Robustness in Timed Systems

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France

Acknowledgment to Nicolas Markey and Ocan Sankur for slides

Outline

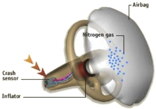
Time-dependent systems

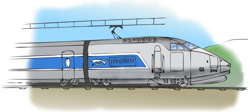
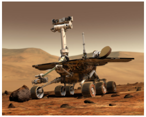
- We are interested in timed systems

Time-dependent systems

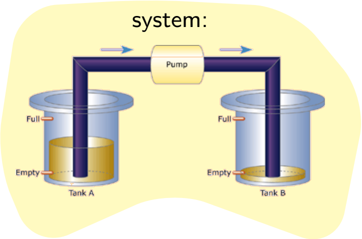
- We are interested in timed systems

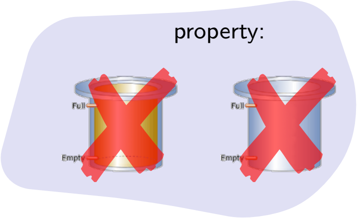




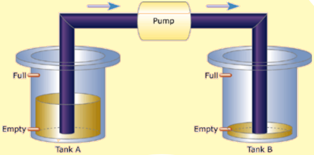
Model-checking and control



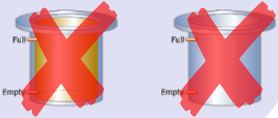


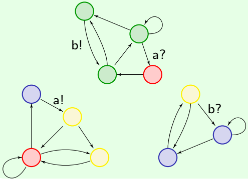
Model-checking and control

system:



property:



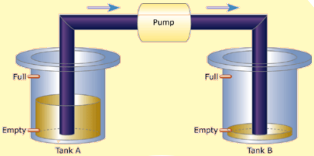


$\mathbf{AG}(\neg\texttt{B.overfull}$

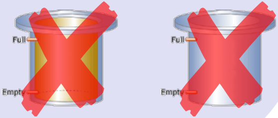
$\quad\wedge\ \neg\texttt{B.dried\_up})$

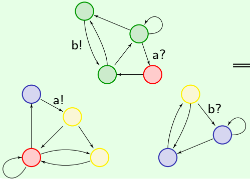
Model-checking and control

system:



property:





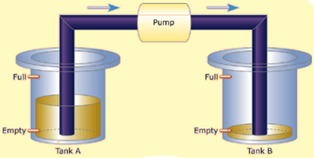
$\Longrightarrow$ algorithm $\Longleftarrow$

$\mathbf{AG}(\neg\texttt{B.overfull}$

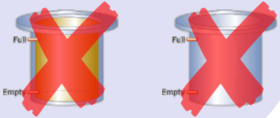
$\wedge\ \neg\texttt{B.dried\_up})$

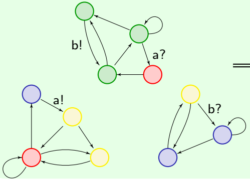
Model-checking and control

system:



property:





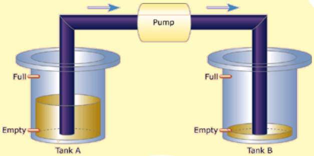
model-checking algorithm

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

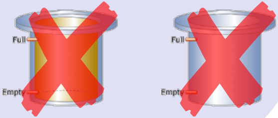
yes/no

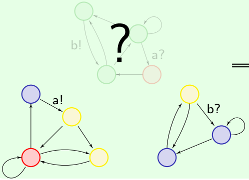
Model-checking and control

system:



property:

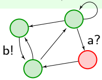




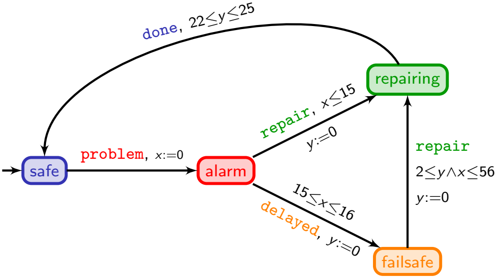
control/synthesis algorithm

AG($\neg$`B.overfull`

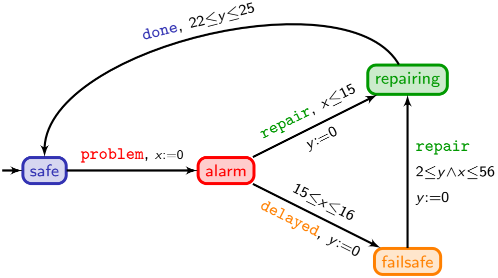
$\wedge$ $\neg$`B.dried_up`)



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	$\cdots$
y	0		23		23		38.6		0	

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

Discrete-time semantics

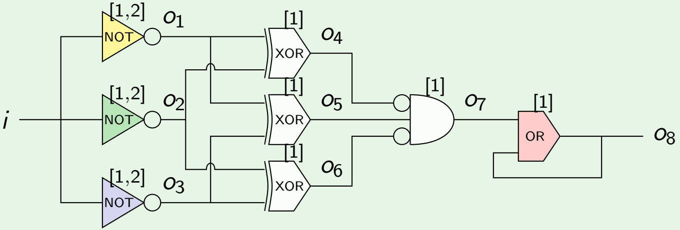
...because computers are digital!

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

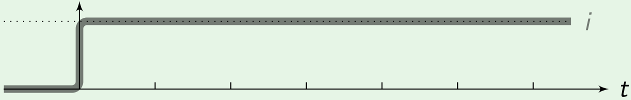
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

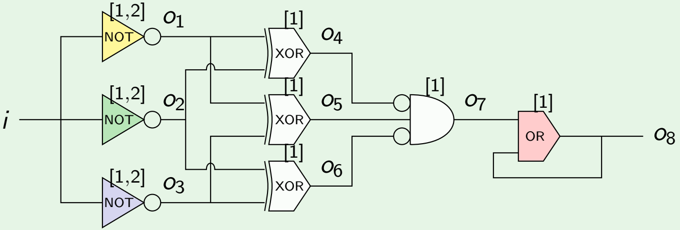


[Alur91] Techniques for automatic verification of real-time systems *(PhD thesis, 1991)*.

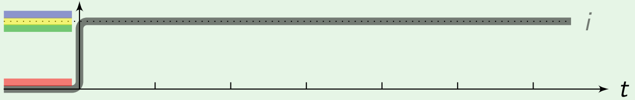
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

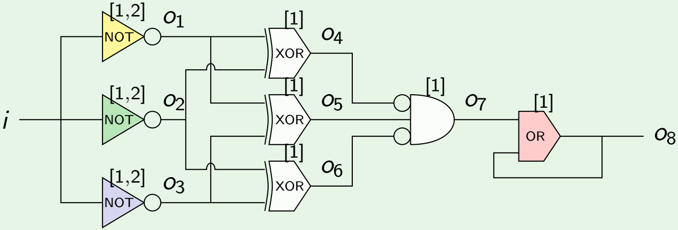


[Alur91] Techniques for automatic verification of real-time systems *(PhD thesis, 1991)*.

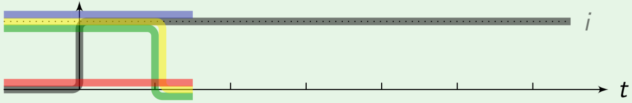
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

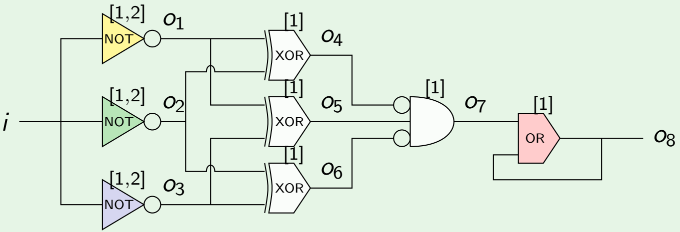


[Alur91] Techniques for automatic verification of real-time systems *(PhD thesis, 1991)*.

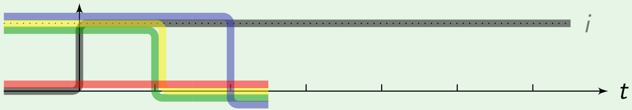
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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

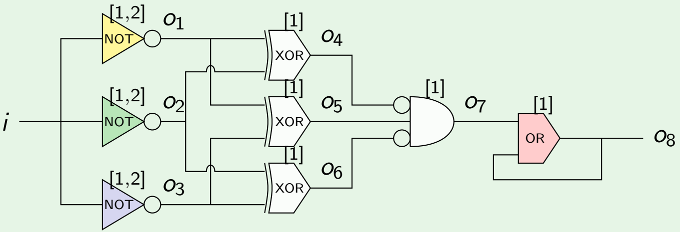


[Alur91] Techniques for automatic verification of real-time systems *(PhD thesis, 1991)*.

Discrete-time semantics

...because computers are digital!

Example [Alur91]



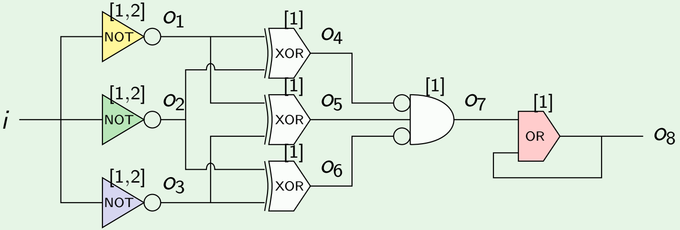
- under discrete-time, the output is always 0:

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

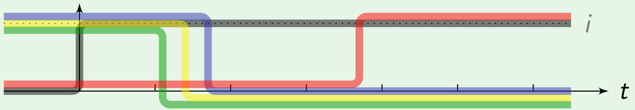
Discrete-time semantics

...because computers are digital!

Example [Alur91]



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



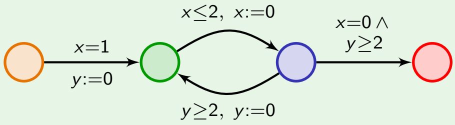
[Alur91] Techniques for automatic verification of real-time systems *(PhD thesis, 1991)*.

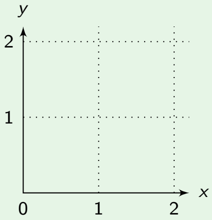
Continuous-time semantics

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

Continuous-time semantics

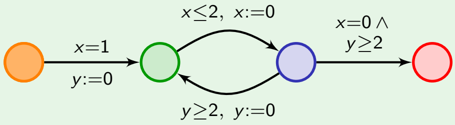
...real-time models for real-time systems!

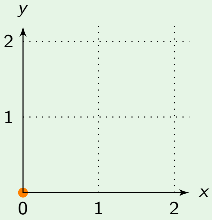




Continuous-time semantics

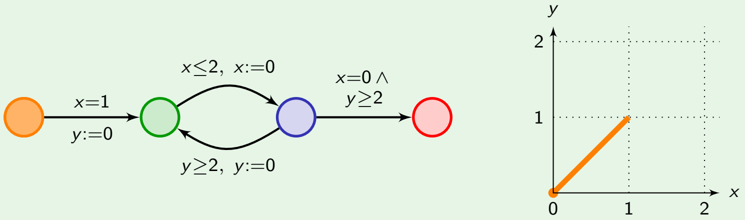
...real-time models for real-time systems!





