On the optimal reachability problem in weighted timed games

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France

Based on former works with Thomas Brihaye, Kim G. Larsen, Nicolas Markey, etc...
And on recent work with Samy Jaziri and Nicolas Markey

Outline

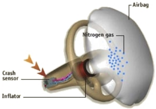
Time-dependent systems

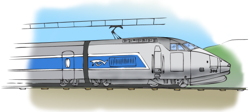
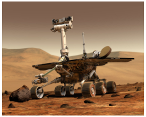
- We are interested in timed systems

Time-dependent systems

- We are interested in timed systems

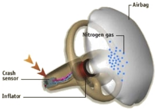


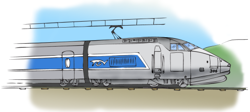


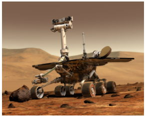




Time-dependent systems

- We are interested in timed systems



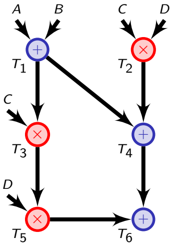




- ... and in their analysis and control

An example: The task graph scheduling problem

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

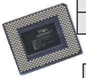


P_1 (fast):

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.

An example: The task graph scheduling problem

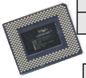
Compute $D\times(C\times(A+B))+(A+B)+(C\times D)$ using two processors:

P_1 (fast):

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

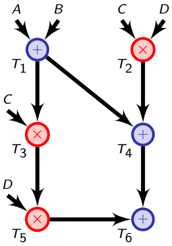
energy	
idle	10 Watt
in use	90 Watts

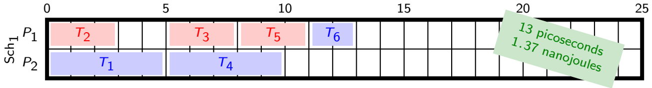
P_2 (slow):



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

energy	
idle	20 Watts
in use	30 Watts

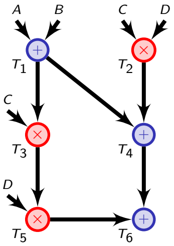




[BFLM10] Bouyer, Fahrenberg, Larsen, Markey. Quantitative Analysis of Real-Time Systems using Priced Timed Automata.

An example: The task graph scheduling problem

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

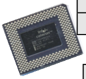


P_1 (fast):

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

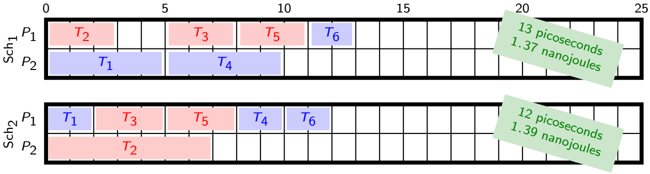
	energy
idle	10 Watt
in use	90 Watts

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.

An example: The task graph scheduling problem

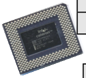
Compute $D \times (C \times (A + B)) + (A + B) + (C \times D)$ using two processors:

P_1 (fast):

	time
+	2 picoseconds
×	3 picoseconds

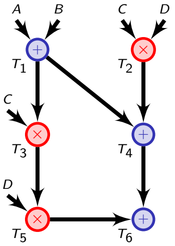
	energy
idle	10 Watt
in use	90 Watts

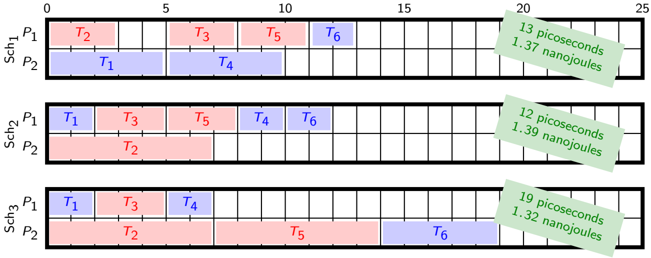
P_2 (slow):



	time
+	5 picoseconds
×	7 picoseconds

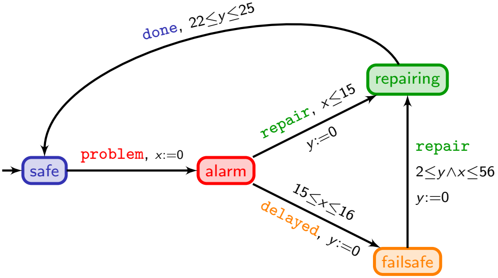
	energy
idle	20 Watts
in use	30 Watts



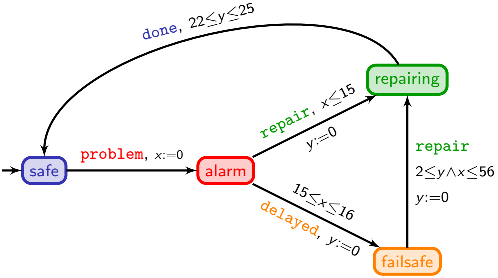


[BFLM10] Bouyer, Fahrenberg, Larsen, Markey. Quantitative Analysis of Real-Time Systems using Priced Timed Automata.

The model of timed automata



The model of timed automata



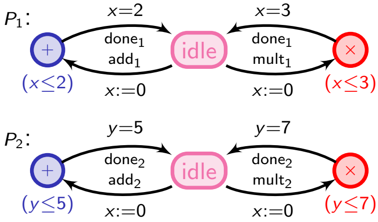
	safe	$\xrightarrow{23}$	safe	$\xrightarrow{\texttt{problem}}$	alarm	$\xrightarrow{15.6}$	alarm	$\xrightarrow{\texttt{delayed}}$	failsafe
x	0		23		0		15.6		15.6 ···
y	0		23		23		38.6		0

	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

Modelling the task graph scheduling problem

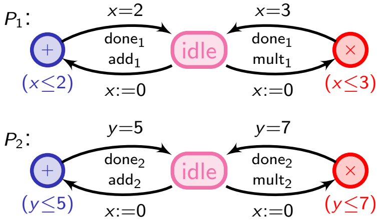
Modelling the task graph scheduling problem

- Processors

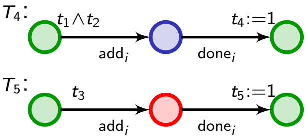


Modelling the task graph scheduling problem

- Processors

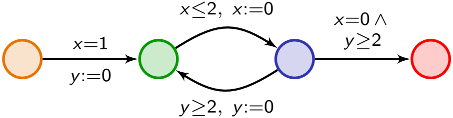


- Tasks

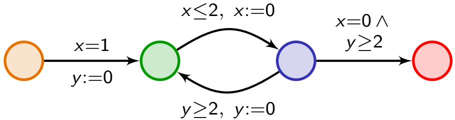


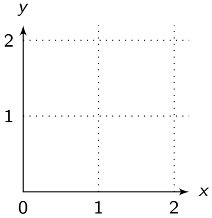
A schedule is a path in the product automaton

Analyzing timed automata

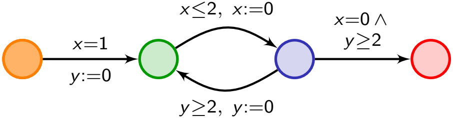


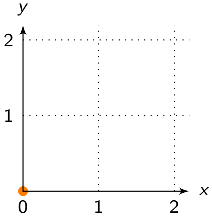
Analyzing timed automata



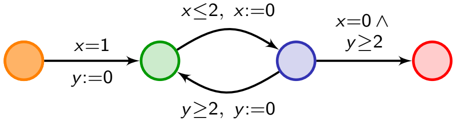


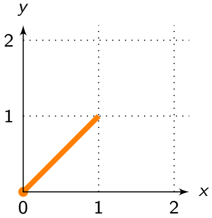
Analyzing timed automata



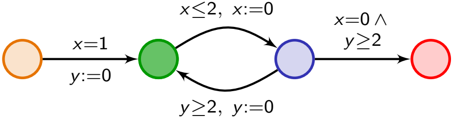


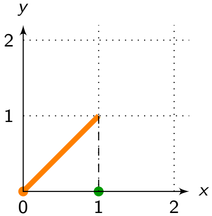
Analyzing timed automata



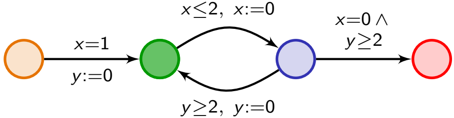


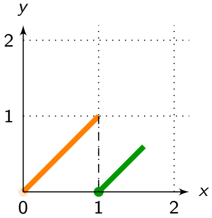
Analyzing timed automata



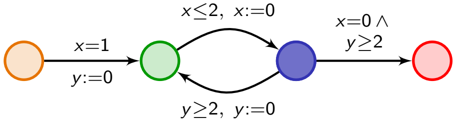


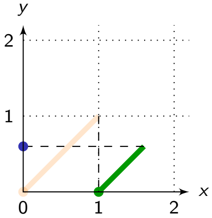
Analyzing timed automata



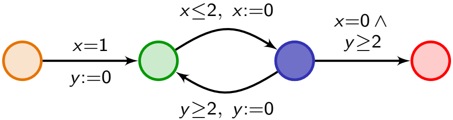


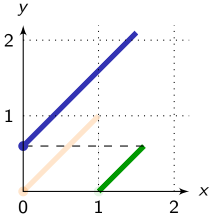
Analyzing timed automata



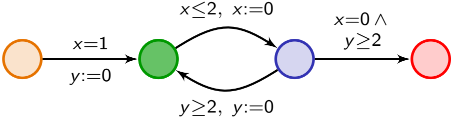


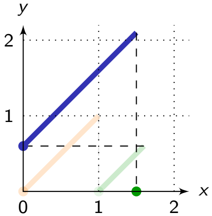
Analyzing timed automata



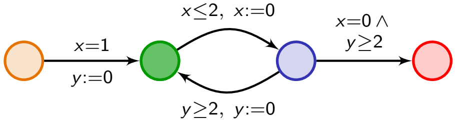


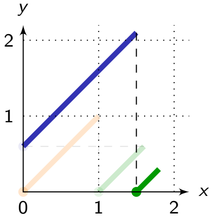
Analyzing timed automata



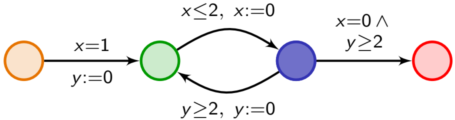


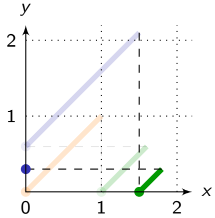
Analyzing timed automata



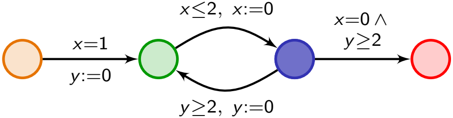


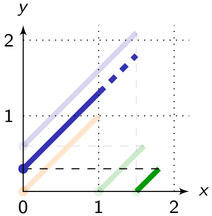
Analyzing timed automata



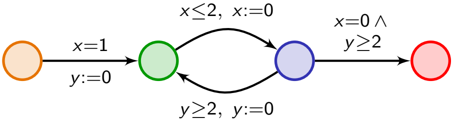


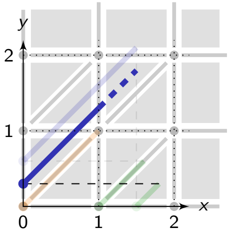
Analyzing timed automata



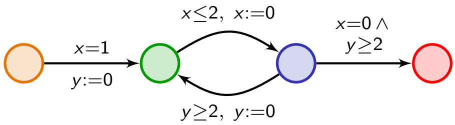


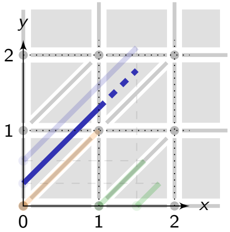
Analyzing timed automata





Analyzing timed automata



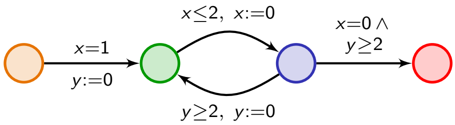


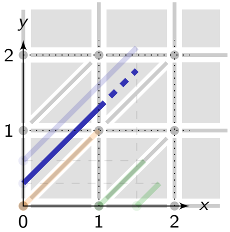
Theorem [AD94]

