_1{,}a_1"] & (\ell_2, v'_2) \arrow[r, "d'_2{,}a_2"] & (\ell_3, v'_3) \arrow[r, "d'_3{,}a_3"] & \cdots & \text{with } v'_i \in R_i
\end{tikzcd}
$$

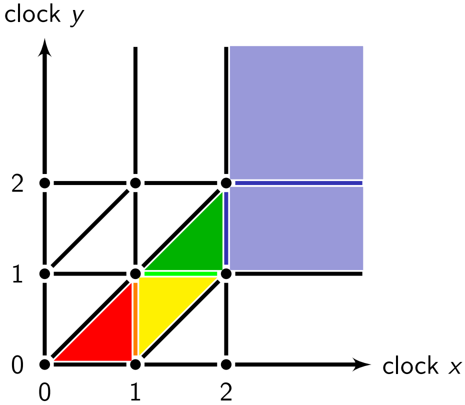
The region abstraction



- region R defined by:

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

The region abstraction

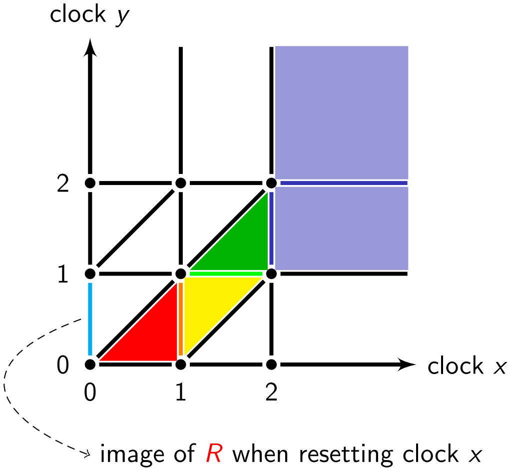


- region R defined by:

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

- time successors of R

The region abstraction



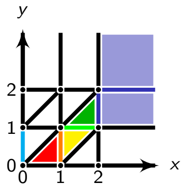
- region R defined by:

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

- time successors of R

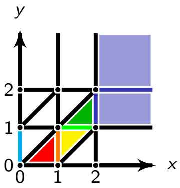
The construction of the region graph

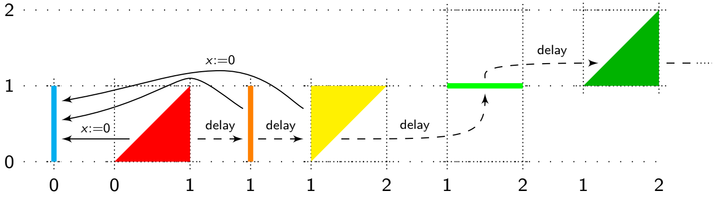
It “mimicks” the behaviours of the clocks.



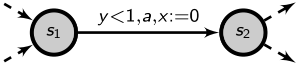
The construction of the region graph

It “mimicks” the behaviours of the clocks.



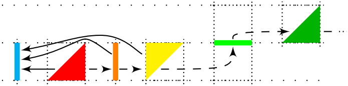


Region automaton $\equiv$ finite bisimulation quotient



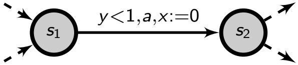
timed automaton

$\otimes$



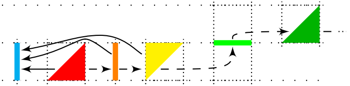
region graph

Region automaton ≡ finite bisimulation quotient

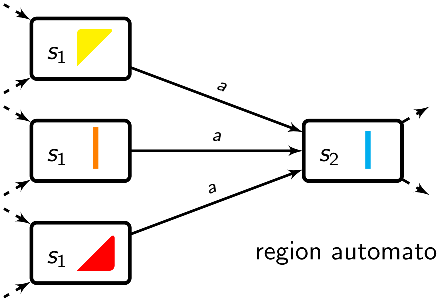


timed automaton

$\otimes$

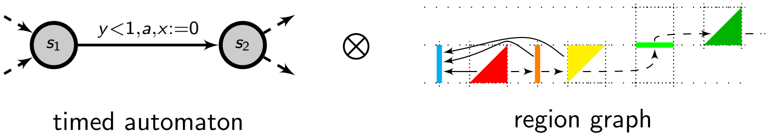


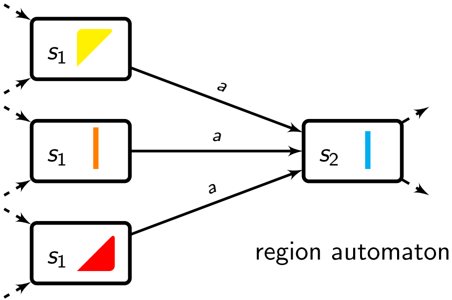
region graph



region automaton

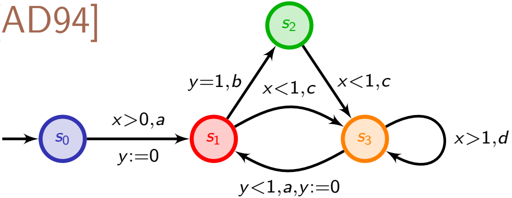
Region automaton ≡ finite bisimulation quotient



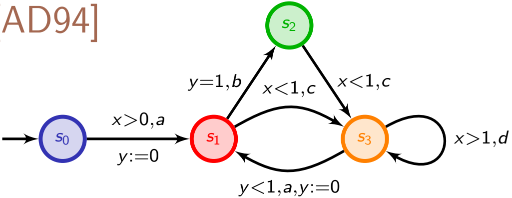


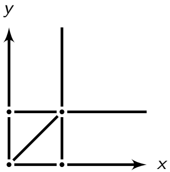
language(reg. aut.) = UNTIME(language(timed aut.))

An example [AD94]

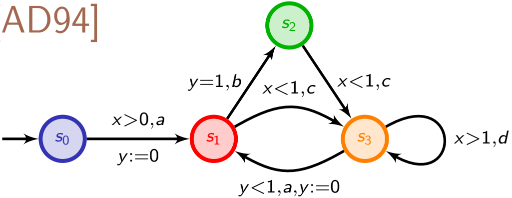


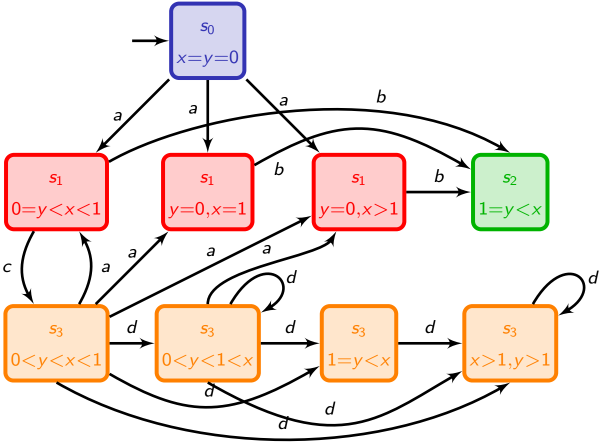
An example [AD94]

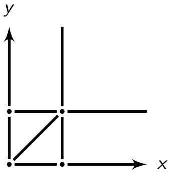


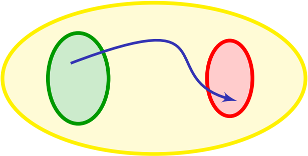


An example [AD94]

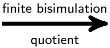


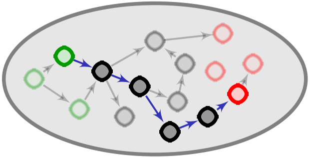




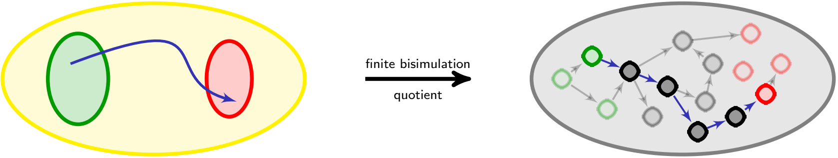


timed automaton





large (but finite) automaton
(region automaton)

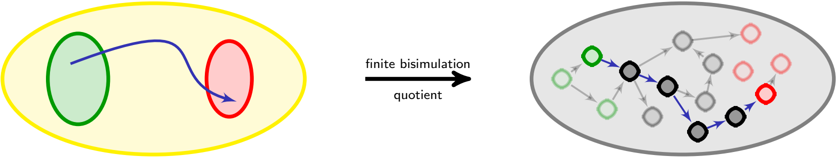


timed automaton

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:

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



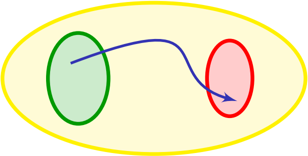
timed automaton

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:

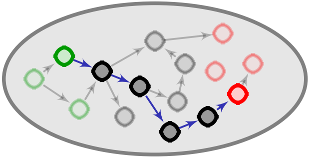
$$\prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|}$$

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



finite bisimulation

quotient



timed automaton

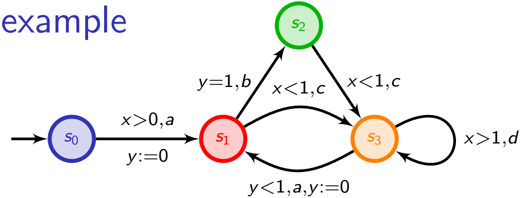
large (but finite) automaton
(region automaton)

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

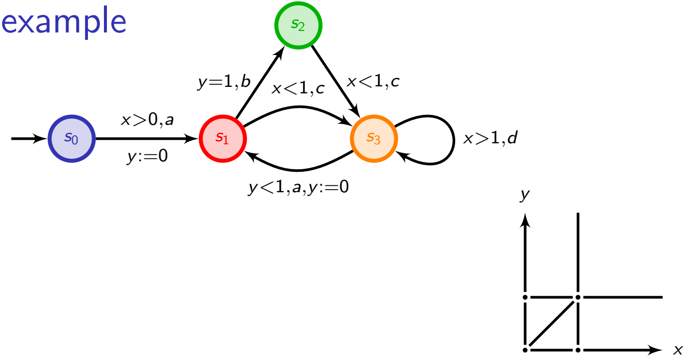
$$\prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|}$$

- 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)

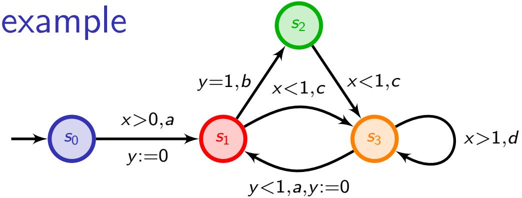
Back to the example

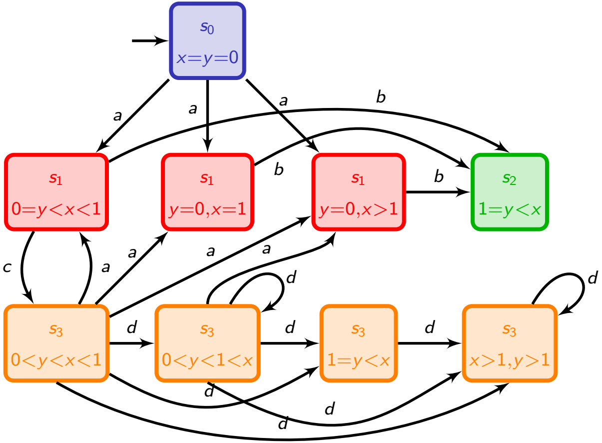


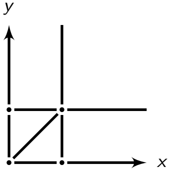
Back to the example



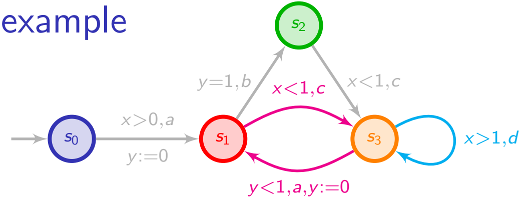
Back to the example

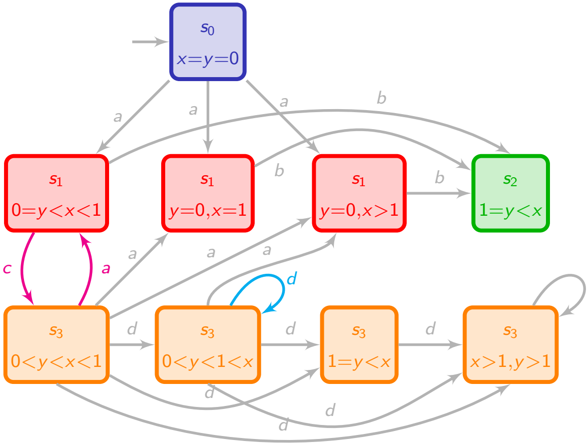


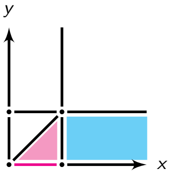




Back to the example

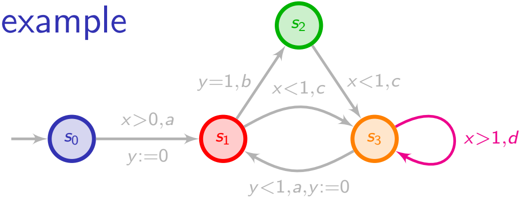


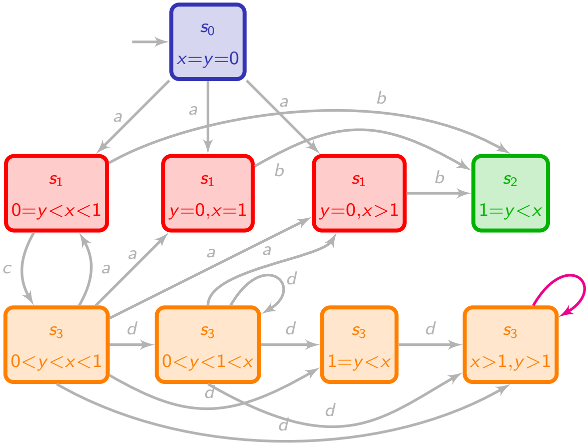


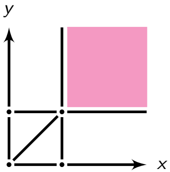


Zeno cycles

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].

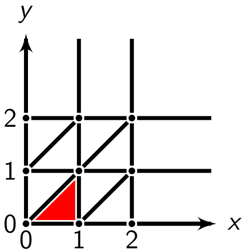
- PSPACE upper bound: guess a path in the region automaton

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].

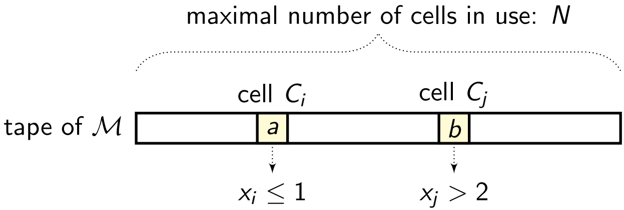
- PSPACE upper bound: guess a path in the region automaton
- PSPACE lower bound: by reduction from a linearly-bounded Turing machine $\mathcal{M}$

Complexity issues

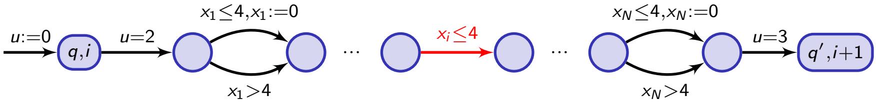
Theorem [AD90,AD94]

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].

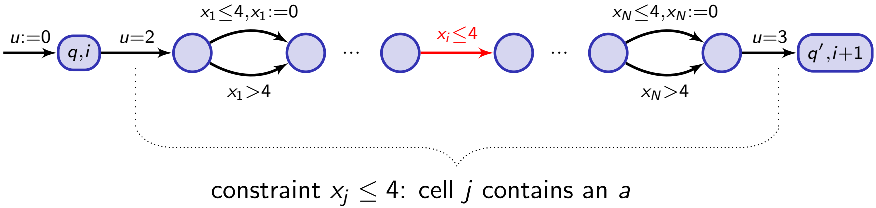
- PSPACE upper bound: guess a path in the region automaton
- PSPACE lower bound: by reduction from a linearly-bounded Turing machine $\mathcal{M}$



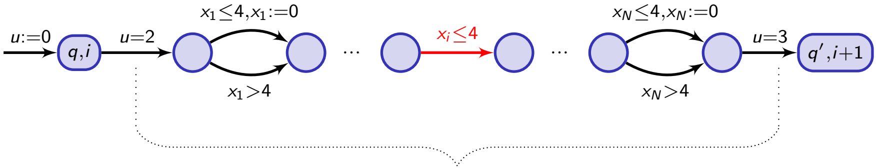
Example of the simulation of a rule $(q, a, b, q', \rightarrow)$:



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

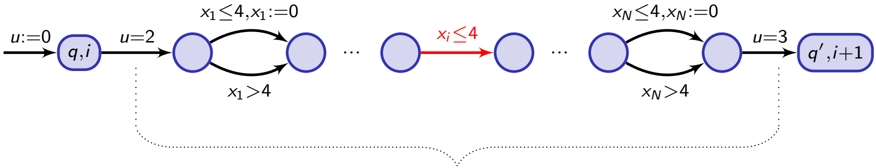


Example of the simulation of a rule $(q, a, b, q', \rightarrow)$:



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

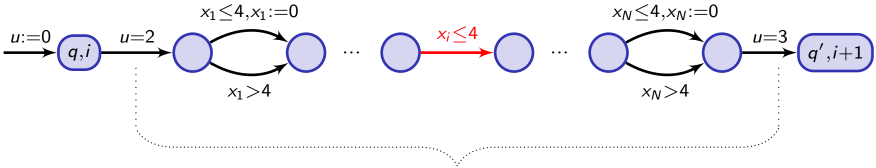
Example of the simulation of a rule $(q, a, b, q', \rightarrow)$:



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

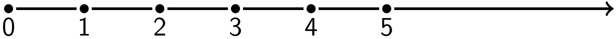
Example of the simulation of a rule $(q, a, b, q', \rightarrow)$:



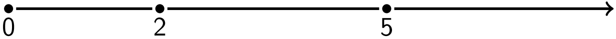
constraint $x_j \leq 4$: cell j contains an a
constraint $x_j > 4$: cell j contains a b

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

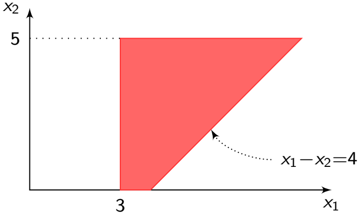
- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
  - o-minimal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

What about the practice?

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

- Symbolic representation: zones

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



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

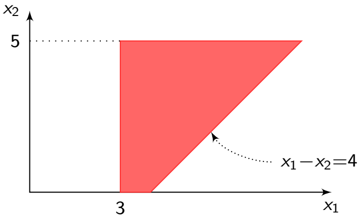
DBM: Difference Bound Matrice [BM83,Dill89]

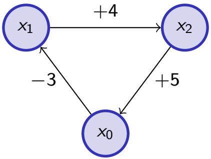
What about the practice?

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

- Symbolic representation: zones

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



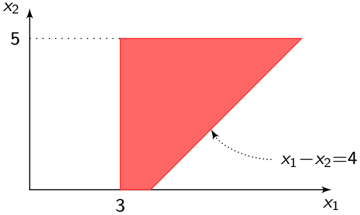


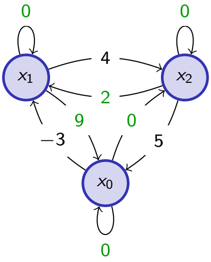
What about the practice?

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

- Symbolic representation: zones

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



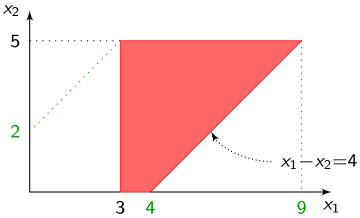


What about the practice?

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

- Symbolic representation: zones

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



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

"normal form"

What about the practice?

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

- Symbolic representation: zones
- Needs of (correct) extrapolation operators... [Bou04,BBLP06]

[Bou04] Bouyer. Forward analysis of updatable timed automata *(Formal Methods in System Design)*.
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata *(International Journal on Software Tools for Technology Transfer)*.

What about the practice?

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

- Symbolic representation: zones
- Needs of (correct) extrapolation operators... [Bou04,BBLP06]
- ... or clever inclusion tests [HSW12,HSW13]

[Bou04] Bouyer. Forward analysis of updatable timed automata *(Formal Methods in System Design)*.
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata *(International Journal on Software Tools for Technology Transfer)*.
[HSW12] Herbreteau, Srivathsan, Walukiewicz. Better abstractions for timed automata *(LICS'12)*.
[HSW13] Herbreteau, Srivathsan, Walukiewicz. Lazy abstractions for timed automata *(CAV'13)*

Before going further...

Which hypotheses did we make?

- timestamps taken in $\mathbb{R}_+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}_+$

Before going further...

Which hypotheses did we make?

- timestamps taken in $\mathbb{R}_+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}_+$
- constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}_+$, but not in $\mathbb{R}_+$!

Before going further...

Which hypotheses did we make?

- timestamps taken in $\mathbb{R}_+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}_+$
- constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}_+$, but not in $\mathbb{R}_+$!
- clock constraints of the form $x \sim c$

Before going further...

Which hypotheses did we make?

- timestamps taken in $\mathbb{R}_+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}_+$
- constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}_+$, but not in $\mathbb{R}_+$!
- clock constraints of the form $x \sim c$
  - $x - y \sim c$ are fine as well
  - no other kind of clock constraints!

Before going further...

Which hypotheses did we make?

- timestamps taken in $\mathbb{R}_+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}_+$
- constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}_+$, but not in $\mathbb{R}_+$!
- clock constraints of the form $x \sim c$
  - $x - y \sim c$ are fine as well
  - no other kind of clock constraints!
- resets of clocks to 0 only; we can reset to integral values as well

Before going further...

Which hypotheses did we make?

- timestamps taken in $\mathbb{R}_+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}_+$
- constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}_+$, but not in $\mathbb{R}_+$!
- clock constraints of the form $x \sim c$
  - $x - y \sim c$ are fine as well
  - no other kind of clock constraints!
- resets of clocks to 0 only; we can reset to integral values as well
  - more involved updates can be used as well, but they don't interact very well with diagonal constraints. So one needs to be careful

Limits of the model

- Any slight extension of the model is undecidable:
  - Richer clock constraints $x + y = c$, $2x \leq y$
  - Richer updates: $x := x + 1$
  - ...

Limits of the model

- Any slight extension of the model is undecidable:
  - Richer clock constraints $x + y = c$, $2x \leq y$
  - Richer updates: $x := x + 1$
  - ...

- The inclusion problem

$$L(\mathcal{A}) \subseteq L(\mathcal{B})$$

  is undecidable [AD94]

Limits of the model

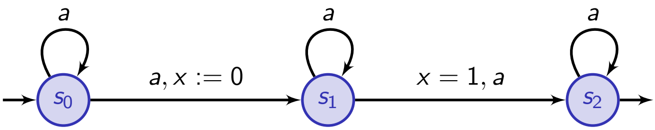
- Any slight extension of the model is undecidable:
  - Richer clock constraints $x + y = c$, $2x \leq y$
  - Richer updates: $x := x + 1$
  - ...