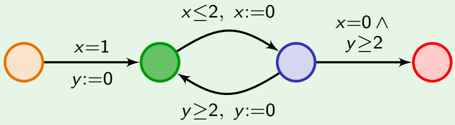
Continuous-time semantics

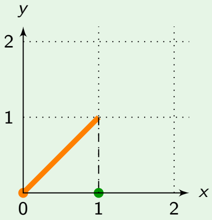
...real-time models for real-time systems!



Continuous-time semantics

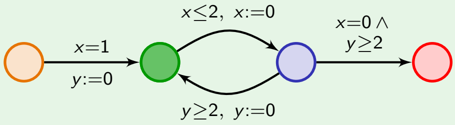
...real-time models for real-time systems!

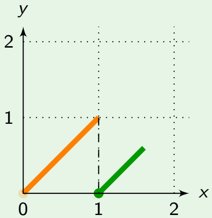




Continuous-time semantics

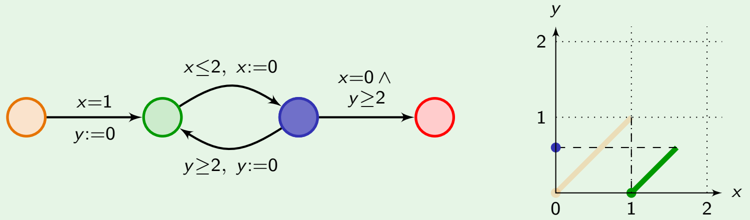
...real-time models for real-time systems!





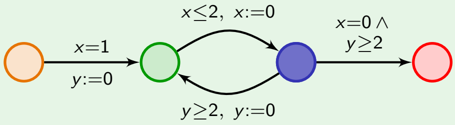
Continuous-time semantics

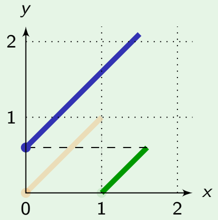
...real-time models for real-time systems!



Continuous-time semantics

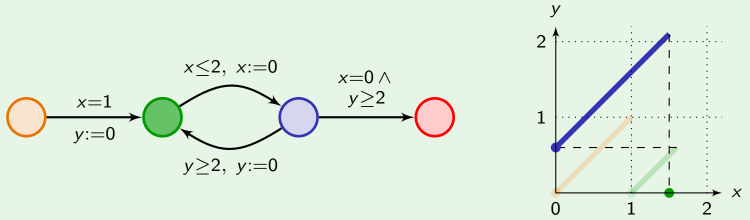
...real-time models for real-time systems!





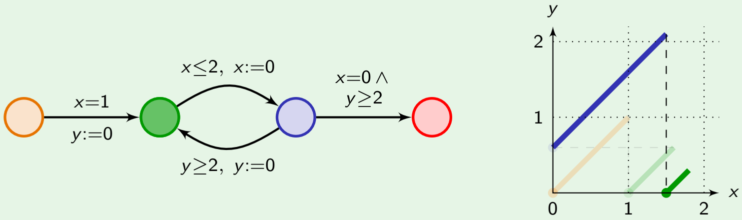
Continuous-time semantics

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



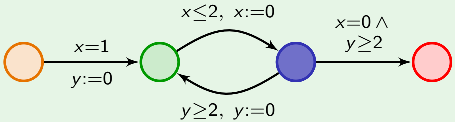
Continuous-time semantics

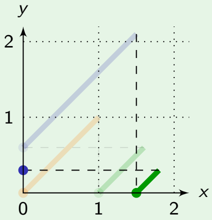
...real-time models for real-time systems!



Continuous-time semantics

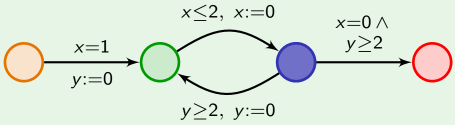
...real-time models for real-time systems!

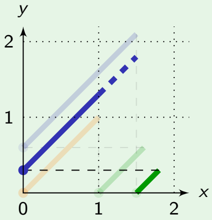




Continuous-time semantics

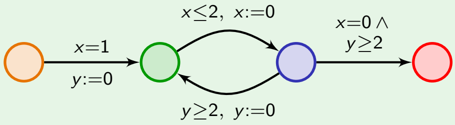
...real-time models for real-time systems!

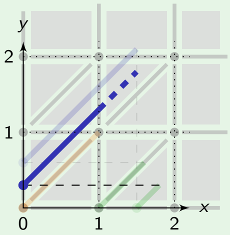




Continuous-time semantics

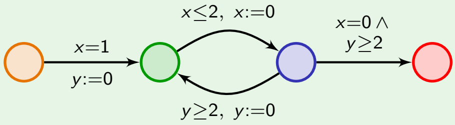
...real-time models for real-time systems!

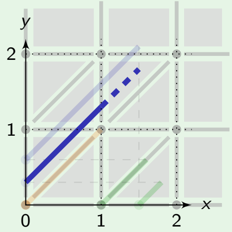




Continuous-time semantics

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





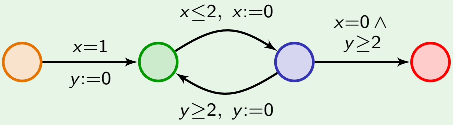
Theorem [AD94]

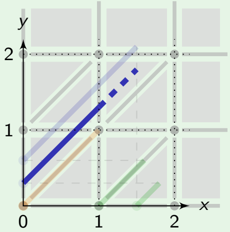
Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

- Technical tool: region abstraction

Continuous-time semantics

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



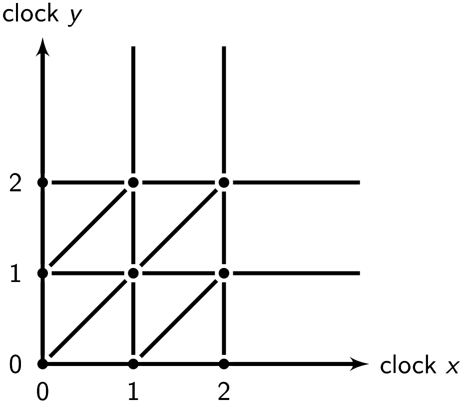


Theorem [AD94]

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

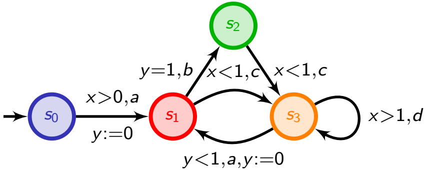
- Technical tool: region abstraction
- Efficient symbolic technics based on zones, implemented in tools

Technical tool: Region abstraction

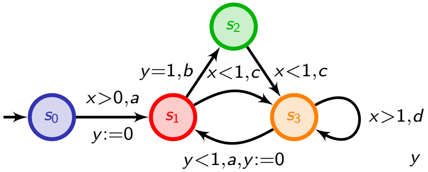


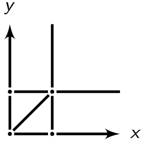
$\rightsquigarrow$ This is a finite time-abstract bisimulation!

Technical tool: Region abstraction – An example [AD94]

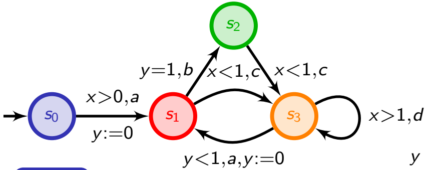


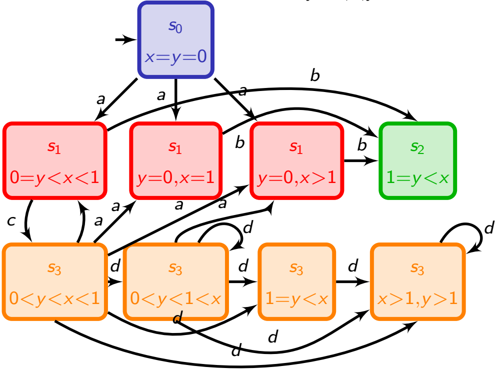
Technical tool: Region abstraction – An example [AD94]

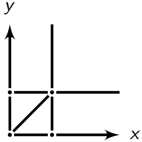




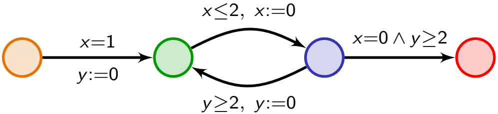
Technical tool: Region abstraction – An example [AD94]



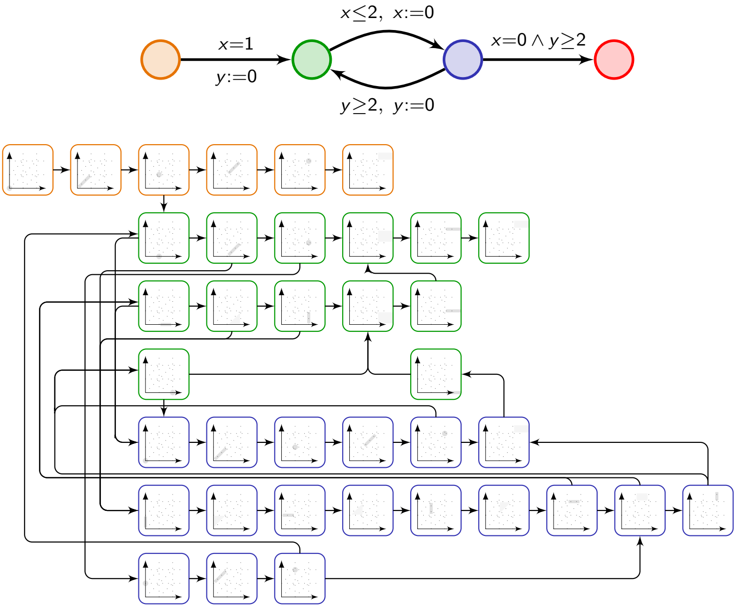




Technical tool: Region abstraction – Another example



Technical tool: Region abstraction – Another example



Technical tool: Zones and DBMs

DBM = Difference Bound Matrix

Technical tool: Zones and DBMs

DBM = Difference Bound Matrix

Zones, or DBMs...

... are used to represent sets of states of timed automata:

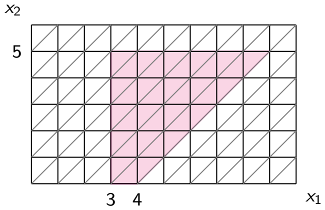
Technical tool: Zones and DBMs

DBM = Difference Bound Matrix

Zones, or DBMs...

... are used to represent sets of states of timed automata:

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



Technical tool: Zones and DBMs

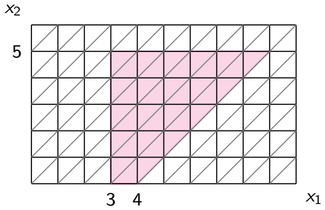
DBM = Difference Bound Matrix

Zones, or DBMs...

... are used to represent sets of states of timed automata:

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

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



Technical tool: Zones and DBMs

DBM = Difference Bound Matrix

Zones, or DBMs...