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

- Technical tool: region abstraction

Analyzing timed automata



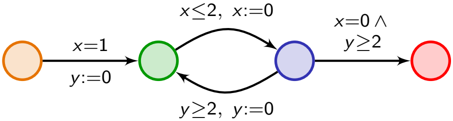


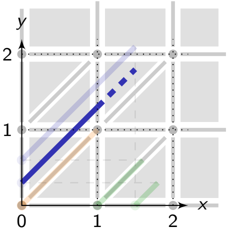
Theorem [AD94]

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

- Technical tool: region abstraction
- Efficient symbolic technics based on zones, implemented in tools

Analyzing timed automata





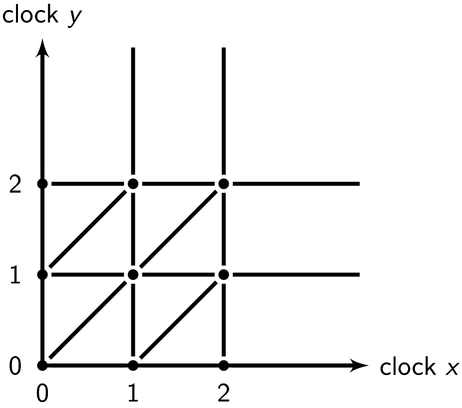
Theorem [AD94]

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

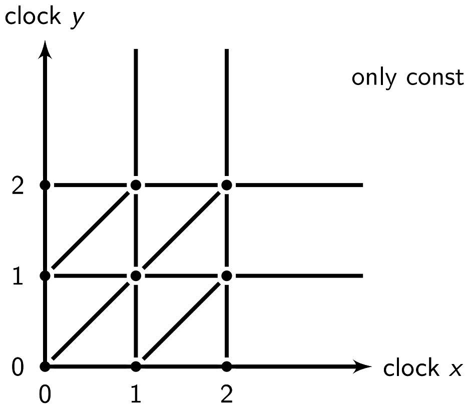
- Technical tool: region abstraction
- Efficient symbolic technics based on zones, implemented in tools

Skip regions

Technical tool: Region abstraction



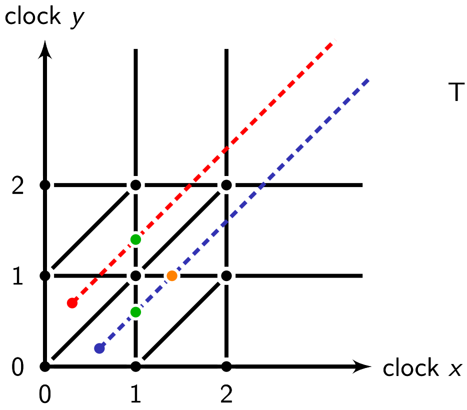
Technical tool: Region abstraction



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

Technical tool: Region abstraction

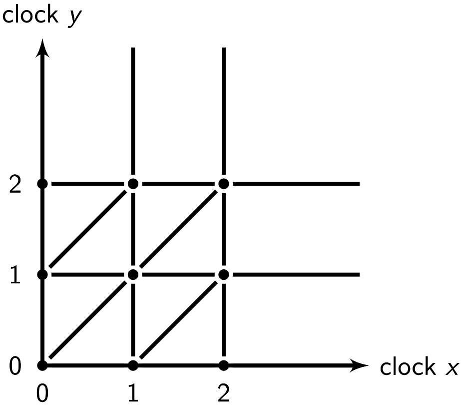


The path 

- can be fired from
- cannot be fired from

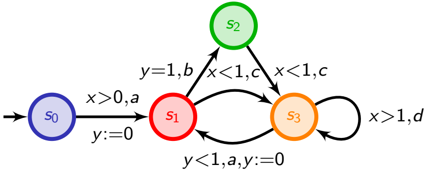
- "compatibility" between regions and constraints
- "compatibility" between regions and time elapsing

Technical tool: Region abstraction

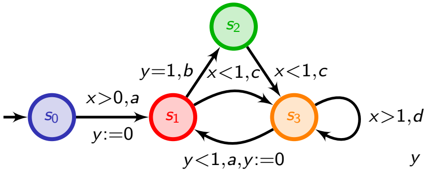


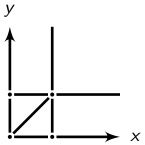
$\rightsquigarrow$ This is a finite time-abstract bisimulation!

Technical tool: Region abstraction – An example [AD94]

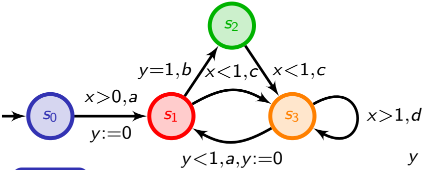


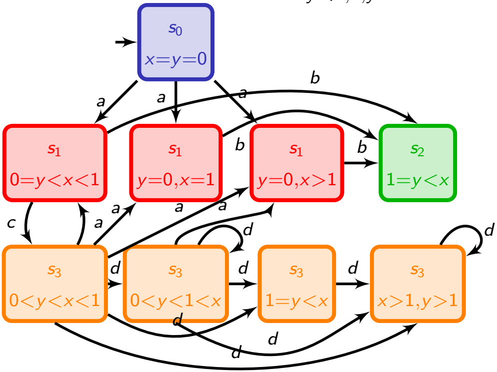
Technical tool: Region abstraction – An example [AD94]

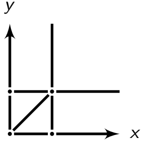




Technical tool: Region abstraction – An example [AD94]







Outline

Outline

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,

$\leadsto$ 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

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

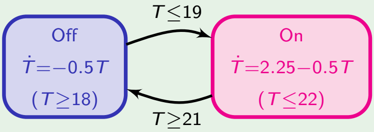
Modelling resources in timed systems

- 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



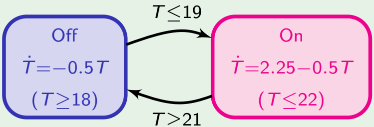
Modelling resources in timed systems

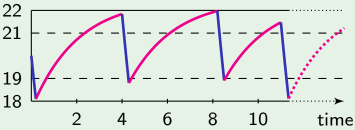
- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,

$\leadsto$ timed automata are not powerful enough!

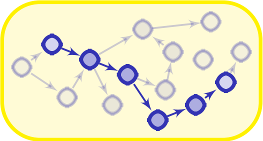
- A possible solution: use hybrid automata

The thermostat example

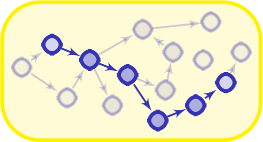




Ok...

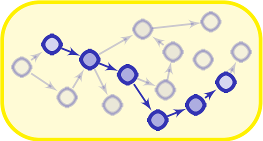


Ok...

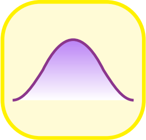


Easy...

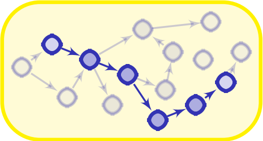
Ok...



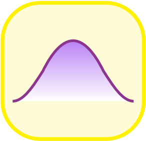
Easy...



Ok...

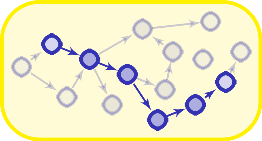


Easy...

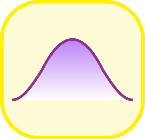


Easy...

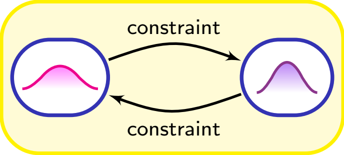
Ok... but?



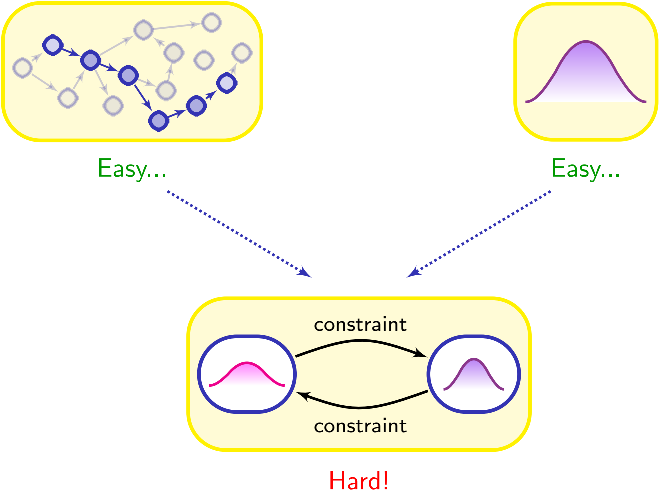
Easy...



Easy...



Ok... but?



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,

  $\leadsto$ timed automata are not powerful enough!

- A possible solution: use hybrid automata

Theorem [HKPV95]

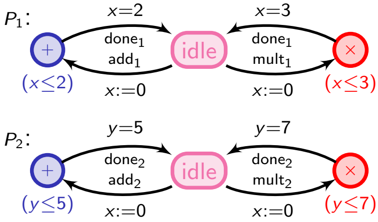
The reachability problem is undecidable in hybrid automata. Even for the simplest, the so-called stopwatch automata (clocks can be stopped).

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

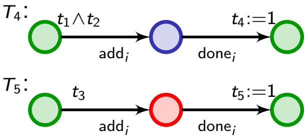
[HKPV95] Henzinger, Kopke, Puri, Varaiya. What's decidable wbout hybrid automata? *(SToC'95)*.[ALP01] Alur, La Torre, Pappas. Optimal paths in weigh
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

Modelling the task graph scheduling problem

- Processors

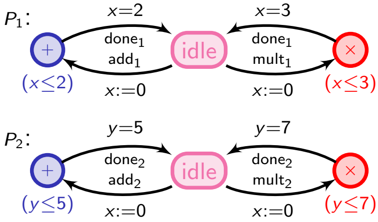


- Tasks

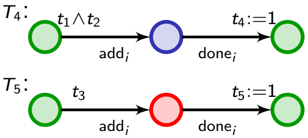


Modelling the task graph scheduling problem

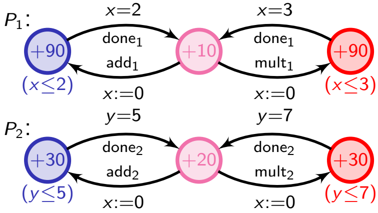
- Processors



- Tasks

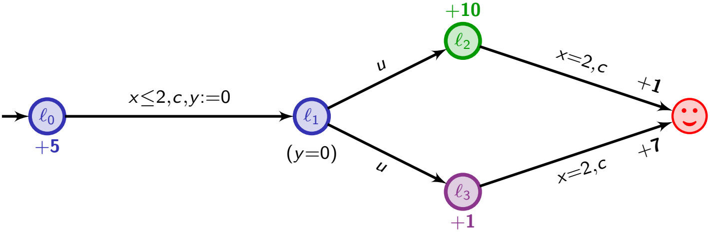


- Modelling energy



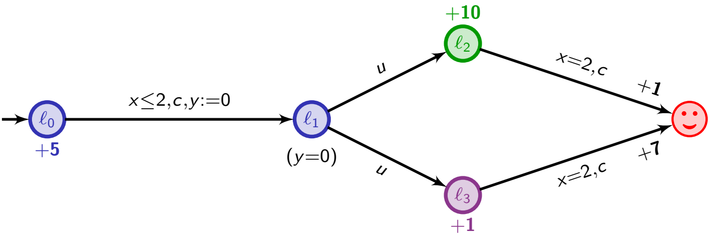
A good schedule is a path in the product automaton with a low cost

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)*.