- The inclusion problem

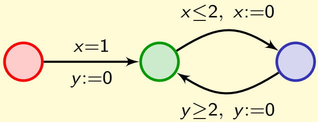
$$L(\mathcal{A}) \subseteq L(\mathcal{B})$$

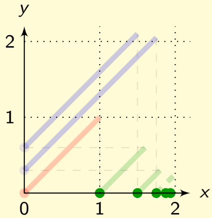
  is undecidable [AD94]

- One cannot complement, determinize timed automata

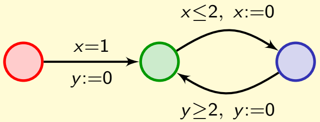


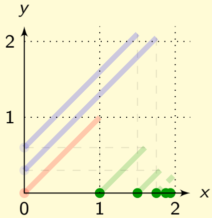
An important issue: Robustness and implementability





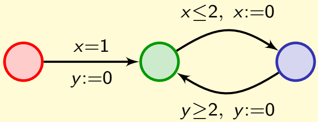
An important issue: Robustness and implementability

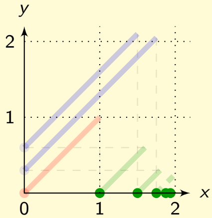




$\rightsquigarrow$ Value of clock x when hitting ○ is converging, even though global time diverges

An important issue: Robustness and implementability

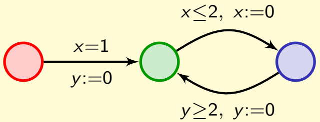


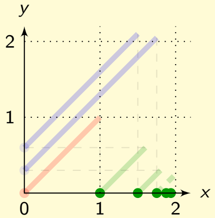


$\rightsquigarrow$ Value of clock x when hitting ○ is converging, even though global time diverges

Can we implement such a strategy??

An important issue: Robustness and implementability





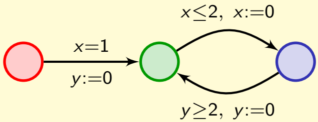
$\rightsquigarrow$ Value of clock x when hitting ○ is converging, even though global time diverges

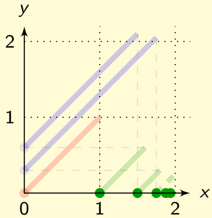
Can we implement such a strategy??

No. But we can detect such behaviours, and give conditions for implementations!

[BMS13] Bouyer, Markey, Sankur. Robustness in timed automata *(RP'13)*.

An important issue: Robustness and implementability





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

Can we implement such a strategy??

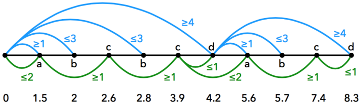
No. But we can detect such behaviours, and give conditions for implementations!

A survey: [BMS13]

[BMS13] Bouyer, Markey, Sankur. Robustness in timed automata *(RP'13)*.

Theoretical recent developments

- Tree automata technics for timed automata analysis [AGK16,AGKS17]

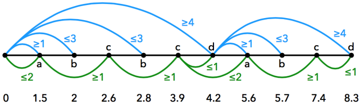


  - Write behaviours as graphs with timing constraints
  - Realize that those graphs have bounded tree-width
  - Express properties using MSO and/or build directly tree automata

[AGK16] Akshay, Gastin, Krishna. Analyzing Timed Systems Using Tree Automata *(CONCUR'16)*.
[AGKS17] Akshay, Gastin, Krishna, Sarkar. Towards an Efficient Tree Automata based technique for Timed Systems *(CONCUR'17)*.

Theoretical recent developments

- Tree automata technics for timed automata analysis [AGK16,AGKS17]



  - Write behaviours as graphs with timing constraints
  - Realize that those graphs have bounded tree-width
  - Express properties using MSO and/or build directly tree automata

- Compute and use the reachability relation [CJ99,QSW17]

[AGK16] Akshay, Gastin, Krishna. Analyzing Timed Systems Using Tree Automata *(CONCUR'16)*.
[AGKS17] Akshay, Gastin, Krishna, Sarkar. Towards an Efficient Tree Automata based technique for Timed Systems *(CONCUR'17)*.
[CJ99] Comon, Jurski. Timed Automata and the Theory of Real Numbers *(CONCUR'99)*.
[QSW17] Quaas, Shirmohammadi, Worrell. Revisiting Reachability in Timed Automata *(LICS'17)*.

Outline

Outline

Back to the task-graph scheduling problem

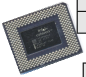
Compute $D\times(C\times(A+B))+(A+B)+(C\times D)$ using two processors:

P_1 (fast):

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

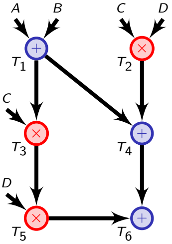
energy	
idle	10 Watt
in use	90 Watts

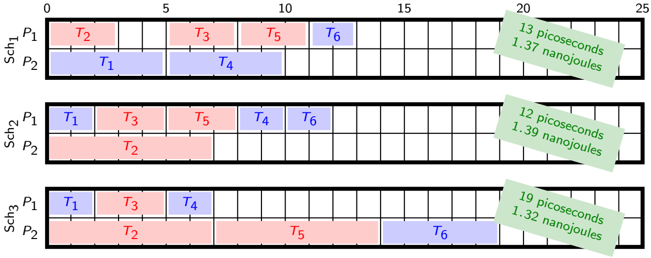
P_2 (slow):



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?

Modelling resources in timed systems

- System resources might be relevant and even crucial information

Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,

Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,

$\rightsquigarrow$ timed automata are not powerful enough!

Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,

$\rightsquigarrow$ timed automata are not powerful enough!

- A possible solution: use hybrid automata

$$\begin{array}{ll} & \text{a discrete control (the mode of the system)} \\ + & \text{continuous evolution of the variables within a mode} \end{array}$$

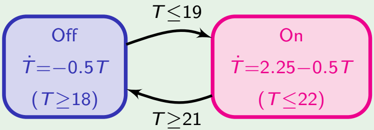
Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,

$\rightsquigarrow$ timed automata are not powerful enough!

- A possible solution: use hybrid automata

The thermostat example



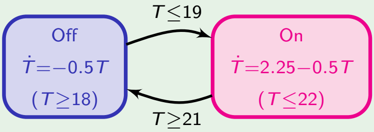
Modelling resources in timed systems

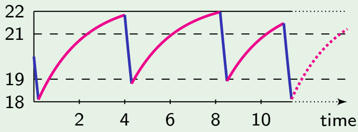
- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,

$\rightsquigarrow$ timed automata are not powerful enough!

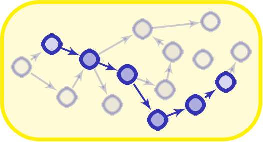
- A possible solution: use hybrid automata

The thermostat example

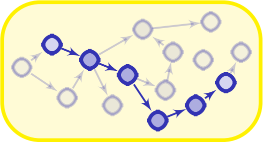




Ok...

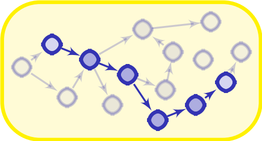


Ok...

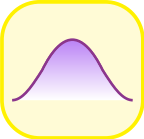


Easy...

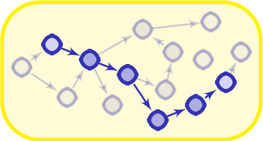
Ok...



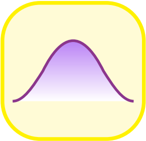
Easy...



Ok...

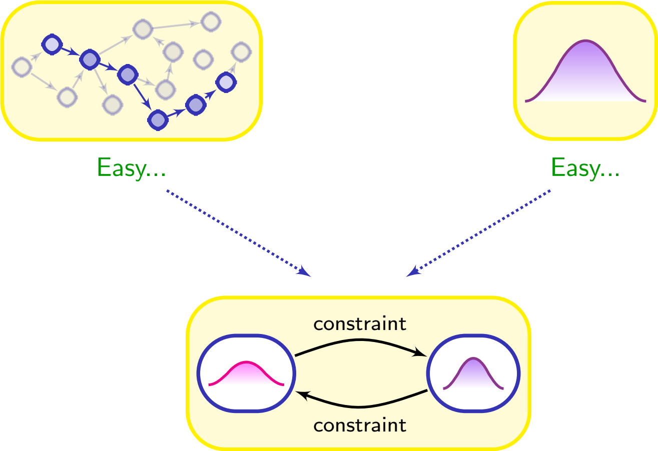


Easy...

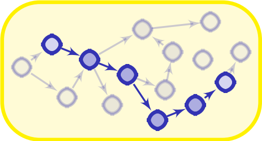


Easy...

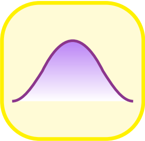
Ok... but?



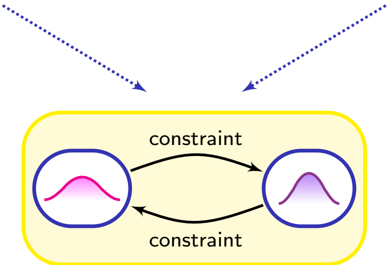
Ok... but?



Easy...



Easy...



Hard!

Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,

  $\rightsquigarrow$ timed automata are not powerful enough!

- A possible solution: use hybrid automata

Theorem [HKPV95]

The reachability problem is undecidable in hybrid automata. Even for the simplest, the so-called stopwatch automata (clocks can be stopped).

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

Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,

  $\rightsquigarrow$ timed automata are not powerful enough!

- A possible solution: use hybrid automata

Theorem [HKPV95]

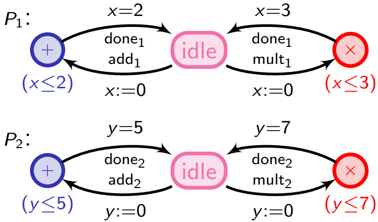
The reachability problem is undecidable in hybrid automata. Even for the simplest, the so-called stopwatch automata (clocks can be stopped).

- An alternative: weighted/priced timed automata [ALP01,BFH+01]
  $\rightsquigarrow$ hybrid variables do not constrain the system
  hybrid variables are observer variables

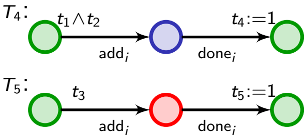
[HKPV95] Henzinger, Kopke, Puri, Varaiya. What's decidable wbout hybrid automata? *(SToC'95)*.
[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Modelling the task graph scheduling problem

- Processors

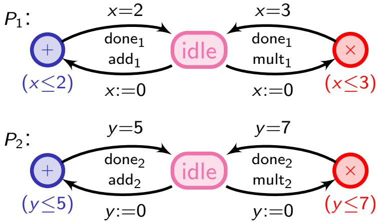


- Tasks

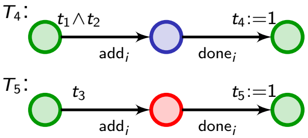


Modelling the task graph scheduling problem

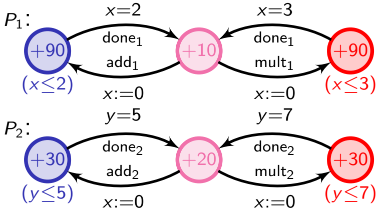
- Processors



- Tasks

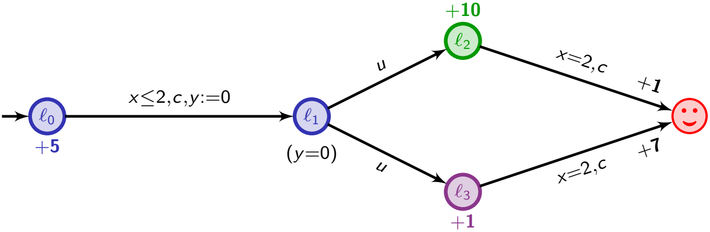


- Modelling energy



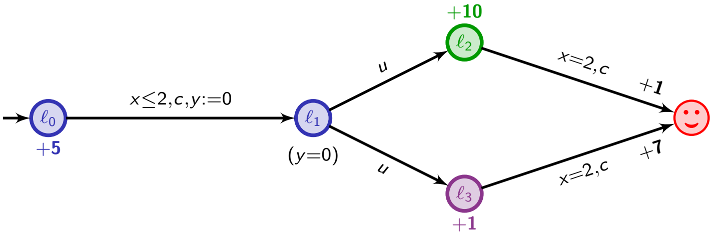
A good schedule is a path in the product automaton with a low cost

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)*.

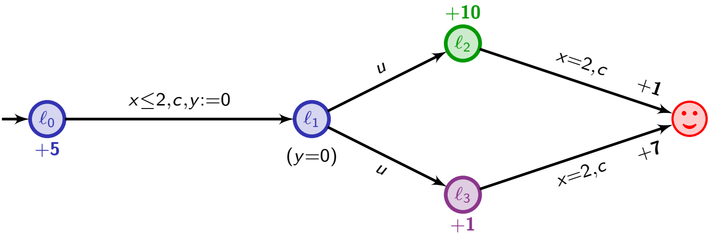
Weighted/priced timed automata [ALP01,BFH+01]



	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]

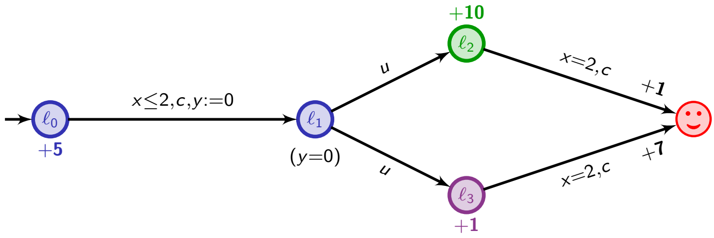


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost :

[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]

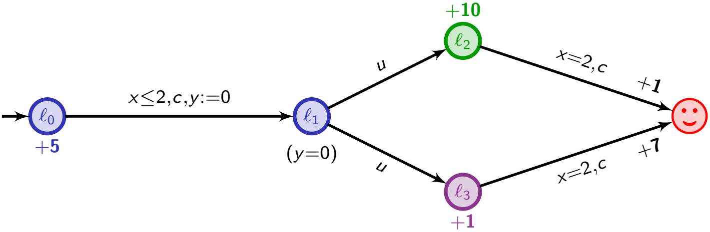


$$
\begin{array}{ccccccccccccc}
 & \ell_0 & \xrightarrow{1.3} & \ell_0 & \xrightarrow{c} & \ell_1 & \xrightarrow{u} & \ell_3 & \xrightarrow{0.7} & \ell_3 & \xrightarrow{c} & ☺ \\
x & 0 & & 1.3 & & 1.3 & & 1.3 & & 2 & & \\
y & 0 & & 1.3 & & 0 & & 0 & & 0.7 & &
\end{array}
$$

cost : 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]

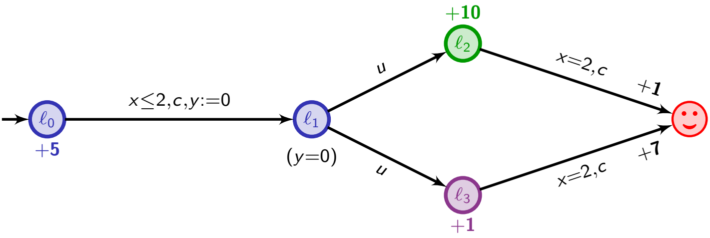


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

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]

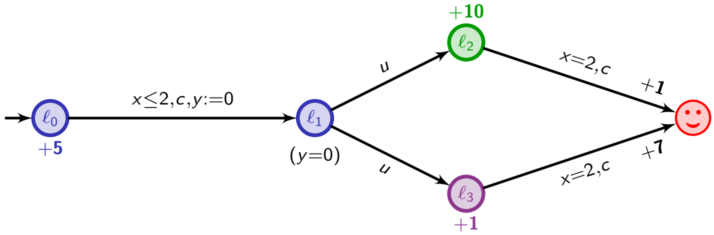


$$
\begin{array}{lcccccccccccc}
 & \ell_0 & \xrightarrow{1.3} & \ell_0 & \xrightarrow{c} & \ell_1 & \xrightarrow{u} & \ell_3 & \xrightarrow{0.7} & \ell_3 & \xrightarrow{c} & ☺ \\
x & 0 & & 1.3 & & 1.3 & & 1.3 & & 2 & & \\
y & 0 & & 1.3 & & 0 & & 0 & & 0.7 & &
\end{array}
$$

cost : 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)*.

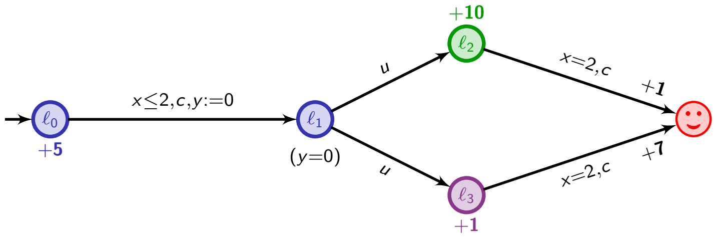
Weighted/priced timed automata [ALP01,BFH+01]



	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]

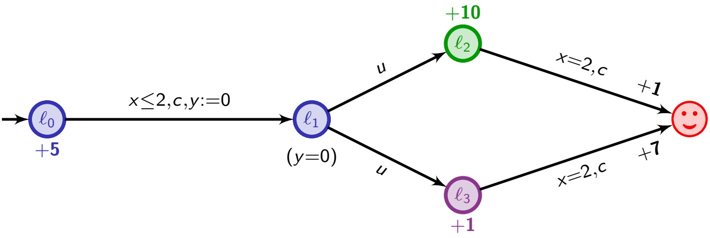


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

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]

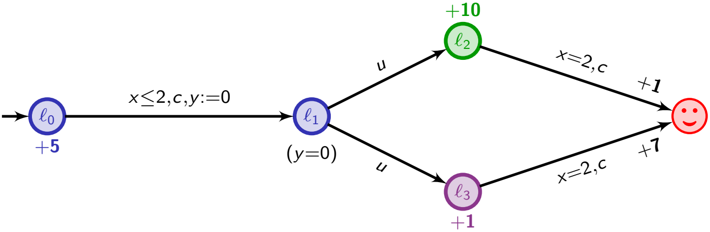


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

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)*.

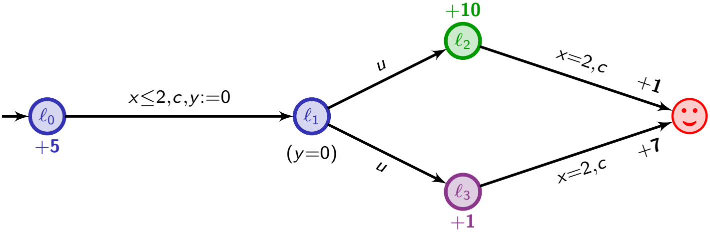
Weighted/priced timed automata [ALP01,BFH+01]



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]

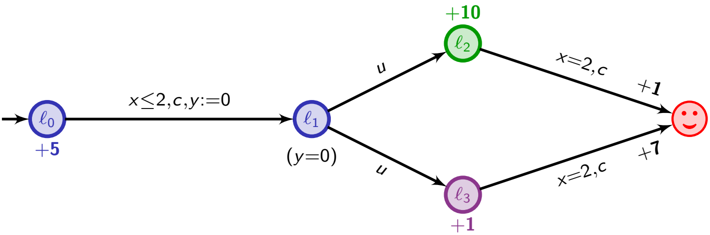


Question: what is the optimal cost for reaching 🙂?

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

[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]

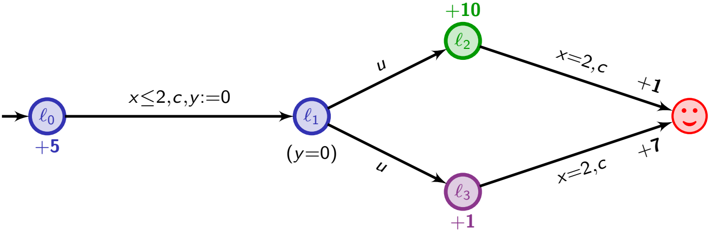


Question: what is the optimal cost for reaching 🙂?

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

[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]

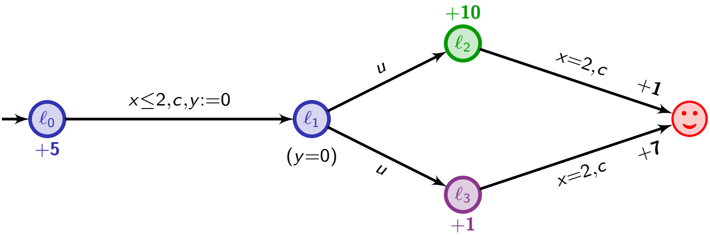


Question: what is the optimal cost for reaching ☺ ?

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

[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]

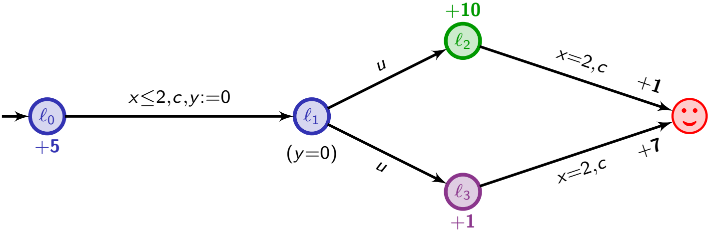


Question: what is the optimal cost for reaching 🙂?

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

[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 ☺?

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

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

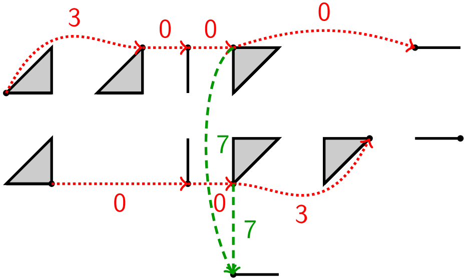
[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Optimal-cost reachability

Theorem [ALP01,BFH+01,BBBR07]

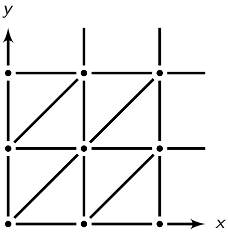
In weighted timed automata, the optimal cost is an integer and can be computed in PSPACE.