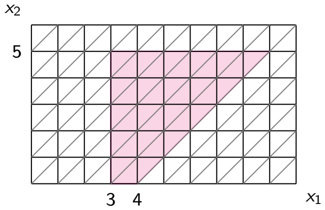
... are used to represent sets of states of timed automata:

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

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

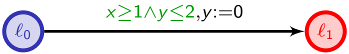
$$|||$$

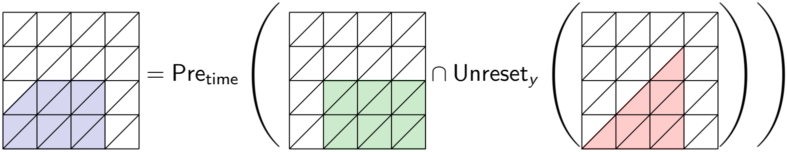
$$\begin{array}{cc} & \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \\ x_2 \end{array} & \begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix} \end{array}$$



Technical tool: Zones and DBMs

They can be used to compute sets of states in timed automata





Are we doing the right job?

The continuous-time semantics is

adequate for abstract design and high-level analysis.

Example: The Patriot anti-ballistic-missile failure

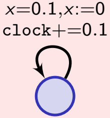
25 February 1991, during Gulf war.
28 soldiers died.

Example: The Patriot anti-ballistic-missile failure

25 February 1991, during Gulf war.
28 soldiers died.

Problem: clock drift

Internal clock incremented by 1/10 every 1/10 s.



Example: The Patriot anti-ballistic-missile failure

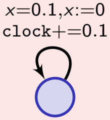
25 February 1991, during Gulf war.
28 soldiers died.

Problem: clock drift

Internal clock incremented by $1/10$ every $1/10$ s.

Clock stored in 24-bit register:

$$\frac{1}{10} - \left\langle \frac{1}{10} \right\rangle_{24\ \text{bit}} \simeq 10^{-7}$$



Example: The Patriot anti-ballistic-missile failure

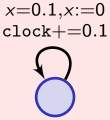
25 February 1991, during Gulf war.
28 soldiers died.

Problem: clock drift

Internal clock incremented by $1/10$ every $1/10$ s.

Clock stored in 24-bit register:

$$\frac{1}{10} - \left\langle \frac{1}{10} \right\rangle_{\text{24 bit}} \simeq 10^{-7}$$



After 100 hours, the total drift was 0.34 seconds.
The incoming missile could not be destroyed.

Are we doing the right job?

The continuous-time semantics is

adequate for abstract design and high-level analysis.

Are we doing the right job?

The continuous-time semantics is
an idealization of a physical system.
It is adequate for abstract design and high-level analysis.

Are we doing the right job?

The continuous-time semantics is
an idealization of a physical system.
It is adequate for abstract design and high-level analysis.

However it suffers from multiple inaccuracies:

Are we doing the right job?

The continuous-time semantics is
an idealization of a physical system.
It is adequate for abstract design and high-level analysis.

However it suffers from multiple inaccuracies:

- It might not be proper for implementation:
  - it assumes zero-delay transitions
  - it assumes infinite precision of the clocks
  - it assumes immediate communication between systems
  - it assumes infinite frequency

Are we doing the right job?

The continuous-time semantics is
an idealization of a physical system.
It is adequate for abstract design and high-level analysis.

However it suffers from multiple inaccuracies:

- It might not be proper for implementation:
- It may generate timing anomalies

Are we doing the right job?

The continuous-time semantics is
an idealization of a physical system.
It is adequate for abstract design and high-level analysis.

However it suffers from multiple inaccuracies:

- It might not be proper for implementation:
- It may generate timing anomalies
- It does not exclude non-realizable behaviours:
  - not only Zeno behaviours
  - many convergence phenomena are hidden

$\rightsquigarrow$ this requires infinite precision and might not be realizable

Are we doing the right job?

The continuous-time semantics is
an idealization of a physical system.
It is adequate for abstract design and high-level analysis.

However it suffers from multiple inaccuracies:

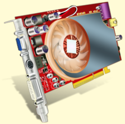
- It might not be proper for implementation:
- It may generate timing anomalies
- It does not exclude non-realizable behaviours:

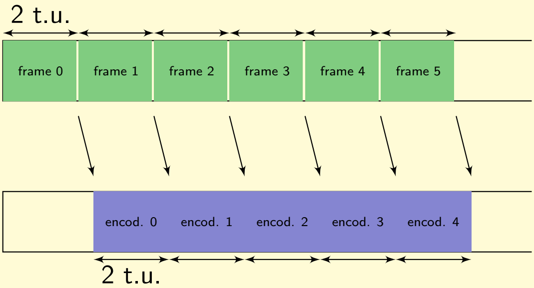
Important questions

- Is the real system correct when it is proven correct on the model?
- Does actual work transfer to real-world systems? To what extent?

Example 1: Imprecision on clock values

Frame capture [ACS10]

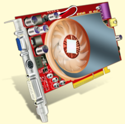


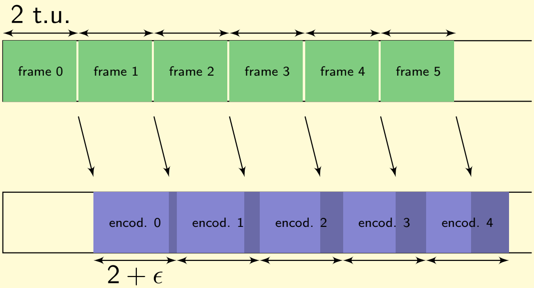


[ACS10] Abdellatif, Combaz, Sifakis. Model-based implementation of real-time applications *(Int. Conf. Embedded Software, 2010)*.

Example 1: Imprecision on clock values

Frame capture [ACS10]



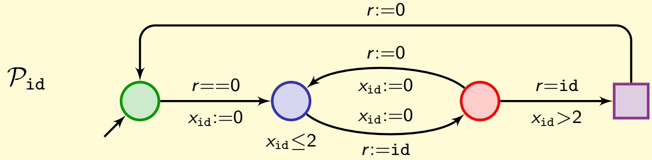


$\rightsquigarrow$ A frame will eventually be skipped

[ACS10] Abdellatif, Combaz, Sifakis. Model-based implementation of real-time applications *(Int. Conf. Embedded Software, 2010).*

Example 2: Strict timing constraints

Mutual exclusion protocol [KLL+97]

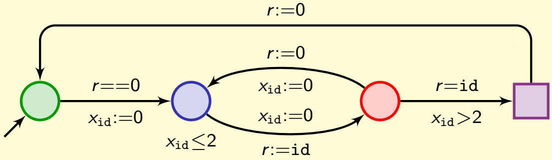


[KLL+97] Kristoffersen, Laroussinie, Larsen, Pettersson, Yi. A compositional proof of a real-time mutual exclusion protocol *(TAPSOFT'97)*.

Example 2: Strict timing constraints

Mutual exclusion protocol [KLL+97]

$\mathcal{P}_{\mathtt{id}}$

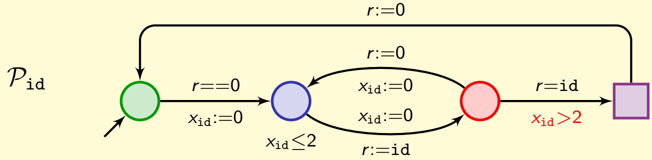


- When $\mathcal{P}_1$ and $\mathcal{P}_2$ run in parallel (sharing variable r), the state where both of them are in ☐ is not reachable.

[KLL+97] Kristoffersen, Laroussinie, Larsen, Pettersson, Yi. A compositional proof of a real-time mutual exclusion protocol *(TAPSOFT'97)*.

Example 2: Strict timing constraints

Mutual exclusion protocol [KLL+97]



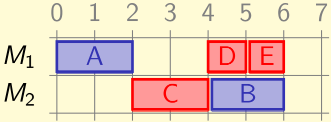
- When $\mathcal{P}_1$ and $\mathcal{P}_2$ run in parallel (sharing variable r), the state where both of them are in □ is not reachable.
- This property is lost when $x_{\texttt{id}} > 2$ is replaced with $x_{\texttt{id}} \geq 2$.

[KLL+97] Kristoffersen, Laroussinie, Larsen, Pettersson, Yi. A compositional proof of a real-time mutual exclusion protocol *(TAPSOFT'97)*.

Example 3: Scheduling and timing anomaly

- Scheduling analysis with timed automata [AAM06]
- **Goal:** analyze a *work-conserving* scheduling policy on given scenarios (no machine is idle if a task is waiting for execution)

Example of a scenario



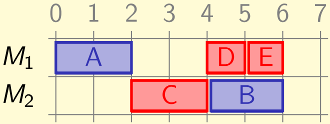
with the dependency constraints: $A \to B$ and $C \to D, E$.

1. A, D, E must be scheduled on machine M_1
2. B, C must be scheduled on machine M_2
3. C starts no sooner than 2 time units

[AAM06] Abdeddaim, Asarin, Maler. Scheduling with timed automata *(Theor. Comp. Science, 2006)*.

Example 3: Scheduling and timing anomaly

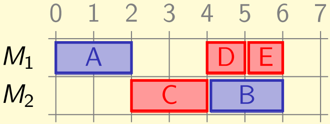
Example of a scenario



$\rightsquigarrow$ Schedulable in 6 time units

Example 3: Scheduling and timing anomaly

Example of a scenario

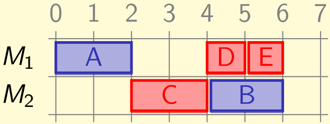


$\rightsquigarrow$ Schedulable in 6 time units

- Unexpectedly, the duration of A drops to 1.999

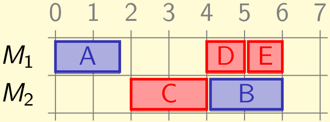
Example 3: Scheduling and timing anomaly

Example of a scenario



$\rightsquigarrow$ Schedulable in 6 time units

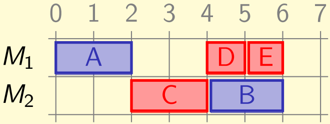
- Unexpectedly, the duration of A drops to 1.999



is not work-conserving

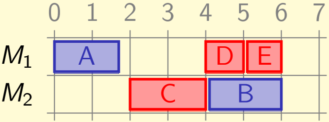
Example 3: Scheduling and timing anomaly

Example of a scenario

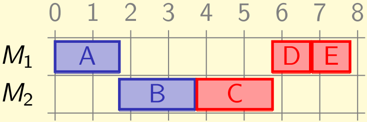


$\rightsquigarrow$ Schedulable in 6 time units

- Unexpectedly, the duration of A drops to 1.999



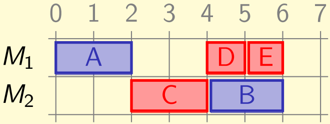
is not work-conserving



is work-conserving
and completes in 7.999 t.u.

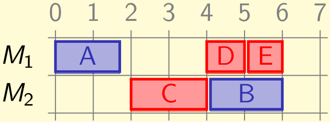
Example 3: Scheduling and timing anomaly

Example of a scenario

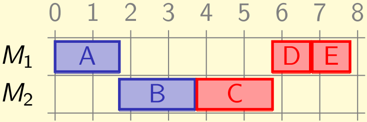


$\rightsquigarrow$ Schedulable in 6 time units

- Unexpectedly, the duration of A drops to 1.999



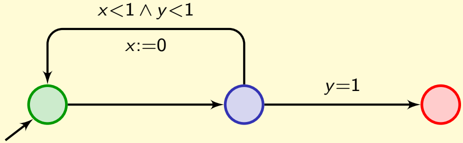
is not work-conserving

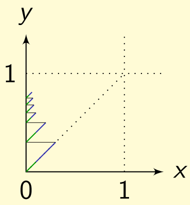


is work-conserving
and completes in 7.999 t.u.