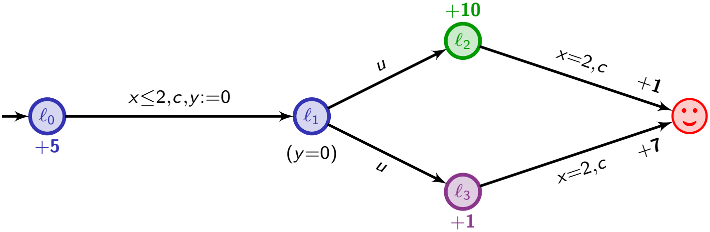
Weighted/priced timed automata [ALP01,BFH+01]



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

[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]

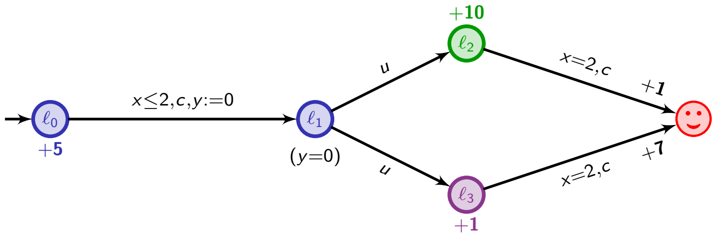


$$
\begin{array}{ccccccccccccc}
 & \ell_0 & \xrightarrow{1.3} & \ell_0 & \xrightarrow{c} & \ell_1 & \xrightarrow{u} & \ell_3 & \xrightarrow{0.7} & \ell_3 & \xrightarrow{c} & ☺ \\
x & 0 & & 1.3 & & 1.3 & & 1.3 & & 2 & & \\
y & 0 & & 1.3 & & 0 & & 0 & & 0.7 & &
\end{array}
$$

cost :

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

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

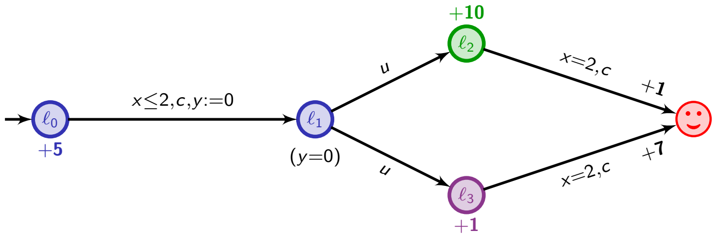


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : 6.5

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

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

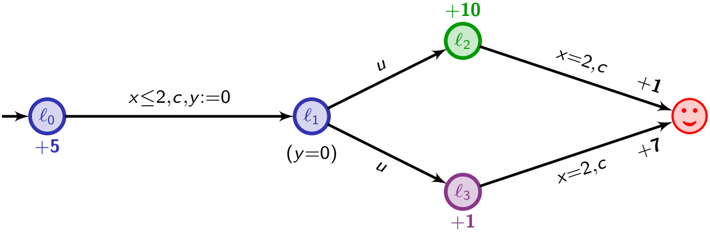


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : 6.5 + 0

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

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

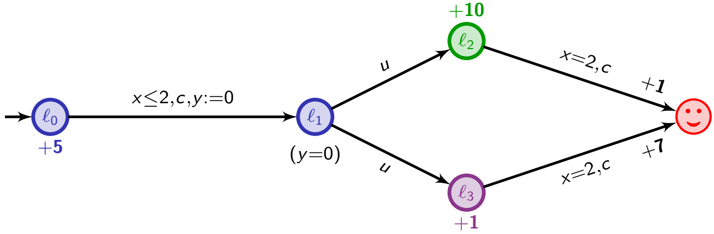


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : 6.5 + 0 + 0

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

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

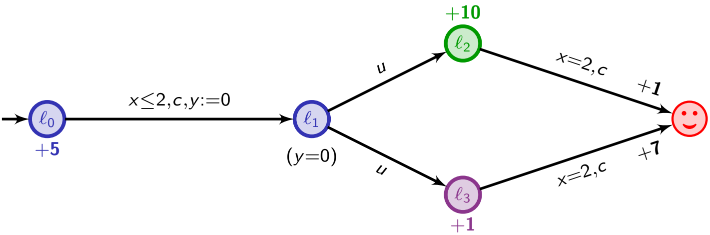


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : 6.5 + 0 + 0 + 0.7

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

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

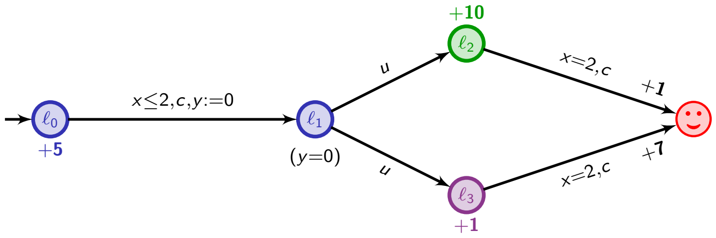


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

cost : 6.5 + 0 + 0 + 0.7 + 7

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

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

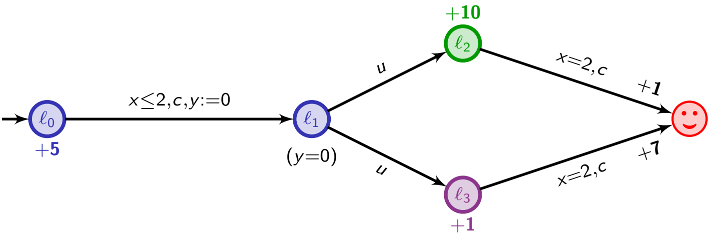


	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺
x	0		1.3		1.3		1.3		2		
y	0		1.3		0		0		0.7		

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

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

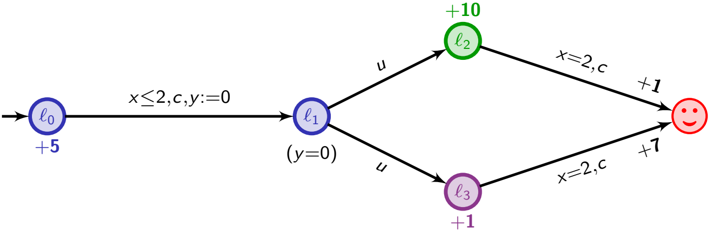
Weighted/priced timed automata [ALP01,BFH+01]



Question: what is the optimal cost for reaching 🙂 ?

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

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

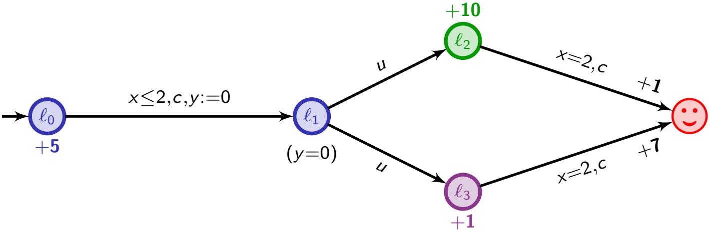


Question: what is the optimal cost for reaching 🙂 ?

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

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

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

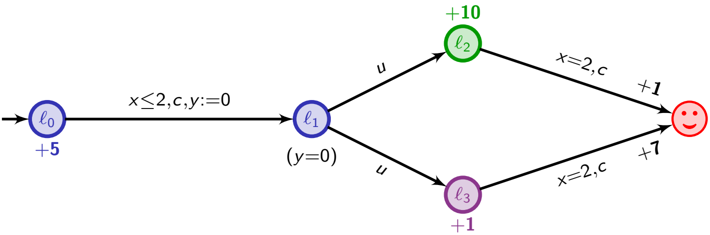


Question: what is the optimal cost for reaching 🙂?

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

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

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

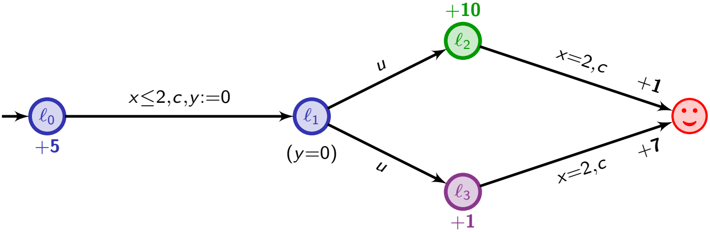


Question: what is the optimal cost for reaching ☺ ?

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

[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]

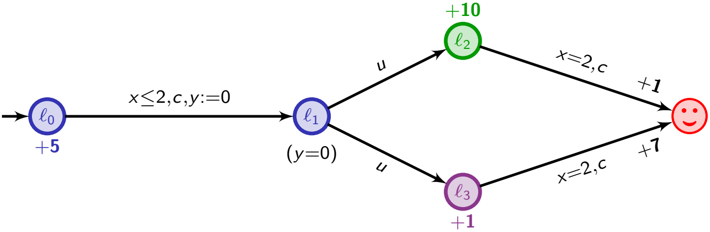


Question: what is the optimal cost for reaching 🙂?

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

[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.

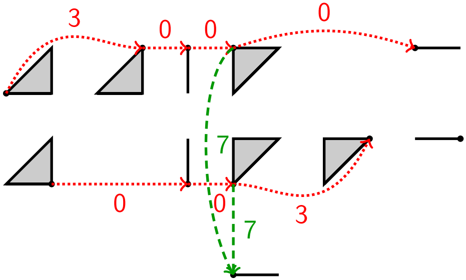
[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata *(HSCC'01)*.
[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC'01)*.

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)*.

Note on the corner-point abstraction

It is a very interesting abstraction, that can be used for many applications:

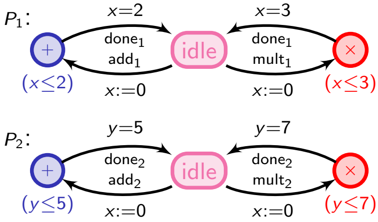
- for mean-cost optimization [BBL04,BBL08]
- for discounted-cost optimization [FL08]
- for all concavely-priced timed automata [JT08]
- for deciding frequency objectives [BBBS11,Sta12]
- ...

[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)*.