- Technical tool: a refinement of the regions, the corner-point abstraction

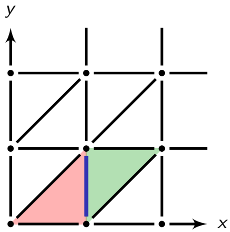


[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.
[BBBR07] Bouyer, Brihaye, Bruyère, Raskin. On the optimal reachability problem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

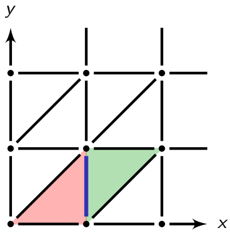
Technical tool: the corner-point abstraction



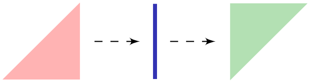
Technical tool: the corner-point abstraction



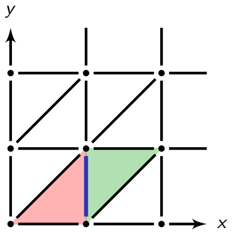
Technical tool: the corner-point abstraction



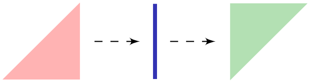
Abstract time successors:



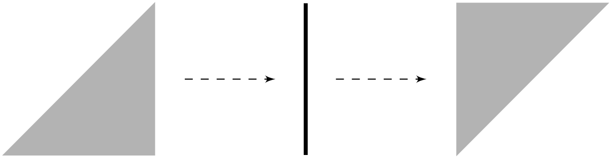
Technical tool: the corner-point abstraction



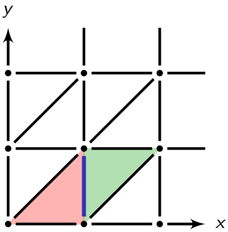
Abstract time successors:



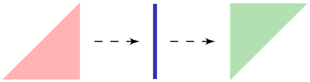
Concrete time successors:



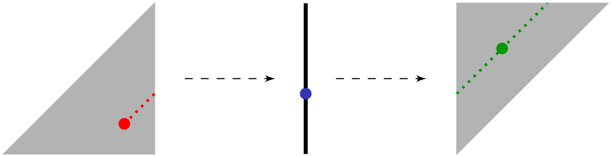
Technical tool: the corner-point abstraction



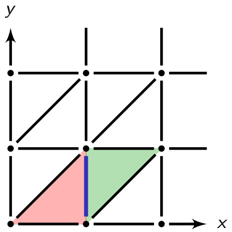
Abstract time successors:



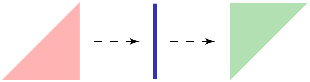
Concrete time successors:



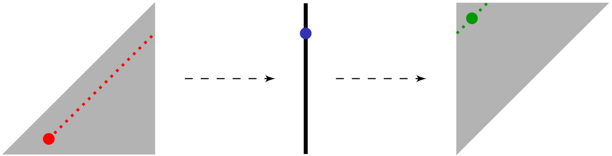
Technical tool: the corner-point abstraction



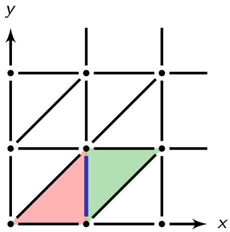
Abstract time successors:



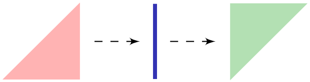
Concrete time successors:



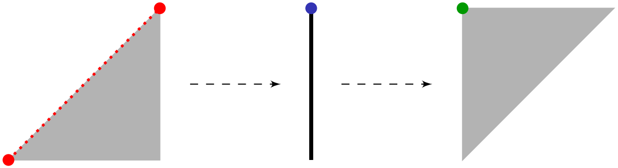
Technical tool: the corner-point abstraction



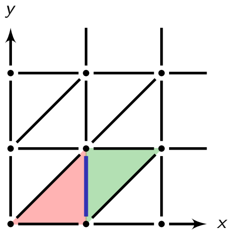
Abstract time successors:



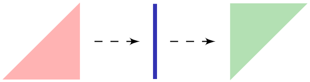
Concrete time successors:



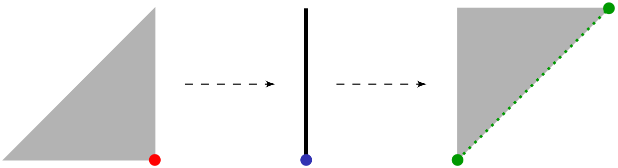
Technical tool: the corner-point abstraction



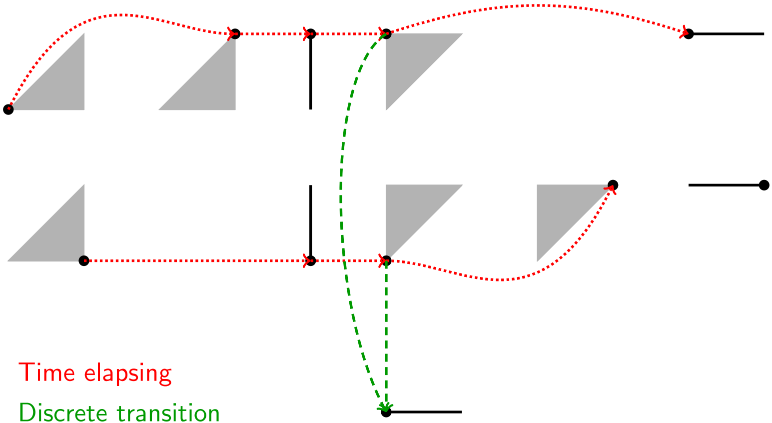
Abstract time successors:



Concrete time successors:



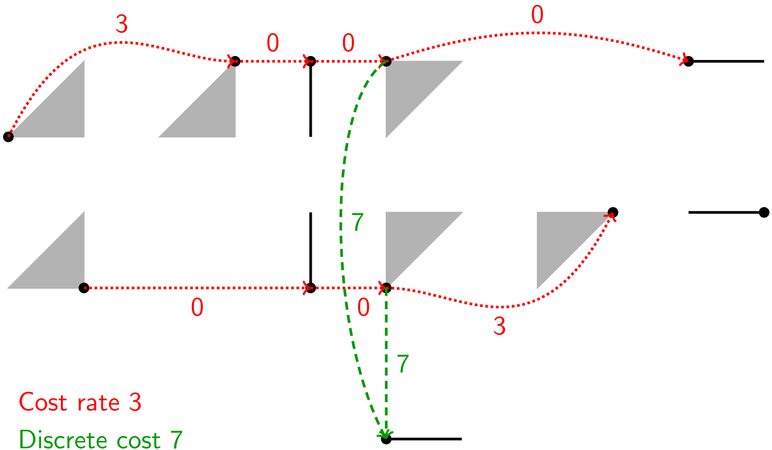
Technical tool: the corner-point abstraction



Time elapsing

Discrete transition

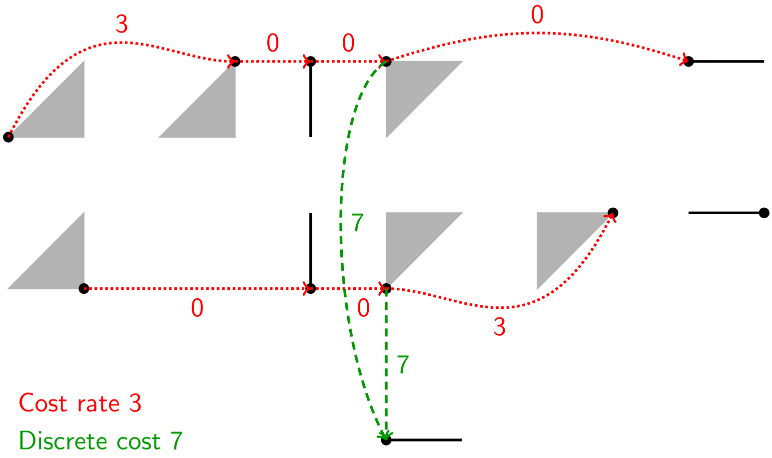
Technical tool: the corner-point abstraction



Cost rate 3

Discrete cost 7

Technical tool: the corner-point abstraction



Optimal cost in the weighted graph
= optimal cost in the weighted timed automaton!

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

$$\circ \xrightarrow{t_1} \circ \xrightarrow[x \leq 2]{t_2} \circ \xrightarrow{t_3} \circ \xrightarrow{t_4} \circ \xrightarrow{t_5} \circ \quad \cdots \quad \left\{ \begin{array}{l} t_1 + t_2 \leq 2 \\ \\ \end{array} \right.$$

From timed to discrete behaviours

Optimal reachability as a linear programming problem

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

From timed to discrete behaviours

Optimal reachability as a linear programming problem

$$\begin{array}{ccccccccccc} & T_1 & & T_2 & & T_3 & & T_4 & & T_5 & \\ \circ & \xrightarrow[y:=0]{t_1} & \circ & \xrightarrow[x\leq 2]{t_2} & \circ & \xrightarrow{t_3} & \circ & \xrightarrow[y\geq 5]{t_4} & \circ & \xrightarrow{t_5} & \circ \quad \cdots \end{array} \quad \begin{cases} t_1+t_2\leq 2 & T_2\leq 2 \\ t_2+t_3+t_4\geq 5 & T_4-T_1\geq 5 \end{cases}$$

From timed to discrete behaviours

Optimal reachability as a linear programming problem

$$T_1 \qquad T_2 \qquad T_3 \qquad T_4 \qquad T_5$$

$$\circ \xrightarrow[y:=0]{t_1} \circ \xrightarrow[x\leq 2]{t_2} \circ \xrightarrow{t_3} \circ \xrightarrow[y\geq 5]{t_4} \circ \xrightarrow{t_5} \circ \quad \cdots \quad \begin{cases} t_1+t_2\leq 2 & T_2\leq 2 \\ t_2+t_3+t_4\geq 5 & T_4-T_1\geq 5 \end{cases}$$

Lemma

Let Z be a bounded zone 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

$$\begin{array}{ccccc} T_1 & T_2 & T_3 & T_4 & T_5 \end{array}$$

$$\circ \xrightarrow[y:=0]{t_1} \circ \xrightarrow[x\leq 2]{t_2} \circ \xrightarrow{t_3} \circ \xrightarrow[y\geq 5]{t_4} \circ \xrightarrow{t_5} \circ \quad \cdots \quad \begin{cases} t_1+t_2\leq 2 & T_2\leq 2 \\ t_2+t_3+t_4\geq 5 & T_4-T_1\geq 5 \end{cases}$$

Lemma

Let Z be a bounded zone 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.

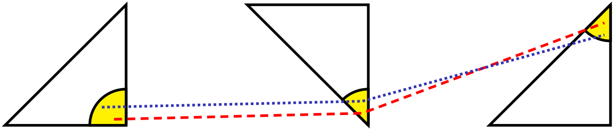
$\rightsquigarrow$ for every finite path π in $\mathcal{A}$, there exists a path Π in $\mathcal{A}_{\mathrm{cp}}$ such that

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

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

From discrete to timed behaviours

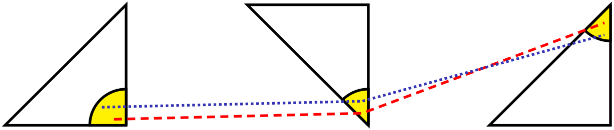
Approximation of abstract paths:



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

From discrete to timed behaviours

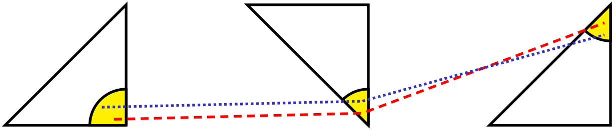
Approximation of abstract paths:



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

From discrete to timed behaviours

Approximation of abstract paths:

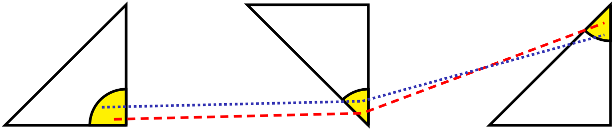


For any path Π of $\mathcal{A}_{\mathsf{cp}}$, for any $\varepsilon > 0$, there exists a path π_ε of $\mathcal{A}$ s.t.

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

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.

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

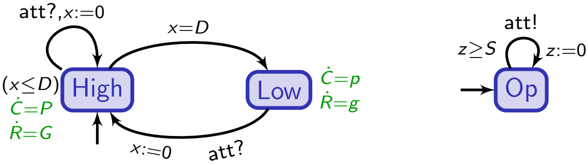
Use of the corner-point abstraction

It is a very interesting abstraction, that can be used in several other contexts:

- for mean-cost optimization [BBL04,BBL08]
- for discounted-cost optimization [FL08]
- for all concavely-priced timed automata [JT08]
- for deciding frequency objectives [BBBS11,Sta12]
- ...

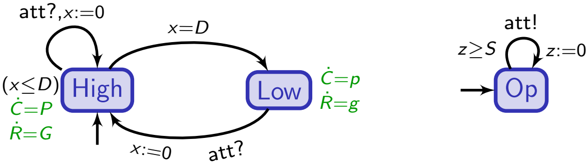
[BBL04] Bouyer, Brinksma, Larsen. Staying Alive As Cheaply As Possible *(HSCC'04)*.
[BBL08] Bouyer, Brinksma, Larsen. Optimal infinite scheduling for multi-priced timed automata *(Formal Methods in System Designs)*.
[FL08] Fahrenberg, Larsen. Discount-optimal infinite runs in priced timed automata *(INFINITY'08)*.
[JT08] Judziński, Trivedi. Concavely-priced timed automata *(FORMATS'08)*.
[BBBS11] Bertrand, Bouyer, Brihaye, Stainer. Emptiness and universality problems in timed automata with positive frequency *(ICALP'11)*.
[Sta12] Stainer. Frequencies in forgetful timed automata *(FORMATS'12)*.

Going further 1: mean-cost optimization



[BBL08] Bouyer, Brinksma, Larsen. Optimal infinite scheduling for multi-priced timed automata *(Formal Methods in System Designs)*.

Going further 1: mean-cost optimization

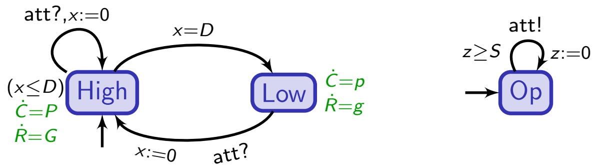


$\rightsquigarrow$ compute optimal infinite schedules that minimize

$$\text{mean-cost}(\pi) = \limsup_{n \to +\infty} \frac{\text{cost}(\pi_n)}{\text{reward}(\pi_n)}$$

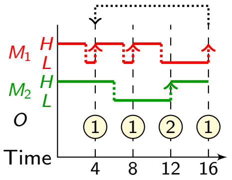
[BBL08] Bouyer, Brinksma, Larsen. Optimal infinite scheduling for multi-priced timed automata *(Formal Methods in System Designs)*.

Going further 1: mean-cost optimization

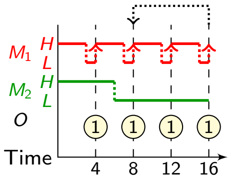


⇝ compute optimal infinite schedules that minimize

$$\text{mean-cost}(\pi) = \limsup_{n\to+\infty} \frac{\text{cost}(\pi_n)}{\text{reward}(\pi_n)}$$



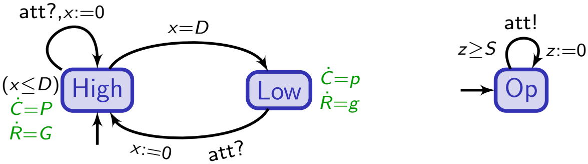
Schedule with ratio ≈1.455



Schedule with ratio ≈1.478

[BBL08] Bouyer, Brinksma, Larsen. Optimal infinite scheduling for multi-priced timed automata *(Formal Methods in System Designs)*.

Going further 1: mean-cost optimization



$\rightsquigarrow$ compute optimal infinite schedules that minimize

$$\text{mean-cost}(\pi) = \limsup_{n \to +\infty} \frac{\text{cost}(\pi_n)}{\text{reward}(\pi_n)}$$

Theorem [BBL08]

In weighted timed automata, the optimal mean-cost can be compute in PSPACE.

$\rightsquigarrow$ the corner-point abstraction can be used

[BBL08] Bouyer, Brinksma, Larsen. Optimal infinite scheduling for multi-priced timed automata *(Formal Methods in System Designs)*.

From timed to discrete behaviours

- **Finite behaviours:** based on the following property

Lemma

Let Z be a bounded zone and f be a function

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

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

- **Finite behaviours:** based on the following property

Lemma

Let Z be a bounded zone and f be a function

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

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

$\leadsto$ for every finite path π in $\mathcal{A}$, there exists a path Π in $\mathcal{A}_{\text{cp}}$ s.t.
mean-cost$(\Pi) \leq$ mean-cost(π)

From timed to discrete behaviours

- **Finite behaviours:** based on the following property

Lemma

Let Z be a bounded zone and f be a function

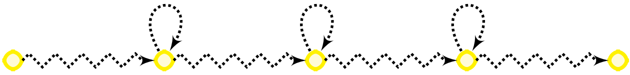
$$f : (t_1, \ldots, t_n) \mapsto \frac{\sum_{i=1}^{n} c_i t_i + c}{\sum_{i=1}^{n} r_i t_i + r}$$

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 $\mathcal{A}_{\text{cp}}$ s.t.

$$\text{mean-cost}(\Pi) \leq \text{mean-cost}(\pi)$$

- **Infinite behaviours:** decompose each sufficiently long projection into cycles:



The (acyclic) linear part will be negligible!

From timed to discrete behaviours

- **Finite behaviours:** based on the following property

Lemma

Let Z be a bounded zone and f be a function

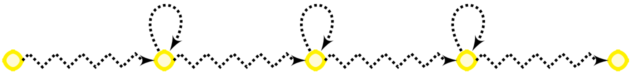
$$f : (t_1, \ldots, t_n) \mapsto \frac{\sum_{i=1}^{n} c_i t_i + c}{\sum_{i=1}^{n} r_i t_i + r}$$

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

$\leadsto$ for every finite path π in $\mathcal{A}$, there exists a path Π in $\mathcal{A}_{\mathrm{cp}}$ s.t.

$$\text{mean-cost}(\Pi) \leq \text{mean-cost}(\pi)$$

- **Infinite behaviours:** decompose each sufficiently long projection into cycles:

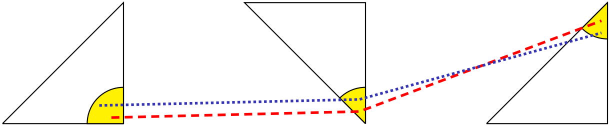


The (acyclic) linear part will be negligible!

$\leadsto$ the optimal cycle of $\mathcal{A}_{\mathrm{cp}}$ is better than any infinite path of $\mathcal{A}$!

From discrete to timed behaviours

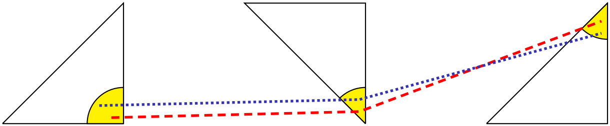
Approximation of abstract paths:



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

From discrete to timed behaviours

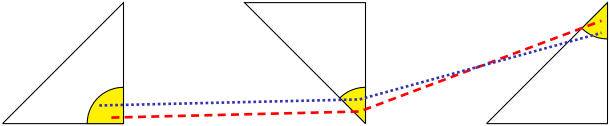
Approximation of abstract paths:



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

From discrete to timed behaviours

Approximation of abstract paths:

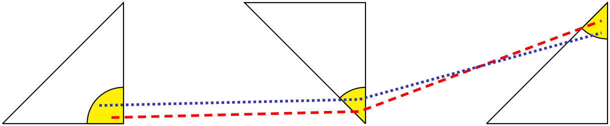


For any path Π of $\mathcal{A}_{\text{cp}}$, for any $\varepsilon > 0$, there exists a path π_ε of $\mathcal{A}$ s.t.

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

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.

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

Going further 2: concavely-priced cost functions

$\rightsquigarrow$ A general abstract framework for quantitative timed systems

Theorem [JT08]

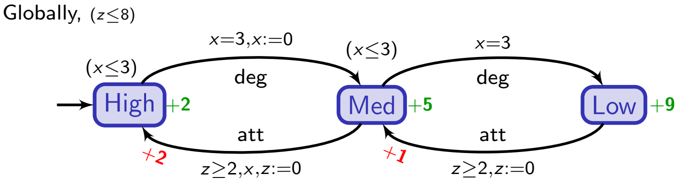
In concavely-priced timed automata, optimal cost is computable, if we restrict to quasi-concave cost functions. For the following cost functions, the (decision) problem is even PSPACE-complete:

- optimal-time and optimal-cost reachability;
- optimal discrete discounted cost;
- optimal mean-cost.

$\rightsquigarrow$ the corner-point abstraction can be used

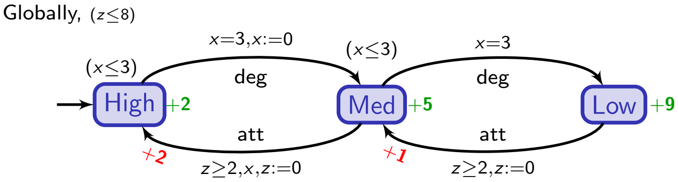
[JT08] Judziński, Trivedi. Concavely-priced timed automata *(FORMATS'08)*.

Going further 3: discounted-time cost optimization



[FL08] Fahrenberg, Larsen. Discount-optimal infinite runs in priced timed automata *(INFINITY'08)*.

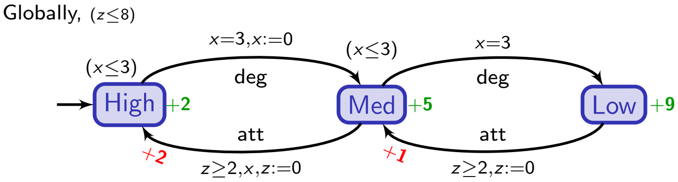
Going further 3: discounted-time cost optimization



$\rightsquigarrow$ compute optimal infinite schedules that minimize
discounted cost over time

[FL08] Fahrenberg, Larsen. Discount-optimal infinite runs in priced timed automata *(INFINITY'08)*.

Going further 3: discounted-time cost optimization



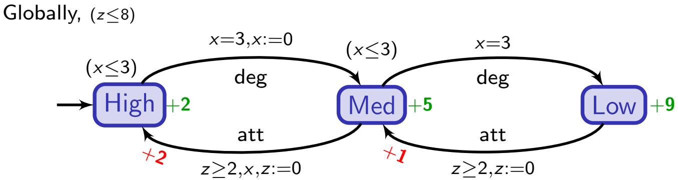
$\rightsquigarrow$ compute optimal infinite schedules that minimize

$$\text{discounted-cost}_\lambda(\pi) = \sum_{n \geq 0} \lambda^{T_n} \int_{t=0}^{\tau_{n+1}} \lambda^t \text{cost}(\ell_n)\, \mathrm{d}t + \lambda^{T_{n+1}} \text{cost}(\ell_n \xrightarrow{a_{n+1}} \ell_{n+1})$$

if $\pi = (\ell_0, v_0) \xrightarrow{\tau_1, a_1} (\ell_1, v_1) \xrightarrow{\tau_2, a_2} \cdots$ and $T_n = \sum_{i \leq n} \tau_i$

[FL08] Fahrenberg, Larsen. Discount-optimal infinite runs in priced timed automata *(INFINITY'08)*.

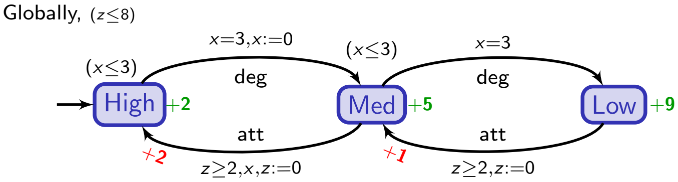
Going further 3: discounted-time cost optimization



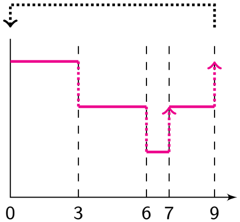
$\rightsquigarrow$ compute optimal infinite schedules that minimize discounted cost over time

[FL08] Fahrenberg, Larsen. Discount-optimal infinite runs in priced timed automata *(INFINITY'08)*.

Going further 3: discounted-time cost optimization



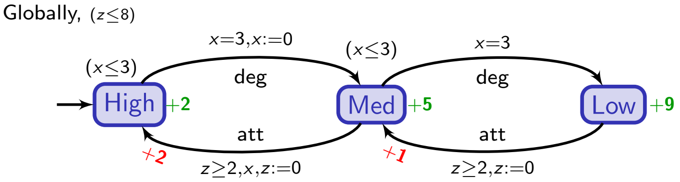
$\rightsquigarrow$ compute optimal infinite schedules that minimize
discounted cost over time



if $\lambda = e^{-1}$, the discounted cost of that infinite schedule is ≈ 2.16

[FL08] Fahrenberg, Larsen. Discount-optimal infinite runs in priced timed automata *(INFINITY'08)*.

Going further 3: discounted-time cost optimization



$\rightsquigarrow$ compute optimal infinite schedules that minimize
discounted cost over time

Theorem [FL08]

In weighted timed automata, the optimal discounted cost is computable in EXPTIME.

$\rightsquigarrow$ the corner-point abstraction can be used

[FL08] Fahrenberg, Larsen. Discount-optimal infinite runs in priced timed automata *(INFINITY'08)*.

And symbolically?

- Non-obvious in general...

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost- optimal reachability for priced timed automata *(CAV'01)*.

And symbolically?

- Non-obvious in general...
- Only for optimal reachability

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost- optimal reachability for priced timed automata *(CAV'01)*.

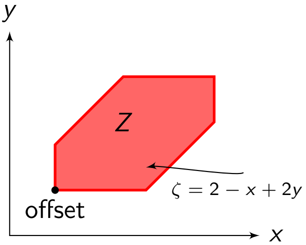
And symbolically?

- Non-obvious in general...
- Only for optimal reachability

Priced zones

priced zone = zone + affine cost function

☺ efficient representation: DBM + offset cost + affine coefficient for each clock



Represented by: zone Z
offset cost: $+4$
rate for x: -1
rate for y: $+2$

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost- optimal reachability for priced timed automata *(CAV'01)*.

Results

Theorem [LBB+01,RLS06]

The forward algorithm with standard inclusion is correct and terminates for **bounded** timed automata with non-negative costs.

Termination: well-quasi-order on priced zones

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost- optimal reachability for priced timed automata *(CAV'01)*.

[RLS06] Rasmussen, Larsen, Subramani. On using priced timed automata to achieve optimal scheduling *(Formal Methods in System Design)*.

Results

Theorem [LBB+01,RLS06]

The forward algorithm with standard inclusion is correct and terminates for **bounded** timed automata with non-negative costs.