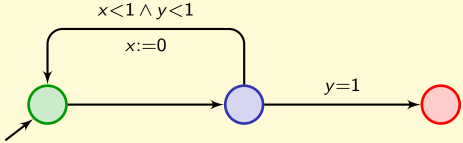
$\rightsquigarrow$ Standard analysis does not capture this **timing anomaly**

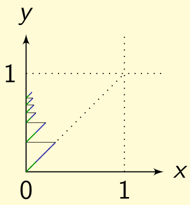
Example 4: Zeno behaviours





Example 4: Zeno behaviours

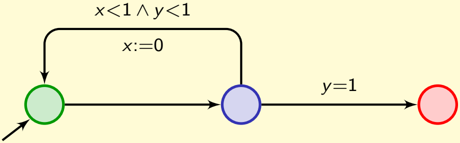


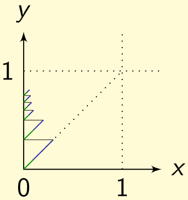


- Those are easy to detect and can be handled; [HS11]

[HS11] Herbreteau, Srivathsan. Coarse abstractions make Zeno behaviours difficult to detect *(Logic. Meth. Comp. Science, 2011)*.

Example 4: Zeno behaviours

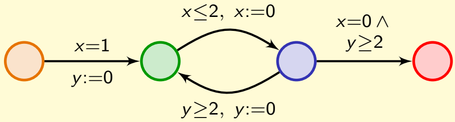




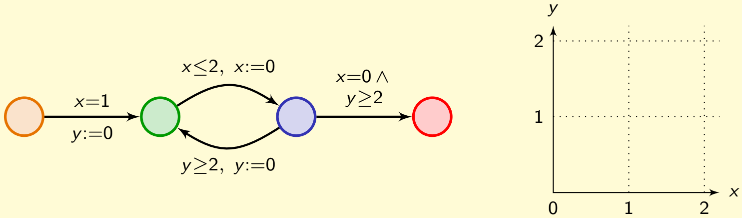
- Those are easy to detect and can be handled; [HS11]
- They are easy to remove by construction.

[HS11] Herbreteau, Srivathsan. Coarse abstractions make Zeno behaviours difficult to detect *(Logic. Meth. Comp. Science, 2011)*.

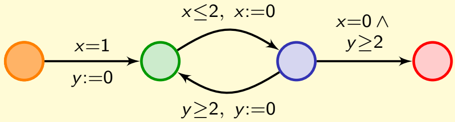
Example 5: More complex convergence phenomena

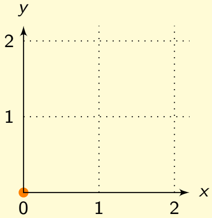


Example 5: More complex convergence phenomena

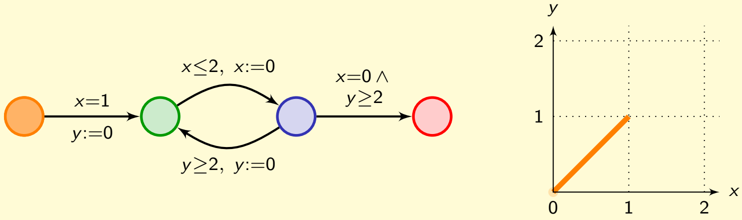


Example 5: More complex convergence phenomena

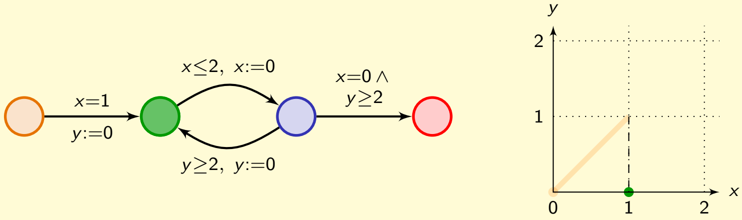




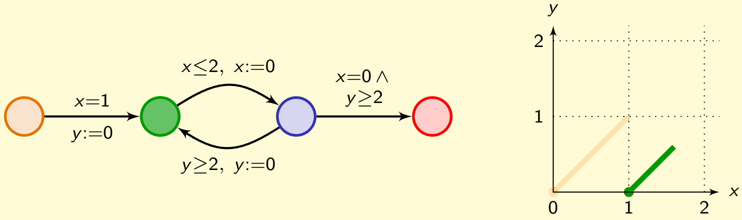
Example 5: More complex convergence phenomena



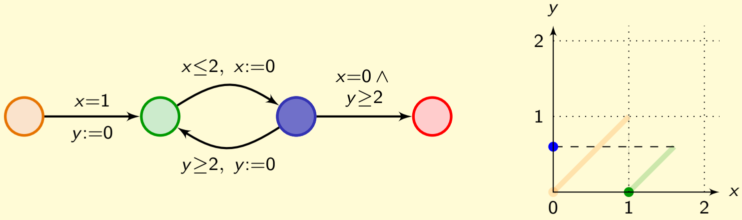
Example 5: More complex convergence phenomena



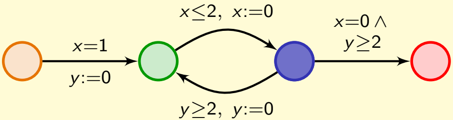
Example 5: More complex convergence phenomena

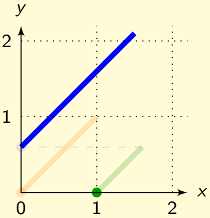


Example 5: More complex convergence phenomena

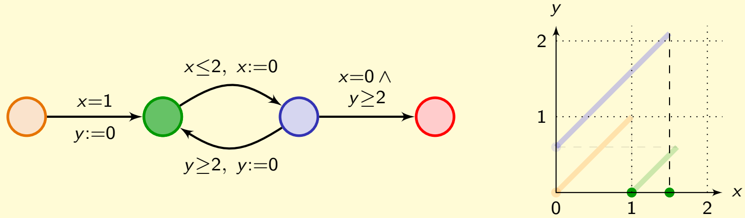


Example 5: More complex convergence phenomena

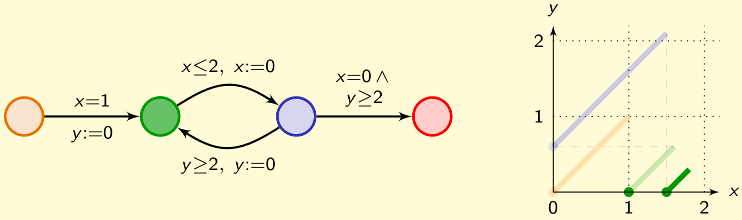




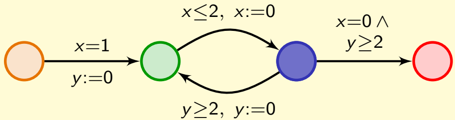
Example 5: More complex convergence phenomena

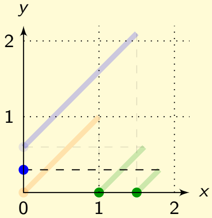


Example 5: More complex convergence phenomena

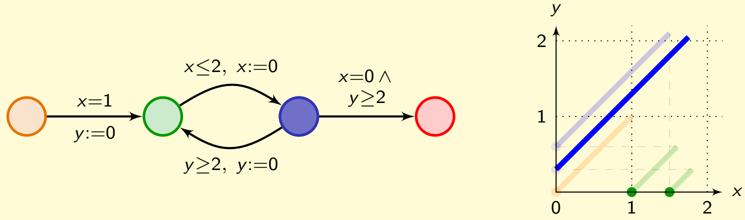


Example 5: More complex convergence phenomena

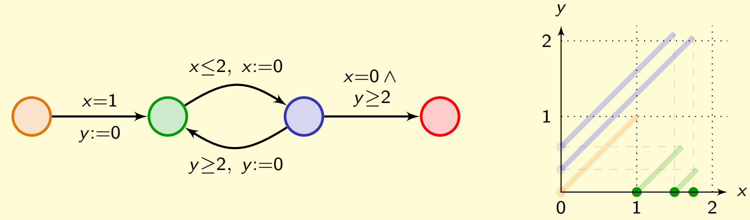




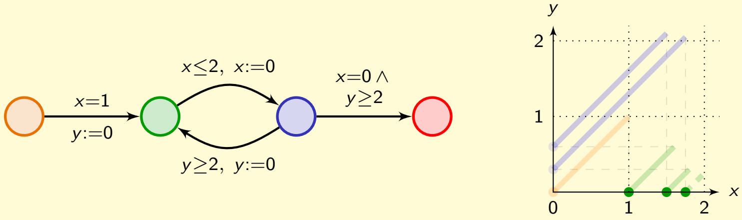
Example 5: More complex convergence phenomena



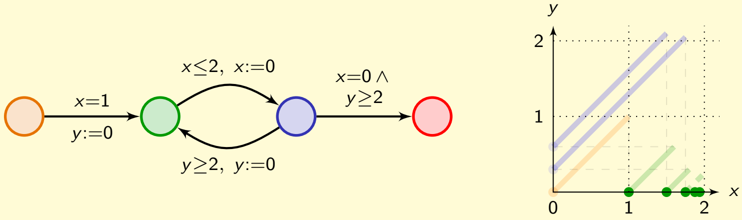
Example 5: More complex convergence phenomena



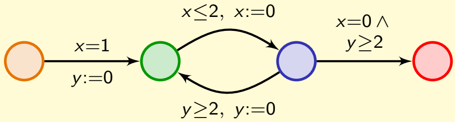
Example 5: More complex convergence phenomena

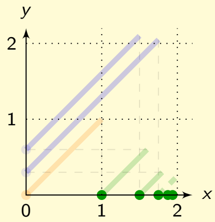


Example 5: More complex convergence phenomena



Example 5: More complex convergence phenomena





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

The goal

Add robustness to the theory of timed automata

The goal

Add robustness to the theory of timed automata

- Understand the real system behind the mathematical model

The goal

Add robustness to the theory of timed automata

- Understand the real system behind the mathematical model
- Describe frameworks and provide tools to develop robustly correct systems

The goal

Add robustness to the theory of timed automata

- Understand the real system behind the mathematical model
- Describe frameworks and provide tools to develop robustly correct systems
  $\rightsquigarrow$ Notion of robustness may depend on application areas

The goal

Add robustness to the theory of timed automata

- Understand the real system behind the mathematical model
- Describe frameworks and provide tools to develop robustly correct systems
  $\rightsquigarrow$ Notion of robustness may depend on application areas

Rest of the talk

- We present a couple of frameworks that have been developed recently
- We focus on the tolerance to slight timing perturbations, that is, to perturbations on time measurements and jitter

Outline

Robust model-checking approach

Idea

Capture any real (or approximate) behaviours (e.g. the implementation) in the verification process

Robust model-checking approach

Idea

Capture any real (or approximate) behaviours (e.g. the implementation) in the verification process

Due to imprecisions,

$$\text{“standard” correctness of } \mathcal{A} \quad \not\Rightarrow \quad \text{correctness of } \mathcal{A}_{\texttt{real}}$$