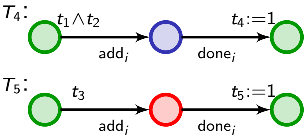
Outline

Modelling the task graph scheduling problem

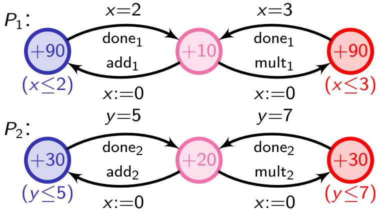
- Processors



- Tasks

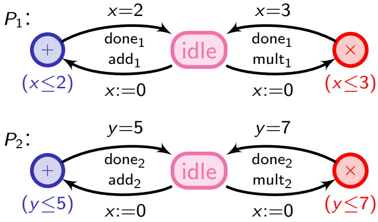


- Modelling energy

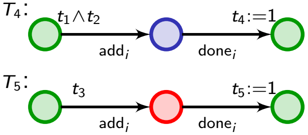


Modelling the task graph scheduling problem

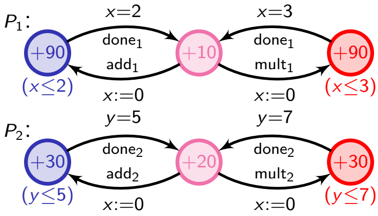
- Processors



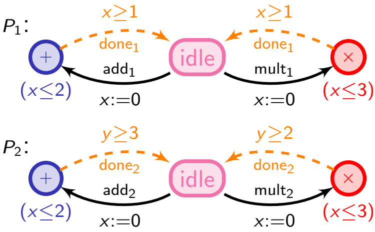
- Tasks



- Modelling energy



- Modelling uncertainty



Modelling the task graph scheduling problem

- Processors

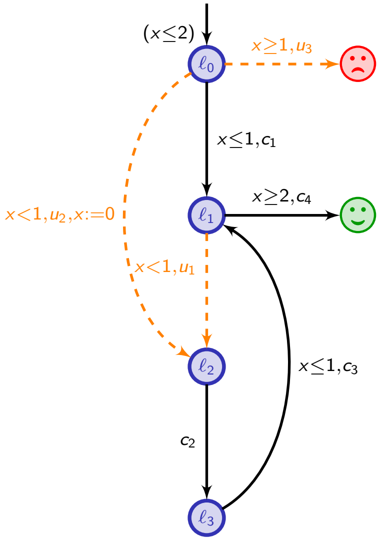
- Tasks

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

- Modelling energy

- Modelling uncertainty

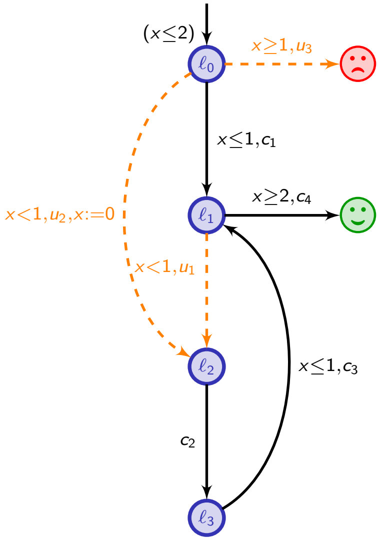
An example of a timed game



Rule of the game

- Aim: avoid ☹ and reach ☺

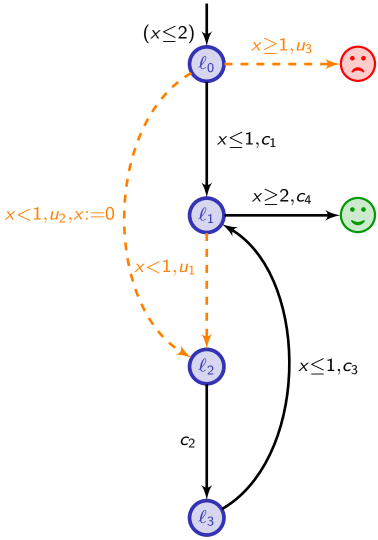
An example of a timed game



Rule of the game

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

An example of a timed game

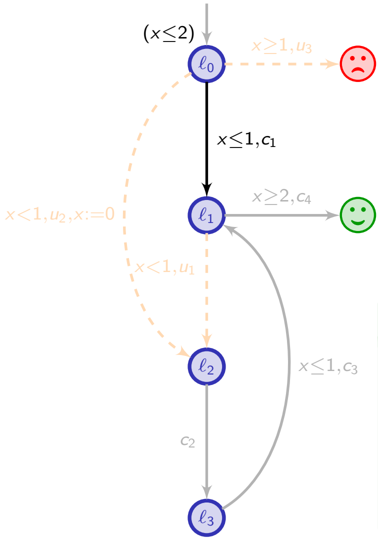


Rule of the game

- 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

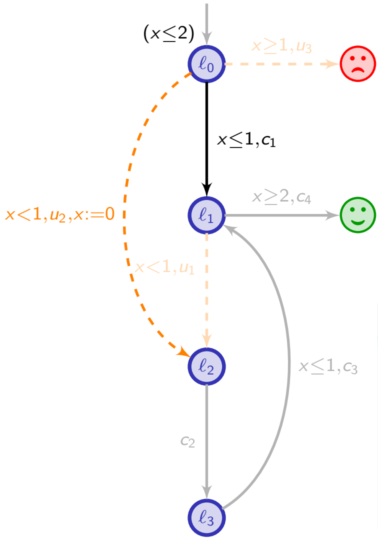
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

A (memoryless) winning strategy

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

An example of a timed game



Rule of the game

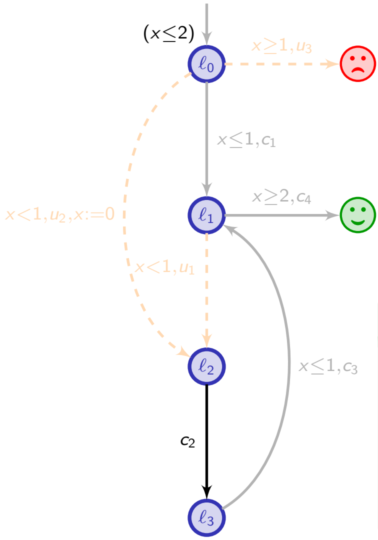
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

A (memoryless) winning strategy

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

An example of a timed game



Rule of the game

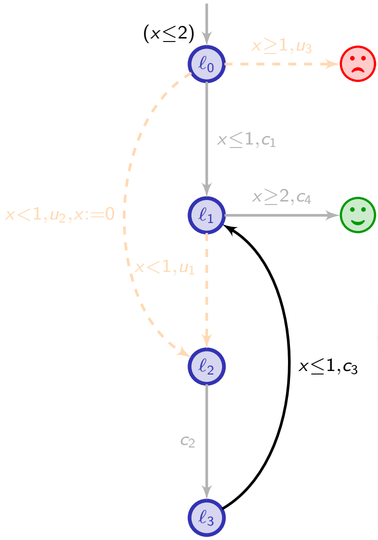
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

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

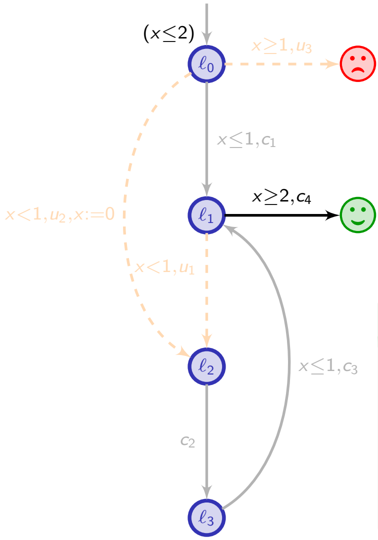
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

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

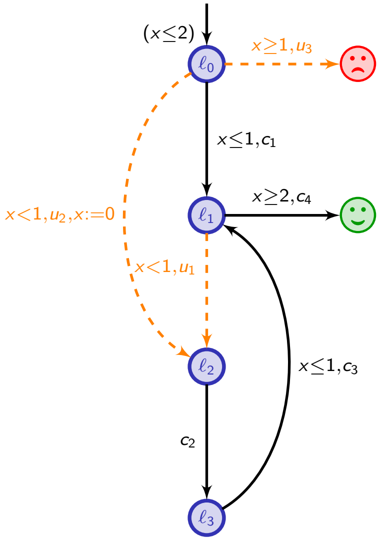
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

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



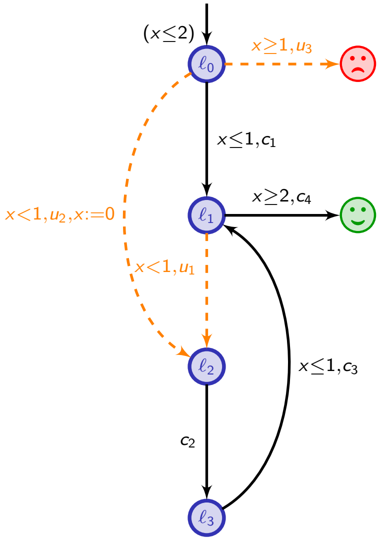
Rule of the game

- 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

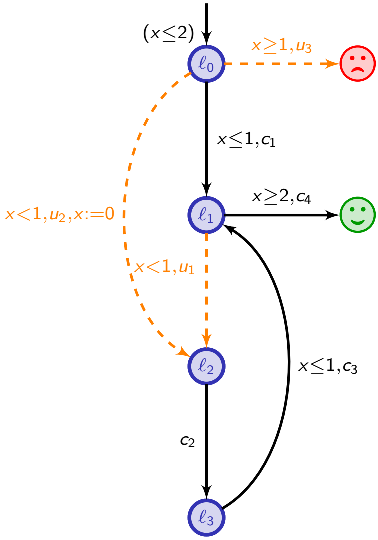
- Aim: avoid ☹ and reach ☺
- How do we play? According to a strategy:

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

Problems to be considered

- Does there exist a winning strategy?

An example of a timed game



Rule of the game

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

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

Problems to be considered

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

Decidability of timed games

Theorem [AMPS98,HK99]

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

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

Decidability of timed games

Theorem [AMPS98,HK99]

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

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

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

Decidability of timed games

Theorem [AMPS98,HK99]

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

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

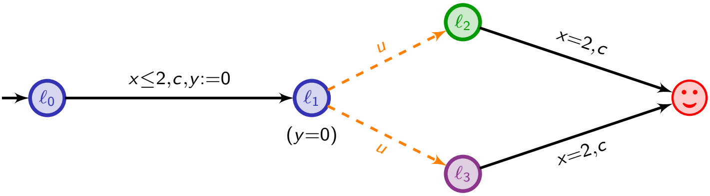
Theorem [AM99,BHPR07,JT07]

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

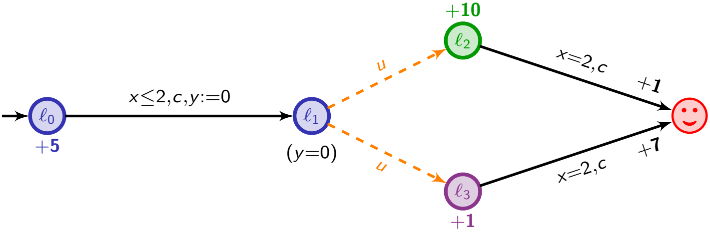
[AM99] Asarin, Maler. As soon as possible: time optimal control for timed automata *(HSCC'99)*.
[BHPR07] Brihaye, Henzinger, Prabhu, Raskin. Minimum-time reachability in timed games *(ICALP'07)*.
[JT07] Jurdziński, Trivedi. Reachability-time games on timed automata *(ICALP'07)*.

Outline

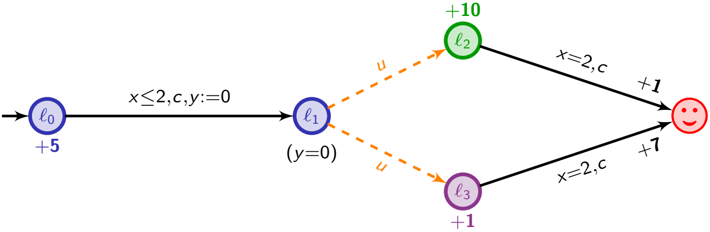
A simple timed game



A simple weighted timed game

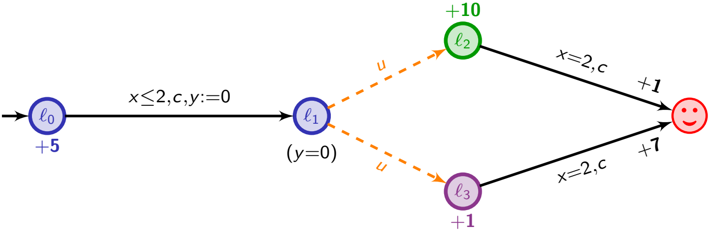


A simple weighted timed game



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

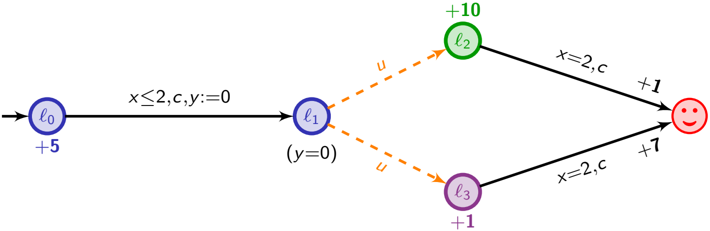
A simple weighted timed game



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

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

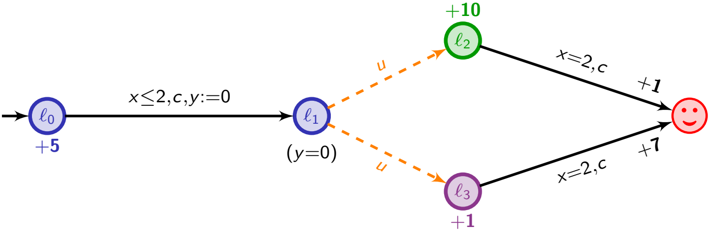
A simple weighted timed game



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

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

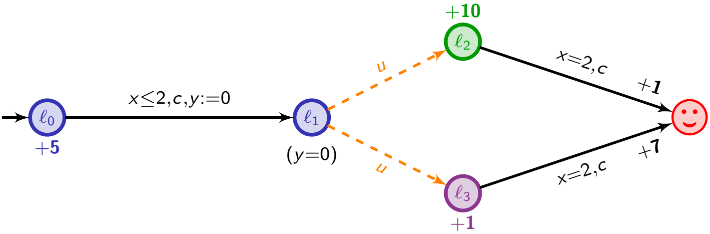
A simple weighted timed game



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

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

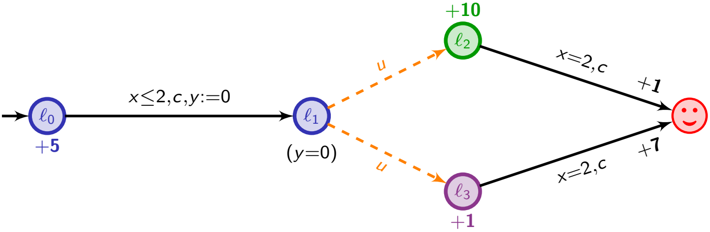
A simple weighted timed game



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...

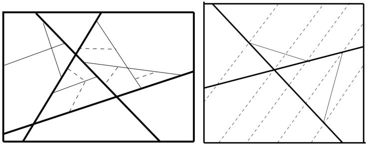
[LMM02,ABM04,BCFL04,BBR05,BBM06,BLMR06,Rut11,HIM13,BGK+14]

[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]

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

Optimal reachability in weighted timed games (2)

[BBR05,BBM06]

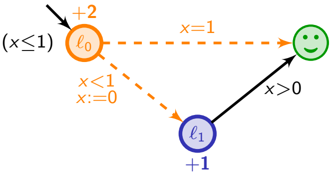
In weighted timed games, the optimal cost cannot be computed, as soon as games have three clocks or more.