Termination: well-quasi-order on priced zones

- Development of an (abstract) inclusion test $\sqsubseteq_M$ on priced zones

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost- optimal reachability for priced timed automata *(CAV'01)*.
[RLS06] Rasmussen, Larsen, Subramani. On using priced timed automata to achieve optimal scheduling *(Formal Methods in System Design)*.
[BCM16] Bouyer, Colange, Markey. Symbolic Optimal Reachability in Weighted Timed Automata *(CAV'16)*.

Results

Theorem [LBB+01,RLS06]

The forward algorithm with standard inclusion is correct and terminates for **bounded** timed automata with non-negative costs.

Termination: well-quasi-order on priced zones

- Development of an (abstract) inclusion test $\sqsubseteq_M$ on priced zones
- $\mathcal{Z} \sqsubseteq_M \mathcal{Z}'$ reduces to several bilevel linear optimization problems

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost- optimal reachability for priced timed automata *(CAV'01)*.
[RLS06] Rasmussen, Larsen, Subramani. On using priced timed automata to achieve optimal scheduling *(Formal Methods in System Design)*.
[BCM16] Bouyer, Colange, Markey. Symbolic Optimal Reachability in Weighted Timed Automata *(CAV'16)*.

Results

Theorem [LBB+01,RLS06]

The forward algorithm with standard inclusion is correct and terminates for **bounded** timed automata with non-negative costs.

Termination: well-quasi-order on priced zones

- Development of an (abstract) inclusion test $\sqsubseteq_M$ on priced zones
- $\mathcal{Z} \sqsubseteq_M \mathcal{Z}'$ reduces to several bilevel linear optimization problems

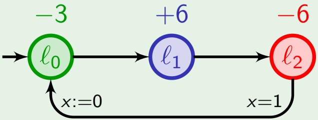
Theorem [BCM16]

The forward algorithm with inclusion test $\sqsubseteq_M$ is correct and terminates for timed automata with some conditions on the cost.
It is always better than standard inclusion for bounded timed automata.

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost- optimal reachability for priced timed automata *(CAV'01)*.
[RLS06] Rasmussen, Larsen, Subramani. On using priced timed automata to achieve optimal scheduling *(Formal Methods in System Design)*.
[BCM16] Bouyer, Colange, Markey. Symbolic Optimal Reachability in Weighted Timed Automata *(CAV'16)*.

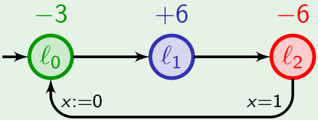
Further problems: Energy management

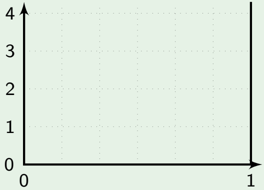
Example



Further problems: Energy management

Example



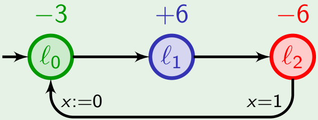


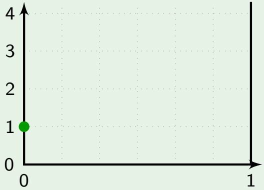
"energy is ≥ 0"

- Lower-bound problem (**L**)

Further problems: Energy management

Example



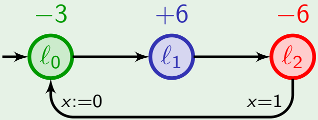


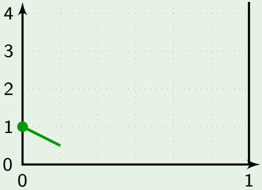
"energy is ≥ 0"

- Lower-bound problem (**L**)

Further problems: Energy management

Example



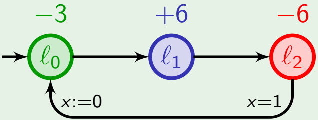


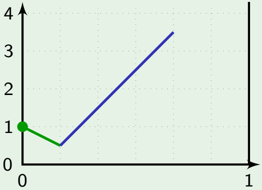
"energy is ≥ 0"

- Lower-bound problem (**L**)

Further problems: Energy management

Example



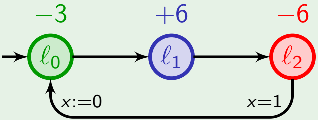


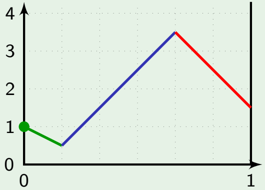
"energy is ≥ 0"

- Lower-bound problem (**L**)

Further problems: Energy management

Example



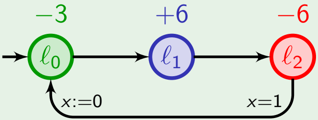


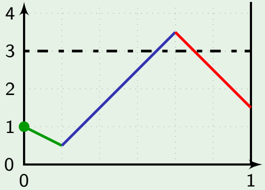
"energy is ≥ 0"

- Lower-bound problem (**L**)

Further problems: Energy management

Example



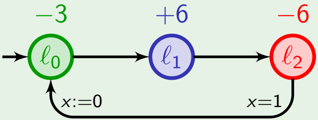


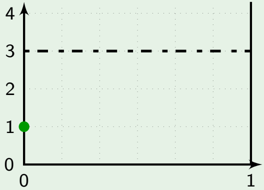
"energy is in [0,3]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



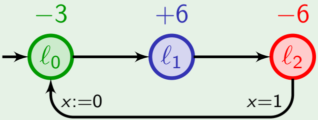


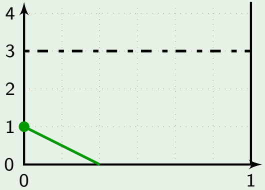
"energy is in [0,3]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



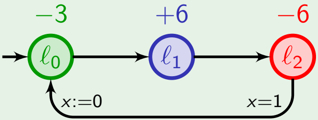


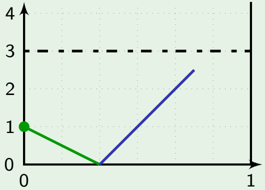
"energy is in [0,3]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



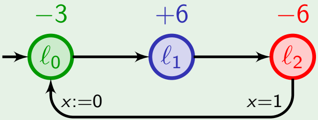


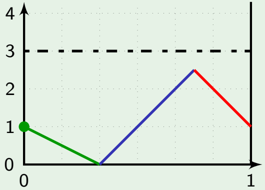
"energy is in [0,3]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



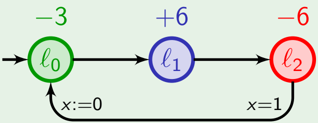


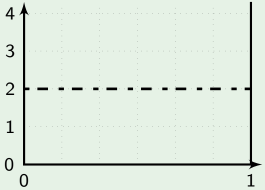
"energy is in [0,3]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



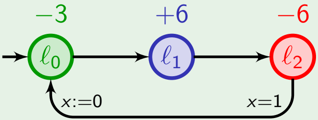


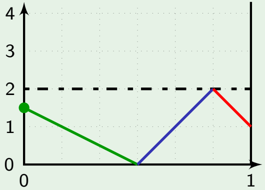
"energy is in [0,2]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



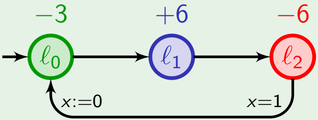


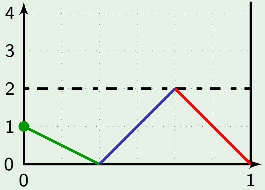
"energy is in [0,2]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



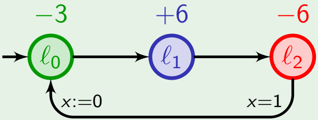


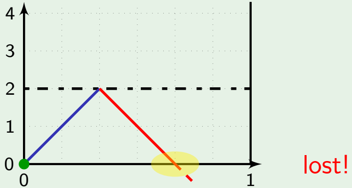
"energy is in [0,2]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



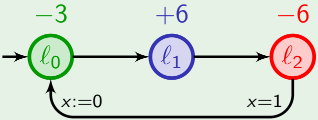


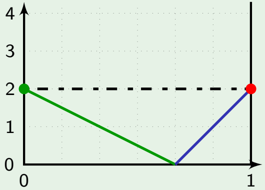
"energy is in [0,2]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



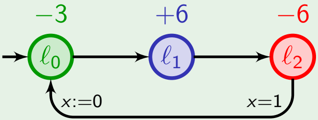


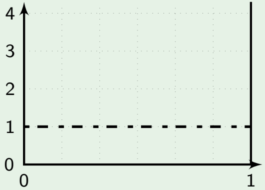
"energy is in [0,2]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



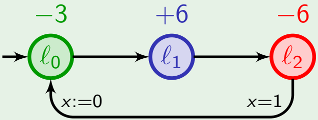


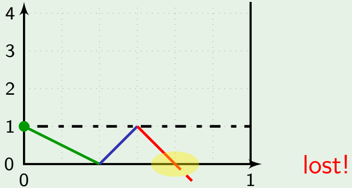
"energy is in [0,1]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example



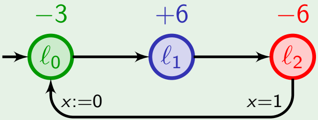


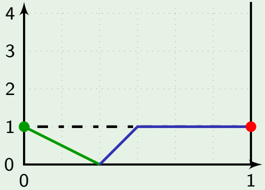
"energy is in [0,1]"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)

Further problems: Energy management

Example





"energy is in [0,1] with a weak upper bound"

- Lower-bound problem (**L**)
- Lower-and-upper-bound problem (**L+U**)
- Lower-and-weak-upper-bound problem (**L+W**)

The **L**-problem: use the corner-point abstraction?

Idea: delay in the most profitable location

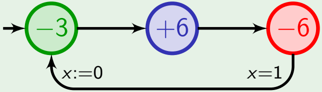
$\rightsquigarrow$ the corner-point abstraction

The **L**-problem: use the corner-point abstraction?

Idea: delay in the most profitable location

$\rightsquigarrow$ the corner-point abstraction

Example

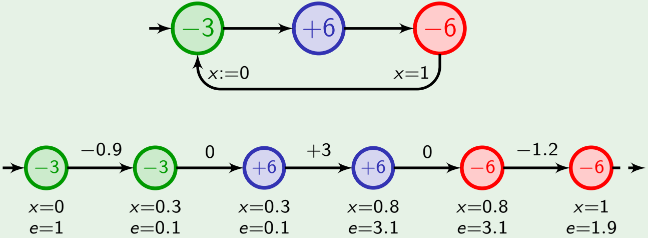


The **L**-problem: use the corner-point abstraction?

Idea: delay in the most profitable location

$\rightsquigarrow$ the corner-point abstraction

Example

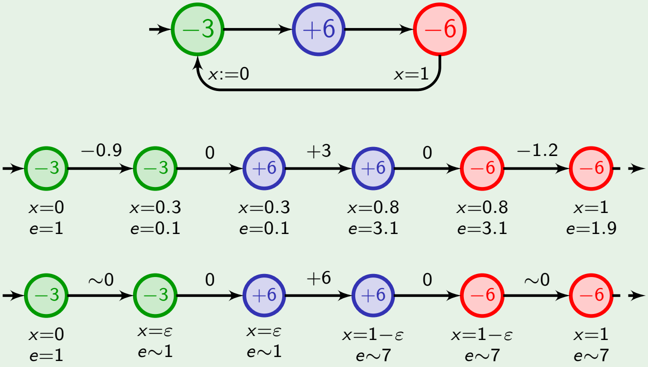


The **L**-problem: use the corner-point abstraction?

Idea: delay in the most profitable location

$\rightsquigarrow$ the corner-point abstraction

Example

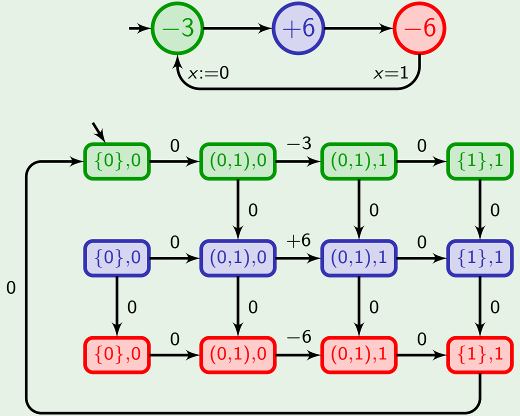


The **L**-problem: use the corner-point abstraction?

Idea: delay in the most profitable location

$\rightsquigarrow$ the corner-point abstraction

Example



The **L**-problem: use the corner-point abstraction?

Idea: delay in the most profitable location

$\rightsquigarrow$ the corner-point abstraction

Theorem [BFLMS08]

The corner-point abstraction is sound and complete for single-clock WTA with no discrete costs. Hence the existential **L**-problem is in PTIME in that case.

[BFLMS08] Bouyer, Fahrenberg, Larsen, Markey, Srba. Infinite runs in weighted timed automata with energy constraints *(FORMATS'08)*.

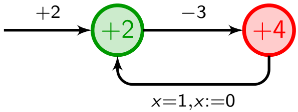
The **L**-problem: use the corner-point abstraction?

Idea: delay in the most profitable location

$\leadsto$ the corner-point abstraction

Remark

The corner-point abstraction is not correct with discrete costs.



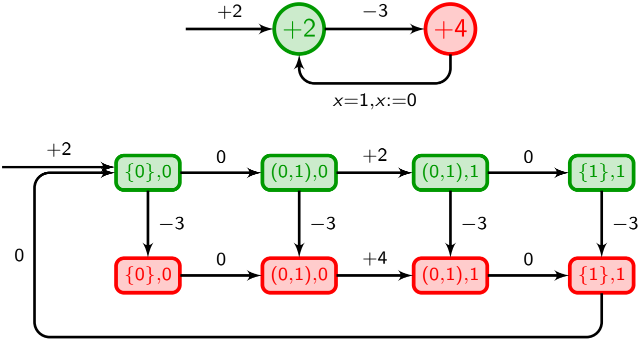
The **L**-problem: use the corner-point abstraction?

Idea: delay in the most profitable location

$\rightsquigarrow$ the corner-point abstraction

Remark

The corner-point abstraction is not correct with discrete costs.



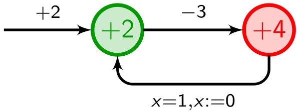
The **L**-problem: use the corner-point abstraction?

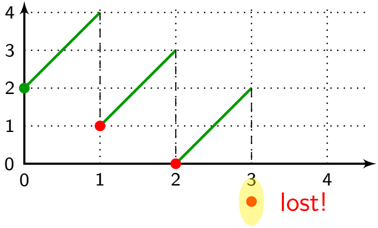
Idea: delay in the most profitable location

$\rightsquigarrow$ the corner-point abstraction

Remark

The corner-point abstraction is not correct with discrete costs.





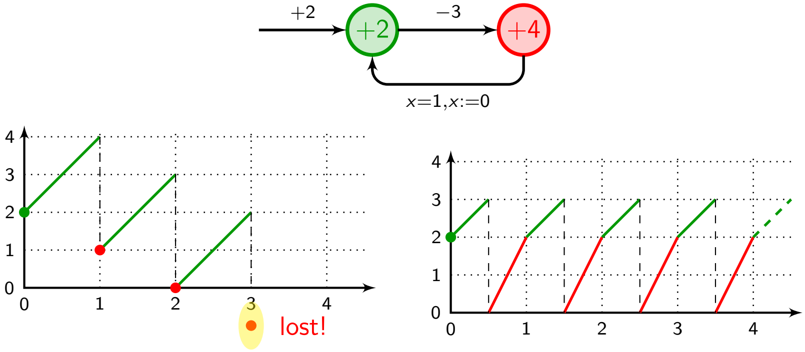
The **L**-problem: use the corner-point abstraction?

Idea: delay in the most profitable location

$\rightsquigarrow$ the corner-point abstraction

Remark

The corner-point abstraction is not correct with discrete costs.



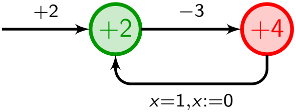
The **L**-problem: use the corner-point abstraction?

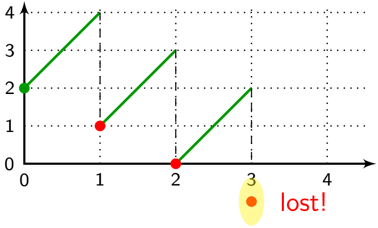
Idea: delay in the most profitable location

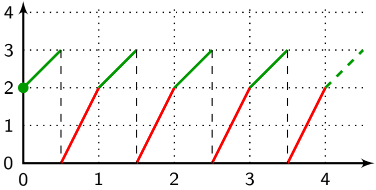
$\rightsquigarrow$ the corner-point abstraction

Remark

The corner-point abstraction is not correct with discrete costs.



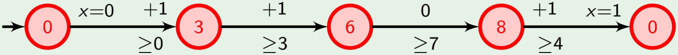




$\rightsquigarrow$ requires new developments!

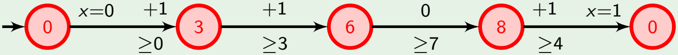
The **L**-problem: computing optimal delays

Example



The **L**-problem: computing optimal delays

Example

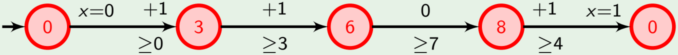


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};

The **L**-problem: computing optimal delays

Example

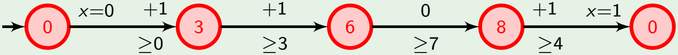


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—				—

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};

The **L**-problem: computing optimal delays

Example

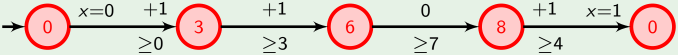


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—			0	—

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};

The **L**-problem: computing optimal delays

Example

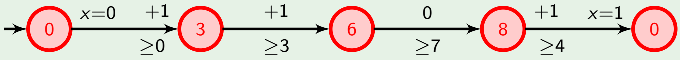


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—		$\frac{1}{2}$	0	—

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};

The **L**-problem: computing optimal delays

Example

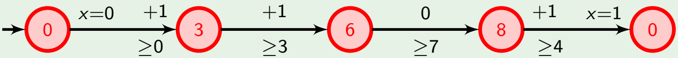


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};

The **L**-problem: computing optimal delays

Example



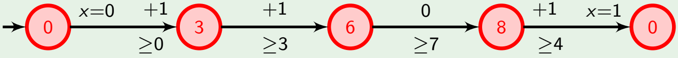
t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—

minimal initial credit required: $\frac{1}{2}$, yields final credit 8.

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};

The **L**-problem: computing optimal delays

Example

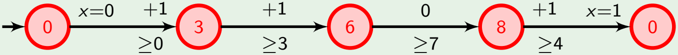


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};
- compute other points on the energy function curve.

The **L**-problem: computing optimal delays

Example



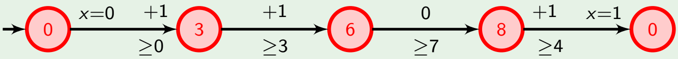
t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—

initial credit
$\frac{1}{2} + \delta$

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};
- compute other points on the energy function curve.

The **L**-problem: computing optimal delays

Example



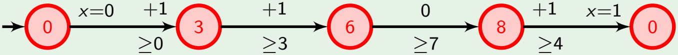
t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—

initial credit $\frac{1}{2} + \delta$ $\quad \frac{1}{2} - \frac{\delta}{3}$

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};
- compute other points on the energy function curve.

The **L**-problem: computing optimal delays

Example

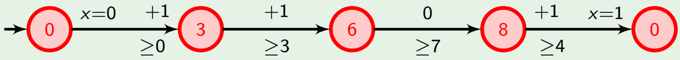


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—
initial credit	$\frac{1}{2} + \delta$	$\frac{1}{2} - \frac{\delta}{3}$	$\frac{1}{2}$		

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};
- compute other points on the energy function curve.

The L-problem: computing optimal delays

Example

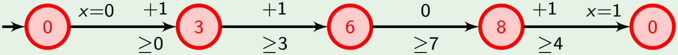


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—
initial credit $\frac{1}{2}+\delta$		$\frac{1}{2}-\frac{\delta}{3}$	$\frac{1}{2}$	$\frac{\delta}{3}$	

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};
- compute other points on the energy function curve.

The **L**-problem: computing optimal delays

Example

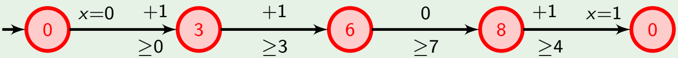


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—
	initial credit $\frac{1}{2}+\delta$	$\frac{1}{2}-\frac{\delta}{3}$	$\frac{1}{2}$	$\frac{\delta}{3}$	final credit $8+\frac{8}{3}\delta$

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};
- compute other points on the energy function curve.

The **L**-problem: computing optimal delays

Example

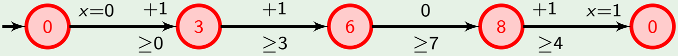


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—
	initial credit 2	0	$\frac{1}{2}$	$\frac{1}{2}$	final credit 12

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};
- compute other points on the energy function curve.

The **L**-problem: computing optimal delays

Example



t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—

initial credit $2+\delta$

0

$\frac{1}{2}-\frac{\delta}{6}$

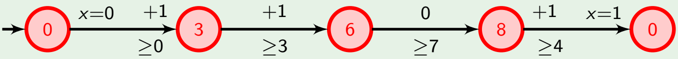
$\frac{1}{2}+\frac{\delta}{6}$

final credit $12+\frac{8}{6}\delta$

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};
- compute other points on the energy function curve.

The **L**-problem: computing optimal delays

Example

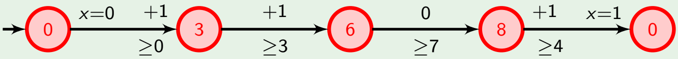


t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—
	initial credit 5	0	0	1	final credit 16

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};
- compute other points on the energy function curve.

The **L**-problem: computing optimal delays

Example

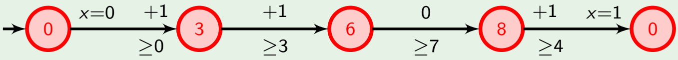


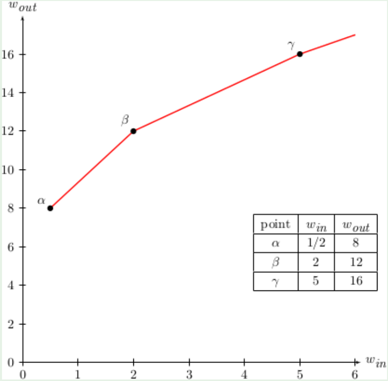
t_{opt}:	—	$\frac{2}{3}$	$\frac{1}{2}$	—	—
t^*:	—	$\frac{1}{2}$	$\frac{1}{2}$	0	—
initial credit $5+\delta$		0	0	1	final credit $16+\delta$

- compute optimal delays t_{opt} in ℓ_1 to ℓ_{n-1};
- compute optimal possible delays t^* in ℓ_1 to ℓ_{n-1};
- compute other points on the energy function curve.