Robust model-checking approach

Idea

Capture any real (or approximate) behaviours (e.g. the implementation) in the verification process

Due to imprecisions,

$$\text{“standard” correctness of } \mathcal{A} \quad \not\Rightarrow \quad \text{correctness of } \mathcal{A}_{\texttt{real}}$$

$\leadsto$ We aim at proposing frameworks in which the correctness of the real system will be ensured once the model is verified

Robust model-checking approach

Idea

Capture any real (or approximate) behaviours (e.g. the implementation) in the verification process

Due to imprecisions,

$$\text{“standard” correctness of } \mathcal{A} \quad \not\Rightarrow \quad \text{correctness of } \mathcal{A}_{\mathtt{real}}$$

$\rightsquigarrow$ We aim at proposing frameworks in which the correctness of the real system will be ensured once the model is verified

We describe two such frameworks:

1. either we implement $\mathcal{A}$ and we prove:

$$\text{“robust” correctness of } \mathcal{A} \quad \Rightarrow \quad \text{correctness of } \mathcal{A}_{\mathtt{real}}$$

Robust model-checking approach

Idea

Capture any real (or approximate) behaviours (e.g. the implementation) in the verification process

Due to imprecisions,

$$\text{“standard” correctness of } \mathcal{A} \quad \not\Rightarrow \quad \text{correctness of } \mathcal{A}_{\texttt{real}}$$

$\rightsquigarrow$ We aim at proposing frameworks in which the correctness of the real system will be ensured once the model is verified

We describe two such frameworks:

1. either we implement $\mathcal{A}$ and we prove:

$$\text{“robust” correctness of } \mathcal{A} \quad \Rightarrow \quad \text{correctness of } \mathcal{A}_{\texttt{real}}$$

2. or we build $\mathcal{A}$ and implement $\mathcal{B}$, and we prove:

$$\text{correctness of } \mathcal{A} \quad \Rightarrow \quad \text{“robust” correctness of } \mathcal{B}$$
$$\Rightarrow \quad \text{correctness of } \mathcal{B}_{\texttt{real}}$$

Outline

1. Introduction
2. Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking
3. Robust realisability and control
   - Excess semantics
   - Strict semantics
4. Conclusion

Parameterized enlarged semantics for timed automata

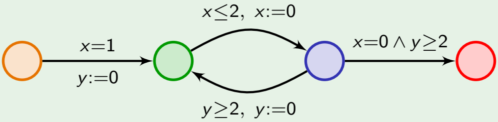
A transition can be taken at any time in $[t - \delta; t + \delta]$

Parameterized enlarged semantics for timed automata

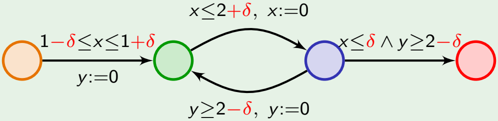
A transition can be taken at any time in $[t-\delta; t+\delta]$

Example

Given a parameter δ,



is transformed into



Parameterized model $\mathcal{A}_\delta$

Parameterized enlarged semantics – Discussion

What is the relevance of this semantics?

- This is a worst-case approach
- This captures approximate behaviours of the system
- One can define program semantics such that for every $\epsilon > 0$:

$$\mathcal{A} \subseteq \mathtt{program}_{\epsilon}(\mathcal{A}) \subseteq \mathcal{A}_{f(\epsilon)}$$

ϵ: parameters of the semantics

[DDR04] De Wulf, Doyen, Raskin. Almost ASAP semantics: From timed models to timed implementations *(HSCC'04)*.
[SBM11] Sankur, Bouyer, Markey. Shrinking Timed Automata *(FSTTCS'11)*.

Parameterized enlarged semantics – Discussion

What is the relevance of this semantics?

- This is a worst-case approach
- This captures approximate behaviours of the system
- One can define program semantics such that for every $\epsilon > 0$:

$$\mathcal{A} \subseteq \mathtt{program}_\epsilon(\mathcal{A}) \subseteq \mathcal{A}_{f(\epsilon)}$$

ϵ: parameters of the semantics

Methodology

- Design $\mathcal{A}$
- Verify $\mathcal{A}_\delta$ (better if δ is a parameter)
- Implement $\mathcal{A}$

Parameterized enlarged semantics – Discussion

What is the relevance of this semantics?

- This is a worst-case approach
- This captures approximate behaviours of the system
- One can define program semantics such that for every $\epsilon > 0$:

$$\mathcal{A} \subseteq \mathtt{program}_\epsilon(\mathcal{A}) \subseteq \mathcal{A}_{f(\epsilon)}$$

ϵ: parameters of the semantics

Methodology

- Design $\mathcal{A}$
- Verify $\mathcal{A}_\delta$ (better if δ is a parameter)
- Implement $\mathcal{A}$

$\rightsquigarrow$ This is a good approach for designing systems with simple timing constraints (e.g. equalities).

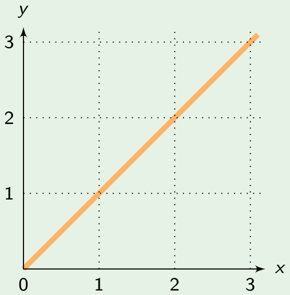
Parameterized enlarged semantics – Algorithmics

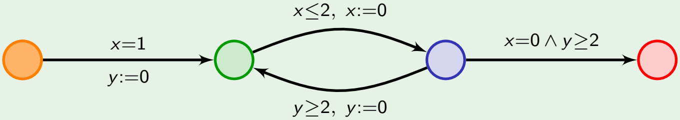
$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