[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.

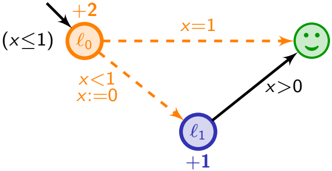
What is easier with a single clock?

- Memoryless strategies can be non-optimal...



What is easier with a single clock?

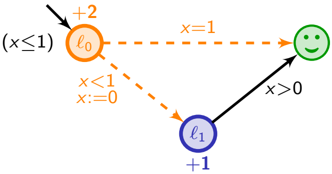
- Memoryless strategies can be non-optimal...



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

What is easier with a single clock?

- Memoryless strategies can be non-optimal...

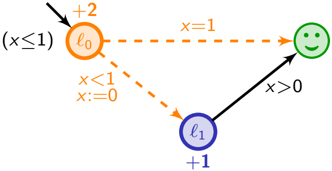


... but memoryless almost-optimal strategies will be sufficient.

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

What is easier with a single clock?

- Memoryless strategies can be non-optimal...

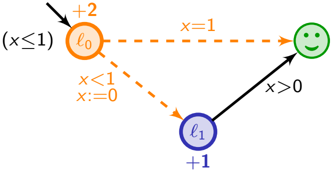


... but memoryless almost-optimal strategies will be sufficient.

- 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.

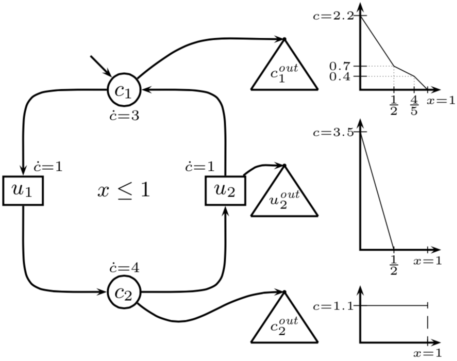
What is easier with a single clock?

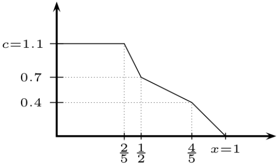
- Memoryless strategies can be non-optimal...



  ... 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





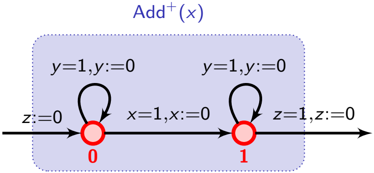
$$\sigma(c_2, x) = \begin{cases} c_2^{out} & \text{if } 0 \leq x < 2/5 \\ c_2 & \text{if } 2/5 \leq x < 1/2 \\ u_2 & \text{if } 1/2 \leq x \leq 1 \end{cases}$$

Computing the optimal cost: why is that hard?

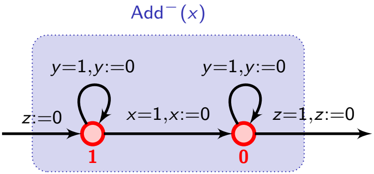
Given two clocks x and y, we can check whether $y = 2x$.

Computing the optimal cost: why is that hard?

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



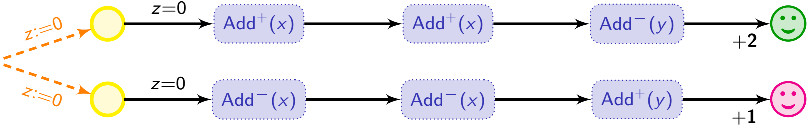
The cost is increased by x_0



The cost is increased by $1-x_0$

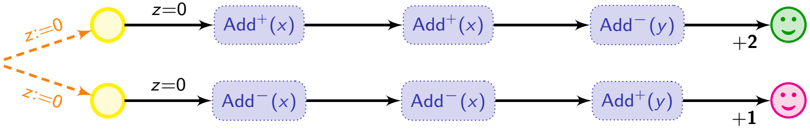
Computing the optimal cost: why is that hard?

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



Computing the optimal cost: why is that hard?

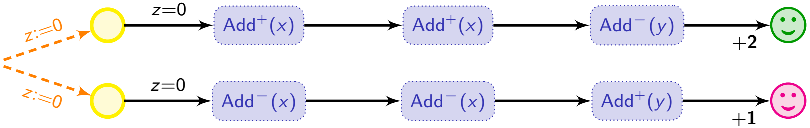
Given two clocks x and y, we can check whether $y = 2x$.



- In ☺, cost $= 2x_0 + (1 - y_0) + 2$

Computing the optimal cost: why is that hard?

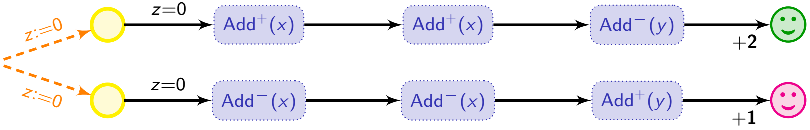
Given two clocks x and y, we can check whether $y = 2x$.



- 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?

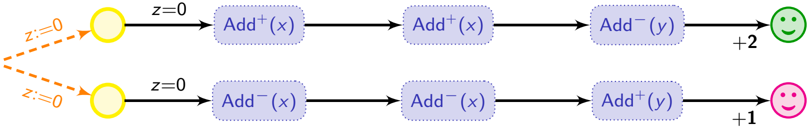
Given two clocks x and y, we can check whether $y = 2x$.



- 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?

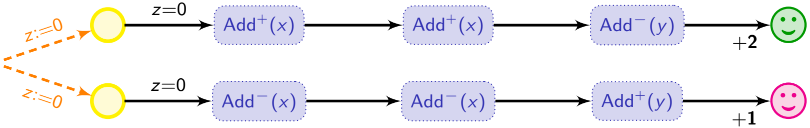
Given two clocks x and y, we can check whether $y = 2x$.



- 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?

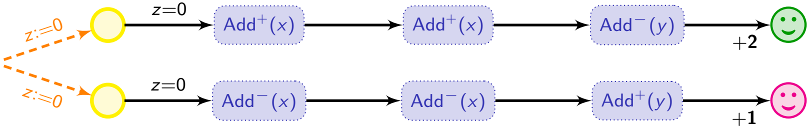
Given two clocks x and y, we can check whether $y = 2x$.



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

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:

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

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:

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

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?

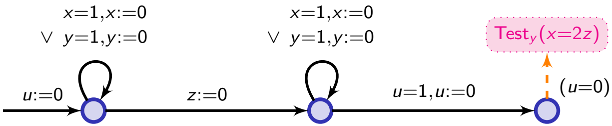
Player 1 will simulate a two-counter machine:

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

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

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

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



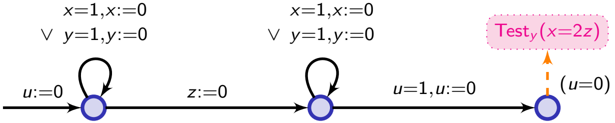
Computing the optimal cost: why is that hard?

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:

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

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



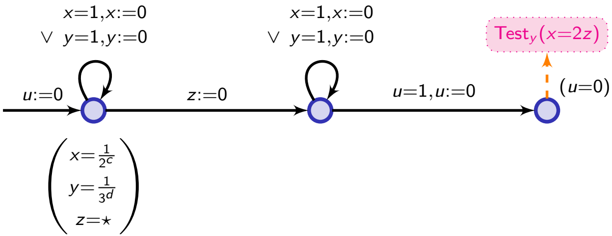
Computing the optimal cost: why is that hard?

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:

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

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



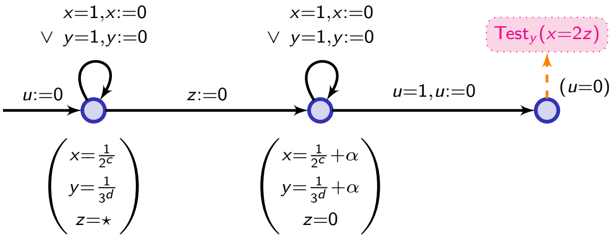
Computing the optimal cost: why is that hard?

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:

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

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



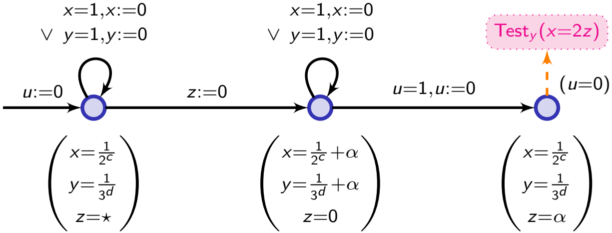
Computing the optimal cost: why is that hard?

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:

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

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



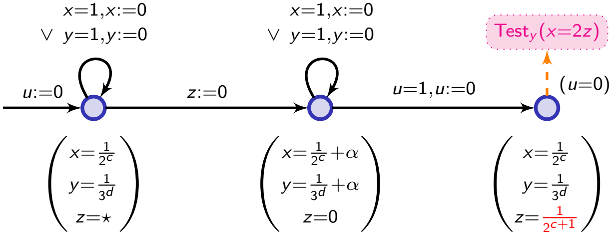
Computing the optimal cost: why is that hard?

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:

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

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



Are we done?

Outline

Are we done?

Are we done? No! Let's be a bit more precise!

Are we done? No! Let's be a bit more precise!

Given $\mathcal{G}$ a weighted timed game,

- a strategy σ is winning whenever all its outcomes are winning;

Are we done? No! Let's be a bit more precise!

Given $\mathcal{G}$ a weighted timed game,

- a strategy σ is winning whenever all its outcomes are winning;
- Cost of a winning strategy σ:

$$\text{cost}(\sigma) = \sup\{\text{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target}\}$$

Are we done? No! Let's be a bit more precise!

Given $\mathcal{G}$ a weighted timed game,

- a strategy σ is winning whenever all its outcomes are winning;
- Cost of a winning strategy σ:

$$\mathsf{cost}(\sigma) = \sup\{\mathsf{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target}\}$$

- Optimal cost:

$$\mathsf{optcost}_{\mathcal{G}} = \inf_{\sigma \text{ winning strat.}} \mathsf{cost}(\sigma)$$

(set it to $+\infty$ if there is no winning strategy)

Are we done? No! Let's be a bit more precise!

Given $\mathcal{G}$ a weighted timed game,

- a strategy σ is winning whenever all its outcomes are winning;
- Cost of a winning strategy σ:

$$\text{cost}(\sigma) = \sup\{\text{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target}\}$$

- Optimal cost:

$$\text{optcost}_{\mathcal{G}} = \inf_{\sigma \text{ winning strat.}} \text{cost}(\sigma)$$

(set it to $+\infty$ if there is no winning strategy)

Two problems of interest

- The value problem asks, given $\mathcal{G}$ and a threshold $\bowtie c$, whether $\text{optcost}_{\mathcal{G}} \bowtie c$?

Are we done? No! Let's be a bit more precise!

Given $\mathcal{G}$ a weighted timed game,

- a strategy σ is winning whenever all its outcomes are winning;
- Cost of a winning strategy σ:

$$\mathsf{cost}(\sigma) = \sup\{\mathsf{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target}\}$$

- Optimal cost:

$$\mathsf{optcost}_{\mathcal{G}} = \inf_{\sigma \text{ winning strat.}} \mathsf{cost}(\sigma)$$

(set it to $+\infty$ if there is no winning strategy)

Two problems of interest

- The value problem asks, given $\mathcal{G}$ and a threshold $\bowtie c$, whether $\mathsf{optcost}_{\mathcal{G}} \bowtie c$?
- The existence problem asks, given $\mathcal{G}$ and a threshold $\bowtie c$, whether there exists a winning strategy in $\mathcal{G}$ such that $\mathsf{cost}(\sigma) \bowtie c$?

Are we done? No! Let's be a bit more precise!

Given $\mathcal{G}$ a weighted timed game,

- a strategy σ is winning whenever all its outcomes are winning;
- Cost of a winning strategy σ:

$$\text{cost}(\sigma) = \sup\{\text{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target}\}$$

- Optimal cost:

$$\text{optcost}_{\mathcal{G}} = \inf_{\sigma \text{ winning strat.}} \text{cost}(\sigma)$$

(set it to $+\infty$ if there is no winning strategy)

Two problems of interest

- The value problem asks, given $\mathcal{G}$ and a threshold $\bowtie c$, whether $\text{optcost}_{\mathcal{G}} \bowtie c$?
- The existence problem asks, given $\mathcal{G}$ and a threshold $\bowtie c$, whether there exists a winning strategy in $\mathcal{G}$ such that $\text{cost}(\sigma) \bowtie c$?