The **L**-problem: computing optimal delays

Example





The **L**-problem: concluding

Theorem

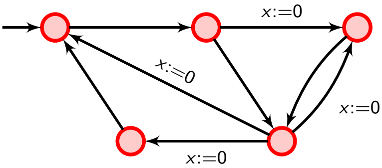
Optimization, reachability and existence of infinite runs satisfying the constraint ≥ 0 can be decided in EXPTIME in single-clock WTA

The **L**-problem: concluding

Theorem

Optimization, reachability and existence of infinite runs satisfying the constraint ≥ 0 can be decided in EXPTIME in single-clock WTA

- transform the automaton into an automaton with energy functions;

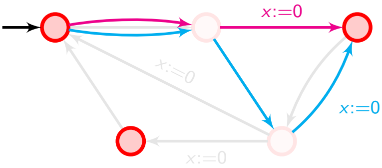


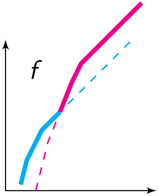
The **L**-problem: concluding

Theorem

Optimization, reachability and existence of infinite runs satisfying the constraint ≥ 0 can be decided in EXPTIME in single-clock WTA

- transform the automaton into an automaton with energy functions;



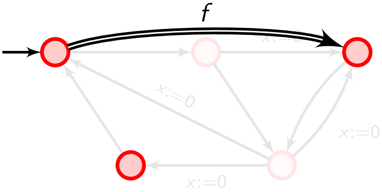


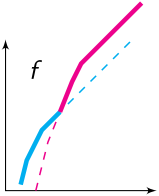
The **L**-problem: concluding

Theorem

Optimization, reachability and existence of infinite runs satisfying the constraint ≥ 0 can be decided in EXPTIME in single-clock WTA

- transform the automaton into an automaton with energy functions;



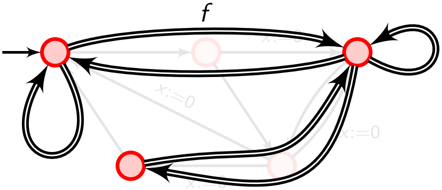


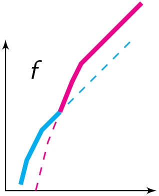
The **L**-problem: concluding

Theorem

Optimization, reachability and existence of infinite runs satisfying the constraint ≥ 0 can be decided in EXPTIME in single-clock WTA

- transform the automaton into an automaton with energy functions;



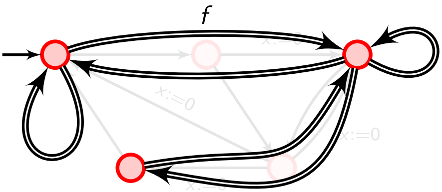


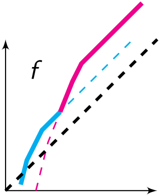
The **L**-problem: concluding

Theorem

Optimization, reachability and existence of infinite runs satisfying the constraint ≥ 0 can be decided in EXPTIME in single-clock WTA

- transform the automaton into an automaton with energy functions;





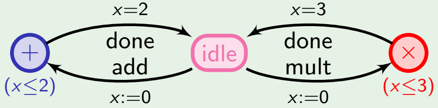
- check if simple cycles can be iterated (or if a Zeno cycle can be reached...)

Outline

Why (timed) games?

- to model uncertainty

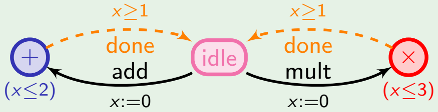
Example of a processor in the taskgraph example



Why (timed) games?

- to model uncertainty

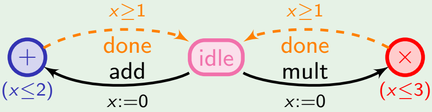
Example of a processor in the taskgraph example



Why (timed) games?

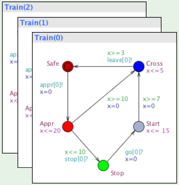
- to model uncertainty

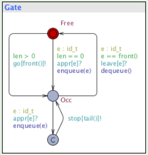
Example of a processor in the taskgraph example



- to model an interaction with the environment

Example of the gate in the train/gate example

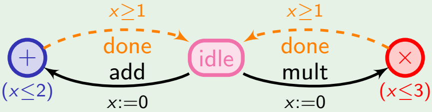




Why (timed) games?

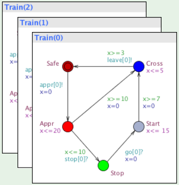
- to model uncertainty

Example of a processor in the taskgraph example



- to model an interaction with the environment

Example of the gate in the train/gate example

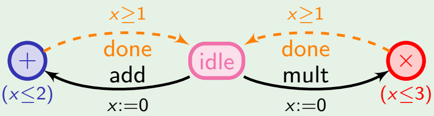


?

Why (timed) games?

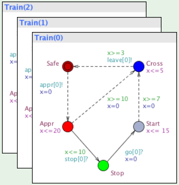
- to model uncertainty

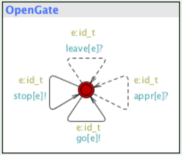
Example of a processor in the taskgraph example



- to model an interaction with the environment

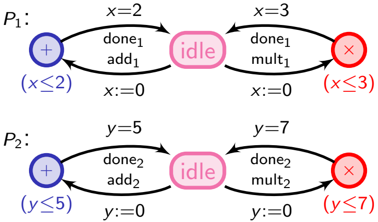
Example of the gate in the train/gate example



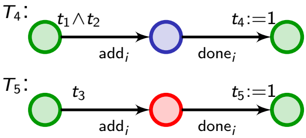


Modelling the task graph scheduling problem

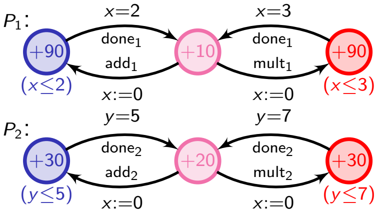
- Processors



- Tasks

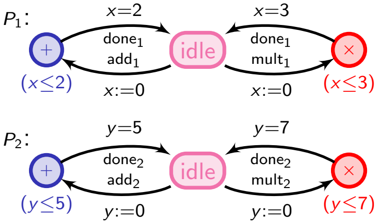


- Modelling energy

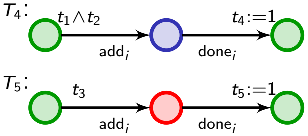


Modelling the task graph scheduling problem

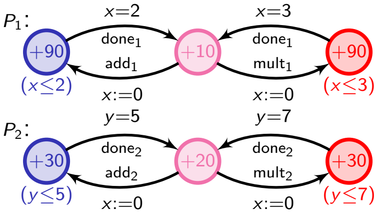
- Processors



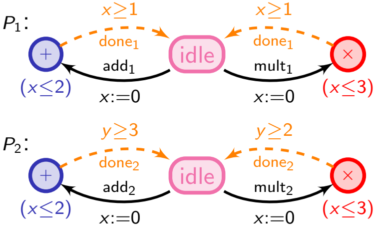
- Tasks



- Modelling energy



- Modelling uncertainty



Modelling the task graph scheduling problem

- Processors

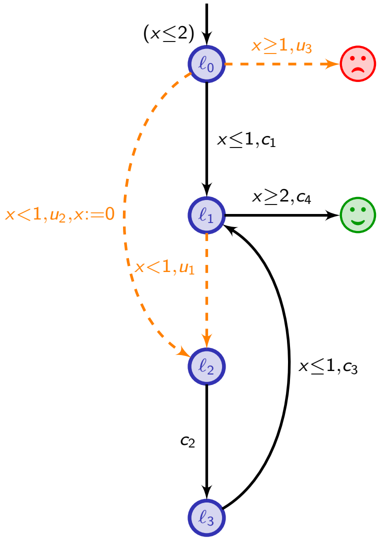
- Tasks

A (good) schedule is a strategy in the product game (with a low cost)

- Modelling energy

- Modelling uncertainty

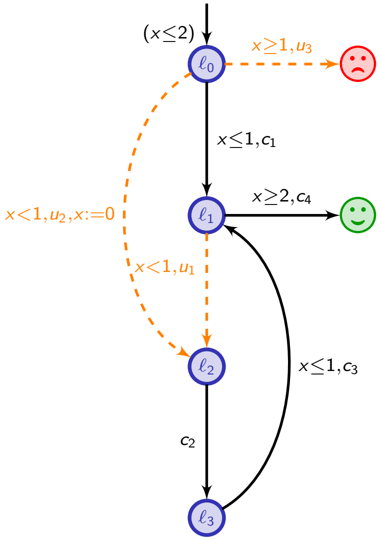
An example of a timed game



Rule of the game

- Aim: avoid ☹ and reach ☺

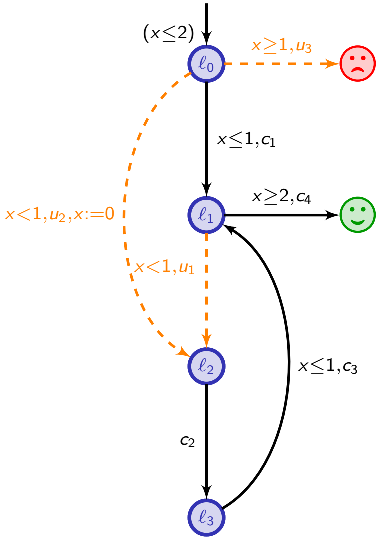
An example of a timed game



Rule of the game

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

An example of a timed game

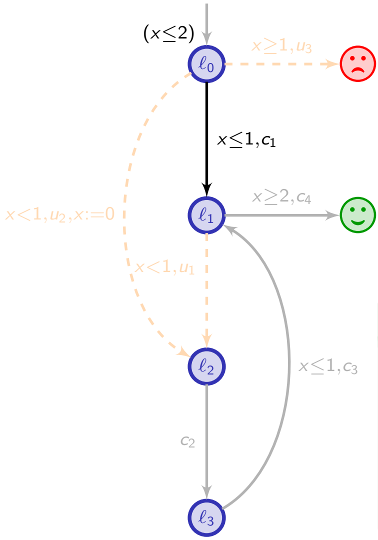


Rule of the game

- 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

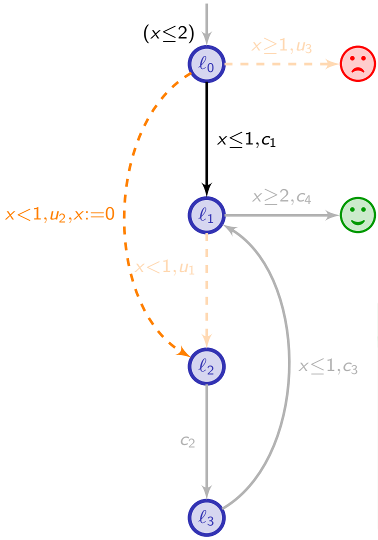
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

A (memoryless) winning strategy

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

An example of a timed game



Rule of the game

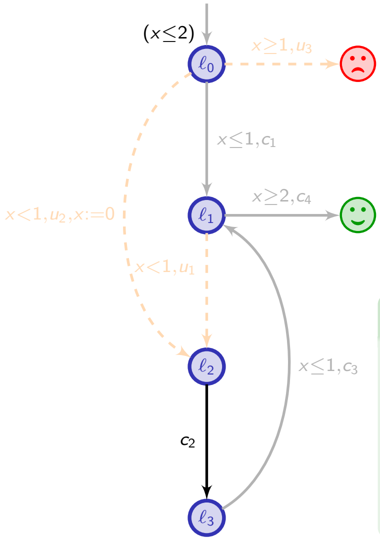
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

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

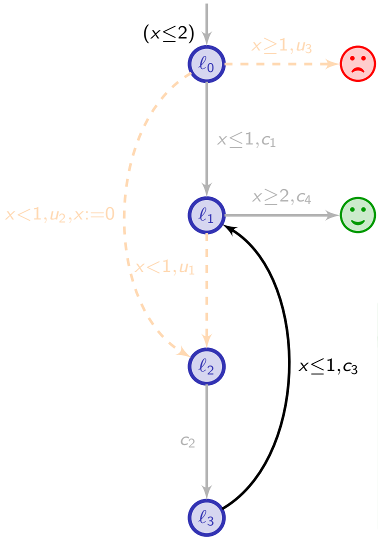
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

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

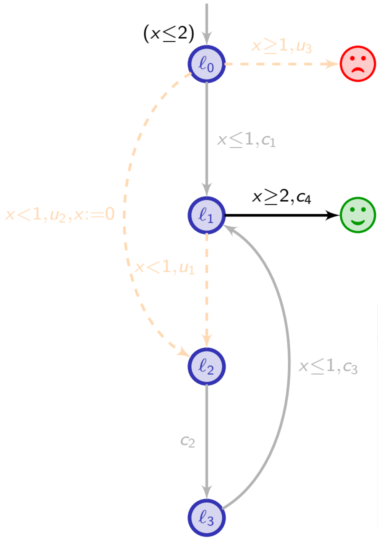
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

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

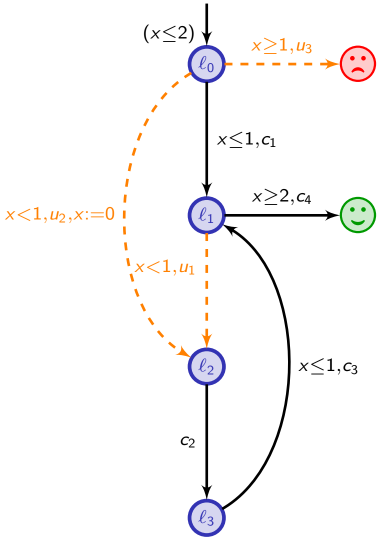
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

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



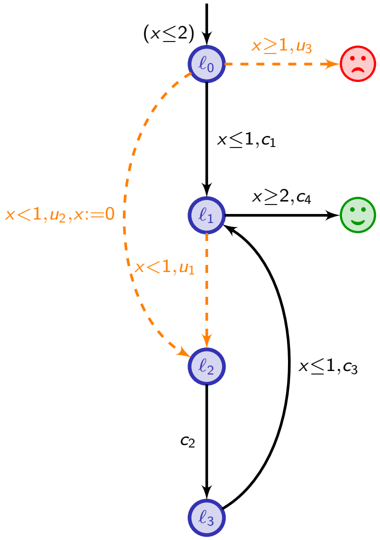
Rule of the game

- 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

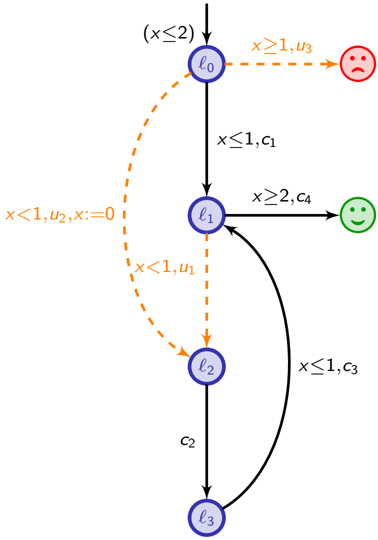
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

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
a region-closed attractor can be computed

[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.

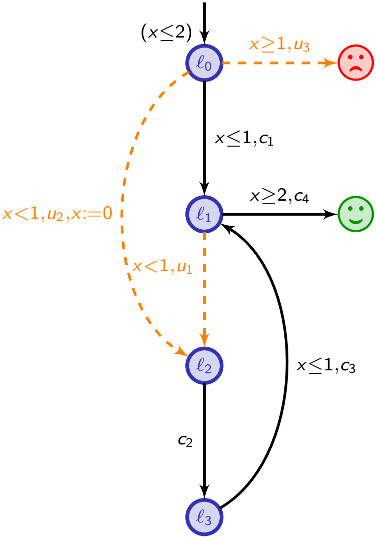
$\rightsquigarrow$ classical regions are sufficient for solving such problems
a region-closed attractor can be computed

Theorem [AM99,BHPR07,JT07]

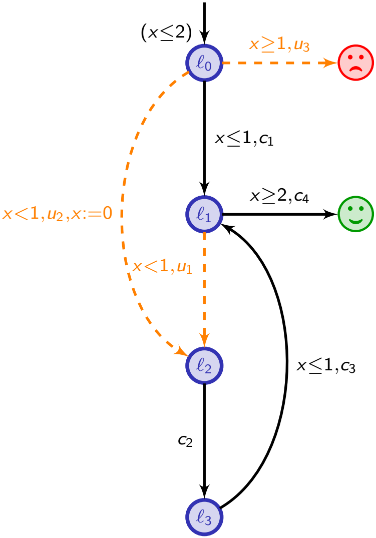
Optimal-time reachability timed games are decidable and EXPTIME-complete.

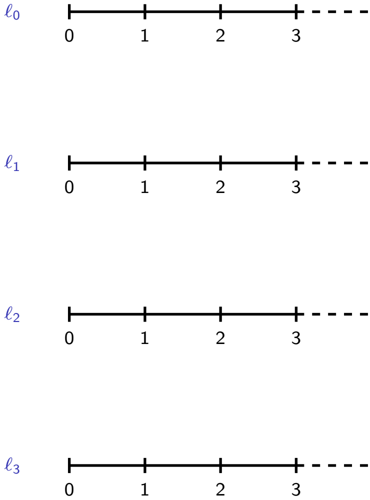
[AM99] Asarin, Maler. As soon as possible: time optimal control for timed automata *(HSCC'99)*.
[BHPR07] Brihaye, Henzinger, Prabhu, Raskin. Minimum-time reachability in timed games *(ICALP'07)*.
[JT07] Jurdziński, Trivedi. Reachability-time games on timed automata *(ICALP'07)*.

Back to the example: computing winning states

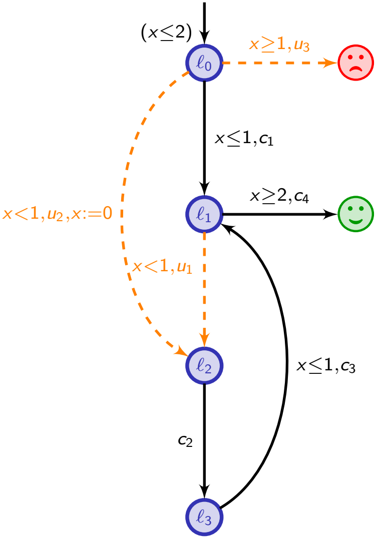


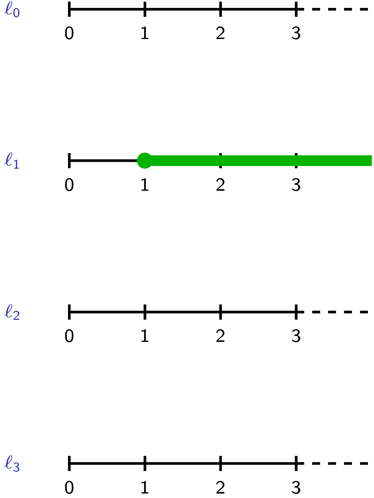
Back to the example: computing winning states



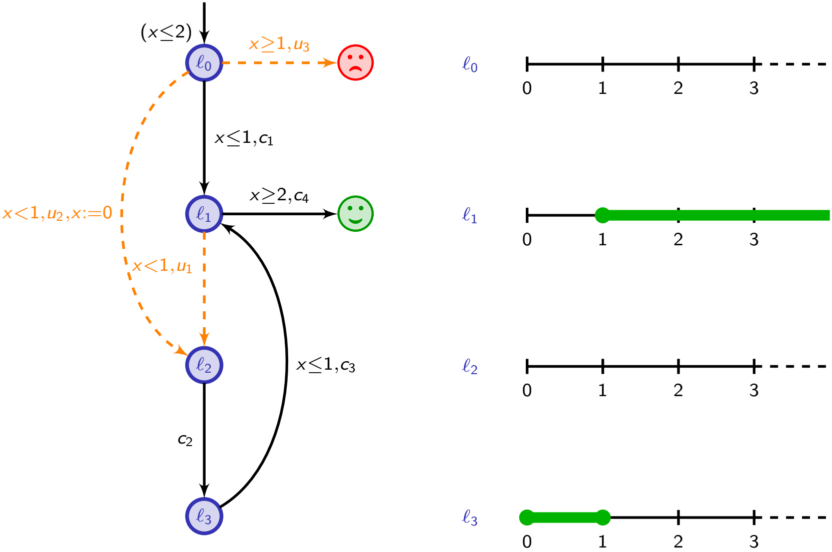


Back to the example: computing winning states

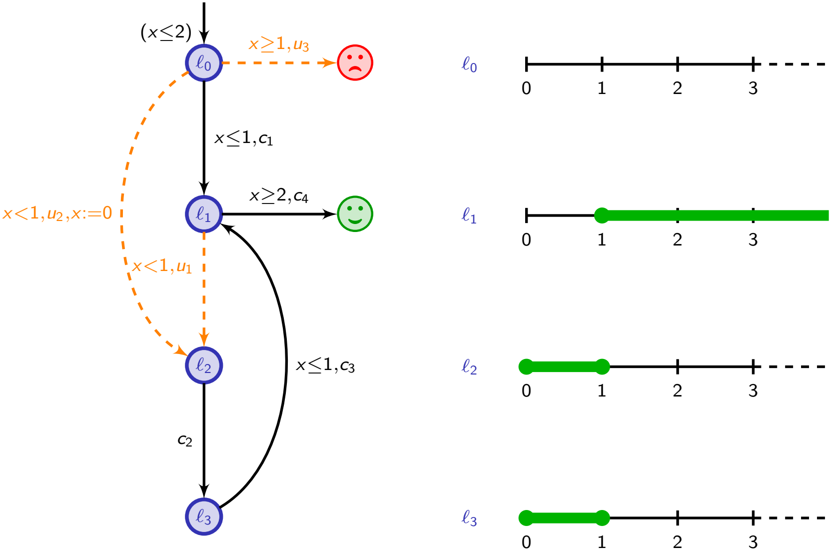




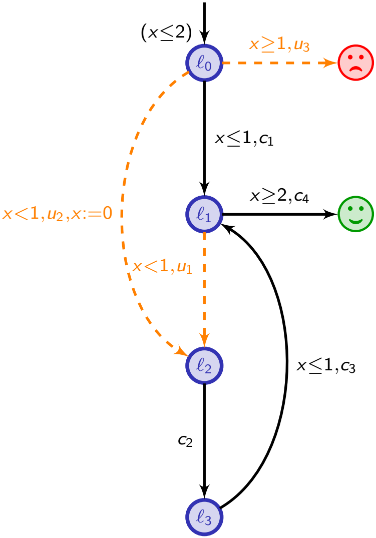
Back to the example: computing winning states

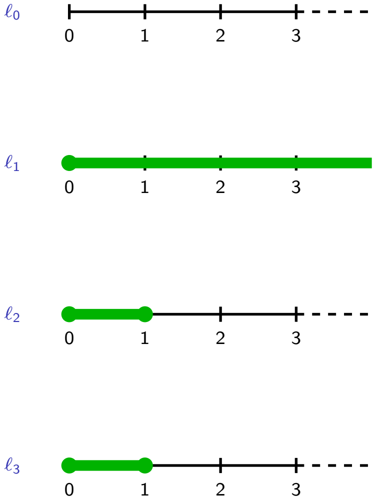


Back to the example: computing winning states

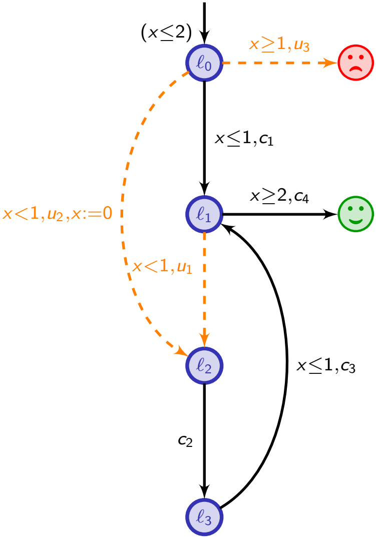


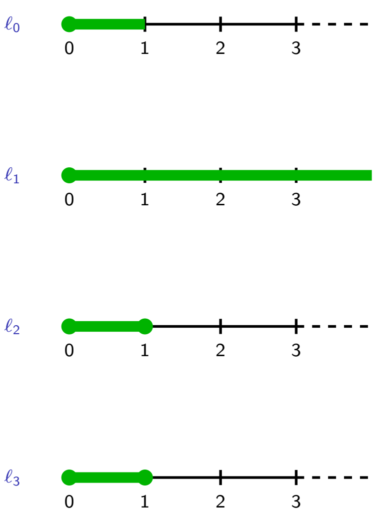
Back to the example: computing winning states



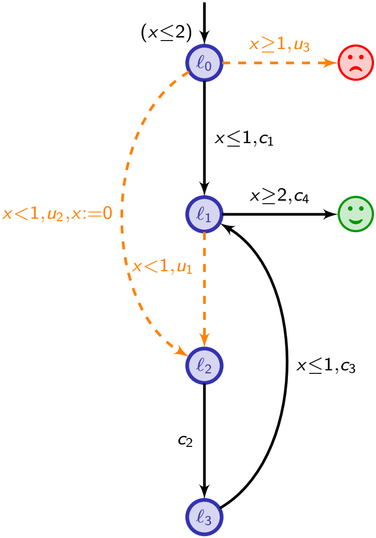


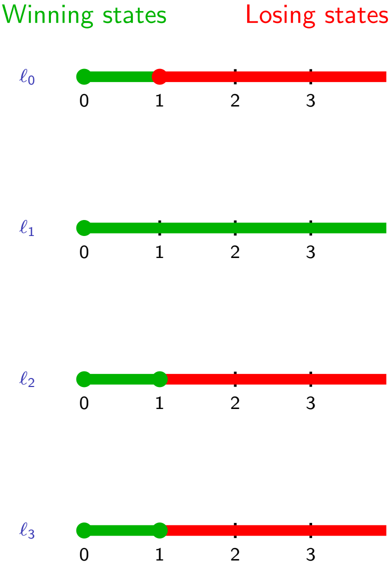
Back to the example: computing winning states





Back to the example: computing winning states





Decidability *via* attractors

Skip attractors

Decidability *via* attractors

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

Decidability *via* attractors

- $\mathsf{Pred}^a(X) = \{\bullet \mid \bullet \xrightarrow{a} \bullet \in X\}$
- controllable and uncontrollable discrete predecessors:

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

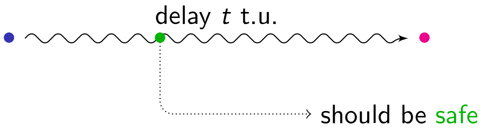
Decidability *via* attractors

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

- 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:

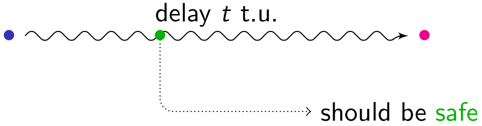


Decidability *via* attractors

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



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

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:

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

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

$$\mathsf{Attr}_1(☺) = \pi(☺)$$

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:

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

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

$$\mathrm{Attr}_1(☺) = \pi(☺)$$

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

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

- ...