Example

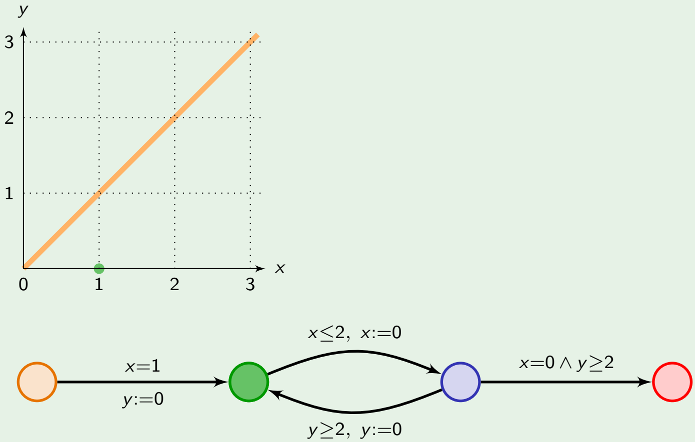




Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

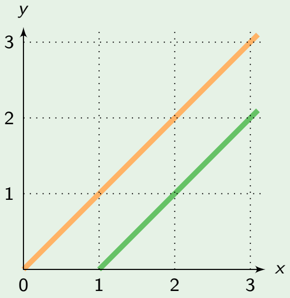
Example

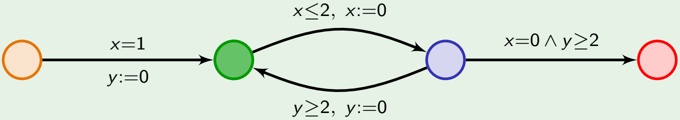


Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

Example

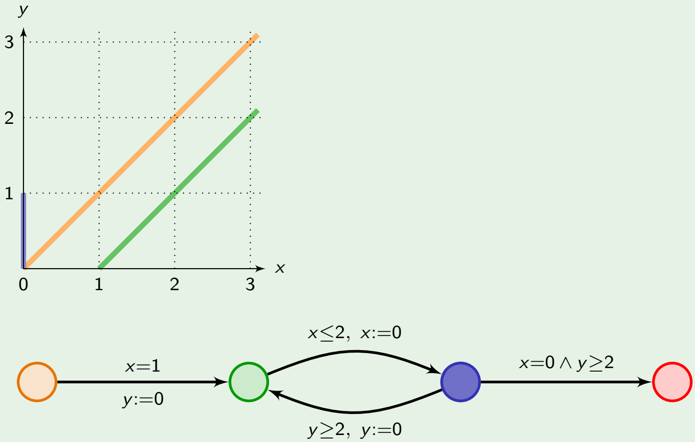




Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

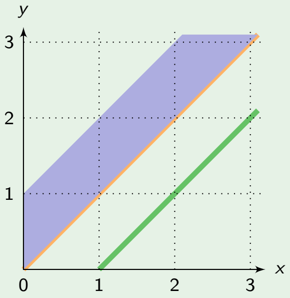
Example

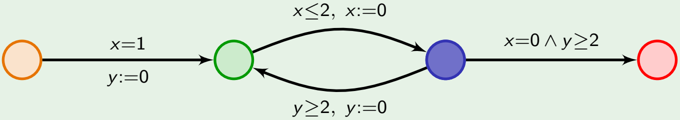


Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

Example

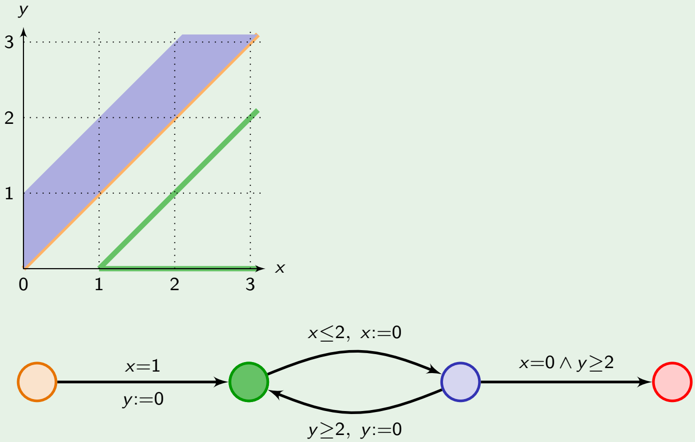




Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

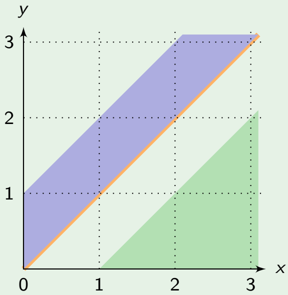
Example

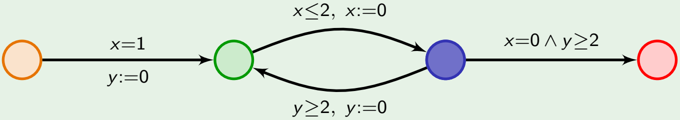


Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

Example

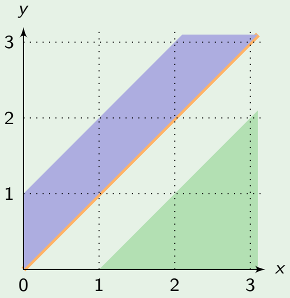


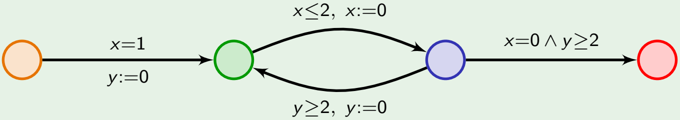


Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

Example

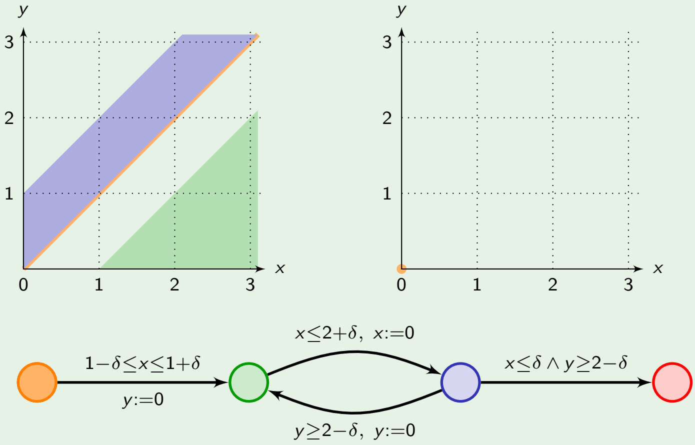




Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

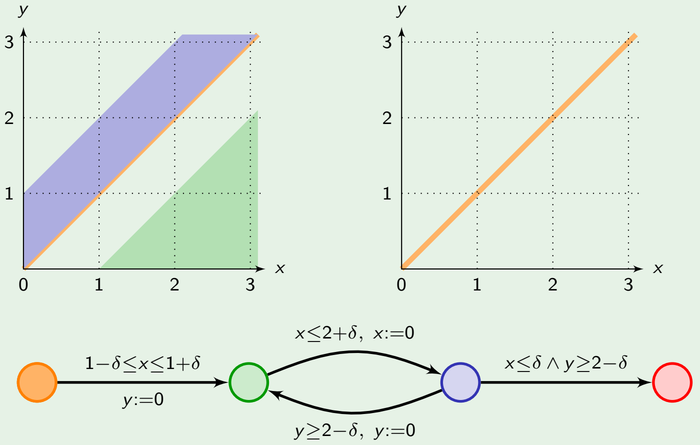
Example



Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

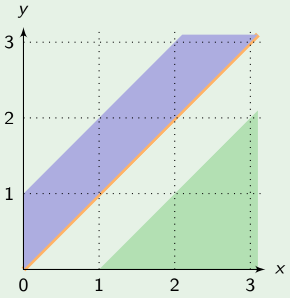
Example

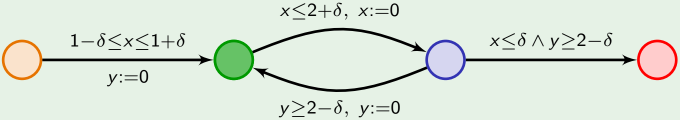


Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

Example

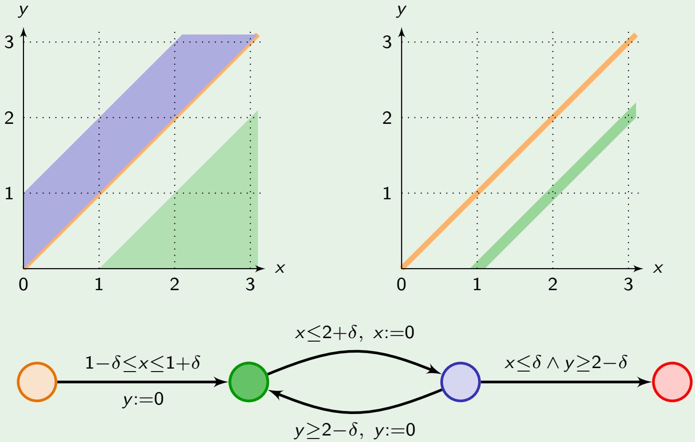




Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

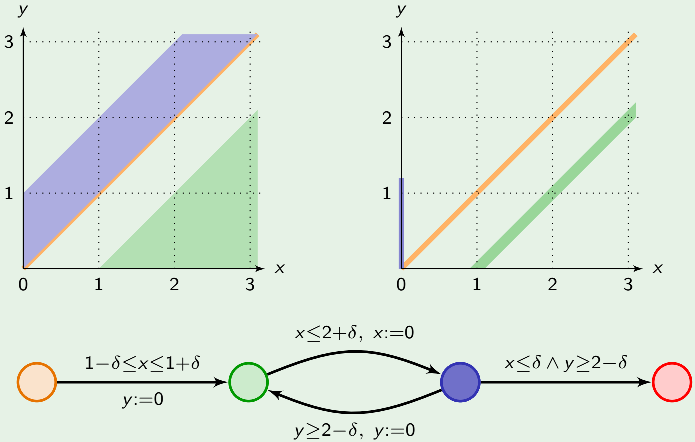
Example



Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

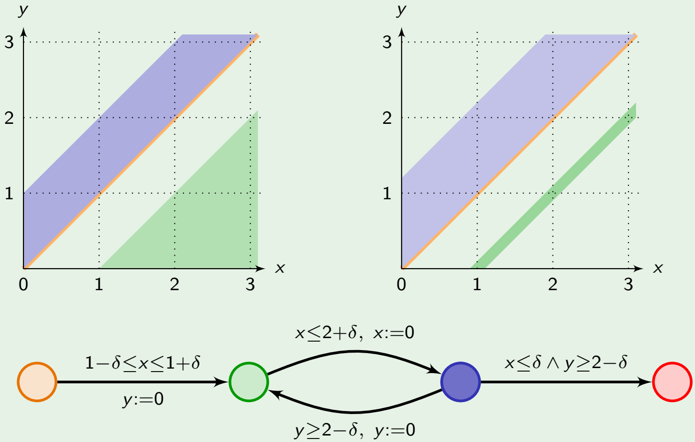
Example



Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

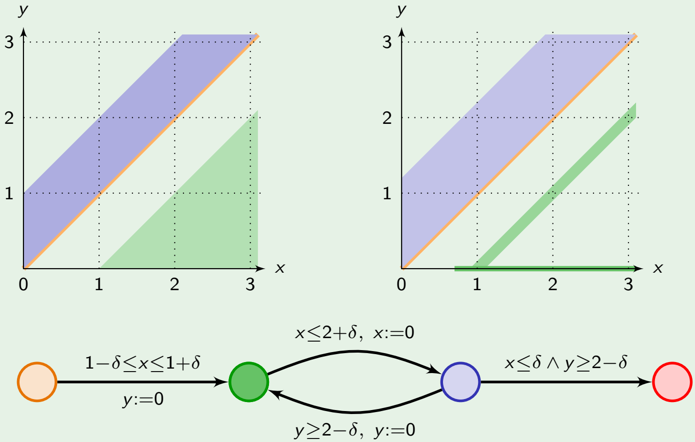
Example



Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

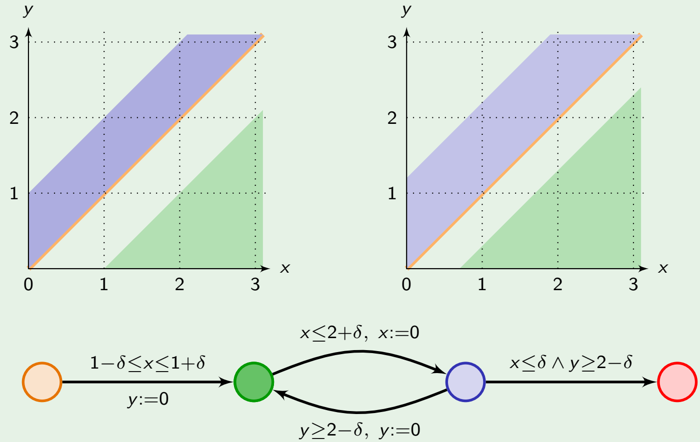
Example



Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

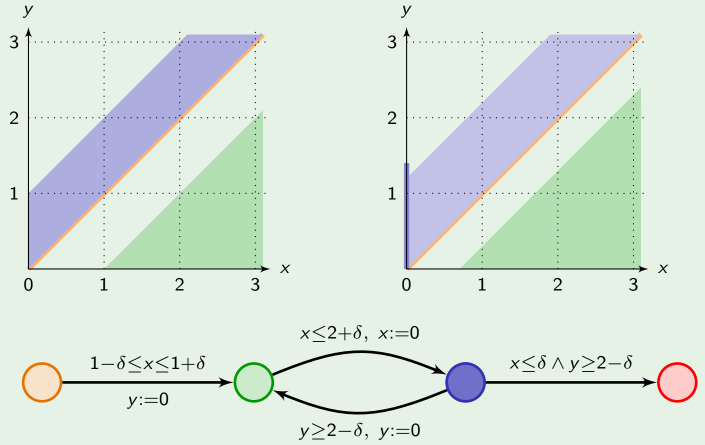
Example



Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

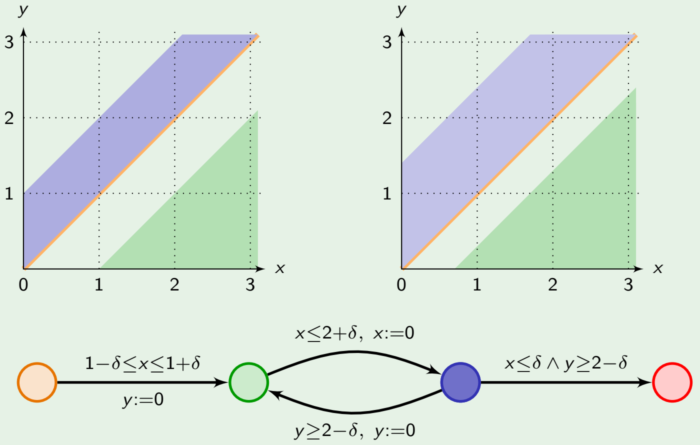
Example



Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

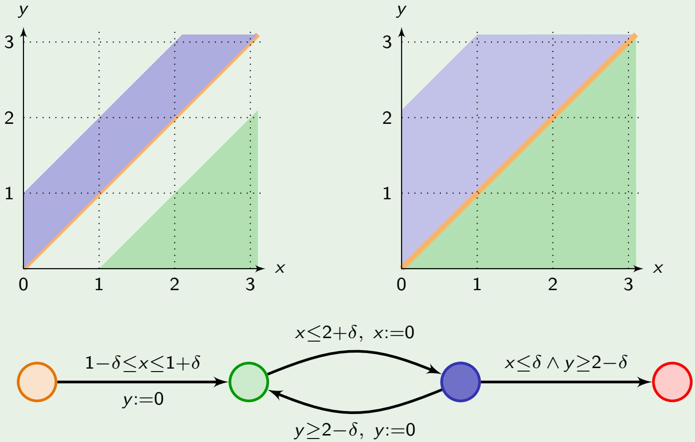
Example



Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

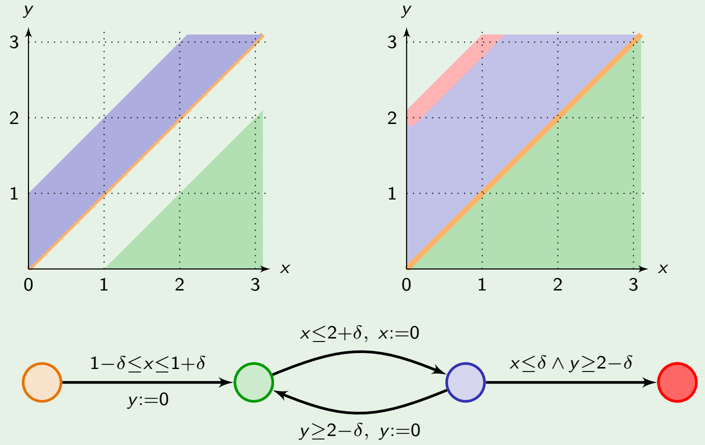
Example



Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

Example



Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

The (parameterized) robust model-checking problem

It asks whether there is $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $\mathcal{A}_\delta \models \varphi$.

Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

The (parameterized) robust model-checking problem

It asks whether there is $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $\mathcal{A}_\delta \models \varphi$.

- When δ is small, truth of φ is independent of δ

[Mil00] Miller. Decidability and Complexity Results for Timed Automata and Semi-linear Hybrid Automata *(HSCC'00)*.

Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

The (parameterized) robust model-checking problem

It asks whether there is $\delta_0 > 0$ s.t. for every $0 \le \delta \le \delta_0$, $\mathcal{A}_\delta \models \varphi$.

- When δ is small, truth of φ is independent of δ

$$\delta_1 \le \delta_2 \quad \Rightarrow \quad \mathsf{Reach}(\mathcal{A}_{\delta_1}) \subseteq \mathsf{Reach}(\mathcal{A}_{\delta_2})$$

Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

The (parameterized) robust model-checking problem

It asks whether there is $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $\mathcal{A}_\delta \models \varphi$.

- When δ is small, truth of φ is independent of δ

$$\begin{array}{rcl} \delta_1 \leq \delta_2 & \Rightarrow & \text{Reach}(\mathcal{A}_{\delta_1}) \subseteq \text{Reach}(\mathcal{A}_{\delta_2}) \\ & \Rightarrow & (\mathcal{A}_{\delta_2} \text{ safe} \Rightarrow \mathcal{A}_{\delta_1} \text{ safe}) \end{array}$$

Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

The (parameterized) robust model-checking problem

It asks whether there is $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $\mathcal{A}_\delta \models \varphi$.

- When δ is small, truth of φ is independent of δ
- Timed automata with parameters: undecidable in general

Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

The (parameterized) robust model-checking problem

It asks whether there is $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $\mathcal{A}_\delta \models \varphi$.

- When δ is small, truth of φ is independent of δ
- Timed automata with parameters: undecidable in general
- Here, an extension of the region automaton will do the job!

Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

The (parameterized) robust model-checking problem

It asks whether there is $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $\mathcal{A}_\delta \models \varphi$.

- When δ is small, truth of φ is independent of δ
- Timed automata with parameters: undecidable in general
- Here, an extension of the region automaton will do the job!

Theorem

Robust model-checking of safety, LTL, CoflatMTL properties is decidable.
Complexities are those of standard non robust model-checking problems.

[Puri00] Puri. Dynamical properties of timed automata *(Disc. Event Dyn. Syst., 2000)*.
[DDMR08] De Wulf, Doyen, Markey, Raskin. Robust safety of timed automata *(Form. Meth. Syst. Design, 2008)*.
[BMR06] Bouyer, Markey, Reynier. Robust model-checking of timed automata *(LATIN'06)*.
[BMR08] Bouyer, Markey, Reynier. Robust analysis of timed automata via channel machines *(FoSSaCS'08)*.
[BMS11] Bouyer, Markey, Sankur. Robust Model-Checking of Timed Automata via Pumping in Channel Machines *(FORMATS'11)*.

Parameterized enlarged semantics – Algorithmics

$\rightsquigarrow$ It adds extra behaviours, however small may be parameter δ

The (parameterized) robust model-checking problem

It asks whether there is $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $\mathcal{A}_\delta \models \varphi$.

- When δ is small, truth of φ is independent of δ
- Timed automata with parameters: undecidable in general
- Here, an extension of the region automaton will do the job!

Theorem

Robust model-checking of safety, LTL, CoflatMTL properties is decidable. Complexities are those of standard non robust model-checking problems.

(An exponential bound on δ_0 is proven)

[Puri00] Puri. Dynamical properties of timed automata *(Disc. Event Dyn. Syst., 2000)*.
[DDMR08] De Wulf, Doyen, Markey, Raskin. Robust safety of timed automata *(Form. Meth. Syst. Design, 2008)*.
[BMR06] Bouyer, Markey, Reynier. Robust model-checking of timed automata *(LATIN'06)*.
[BMR08] Bouyer, Markey, Reynier. Robust analysis of timed automata via channel machines *(FoSSaCS'08)*.
[BMS11] Bouyer, Markey, Sankur. Robust Model-Checking of Timed Automata via Pumping in Channel Machines *(FORMATS'11)*.

Technical tool: extended region automaton

Extended region automaton

For any location ℓ and any two regions r and r', if

- $\overline{r} \cap \overline{r'} \neq \varnothing$ and
- (ℓ, r') belongs to an SCC of $\mathcal{R}(\mathcal{A})$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$

(under slight technical restrictions)

Technical tool: extended region automaton

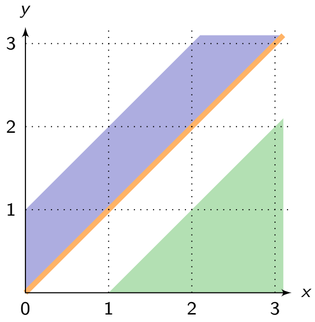
Extended region automaton

For any location ℓ and any two regions r and r', if

- $\overline{r} \cap \overline{r'} \neq \varnothing$ and
- (ℓ, r') belongs to an SCC of $\mathcal{R}(\mathcal{A})$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$

(under slight technical restrictions)



Technical tool: extended region automaton

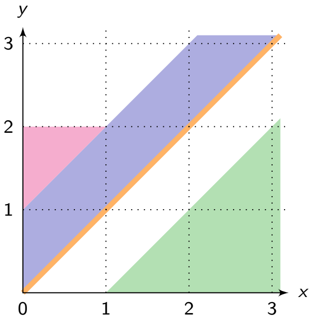
Extended region automaton

For any location ℓ and any two regions r and r', if

- $\overline{r} \cap \overline{r'} \neq \varnothing$ and
- (ℓ, r') belongs to an SCC of $\mathcal{R}(\mathcal{A})$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$

(under slight technical restrictions)



Technical tool: extended region automaton

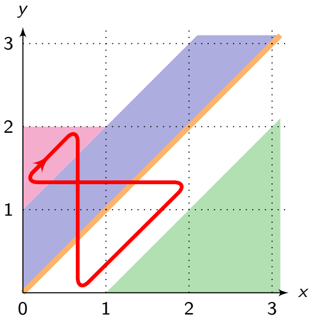
Extended region automaton

For any location ℓ and any two regions r and r', if

- $\overline{r} \cap \overline{r'} \neq \varnothing$ and
- (ℓ, r') belongs to an SCC of $\mathcal{R}(\mathcal{A})$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$

(under slight technical restrictions)



Technical tool: extended region automaton

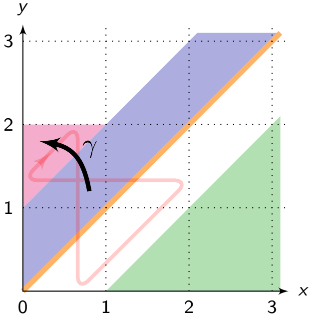
Extended region automaton

For any location ℓ and any two regions r and r', if

- $\overline{r} \cap \overline{r'} \neq \varnothing$ and
- (ℓ, r') belongs to an SCC of $\mathcal{R}(\mathcal{A})$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$

(under slight technical restrictions)



Technical tool: extended region automaton

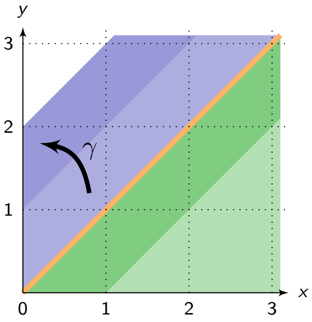
Extended region automaton

For any location ℓ and any two regions r and r', if

- $\overline{r} \cap \overline{r'} \neq \varnothing$ and
- (ℓ, r') belongs to an SCC of $\mathcal{R}(\mathcal{A})$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$

(under slight technical restrictions)



Technical tool: extended region automaton

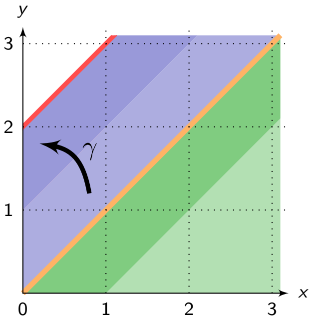
Extended region automaton

For any location ℓ and any two regions r and r', if

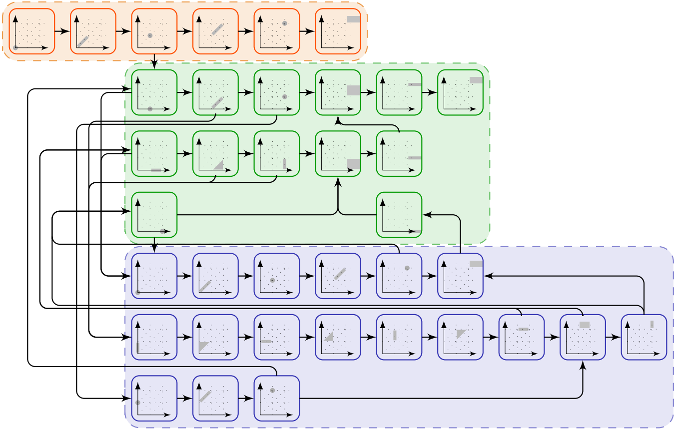
- $\overline{r} \cap \overline{r'} \neq \varnothing$ and
- (ℓ, r') belongs to an SCC of $\mathcal{R}(\mathcal{A})$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$

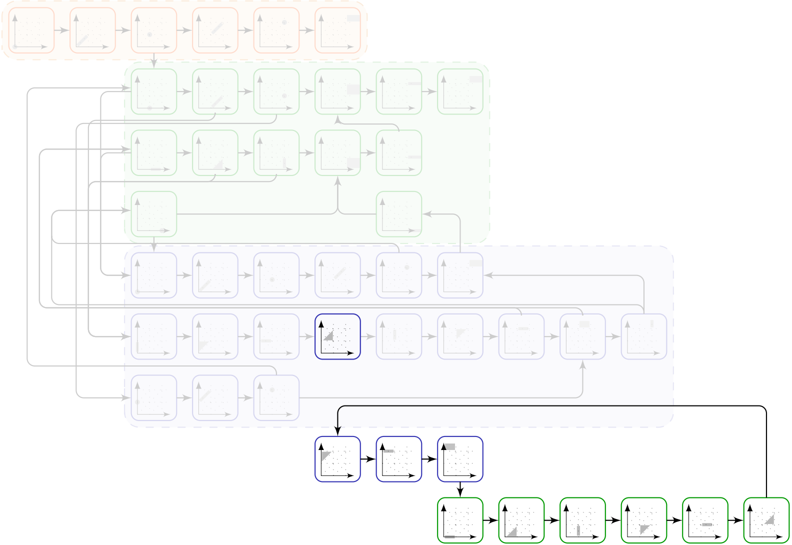
(under slight technical restrictions)