Note: These problems are distinct...

- Weighted timed automata

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

- Weighted timed automata

~~In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.~~
The value problem is PSPACE-complete in weighted timed automata.
Almost-optimal winning schedules can be computed.

- Weighted timed automata

~~In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.~~
The value problem is PSPACE-complete in weighted timed automata.
Almost-optimal winning schedules can be computed.

- Weighted timed games

Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock.

- Weighted timed automata

~~In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.~~
The value problem is PSPACE-complete in weighted timed automata.
Almost-optimal winning schedules can be computed.

- Weighted timed games

~~Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock.~~
The value problem is decidable in EXPTIME in single-clock weighted timed games. Almost-optimal memoryless winning strategies can be computed.

- Weighted timed automata

~~In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.~~
The value problem is PSPACE-complete in weighted timed automata. Almost-optimal winning schedules can be computed.

- Weighted timed games

~~Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock.~~
The value problem is decidable in EXPTIME in single-clock weighted timed games. Almost-optimal memoryless winning strategies can be computed.

There is a symbolic algorithm to solve weighted timed games with a strongly non-Zeno cost.

- Weighted timed automata

~~In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.~~
The value problem is PSPACE-complete in weighted timed automata.
Almost-optimal winning schedules can be computed.

- Weighted timed games

~~Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock.~~
The value problem is decidable in EXPTIME in single-clock weighted timed games. Almost-optimal memoryless winning strategies can be computed.

~~There is a symbolic algorithm to solve weighted timed games with a strongly non-Zeno cost.~~
The value problem can be decided in EXPTIME in weighted timed games with a strongly non-Zeno cost. Almost-optimal winning strategies can be computed.

- Weighted timed automata

~~In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.~~
The value problem is PSPACE-complete in weighted timed automata. Almost-optimal winning schedules can be computed.

- Weighted timed games

~~Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock.~~
The value problem is decidable in EXPTIME in single-clock weighted timed games. Almost-optimal memoryless winning strategies can be computed.

~~There is a symbolic algorithm to solve weighted timed games with a strongly non-Zeno cost.~~
The value problem can be decided in EXPTIME in weighted timed games with a strongly non-Zeno cost. Almost-optimal winning strategies can be computed.

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

- Weighted timed automata

~~In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.~~
The value problem is PSPACE-complete in weighted timed automata.
Almost-optimal winning schedules can be computed.

- Weighted timed games

~~Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock.~~
The value problem is decidable in EXPTIME in single-clock weighted timed games. Almost-optimal memoryless winning strategies can be computed.

~~There is a symbolic algorithm to solve weighted timed games with a strongly non-Zeno cost.~~
The value problem can be decided in EXPTIME in weighted timed games with a strongly non-Zeno cost. Almost-optimal winning strategies can be computed.

~~In weighted timed games, the optimal cost cannot be computed, as soon as games have three clocks or more.~~
The existence problem is undecidable in weighted timed games.

Outline of the rest of the talk

1. Show that the value problem is undecidable in weighted timed games

Outline of the rest of the talk

1. Show that the value problem is undecidable in weighted timed games
   - $\rightsquigarrow$ This is intellectually satisfactory to not have this discrepancy in the set of results

Outline of the rest of the talk

1. Show that the value problem is undecidable in weighted timed games
   - $\rightsquigarrow$ This is intellectually satisfactory to not have this discrepancy in the set of results
   - $\rightsquigarrow$ An original undecidability proof, based on a diagonal construction
     - This method has been introduced in the context of quantitative temporal logics [BMM14]
     - It might be useful in some different contexts

[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL *(CONCUR'14)*.

Outline of the rest of the talk

1. Show that the value problem is undecidable in weighted timed games
   - $\rightsquigarrow$ This is intellectually satisfactory to not have this discrepancy in the set of results
   - $\rightsquigarrow$ An original undecidability proof, based on a diagonal construction
     - This method has been introduced in the context of quantitative temporal logics [BMM14]
     - It might be useful in some different contexts

2. Propose an approximation algorithm for a large class of weighted timed games (that comprises the class of games used for proving the above undecidability)
   - 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...

[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL *(CONCUR'14)*.

Outline

1. Introduction
2. Overview of “old” results
    - Weighted timed automata
    - Timed games
    - Weighted timed games
3. Some recent developments
    - Undecidability of the value problem
    - Approximation of the optimal cost
4. Conclusion

What about the value problem?

$\mathcal{M} \rightsquigarrow$ the previous game $\mathcal{G}_{\mathcal{M}}$

What about the value problem?

$\mathcal{M} \rightsquigarrow$ the previous game $\mathcal{G}_\mathcal{M}$

It is always the case that $\text{optcost}_{\mathcal{G}_\mathcal{M}} \geq 3$

What about the value problem?

$\mathcal{M} \rightsquigarrow$ the previous game $\mathcal{G}_{\mathcal{M}}$

It is always the case that $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} \geq 3$

- If $\mathcal{M}$ halts, then $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$

What about the value problem?

$\mathcal{M} \leadsto$ the previous game $\mathcal{G}_{\mathcal{M}}$

It is always the case that $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} \geq 3$

- If $\mathcal{M}$ halts, then $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$
- It might be the case that $\mathcal{M}$ does not halt but $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$

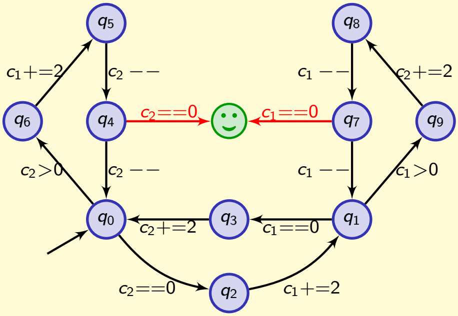
What about the value problem?

$\mathcal{M} \rightsquigarrow$ the previous game $\mathcal{G}_{\mathcal{M}}$

It is always the case that $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} \geq 3$

- If $\mathcal{M}$ halts, then $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$
- It might be the case that $\mathcal{M}$ does not halt but $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$

$\mathcal{M}$ does not halt but $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$



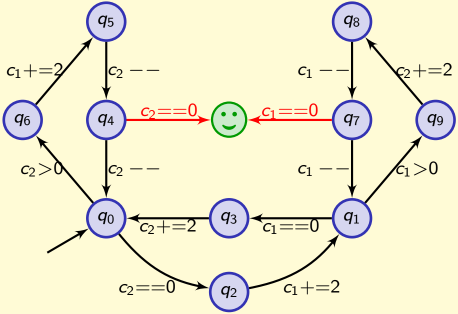
What about the value problem?

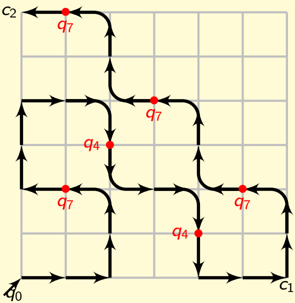
$\mathcal{M} \rightsquigarrow$ the previous game $\mathcal{G}_{\mathcal{M}}$

It is always the case that $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} \geq 3$

- If $\mathcal{M}$ halts, then $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$
- It might be the case that $\mathcal{M}$ does not halt but $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$

$\mathcal{M}$ does not halt but $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$





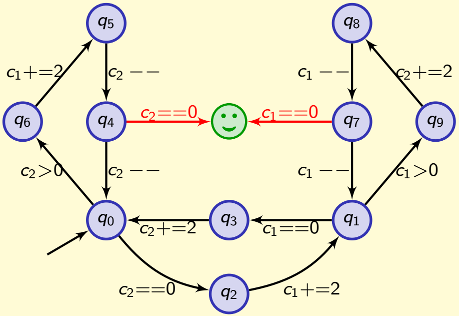
What about the value problem?

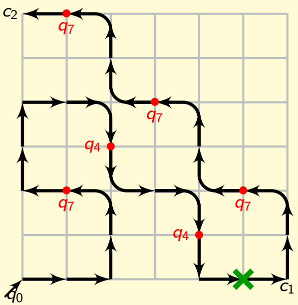
$\mathcal{M} \rightsquigarrow$ the previous game $\mathcal{G}_{\mathcal{M}}$

It is always the case that $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}}} \geq 3$

- If $\mathcal{M}$ halts, then $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$
- It might be the case that $\mathcal{M}$ does not halt but $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$

$\mathcal{M}$ does not halt but $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$





- We need to be able to distinguish between machines that halt and machines that have a converging phenomenon

- We need to be able to distinguish between machines that halt and machines that have a converging phenomenon
- We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14]

[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL *(CONCUR'14)*.

- We need to be able to distinguish between machines that halt and machines that have a converging phenomenon
- We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14]

Technical lemma

We assume two halting states: accept and reject.

[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL *(CONCUR'14)*.

- We need to be able to distinguish between machines that halt and machines that have a converging phenomenon
- We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14]

Technical lemma

We assume two halting states: accept and reject.
If $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$ but no strategy has cost 3

[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL *(CONCUR'14)*.

- We need to be able to distinguish between machines that halt and machines that have a converging phenomenon
- We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14]

Technical lemma

We assume two halting states: accept and reject.
If $\mathsf{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$ but no strategy has cost 3 (or equivalently, the unique valid run of $\mathcal{M}$ is not accepting),

[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL *(CONCUR'14)*.

- We need to be able to distinguish between machines that halt and machines that have a converging phenomenon
- We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14]

Technical lemma

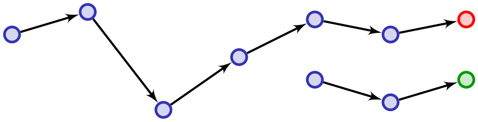
We assume two halting states: accept and reject.
If $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$ but no strategy has cost 3 (or equivalently, the unique valid run of $\mathcal{M}$ is not accepting), then the unique valid run of $\mathcal{M}$ is infinite.

[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL *(CONCUR'14)*.

- We need to be able to distinguish between machines that halt and machines that have a converging phenomenon
- We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14]

Technical lemma

We assume two halting states: accept and reject.
If $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$ but no strategy has cost 3 (or equivalently, the unique valid run of $\mathcal{M}$ is not accepting), then the unique valid run of $\mathcal{M}$ is infinite.

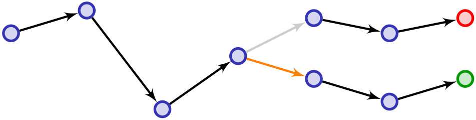


[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL *(CONCUR'14)*.

- We need to be able to distinguish between machines that halt and machines that have a converging phenomenon
- We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14]

Technical lemma

We assume two halting states: accept and reject.
If $\text{optcost}_{\mathcal{G}_{\mathcal{M}}} = 3$ but no strategy has cost 3 (or equivalently, the unique valid run of $\mathcal{M}$ is not accepting), then the unique valid run of $\mathcal{M}$ is infinite.



big impact on the cost!

[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL *(CONCUR'14)*.

The diagonal argument

$$\begin{array}{ll} & B \text{ det. Turing machine can either accept, reject, or not halt} \\ \rightarrow & \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \end{array}$$

The diagonal argument

$$\begin{array}{rl} & B \text{ det. Turing machine can either accept, reject, or not halt} \\ \rightarrow & \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \end{array}$$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The diagonal argument

B det. Turing machine can either accept, reject, or not halt
$\rightarrow$ $\mathcal{M}(B)$ two-counter machine which simulates B on B

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

The diagonal argument

B det. Turing machine can either accept, reject, or not halt

$\rightarrow$ $\mathcal{M}(B)$ two-counter machine which simulates B on B

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} accept & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ reject & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$,

The diagonal argument

B det. Turing machine can either accept, reject, or not halt
$\rightarrow$ $\mathcal{M}(B)$ two-counter machine which simulates B on B

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} accept & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ reject & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$\mathcal{C}(B)$: Simulate $\mathcal{T}_{\mathcal{H}}$ on B;
If $\mathcal{T}_{\mathcal{H}}$ accepts B then reject, otherwise accept

The diagonal argument

$$\begin{array}{rl} & B \text{ det. Turing machine can either accept, reject, or not halt} \\ \to & \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \end{array}$$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$$\begin{array}{ll} \mathcal{C}(B): & \text{Simulate } \mathcal{T}_{\mathcal{H}} \text{ on } B; \\ & \text{If } \mathcal{T}_{\mathcal{H}} \text{ accepts } B \text{ then reject, otherwise accept} \end{array}$$

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$

The diagonal argument

$$\begin{array}{ll} & B \text{ det. Turing machine can either accept, reject, or not halt} \\ \rightarrow & \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \end{array}$$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_\mathcal{H}$, and define the program:

$$\mathcal{C}(B): \begin{array}{l} \text{Simulate } \mathcal{T}_\mathcal{H} \text{ on } B; \\ \text{If } \mathcal{T}_\mathcal{H} \text{ accepts } B \text{ then reject, otherwise accept} \end{array}$$

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$:

The diagonal argument

B det. Turing machine can either accept, reject, or not halt
$\rightarrow$ $\mathcal{M}(B)$ two-counter machine which simulates B on B

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$\mathcal{C}(B)$: Simulate $\mathcal{T}_{\mathcal{H}}$ on B;
If $\mathcal{T}_{\mathcal{H}}$ accepts B then reject, otherwise accept

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$: this means that $\mathcal{H}(\mathcal{C}) = \textit{reject}$,

The diagonal argument

B det. Turing machine can either accept, reject, or not halt
$\rightarrow$ $\mathcal{M}(B)$ two-counter machine which simulates B on B

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$\mathcal{C}(B)$: Simulate $\mathcal{T}_{\mathcal{H}}$ on B;
If $\mathcal{T}_{\mathcal{H}}$ accepts B then reject, otherwise accept

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$: this means that $\mathcal{H}(\mathcal{C}) = \textit{reject}$, hence $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} > 3$.