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 $\smiley$ in no more than 1 step is:

$$\mathsf{Attr}_1(\smiley) = \pi(\smiley)$$

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

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

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

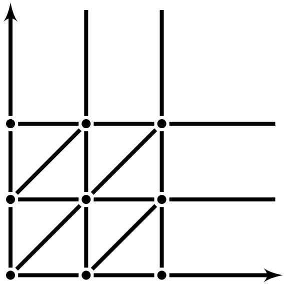
$$\begin{array}{rcl} \mathsf{Attr}_n(\smiley) & = & \pi(\mathsf{Attr}_{n-1}(\smiley)) \\ & = & \pi^n(\smiley) \end{array}$$

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 $\mathrm{uPred}(X)$.

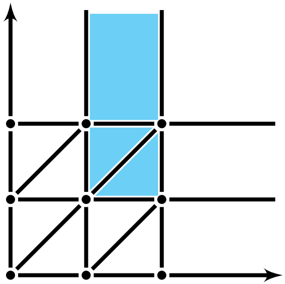
Stability w.r.t. regions

- 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

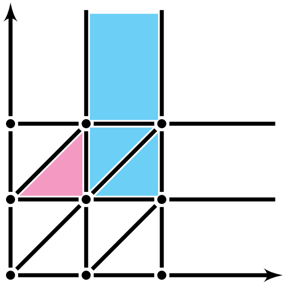
- if X is a union of regions, then:
  - $\mathrm{Pred}_a(X)$ is a union of regions,
  - and so are $\mathrm{cPred}(X)$ and $\mathrm{uPred}(X)$.
- Does π also preserve unions of regions?



cPred(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?

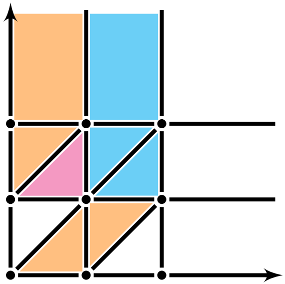


$\text{cPred}(X)$

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

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?



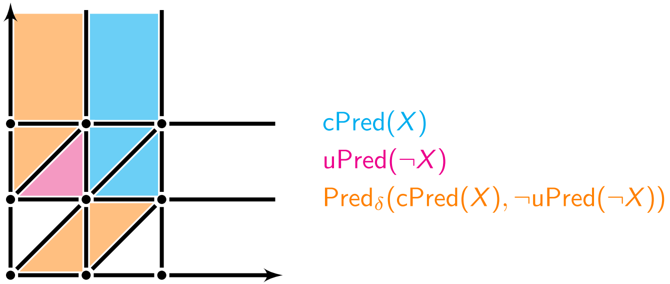
$\mathrm{cPred}(X)$

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

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

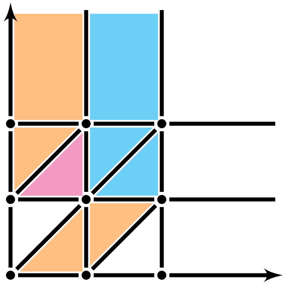
Stability w.r.t. regions

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



$\text{cPred}(X)$

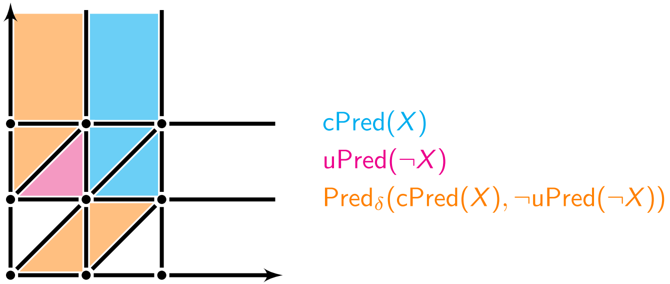
$\text{uPred}(\neg X)$

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

(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!

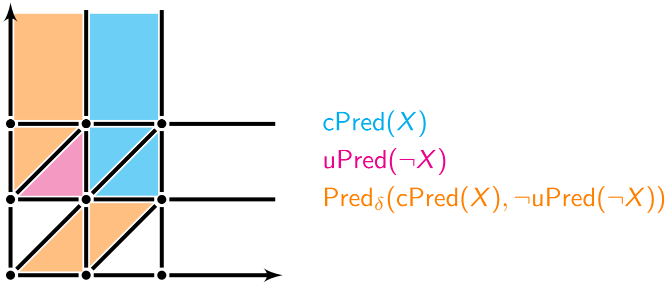


(but it generates non-convex unions of regions...)

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

Stability w.r.t. regions

- if X is a union of regions, then:
  - $\text{Pred}_a(X)$ 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!
... and is correct

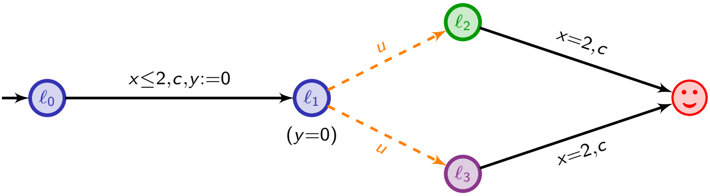
And in practice?

- A zone-based forward algorithm with backtracking [CDF+05,BCD+07]

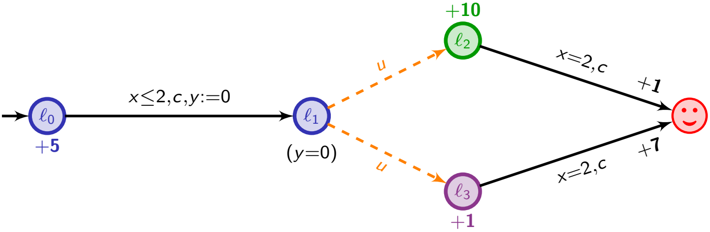
[CDF+05] Cassez, David, Fleury, Larsen, Lime. Efficient On-the-Fly Algorithms for the Analysis of Timed Games *(CONCUR'05)*.
[BCD+07] Behrmann, Cougnard, David, Fleury, Larsen, Lime. UPPAAL-Tiga: Time for Playing Games! *(CAV'07)*.

Outline

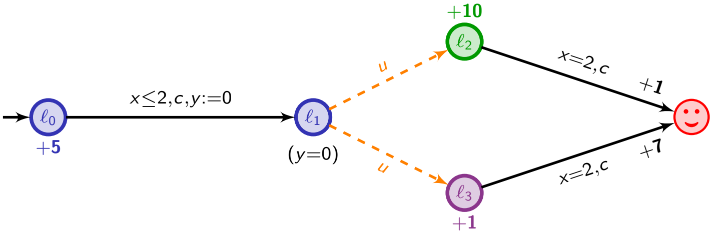
A simple timed game



A simple weighted timed game

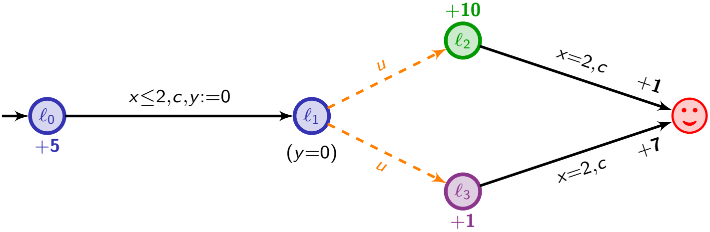


A simple weighted timed game



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

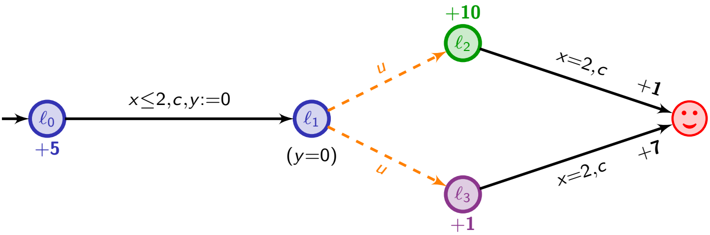
A simple weighted timed game



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

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

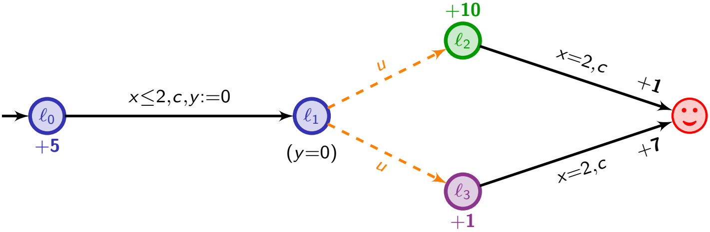
A simple weighted timed game



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

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

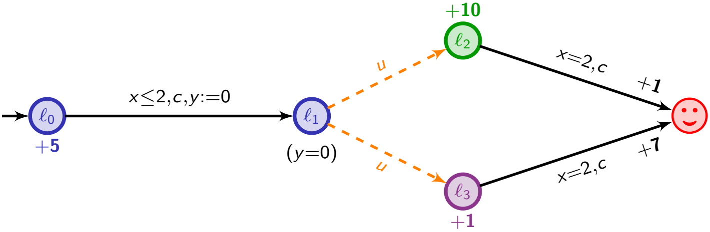
A simple weighted timed game



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

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

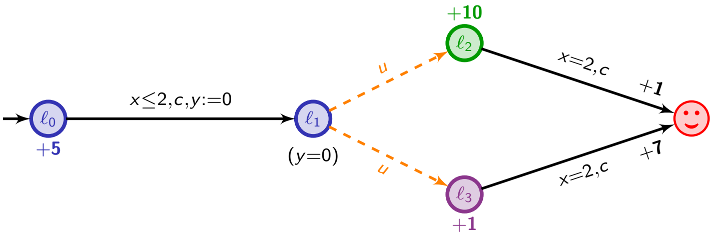
A simple weighted timed game



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



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

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

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

Optimal reachability in weighted timed games (1)

This topic has been fairly hot these last fifteen 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 (1)

This topic has been fairly hot these last fifteen years...

[LMM02,ABM04,BCFL04,BBR05,BBM06,BLMR06,Rut11,HIM13,BGK+14]

[LMM02]

Tree-like weighted timed games can be solved in 2EXPTIME.

Optimal reachability in weighted timed games (1)

This topic has been fairly hot these last fifteen years...

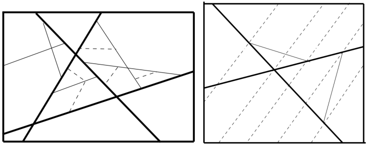
[LMM02,ABM04,BCFL04,BBR05,BBM06,BLMR06,Rut11,HIM13,BGK+14]

[LMM02]

Tree-like weighted timed games can be solved in 2EXPTIME.

[ABM04,BCFL04]

Depth-k weighted timed games can be solved in EXPTIME. There is a symbolic algorithm to solve weighted timed games **with a strongly non-Zeno cost**.



Optimal reachability in weighted timed games (2)

[BBR05,BBM06,BJM15]

In weighted timed games, the optimal cost (and the value) cannot be computed, as soon as games have three clocks or more.

Optimal reachability in weighted timed games (2)

[BBR05,BBM06,BJM15]

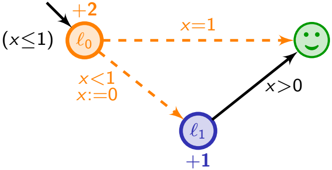
In weighted timed games, the optimal cost (and the value) cannot be computed, as soon as games have three clocks or more.

[BLMR06,Rut11,HIM13,BGK+14]

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

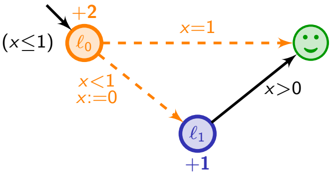
What is easier with a single clock?

- Memoryless strategies can be non-optimal...



What is easier with a single clock?

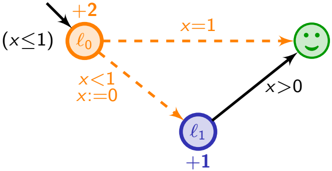
- Memoryless strategies can be non-optimal...



... but memoryless almost-optimal strategies will be sufficient.

What is easier with a single clock?

- Memoryless strategies can be non-optimal...

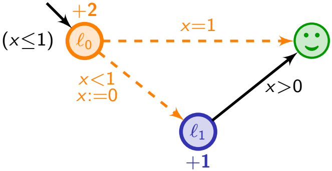


... but memoryless almost-optimal strategies will be sufficient.

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

What is easier with a single clock?

- Memoryless strategies can be non-optimal...

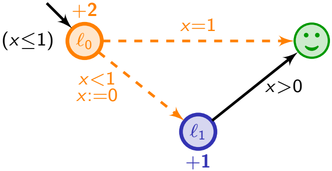


... but memoryless almost-optimal strategies will be sufficient.

- Key: resetting the clock somehow resets the history...
- By unfolding and removing one by one the locations,we can synthesize memoryless almost-optimal winning strategies.

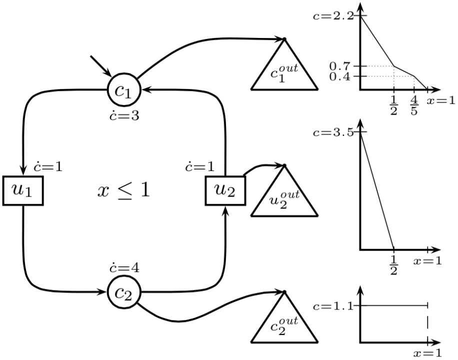
What is easier with a single clock?

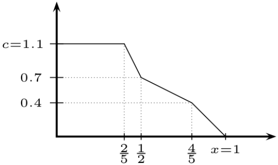
- Memoryless strategies can be non-optimal...



... but memoryless almost-optimal strategies will be sufficient.

- Key: resetting the clock somehow resets the history...
- By unfolding and removing one by one the locations,we can synthesize memoryless almost-optimal winning strategies.
- Rather involved proofs of correctness





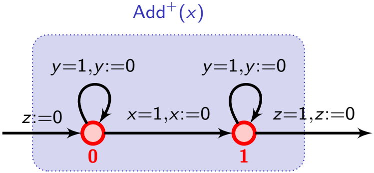
$$\sigma(c_2, x) = \begin{cases} c_2^{out} & \text{if } 0 \leq x < 2/5 \\ c_2 & \text{if } 2/5 \leq x < 1/2 \\ u_2 & \text{if } 1/2 \leq x \leq 1 \end{cases}$$

Computing the optimal cost: why is that hard?

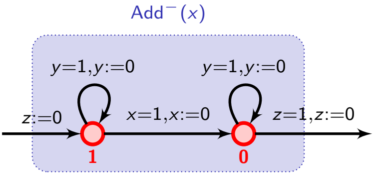
Given two clocks x and y, we can check whether $y = 2x$.

Computing the optimal cost: why is that hard?

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



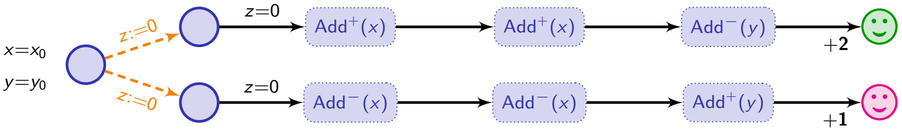
The cost is increased by x_0



The cost is increased by $1-x_0$

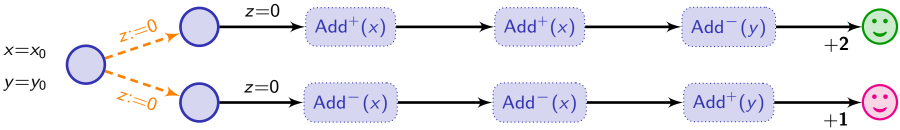
Computing the optimal cost: why is that hard?

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



Computing the optimal cost: why is that hard?

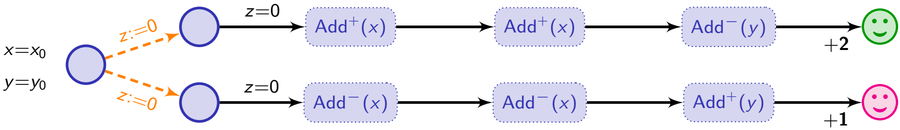
Given two clocks x and y, we can check whether $y = 2x$.



- In 🙂, cost $= 2x_0 + (1 - y_0) + 2$

Computing the optimal cost: why is that hard?

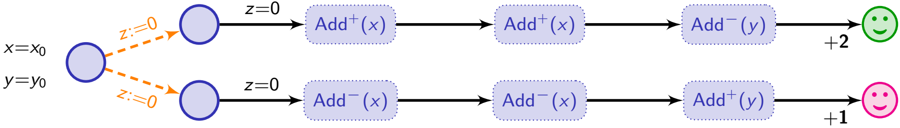
Given two clocks x and y, we can check whether $y = 2x$.



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

Computing the optimal cost: why is that hard?

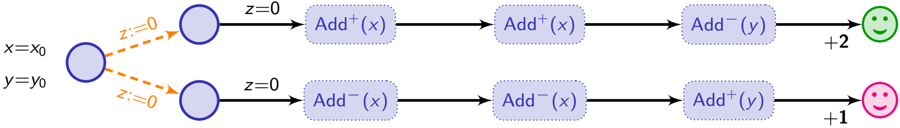
Given two clocks x and y, we can check whether $y = 2x$.



- 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

Computing the optimal cost: why is that hard?

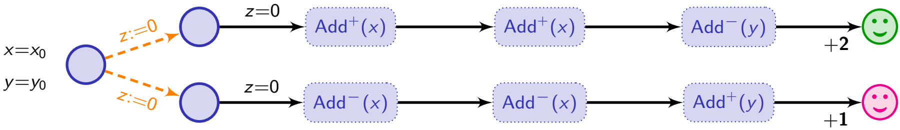
Given two clocks x and y, we can check whether $y = 2x$.



- 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

Computing the optimal cost: why is that hard?

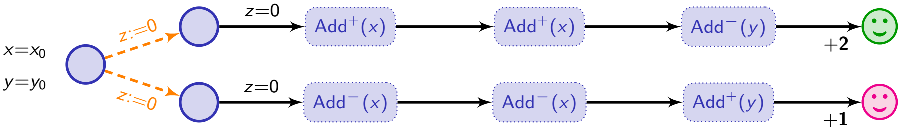
Given two clocks x and y, we can check whether $y = 2x$.



- 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 $= 3$

Computing the optimal cost: why is that hard?

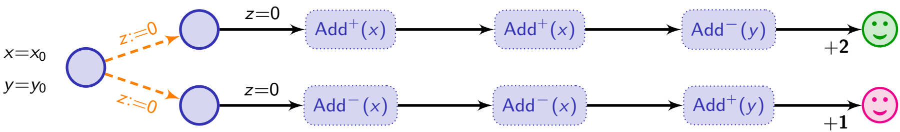
Given two clocks x and y, we can check whether $y = 2x$.



- 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 $= 3$
  $\rightsquigarrow$ player 2 can enforce cost $3 + |y_0 - 2x_0|$

Computing the optimal cost: 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 $= 3$

  $\leadsto$ player 2 can enforce cost $3 + |y_0 - 2x_0|$

- Player 1 has a winning strategy with cost ≤ 3 iff $y_0 = 2x_0$

Computing the optimal cost: 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:

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

Computing the optimal cost: 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:

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

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

Computing the optimal cost: why is that hard?

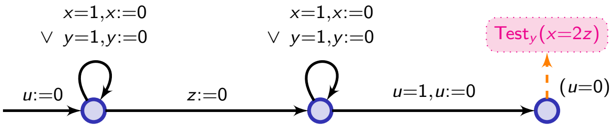
Player 1 will simulate a two-counter machine:

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

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

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

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



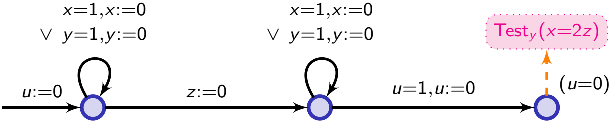
Computing the optimal cost: why is that hard?

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:

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

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



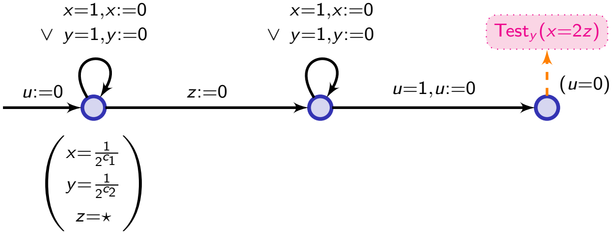
Computing the optimal cost: why is that hard?

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:

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

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



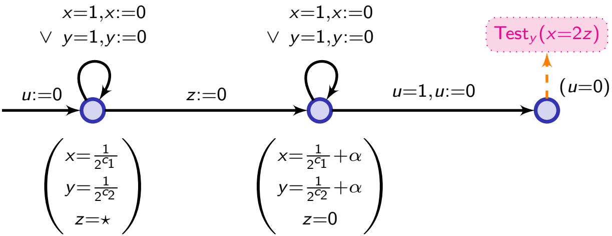
Computing the optimal cost: why is that hard?

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:

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

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



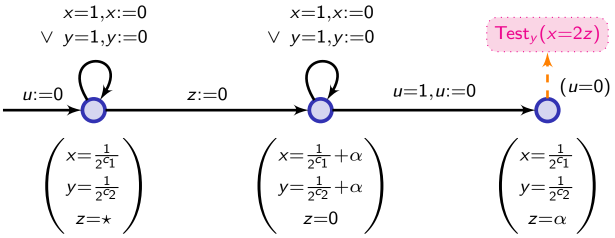
Computing the optimal cost: why is that hard?

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:

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

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



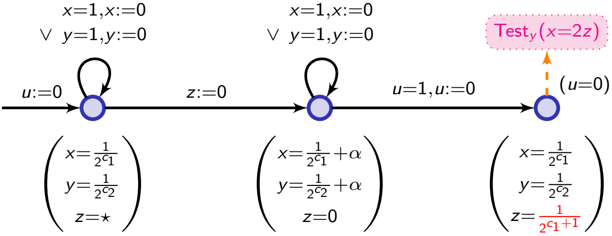
Computing the optimal cost: why is that hard?

Player 1 will simulate a two-counter machine:

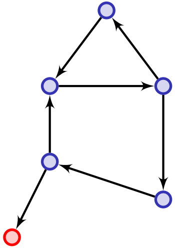
- each instruction is encoded as a module;
- the counter values c_1 and c_2 are encoded by two clocks:

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

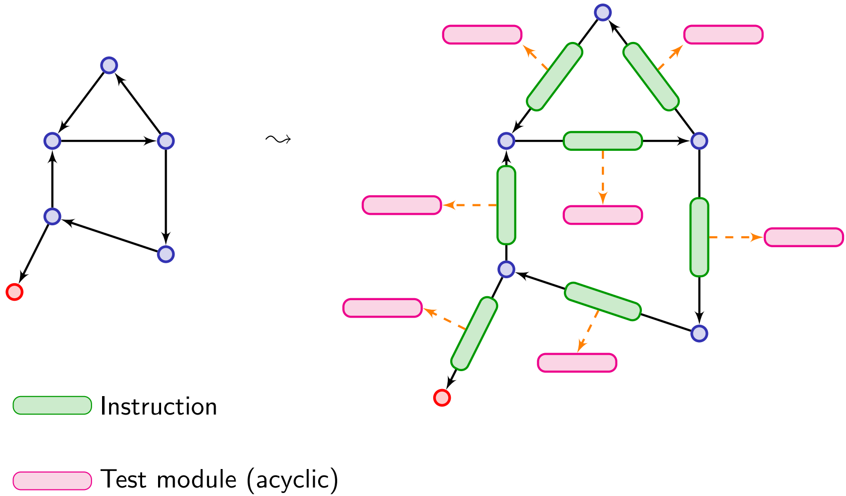
The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.



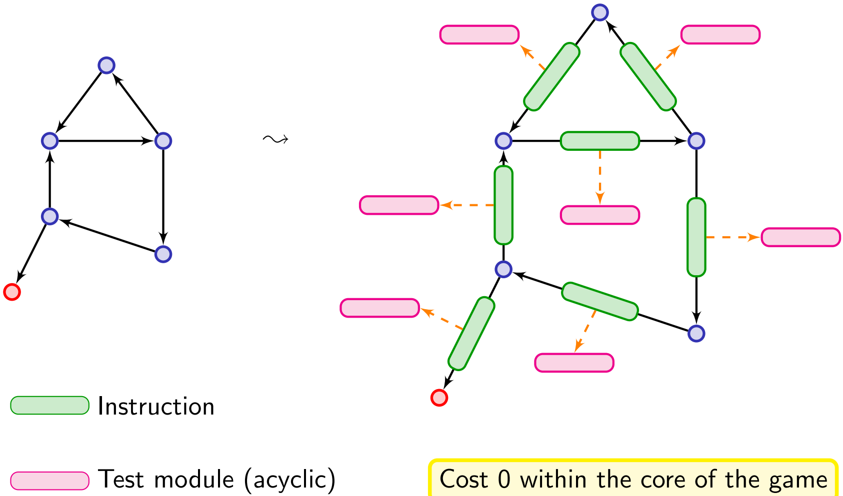
Shape of the reduction



Shape of the reduction



Shape of the reduction



Cost 0 within the core of the game

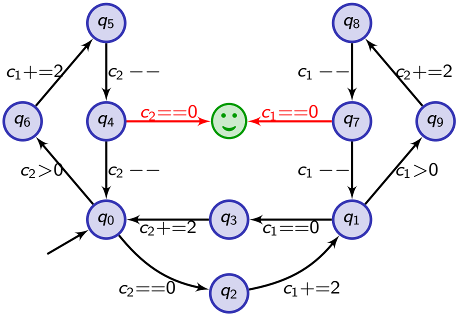
Some further subtlety

Value of the game = infimum of all costs of strategies

Some further subtlety

Value of the game = infimum of all costs of strategies

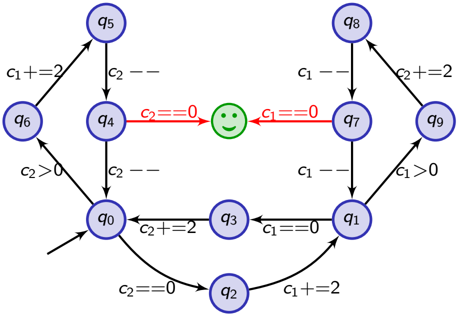
The value of the game is 3, but no strategy has cost 3.

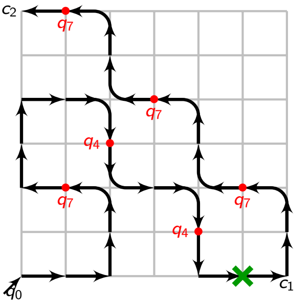


Some further subtlety

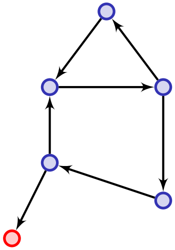
Value of the game = infimum of all costs of strategies

The value of the game is 3, but no strategy has cost 3.

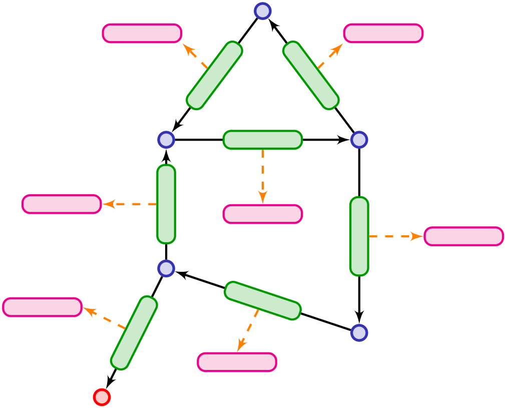




A snapshot on the undecidability proof



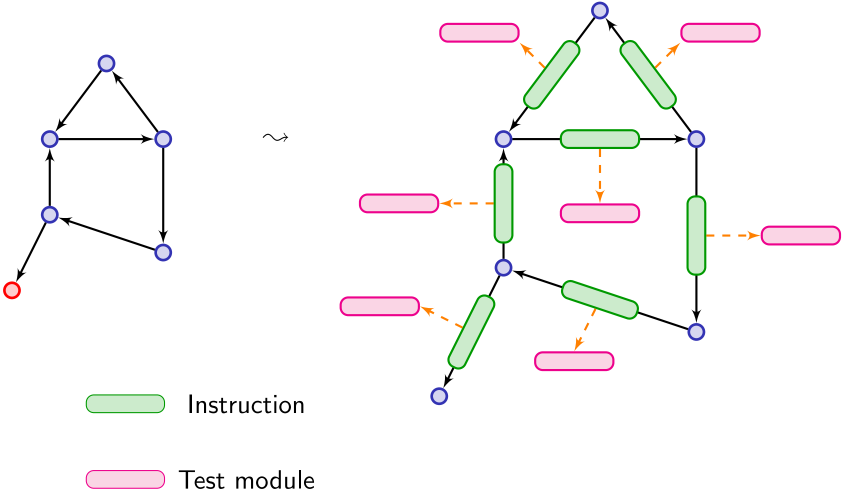
$\rightsquigarrow$



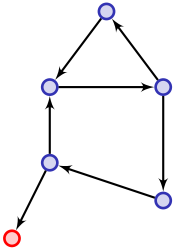
Instruction

Test module

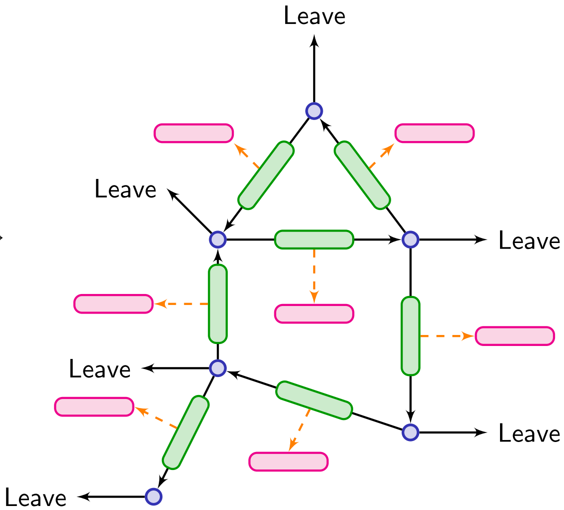
A snapshot on the undecidability proof



A snapshot on the undecidability proof



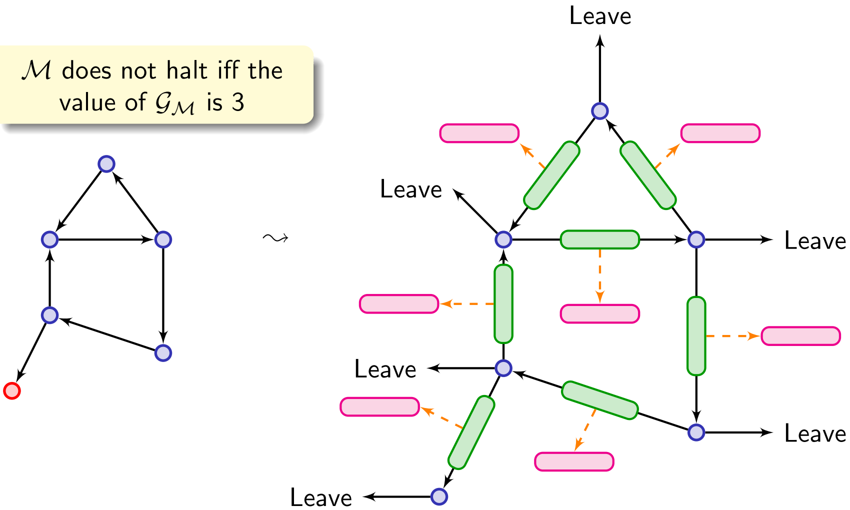
$\rightsquigarrow$



Leave with cost $3 + 1/2^n$ (n: length of the path)

A snapshot on the undecidability proof

$\mathcal{M}$ does not halt iff the value of $\mathcal{G}_{\mathcal{M}}$ is 3



Leave with cost $3 + 1/2^n$ (n: length of the path)

Are we done?

Are we done? No!

Are we done? No!

Optimal cost is computable...

... when cost is strongly non-zeno. [AM04,BCFL04]

There is $\kappa > 0$ s.t. for every region cycle C, for every real run ϱ read on C,

$$\mathsf{cost}(\varrho) \geq \kappa$$

Optimal cost is not computable...

... when cost is almost-strongly non-zeno.

There is $\kappa > 0$ s.t. for every region cycle C, for every real run ϱ read on C,

$$\mathsf{cost}(\varrho) \geq \kappa \quad \text{or} \quad \mathsf{cost}(\varrho) = 0$$

Are we done? No!

Optimal cost is computable...

... when cost is strongly non-zeno. [AM04,BCFL04]

There is $\kappa > 0$ s.t. for every region cycle C, for every real run ϱ read on C,

$$\mathsf{cost}(\varrho) \geq \kappa$$

Optimal cost is not computable... but is approximable!

... when cost is almost-strongly non-zeno. [BJM15]

There is $\kappa > 0$ s.t. for every region cycle C, for every real run ϱ read on C,

$$\mathsf{cost}(\varrho) \geq \kappa \quad \text{or} \quad \mathsf{cost}(\varrho) = 0$$

[BJM15] Bouyer, Jaziri, Markey. On the value problem in weighted timed games *(CONCUR'15)*.

Are we done? No!

Optimal cost is computable...

... when cost is strongly non-zeno. [AM04,BCFL04]

There is $\kappa > 0$ s.t. for every region cycle C, for every real run ϱ read on C,

$$\mathrm{cost}(\varrho) \geq \kappa$$

Optimal cost is not computable... but is approximable!

... when cost is almost-strongly non-zeno. [BJM15]

There is $\kappa > 0$ s.t. for every region cycle C, for every real run ϱ read on C,

$$\mathrm{cost}(\varrho) \geq \kappa \quad \text{or} \quad \mathrm{cost}(\varrho) = 0$$

- Almost-optimality in practice should be sufficient
- Even when we know how to compute the value, we are only able to synthesize almost-optimal strategies...

[BJM15] Bouyer, Jaziri, Markey. On the value problem in weighted timed games *(CONCUR'15)*.

Approximation of the optimal cost

Theorem

Let $\mathcal{G}$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values v_ϵ^- and v_ϵ^+ such that

$$|v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \text{and} \quad v_\epsilon^- \leq \mathsf{optcost}_{\mathcal{G}} \leq v_\epsilon^+$$

Approximation of the optimal cost

Theorem

Let $\mathcal{G}$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values v_ϵ^- and v_ϵ^+ such that

$$|v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \text{and} \quad v_\epsilon^- \leq \mathsf{optcost}_\mathcal{G} \leq v_\epsilon^+$$

- one strategy σ_ϵ such that

$$\mathsf{optcost}_\mathcal{G} \leq \mathsf{cost}(\sigma_\epsilon) \leq \mathsf{optcost}_\mathcal{G} + \epsilon$$

[it is an ϵ-optimal winning strategy]

Approximation of the optimal cost

Theorem

Let $\mathcal{G}$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values v_ϵ^- and v_ϵ^+ such that

$$|v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \text{and} \quad v_\epsilon^- \leq \mathsf{optcost}_\mathcal{G} \leq v_\epsilon^+$$

- one strategy σ_ϵ such that

$$\mathsf{optcost}_\mathcal{G} \leq \mathsf{cost}(\sigma_\epsilon) \leq \mathsf{optcost}_\mathcal{G} + \epsilon$$

  [it is an ϵ-optimal winning strategy]

Skip approximation scheme

Approximation of the optimal cost

Theorem

Let $\mathcal{G}$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values v_ϵ^- and v_ϵ^+ such that

$$|v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \text{and} \quad v_\epsilon^- \leq \mathsf{optcost}_\mathcal{G} \leq v_\epsilon^+$$

- one strategy σ_ϵ such that

$$\mathsf{optcost}_\mathcal{G} \leq \mathsf{cost}(\sigma_\epsilon) \leq \mathsf{optcost}_\mathcal{G} + \epsilon$$

  [it is an ϵ-optimal winning strategy]

- Standard technics: unfold the game to get more precision, and compute two adjacency sequences

Approximation of the optimal cost

Theorem

Let $\mathcal{G}$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values v_ϵ^- and v_ϵ^+ such that

$$|v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \text{and} \quad v_\epsilon^- \leq \mathsf{optcost}_\mathcal{G} \leq v_\epsilon^+$$

- one strategy σ_ϵ such that

$$\mathsf{optcost}_\mathcal{G} \leq \mathsf{cost}(\sigma_\epsilon) \leq \mathsf{optcost}_\mathcal{G} + \epsilon$$

  [it is an ϵ-optimal winning strategy]

- Standard technics: unfold the game to get more precision, and compute two adjacency sequences

$\rightsquigarrow$ This is not possible here
There might be runs with prefixes of arbitrary length and cost 0 (e.g. the game of the undecidability proof)

Idea for approximation

Idea

Only partially unfold the game:

- Keep components with cost 0 untouched – we call it the kernel
- Unfold the rest of the game

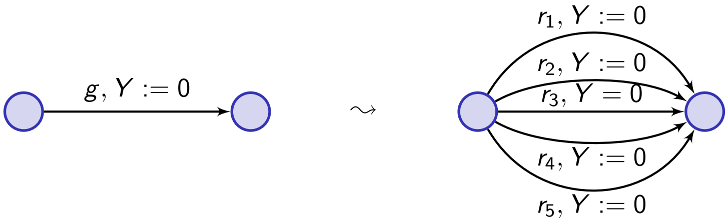
Idea for approximation

Idea

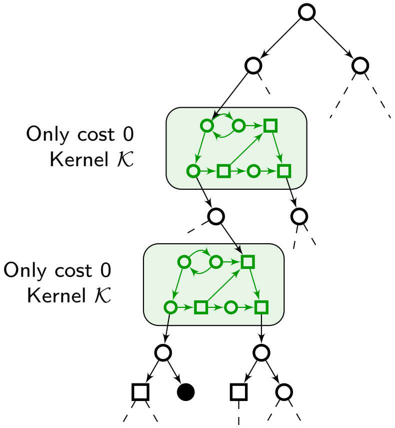
Only partially unfold the game:

- Keep components with cost 0 untouched – we call it the kernel
- Unfold the rest of the game

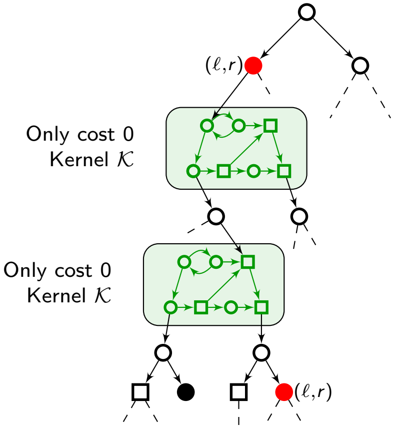
First: split the game along regions!



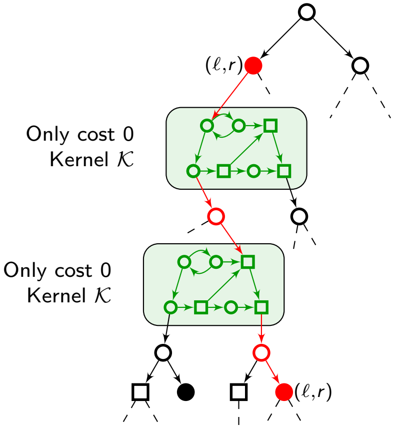
Idea of the proof: Semi-unfolding



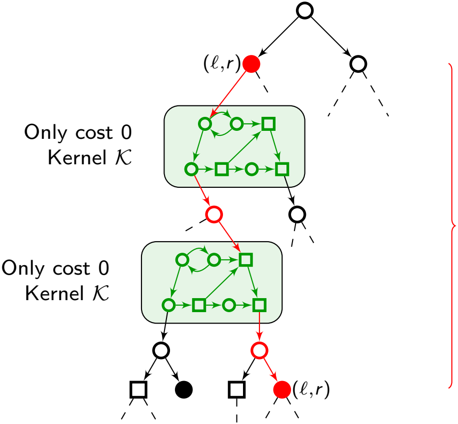
Idea of the proof: Semi-unfolding



Idea of the proof: Semi-unfolding

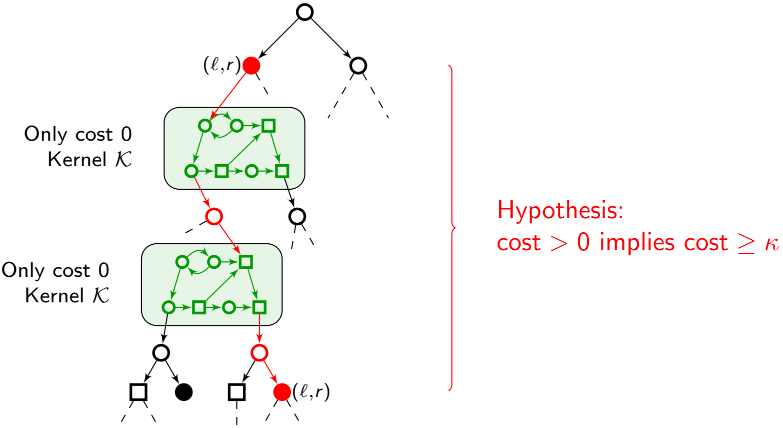


Idea of the proof: Semi-unfolding



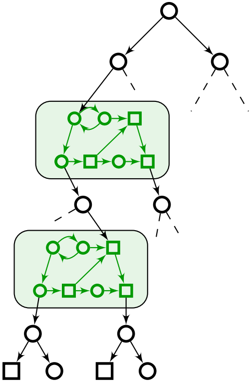
Hypothesis:
cost > 0 implies cost $\geq \kappa$

Idea of the proof: Semi-unfolding

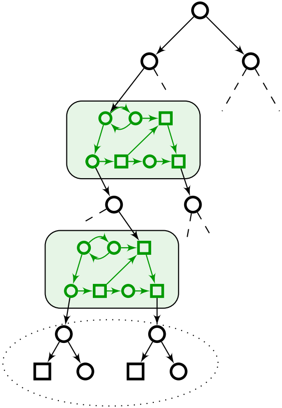


Conclusion: we can stop unfolding the game after finitely many steps

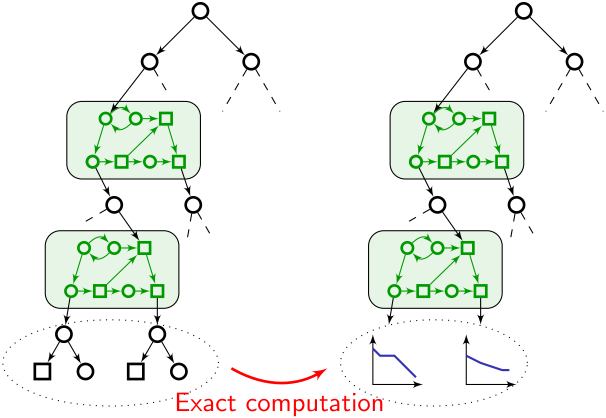
Approximation scheme



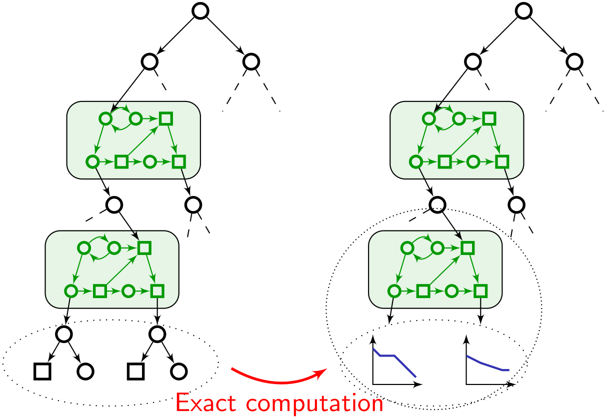
Approximation scheme



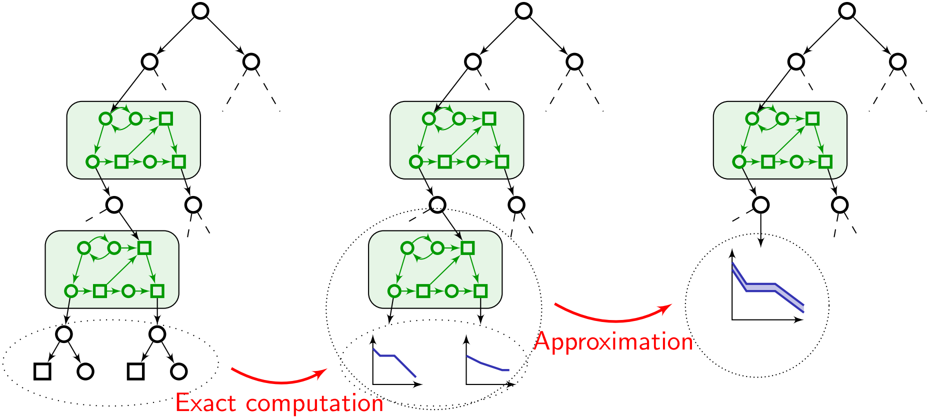
Approximation scheme



Approximation scheme



Approximation scheme

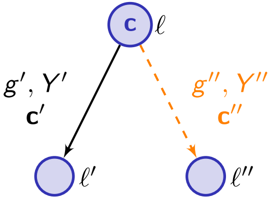


First step: Tree-like parts

$\rightsquigarrow$ Goes back to [LMM02]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.

First step: Tree-like parts

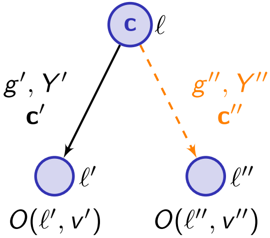


$\rightsquigarrow$ Goes back to [LMM02]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.

First step: Tree-like parts

$\rightsquigarrow$ Goes back to [LMM02]

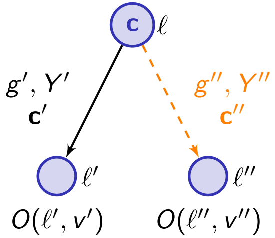


$O(\ell, v) =$

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.

First step: Tree-like parts

$\rightsquigarrow$ Goes back to [LMM02]

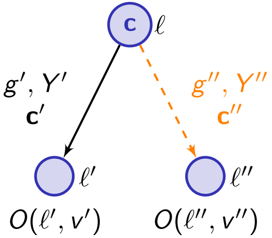


$$O(\ell, v) = \inf_{t' \mid v+t' \models g'}$$

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.