The diagonal argument

$$\begin{array}{rl} & B \text{ det. Turing machine can either accept, reject, or not halt} \\ \rightarrow & \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \end{array}$$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$$\begin{array}{rl} \mathcal{C}(B): & \text{Simulate } \mathcal{T}_{\mathcal{H}} \text{ on } B; \\ & \text{If } \mathcal{T}_{\mathcal{H}} \text{ accepts } B \text{ then reject, otherwise accept} \end{array}$$

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$: this means that $\mathcal{H}(\mathcal{C}) = \textit{reject}$, hence $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} > 3$.
  This implies $\mathcal{M}(\mathcal{C})$ does not accept,

The diagonal argument

B det. Turing machine can either accept, reject, or not halt

$\rightarrow$ $\mathcal{M}(B)$ two-counter machine which simulates B on B

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$\mathcal{C}(B)$: Simulate $\mathcal{T}_{\mathcal{H}}$ on B;
If $\mathcal{T}_{\mathcal{H}}$ accepts B then reject, otherwise accept

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$: this means that $\mathcal{H}(\mathcal{C}) = \textit{reject}$, hence $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} > 3$. This implies $\mathcal{M}(\mathcal{C})$ does not accept, and therefore $\mathcal{C}$ does not accept $\mathcal{C}$,

The diagonal argument

$$\begin{array}{rl} & B \text{ det. Turing machine can either accept, reject, or not halt} \\ \rightarrow & \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \end{array}$$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$$\begin{array}{rl} \mathcal{C}(B): & \text{Simulate } \mathcal{T}_{\mathcal{H}} \text{ on } B; \\ & \text{If } \mathcal{T}_{\mathcal{H}} \text{ accepts } B \text{ then reject, otherwise accept} \end{array}$$

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$: this means that $\mathcal{H}(\mathcal{C}) = \textit{reject}$, hence $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} > 3$. This implies $\mathcal{M}(\mathcal{C})$ does not accept, and therefore $\mathcal{C}$ does not accept $\mathcal{C}$, contradiction: $\mathcal{C}$ rejects $\mathcal{C}$.

The diagonal argument

$$\begin{array}{ll} & B \text{ det. Turing machine can either accept, reject, or not halt} \\ \rightarrow & \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \end{array}$$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$$\begin{array}{ll} \mathcal{C}(B): & \text{Simulate } \mathcal{T}_{\mathcal{H}} \text{ on } B; \\ & \text{If } \mathcal{T}_{\mathcal{H}} \text{ accepts } B \text{ then reject, otherwise accept} \end{array}$$

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$: this means that $\mathcal{H}(\mathcal{C}) = \textit{reject}$, hence $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} > 3$. This implies $\mathcal{M}(\mathcal{C})$ does not accept, and therefore $\mathcal{C}$ does not accept $\mathcal{C}$, contradiction: $\mathcal{C}$ rejects $\mathcal{C}$.
- $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} = 3$.

The diagonal argument

$$\begin{array}{rl} & B \text{ det. Turing machine can either accept, reject, or not halt} \\ \to & \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \end{array}$$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$$\begin{array}{rl} \mathcal{C}(B) : & \text{Simulate } \mathcal{T}_{\mathcal{H}} \text{ on } B; \\ & \text{If } \mathcal{T}_{\mathcal{H}} \text{ accepts } B \text{ then reject, otherwise accept} \end{array}$$

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$: this means that $\mathcal{H}(\mathcal{C}) = \textit{reject}$, hence $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} > 3$. This implies $\mathcal{M}(\mathcal{C})$ does not accept, and therefore $\mathcal{C}$ does not accept $\mathcal{C}$, contradiction: $\mathcal{C}$ rejects $\mathcal{C}$.
- $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} = 3$. Since $\mathcal{C}$ does not accept $\mathcal{C}$, the unique valid computation of $\mathcal{M}(\mathcal{C})$ is either infinite or rejecting.

The diagonal argument

$$\begin{array}{rl} & B \text{ det. Turing machine can either accept, reject, or not halt} \\ \to & \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \end{array}$$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$$\begin{array}{rl} \mathcal{C}(B): & \text{Simulate } \mathcal{T}_{\mathcal{H}} \text{ on } B; \\ & \text{If } \mathcal{T}_{\mathcal{H}} \text{ accepts } B \text{ then reject, otherwise accept} \end{array}$$

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$: this means that $\mathcal{H}(\mathcal{C}) = \textit{reject}$, hence $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} > 3$. This implies $\mathcal{M}(\mathcal{C})$ does not accept, and therefore $\mathcal{C}$ does not accept $\mathcal{C}$, contradiction: $\mathcal{C}$ rejects $\mathcal{C}$.
- $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} = 3$. Since $\mathcal{C}$ does not accept $\mathcal{C}$, the unique valid computation of $\mathcal{M}(\mathcal{C})$ is either infinite or rejecting. Applying the lemma on previous slide, it is infinite,

The diagonal argument

$$\begin{array}{ll} & B \text{ det. Turing machine can either accept, reject, or not halt} \\ \rightarrow & \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \end{array}$$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$$\begin{array}{ll} \mathcal{C}(B): & \text{Simulate } \mathcal{T}_{\mathcal{H}} \text{ on } B; \\ & \text{If } \mathcal{T}_{\mathcal{H}} \text{ accepts } B \text{ then reject, otherwise accept} \end{array}$$

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$: this means that $\mathcal{H}(\mathcal{C}) = \textit{reject}$, hence $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} > 3$. This implies $\mathcal{M}(\mathcal{C})$ does not accept, and therefore $\mathcal{C}$ does not accept $\mathcal{C}$, contradiction: $\mathcal{C}$ rejects $\mathcal{C}$.
- $\mathrm{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} = 3$. Since $\mathcal{C}$ does not accept $\mathcal{C}$, the unique valid computation of $\mathcal{M}(\mathcal{C})$ is either infinite or rejecting. Applying the lemma on previous slide, it is infinite, which contradicts the fact that $\mathcal{C}$ always terminates.

The diagonal argument

B det. Turing machine can either accept, reject, or not halt
$\rightarrow$ $\mathcal{M}(B)$ two-counter machine which simulates B on B

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} \textit{accept} & \text{if } \mathsf{optcost}_{\mathcal{G}_{\mathcal{M}(B)}} = 3 \\ \textit{reject} & \text{otherwise} \end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $\mathcal{T}_{\mathcal{H}}$, and define the program:

$\mathcal{C}(B)$: Simulate $\mathcal{T}_{\mathcal{H}}$ on B;
If $\mathcal{T}_{\mathcal{H}}$ accepts B then reject, otherwise accept

Program $\mathcal{C}$ is deterministic hence we can run $\mathcal{C}$ on $\mathcal{C}$
Program $\mathcal{C}$ always terminates:

- Assume $\mathcal{C}$ accepts $\mathcal{C}$: this means that $\mathcal{H}(\mathcal{C}) = \textit{reject}$, hence $\mathsf{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} > 3$. This implies $\mathcal{M}(\mathcal{C})$ does not accept, and therefore $\mathcal{C}$ does not accept $\mathcal{C}$, contradiction: $\mathcal{C}$ rejects $\mathcal{C}$.
- $\mathsf{optcost}_{\mathcal{G}_{\mathcal{M}(\mathcal{C})}} = 3$. Since $\mathcal{C}$ does not accept $\mathcal{C}$, the unique valid computation of $\mathcal{M}(\mathcal{C})$ is either infinite or rejecting. Applying the lemma on previous slide, it is infinite, which contradicts the fact that $\mathcal{C}$ always terminates.

Therefore, $\mathcal{H}$ is not computable.

Theorem [BJM15]

The value problem is undecidable in weighted timed games (with four clocks or more).

- Remark on the reduction:
  - Cost 0 within the core of the game
  - The rest of the game is acyclic

[BJM15] Bouyer, Jaziri, Markey. On the value problem in weighted timed games.

Outline

Optimal cost is computable...

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

That is, there exists $\kappa > 0$ such that 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. [BJM15]

That is, there exists $\kappa > 0$ such that for every region cycle C, for every real run ϱ read on C,

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

Note: In both cases, we can assume $\kappa = 1$.

[BJM15] Bouyer, Jaziri, Markey. On the value problem in weighted timed games.

Optimal cost is computable...

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

That is, there exists $\kappa > 0$ such that for every region cycle C, for every real run ϱ read on C,

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

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

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

That is, there exists $\kappa > 0$ such that for every region cycle C, for every real run ϱ read on C,

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

Note: In both cases, we can assume $\kappa = 1$.

[BJM15] Bouyer, Jaziri, Markey. On the value problem in weighted timed games.

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]

- 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

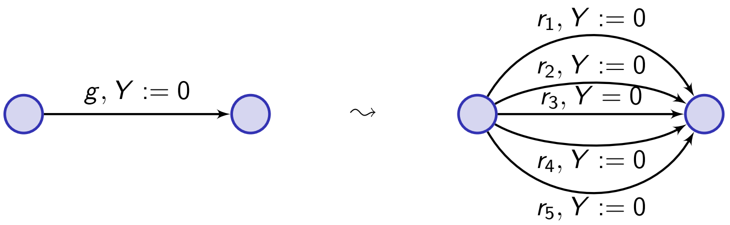
Idea for approximation

Idea

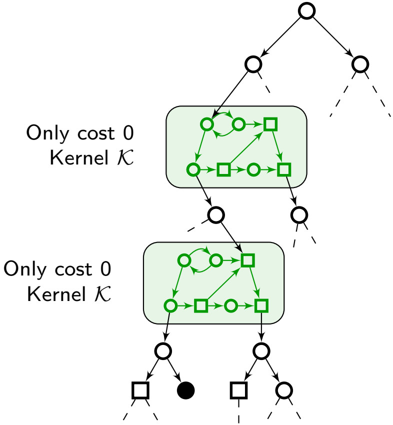
Only partially unfold the game:

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

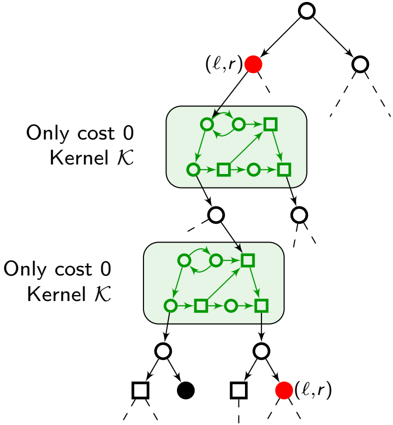
First: split the game along regions!