First step: Tree-like parts

$\rightsquigarrow$ Goes back to [LMM02]

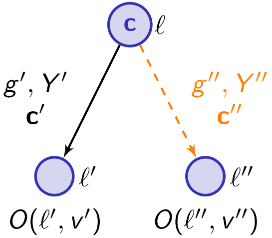


$$O(\ell, v) = \inf_{t' \mid v+t' \models g'} \max(\quad , \quad)$$

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.

First step: Tree-like parts

$\rightsquigarrow$ Goes back to [LMM02]



$$O(\ell, v) = \inf_{t' \mid v+t' \models g'} \max\big((\alpha), \quad\big)$$

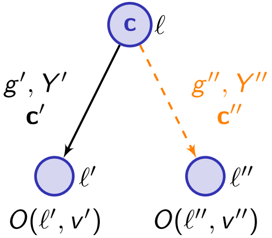
$$(\alpha) = t'\mathbf{c} + \mathbf{c}' + O(\ell', v')$$

$v' = [Y' \leftarrow 0](v + t')$

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.

First step: Tree-like parts

$\rightsquigarrow$ Goes back to [LMM02]



$$O(\ell, v) = \inf_{t' \mid v+t' \models g'} \max\big((\alpha), (\beta)\big)$$

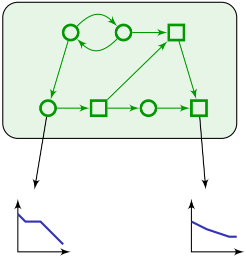
$$(\alpha) = t'\mathbf{c} + \mathbf{c}' + O(\ell', v')$$

$$(\beta) = \sup_{t'' \leq t' \mid v+t'' \models g''} t''\mathbf{c} + \mathbf{c}'' + O(\ell'', v'')$$

$$v' = [Y' \leftarrow 0](v+t')$$
$$v'' = [Y'' \leftarrow 0](v+t'')$$

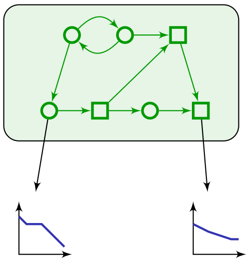
[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.

Second step: Kernels



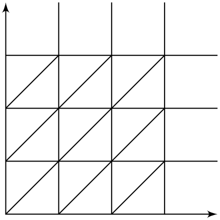
Output cost functions f

Second step: Kernels

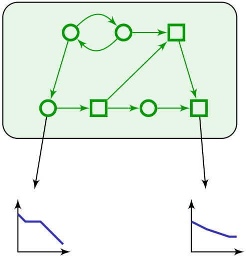


Output cost functions f

1. Refine the regions such that f differs of at most ϵ within a small region

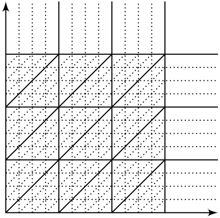


Second step: Kernels

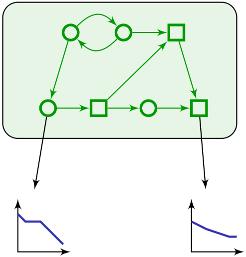


Output cost functions f

1. Refine the regions such that f differs of at most ϵ within a small region

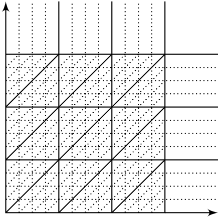


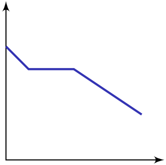
Second step: Kernels



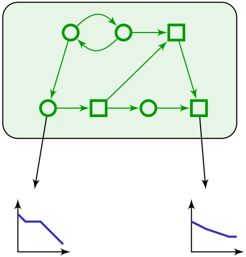
Output cost functions f

1. Refine the regions such that f differs of at most ϵ within a small region



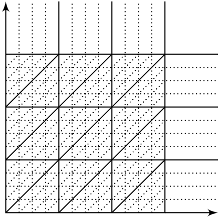


Second step: Kernels

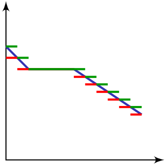


Output cost functions f

1. Refine the regions such that f differs of at most ϵ within a small region

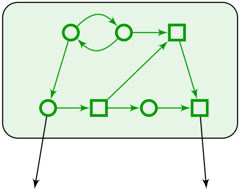


2. Under- and over-approximate by piecewise constant functions f_ϵ^- and f_ϵ^+



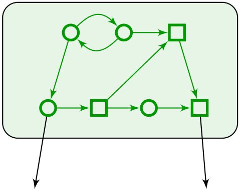
Second step: Kernels

3. Refine/split the kernel along the new small regions and fix f_ϵ^- or f_ϵ^+, write f_ϵ



f_ϵ: constant $\qquad f_\epsilon$: constant

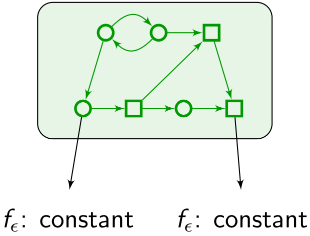
Second step: Kernels



f_ϵ: constant f_ϵ: constant

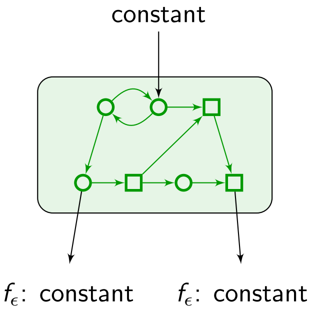
3. Refine/split the kernel along the new small regions and fix f_ϵ^- or f_ϵ^+, write f_ϵ
4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by f_ϵ)

Second step: Kernels



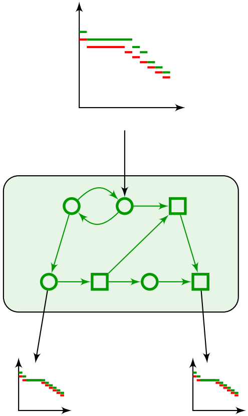
3. Refine/split the kernel along the new small regions and fix f_ϵ^- or f_ϵ^+, write f_ϵ
4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by f_ϵ)
5. Those can be solved using standard technics based on attractors: small regions are sufficient, and the local optimal cost (for output f_ϵ) is constant within a small region

Second step: Kernels



3. Refine/split the kernel along the new small regions and fix f_ϵ^- or f_ϵ^+, write f_ϵ
4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by f_ϵ)
5. Those can be solved using standard technics based on attractors: small regions are sufficient, and the local optimal cost (for output f_ϵ) is constant within a small region

Second step: Kernels



3. Refine/split the kernel along the new small regions and fix f_ϵ^- or f_ϵ^+, write f_ϵ
4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by f_ϵ)
5. Those can be solved using standard technics based on attractors: small regions are sufficient, and the local optimal cost (for output f_ϵ) is constant within a small region

$\rightsquigarrow$ We have computed ϵ-approximations of the optimal cost, which are constant within small regions. Corresponding strategies can be inferred

Outline

Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...

Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...
- Tool-suite Uppaal, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995
  - Uppaal for timed automata
  - Uppaal-TiGa for timed games
  - Uppaal-Cora for weighted timed automata

Uppaal url: http://www.uppaal.org

Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...
- Tool-suite Uppaal, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995

Uppaal url: http://www.uppaal.org

Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...
- Tool-suite Uppaal, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995
- Our new tool TiAMo, developed by Maximilien Colange (formerly at LSV), using code by Ocan Sankur (IRISA, France)

TiAMo = Timed Automata Model-checker

Uppaal url: http://www.uppaal.org
TiAMo url: https://git.lsv.fr/colange/tiamo

[BCM16] Bouyer, Colange, Markey. Symbolic optimal reachability in weighted timed automata *(CAV'16)*.

Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...
- Tool-suite Uppaal, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995
- Our new tool TiAMo, developed by Maximilien Colange (formerly at LSV), using code by Ocan Sankur (IRISA, France)

TiAMo = Timed Automata Model-checker

- Timed automata: (time-optimal) reachability
- Weighted timed automata: optimal rechability

Uppaal url: http://www.uppaal.org
TiAMo url: https://git.lsv.fr/colange/tiamo
[BCM16] Bouyer, Colange, Markey. Symbolic optimal reachability in weighted timed automata *(CAV'16)*.

Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...
- Tool-suite Uppaal, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995
- Our new tool TiAMo, developed by Maximilien Colange (formerly at LSV), using code by Ocan Sankur (IRISA, France)

TiAMo = Timed Automata Model-checker

- Timed automata: (time-optimal) reachability
- Weighted timed automata: optimal rechability

- Aims at being a platform for experiments (open source!)
- Aims at asserting and comparing algorithms

Uppaal url: http://www.uppaal.org
TiAMo url: https://git.lsv.fr/colange/tiamo
[BCM16] Bouyer, Colange, Markey. Symbolic optimal reachability in weighted timed automata *(CAV'16)*.

Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...
- Tool-suite Uppaal, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995
- Our new tool TiAMo, developed by Maximilien Colange (formerly at LSV), using code by Ocan Sankur (IRISA, France)

TiAMo = Timed Automata Model-checker

https://git.lsv.fr/colange/tiamo

Uppaal url: http://www.uppaal.org
TiAMo url: https://git.lsv.fr/colange/tiamo
[BCM16] Bouyer, Colange, Markey. Symbolic optimal reachability in weighted timed automata *(CAV'16)*.

Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...
- Tool-suite Uppaal, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995
- Our new tool TiAMo, developed by Maximilien Colange (formerly at LSV), using code by Ocan Sankur (IRISA, France)

TiAMo = Timed Automata Model-checker

`https://git.lsv.fr/colange/tiamo`

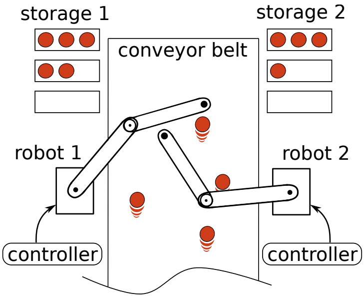
- In the future: TiAMo will merge with TChecker (developed by Frédéric Herbreteau (LaBRI, France))

Uppaal url: `http://www.uppaal.org`
TiAMo url: `https://git.lsv.fr/colange/tiamo`
[BCM16] Bouyer, Colange, Markey. Symbolic optimal reachability in weighted timed automata *(CAV'16)*.

Outline

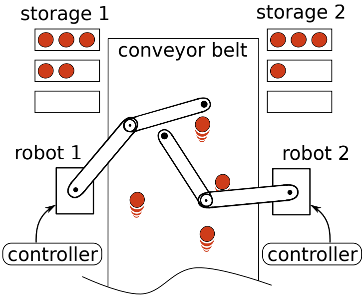
1. Timed automata
2. Timed temporal logics
3. Weighted timed automata
4. Timed games
5. Weighted timed games
6. Tools
7. Towards applying all this theory to robotic systems
8. Conclusion

Example problem, objective and approach



[BMPS15] Bouyer, Markey, Perrin, Schlehuber-Caissier. Timed-Automata Abstraction of Switched Dynamical Systems Using Control Funnels *(FORMATS'15)*.

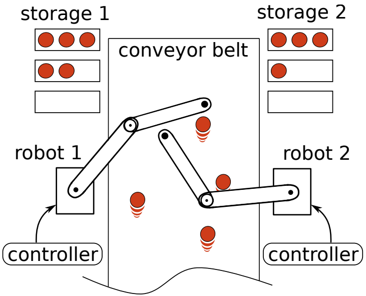
Example problem, objective and approach



- Infinitely many configurations
- Complex behaviour
- Mechanical constraints

[BMPS15] Bouyer, Markey, Perrin, Schlehuber-Caissier. Timed-Automata Abstraction of Switched Dynamical Systems Using Control Funnels *(FORMATS'15)*.

Example problem, objective and approach



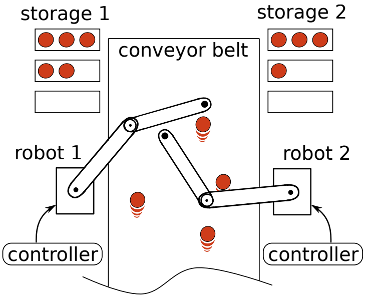
Goal: Synthesize a controller:

- Which robot handles an object
- How to avoid collision
- Don't miss any object

- Infinitely many configurations
- Complex behaviour
- Mechanical constraints

[BMPS15] Bouyer, Markey, Perrin, Schlehuber-Caissier. Timed-Automata Abstraction of Switched Dynamical Systems Using Control Funnels *(FORMATS'15)*.

Example problem, objective and approach



- Infinitely many configurations
- Complex behaviour
- Mechanical constraints

Goal: Synthesize a controller:

- Which robot handles an object
- How to avoid collision
- Don't miss any object

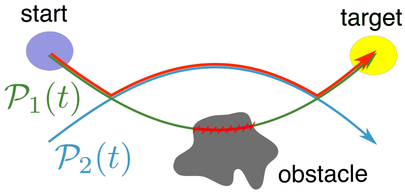
Approach:

- Discretization of the behaviour via a fixed set of continuous controllers
- Create an abstraction and use previous results

[BMPS15] Bouyer, Markey, Perrin, Schlehuber-Caissier. Timed-Automata Abstraction of Switched Dynamical Systems Using Control Funnels *(FORMATS'15)*.

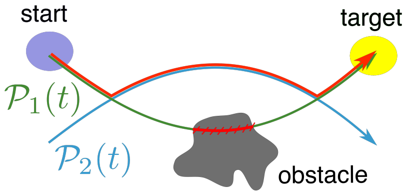
Our approach

Simplistic idea: fixed set of reference trajectories + property

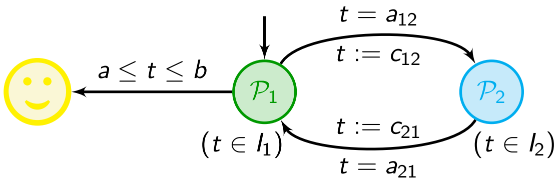


Our approach

Simplistic idea: fixed set of reference trajectories + property

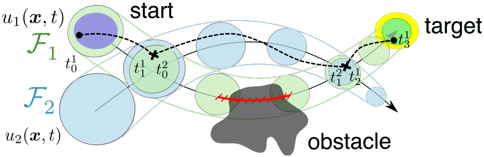


Corresponding timed automaton:



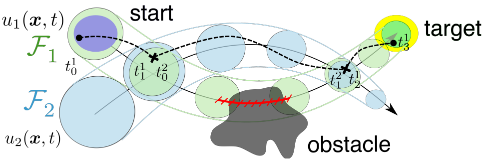
Our approach

More realistic idea: fixed set of funnels for control law + property

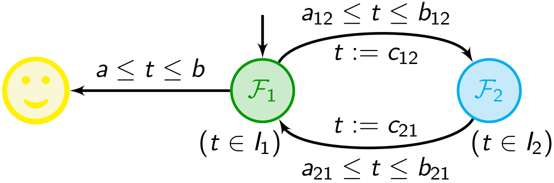


Our approach

More realistic idea: fixed set of funnels for control law + property

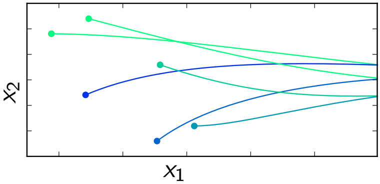


Corresponding timed automaton:



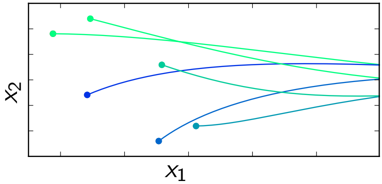
Control funnels

System with continuous dynamics $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t)$



Control funnels

System with continuous dynamics $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t)$

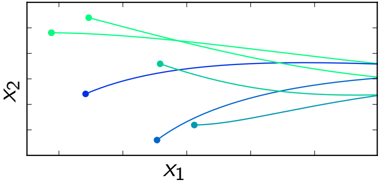


A (control) funnel is a trajectory $\mathcal{F}(t)$ of a set in the state space such that, for any trajectory $\boldsymbol{x}(t)$ of the dynamical system:

$$\forall t_0 \in \mathbb{R},\ \boldsymbol{x}(t_0) \in \mathcal{F}(t_0) \Rightarrow \forall t \geq t_0,\ x(t) \in \mathcal{F}(t)$$

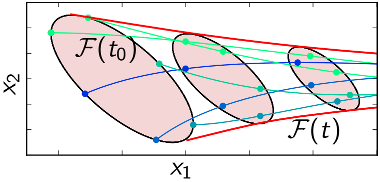
Control funnels

System with continuous dynamics $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t)$

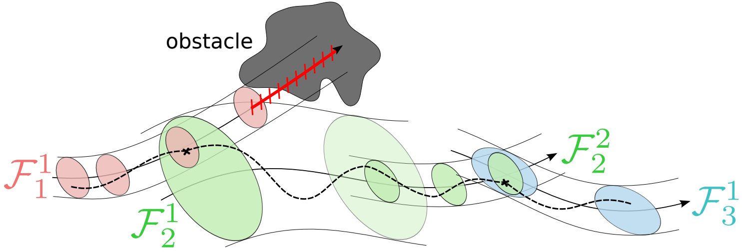


A (control) funnel is a trajectory $\mathcal{F}(t)$ of a set in the state space such that, for any trajectory $\boldsymbol{x}(t)$ of the dynamical system:

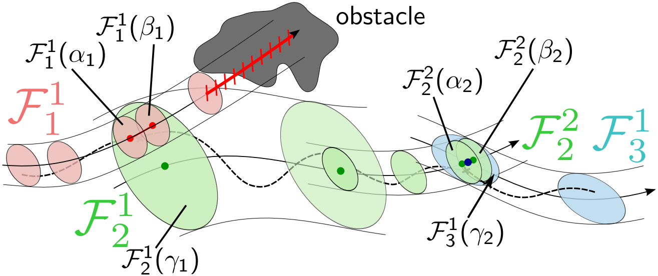
$$\forall t_0 \in \mathbb{R},\ \boldsymbol{x}(t_0) \in \mathcal{F}(t_0) \Rightarrow \forall t \geq t_0,\ x(t) \in \mathcal{F}(t)$$



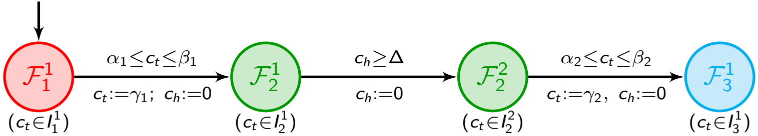
Example



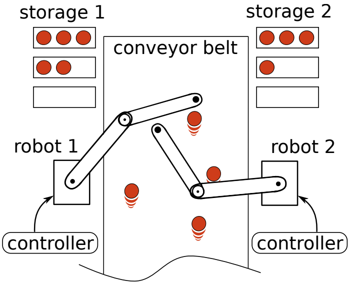
Example



c_t: positional clock; c_h: local clock

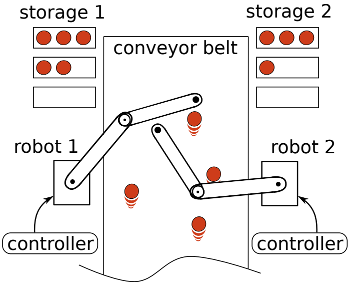


Summary



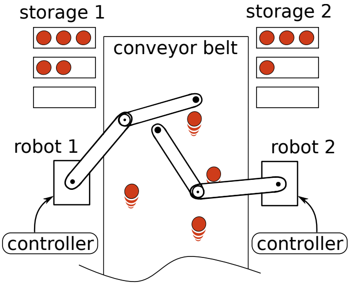
storage 1
storage 2
conveyor belt
robot 1
robot 2
controller
controller

Summary



$\rightsquigarrow$ (huge) timed automata/games (with weights), with few clocks

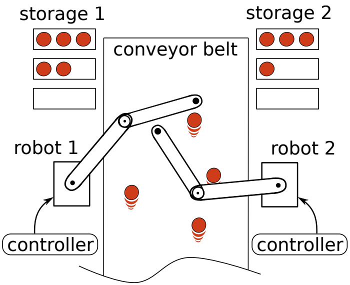
Summary



$\rightsquigarrow$ (huge) timed automata/games (with weights), with few clocks

$\leftarrow$ winning (optimal) strategy

Summary

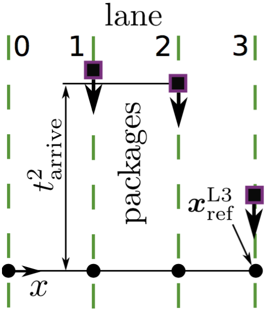


$\rightsquigarrow$ (huge) timed automata/games (with weights), with few clocks

safe (good) controller $\leftarrow$ winning (optimal) strategy

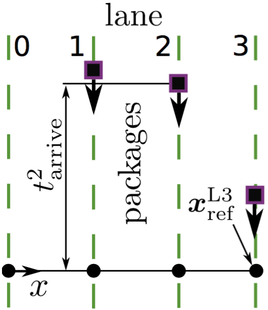
A pick-and-place example

1d point mass

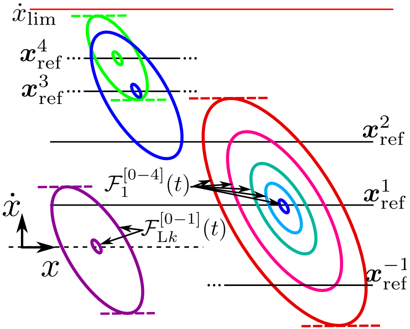


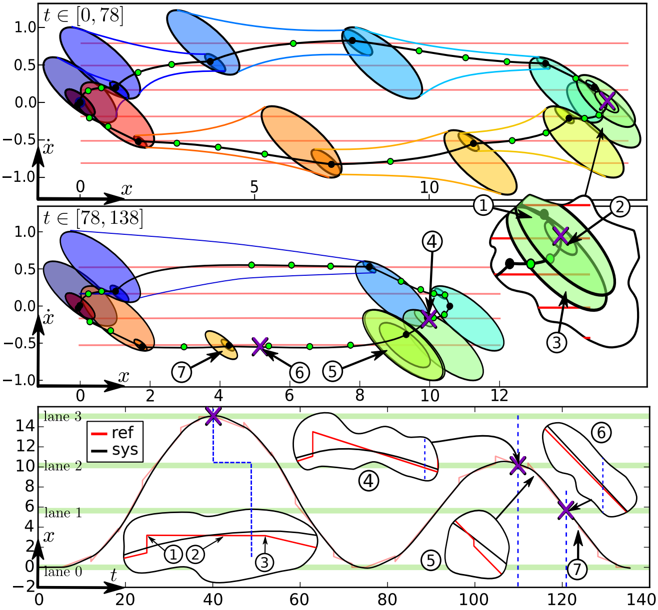
A pick-and-place example

1d point mass



Funnel system




$t \in [0, 78]$
$t \in [78, 138]$
$\dot{x}$
x
t
lane 3
lane 2
lane 1
lane 0
ref
sys
1
2
3
4
5
6
7

Current challenges

For control people

- Handle more non-linear systems (automatically build control funnels)

Current challenges

For control people

- Handle more non-linear systems (automatically build control funnels)

For us

- Does not scale up very well so far (huge timed automata models)
    - Build the model on-demand?
      But, can we give guarantees (optimality) when only part of the model has been built?
    - Develop specific algorithms for the special timed automata we construct?
- Implement efficient approx. algorithm for weighted timed games

Outline

Conclusion

Summary of the talk

- Basics of timed automata verification
- Relevant extensions for applications: weights, games, mix of both
  - We looked at decidability and limits
  - We mentioned algorithmics and tools
- Timed automata can be used as abstractions for more complex systems

Conclusion

Current challenges

- Various theoretical issues
  - Decidability and approximability of weighted timed automata and games
  - New approaches (tree automata, reachability relations) might give a new light on the verification of timed systems
  - Robustness and implementability
- Continue working on algorithms and tools

TiAMo + TChecker

  - Implementation of (weighted) timed games (good data structures, abstractions, etc.)
  - More applications with specific challenges (e.g. robotic problems)