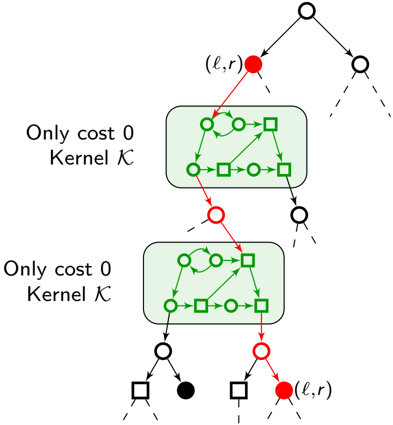
Semi-unfolding



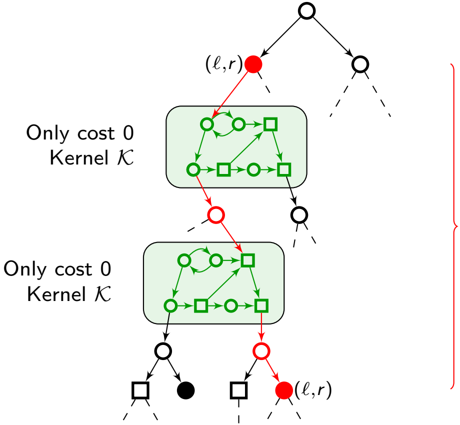
Semi-unfolding



Semi-unfolding

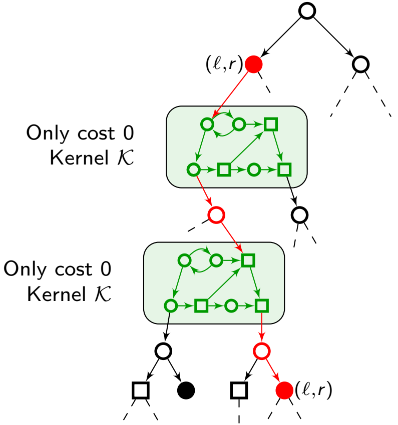


Semi-unfolding



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

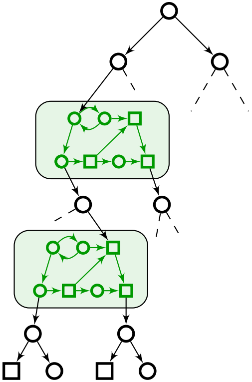
Semi-unfolding



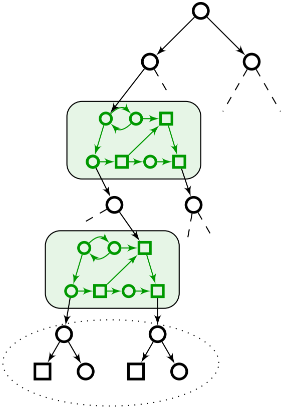
Hypothesis:
cost > 0 implies cost $\geq \kappa$

Conclusion: we can stop unfolding the game after N steps
(e.g. $N = (M+2) \cdot |\mathcal{R}(\mathcal{A})|$, where M is a pre-computed bound on $\mathrm{optcost}_{\mathcal{G}}$)

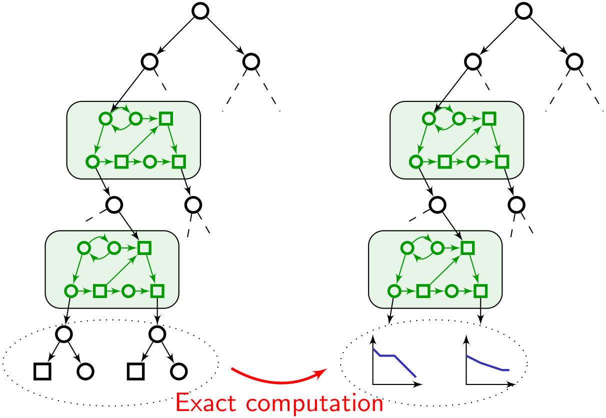
Approximation scheme



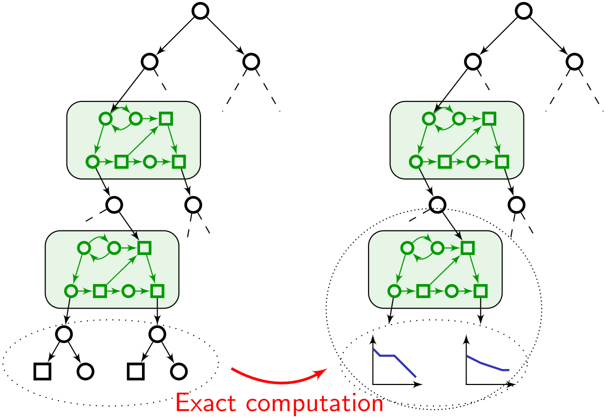
Approximation scheme



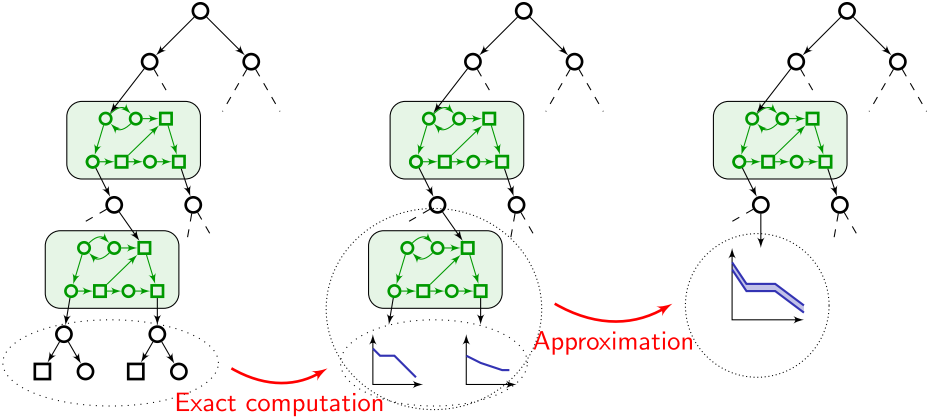
Approximation scheme



Approximation scheme



Approximation scheme



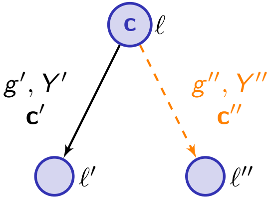
First step: Tree-like parts

$\rightsquigarrow$ 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

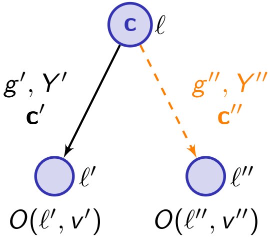
$\rightsquigarrow$ Goes back to [LMM02]



[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]

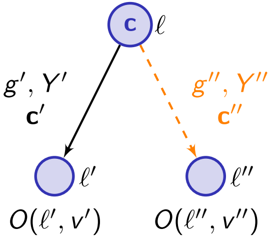


$O(\ell, v) =$

[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]

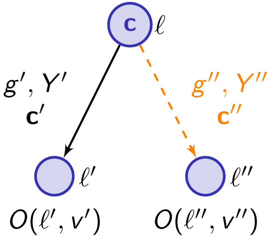


$$O(\ell, v) = \inf_{t' \mid v+t' \models g'}$$

[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]

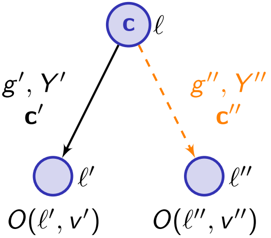


$$O(\ell, v) = \inf_{t' \mid v+t' \models g'} \max(\quad , \quad)$$

[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]



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

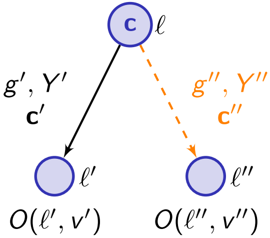
$$(\alpha) = t'\mathbf{c} + \mathbf{c}' + O(\ell', v')$$

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

[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]



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

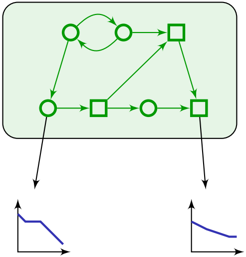
$$(\alpha) = t'\mathbf{c} + \mathbf{c}' + O(\ell', v')$$

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

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

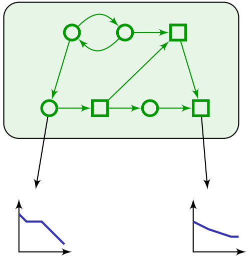
[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata *(TCS@02)*.

Second step: Kernels



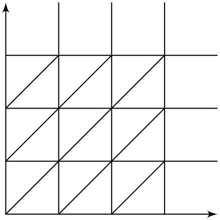
Output cost functions f

Second step: Kernels

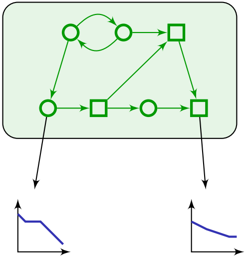


Output cost functions f

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

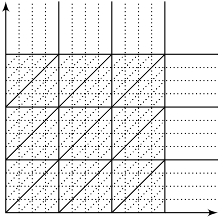


Second step: Kernels

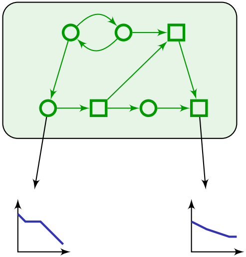


Output cost functions f

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

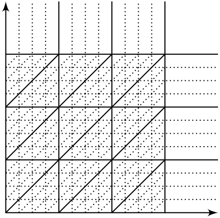


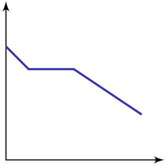
Second step: Kernels



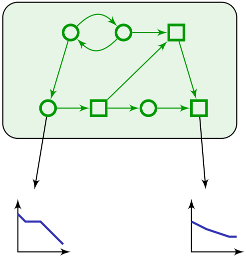
Output cost functions f

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



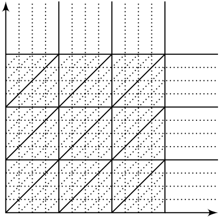


Second step: Kernels

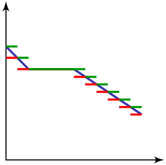


Output cost functions f

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

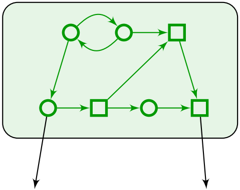


2. Under- and over-approximate by piecewise constant functions f_ϵ^- and f_ϵ^+



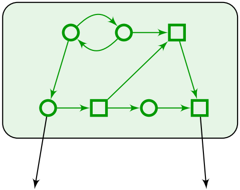
Second step: Kernels

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



f_ϵ: constant f_ϵ: constant

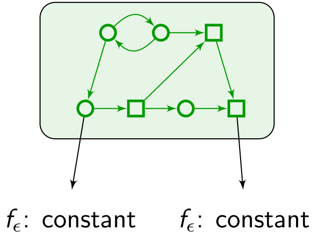
Second step: Kernels



f_ϵ: constant f_ϵ: constant

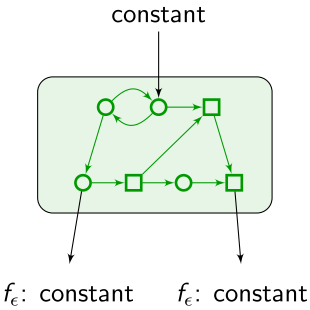
3. Refine/split the kernel along the new small regions and fix f_ϵ^- or f_ϵ^+, write f_ϵ
4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by f_ϵ)

Second step: Kernels



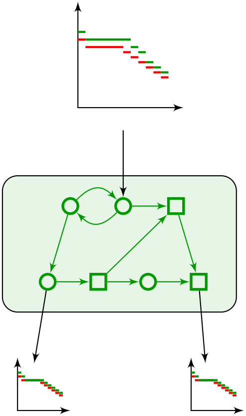
3. Refine/split the kernel along the new small regions and fix f_ϵ^- or f_ϵ^+, write f_ϵ
4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by f_ϵ)
5. Those can be solved using standard technics based on attractors: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

1. Introduction
2. Overview of “old” results
   - Weighted timed automata
   - Timed games
   - Weighted timed games
3. Some recent developments
   - Undecidability of the value problem
   - Approximation of the optimal cost
4. Conclusion

Conclusion

Summary of the talk

- Quick overview of results concerning the optimal reachability problem in weighted timed games
- New insight into the value problem for this model:
  - Undecidability of this problem
  - Approximability of the optimal cost (under some conditions)

Conclusion

Summary of the talk

- Quick overview of results concerning the optimal reachability problem in weighted timed games
- New insight into the value problem for this model:
  - Undecidability of this problem
  - Approximability of the optimal cost (under some conditions)

Future work

- Improve the approximation scheme ($2\mathrm{EXP}(|\mathcal{G}|) \cdot \left(1/\epsilon\right)^{|X|}$), and implement it
- Extend to the whole class of weighted timed games, or understand why it is not possible
- Assume stochastic uncertainty