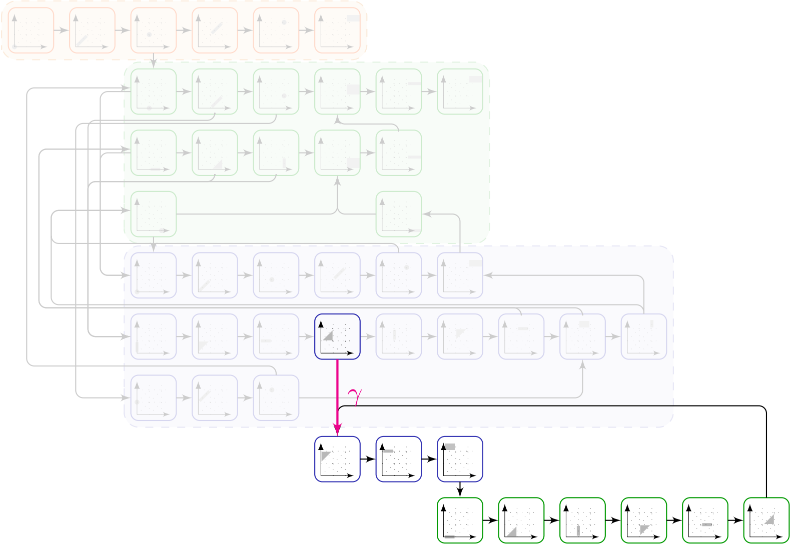
Parameterized enlarged semantics – An example



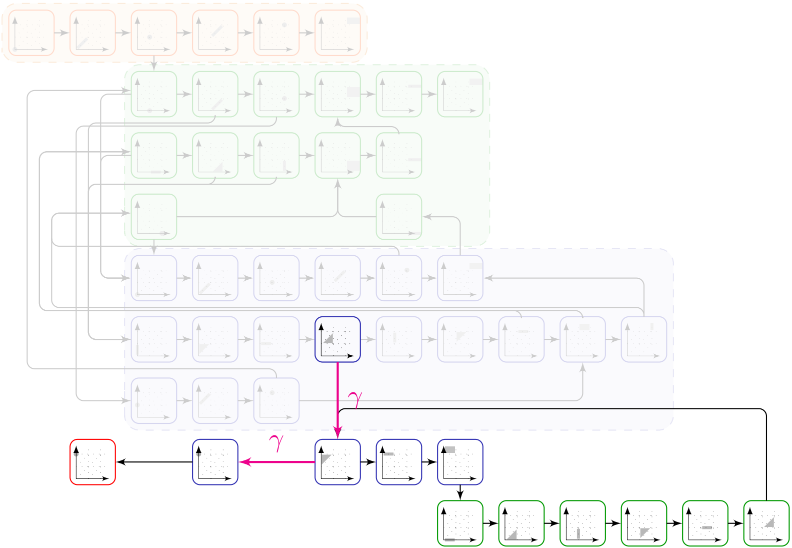
Parameterized enlarged semantics – An example



Parameterized enlarged semantics – An example



Parameterized enlarged semantics – An example



Outline

1. Introduction
2. Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking
3. Robust realisability and control
   - Excess semantics
   - Strict semantics
4. Conclusion

Automatic generation of an implementation

The (approx.) implementation synthesis problem

Given $\mathcal{A}$, build $\mathcal{A}'$ such that:

- $\mathcal{A}'$ 'identical' (e.g. bisimilar) to $\mathcal{A}$
- $\mathcal{A}'$ is 'robust' (that is, good enough for implementation)

[BLM+11] Bouyer, Larsen, Markey, Sankur, Thrane. Timed automata can always be made implementable *(CONCUR'11)*.

Automatic generation of an implementation

The (approx.) implementation synthesis problem

Given $\mathcal{A}$, build $\mathcal{A}'$ such that:

- $\mathcal{A}'$ 'identical' (e.g. bisimilar) to $\mathcal{A}$
- $\mathcal{A}'$ is 'robust' (that is, good enough for implementation)

The second condition can be (for instance) read as $\mathcal{A}'$ is approximately the same as $\mathcal{A}'_\delta$, for small enough δ.

[BLM+11] Bouyer, Larsen, Markey, Sankur, Thrane. Timed automata can always be made implementable *(CONCUR'11)*.

Automatic generation of an implementation

The (approx.) implementation synthesis problem

Given $\mathcal{A}$, build $\mathcal{A}'$ such that:

- $\mathcal{A}'$ 'identical' (e.g. bisimilar) to $\mathcal{A}$
- $\mathcal{A}'$ is 'robust' (that is, good enough for implementation)

The second condition can be (for instance) read as $\mathcal{A}'$ is approximately the same as $\mathcal{A}'_\delta$, for small enough δ.

Theorem

All timed automata are approximately implementable!
(for approx. bisimulation)

- Technical tool: region construction

[BLM+11] Bouyer, Larsen, Markey, Sankur, Thrane. Timed automata can always be made implementable *(CONCUR'11)*.

Automatic generation of an implementation

Methodology

- Design and verify $\mathcal{A}$
- Implement $\mathcal{A}'$ (automatically generated)

[BLM$^+$11] Bouyer, Larsen, Markey, Sankur, Thrane. Timed automata can always be made implementable *(CONCUR'11)*.

Automatic generation of an implementation

Methodology

- Design and verify $\mathcal{A}$
- Implement $\mathcal{A}'$ (automatically generated)

- ☺ Separates design and implementation
- ☹ $\mathcal{A}'$ is much bigger than $\mathcal{A}$

[BLM+11] Bouyer, Larsen, Markey, Sankur, Thrane. Timed automata can always be made implementable *(CONCUR'11)*.

Outline

1. Introduction
2. Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking
3. Robust realisability and control
   - Excess semantics
   - Strict semantics
4. Conclusion

Parameterized shrunk semantics for timed automata

A constraint $[a, b]$ is shrunk to $[a + \delta; b - \delta']$

[SBM11] Sankur, Bouyer, Markey. Shrinking Timed Automata *(FSTTCS'11)*.

Parameterized shrunk semantics for timed automata

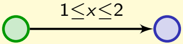
A constraint $[a, b]$ is shrunk to $[a + \delta; b - \delta']$

Why should we do that?

Abstract model

Impl. model

Models

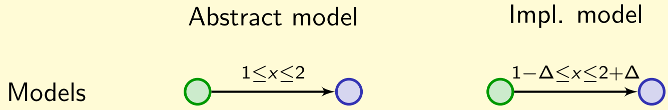


[SBM11] Sankur, Bouyer, Markey. Shrinking Timed Automata *(FSTTCS'11)*.

Parameterized shrunk semantics for timed automata

A constraint $[a, b]$ is shrunk to $[a + \delta; b - \delta']$

Why should we do that?



[SBM11] Sankur, Bouyer, Markey. Shrinking Timed Automata *(FSTTCS'11)*.

Parameterized shrunk semantics for timed automata

A constraint $[a, b]$ **is shrunk to** $[a + \delta; b - \delta']$

Why should we do that?

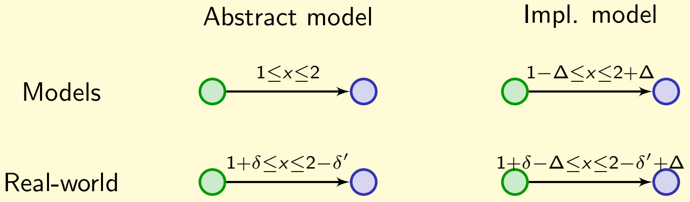
	Abstract model	Impl. model
Models	$\circ \xrightarrow{1 \leq x \leq 2} \circ$	$\circ \xrightarrow{1-\Delta \leq x \leq 2+\Delta} \circ$
Real-world	$\circ \xrightarrow{1+\delta \leq x \leq 2-\delta'} \circ$	

[SBM11] Sankur, Bouyer, Markey. Shrinking Timed Automata *(FSTTCS'11)*.

Parameterized shrunk semantics for timed automata

A constraint $[a, b]$ **is shrunk to** $[a + \delta; b - \delta']$

Why should we do that?



[SBM11] Sankur, Bouyer, Markey. Shrinking Timed Automata *(FSTTCS'11)*.

Parameterized shrunk semantics for timed automata

A constraint $[a, b]$ is shrunk to $[a + \delta; b - \delta']$

Why should we do that?

	Abstract model	Impl. model
Models	$\xrightarrow{1 \leq x \leq 2}$	$\xrightarrow{1-\Delta \leq x \leq 2+\Delta}$
Real-world	$\xrightarrow{1+\delta \leq x \leq 2-\delta'}$	$\xrightarrow{1+\delta-\Delta \leq x \leq 2-\delta'+\Delta}$

It is fine as soon as $[1 + \delta - \Delta; 2 - \delta' + \Delta] \subseteq [1; 2]$,
which is the case when $\delta, \delta' \geq \Delta$.

[SBM11] Sankur, Bouyer, Markey. Shrinking Timed Automata *(FSTTCS'11)*.

Parameterized shrunk semantics for timed automata

A constraint $[a, b]$ is shrunk to $[a + \delta; b - \delta']$

Summary of the approach

$\rightsquigarrow$ Shrink the clock constraints in the model, to prevent additional behaviours in the implementation

- If $\mathcal{B} = \mathcal{A}_{-\mathbf{k}\delta}$, then

$$\mathcal{B} \subseteq \mathtt{program}_\epsilon(\mathcal{B}) \subseteq \mathcal{B}_{f(\epsilon)} = \mathcal{A}_{-\mathbf{k}\delta + f(\epsilon)} \subseteq \mathcal{A}$$

[SBM11] Sankur, Bouyer, Markey. Shrinking Timed Automata *(FSTTCS'11)*.

Parameterized shrunk semantics – Discussion

What is the relevance of that approach?

Anticipate imprecisions to prevent additional behaviours in the real-world

Parameterized shrunk semantics – Discussion

What is the relevance of that approach?

Anticipate imprecisions to prevent additional behaviours in the real-world

Methodology

- Design and verify $\mathcal{A}$
- Implement $\mathcal{A}_{-\mathbf{k}\delta}$ (parameters are **k** and δ)

Parameterized shrunk semantics – Discussion

What is the relevance of that approach?

Anticipate imprecisions to prevent additional behaviours in the real-world

Methodology

- Design and verify $\mathcal{A}$
- Implement $\mathcal{A}_{-\mathbf{k}\delta}$ (parameters are **k** and δ)

$\rightsquigarrow$ This is a good approach for designing systems with strong/hard timing constraints

Parameterized shrunk semantics – Discussion

What is the relevance of that approach?

Anticipate imprecisions to prevent additional behaviours in the real-world

Methodology

- Design and verify $\mathcal{A}$
- Implement $\mathcal{A}_{-\mathbf{k}\delta}$ (parameters are $\mathbf{k}$ and δ)

$\rightsquigarrow$ This is a good approach for designing systems with strong/hard timing constraints

Make sure that no important behaviours are lost in $\mathcal{A}_{-\mathbf{k}\delta}$!!

Parameterized shrunk semantics – Algorithmics

The (parameterized) shrinkability problem

Find parameters **k** and δ such that:

- $\mathcal{A} \sqsubseteq_{\texttt{t.a.}} \mathcal{A}_{-\mathbf{k}\delta}$ (or $\mathcal{F} \sqsubseteq_{\texttt{t.a.}} \mathcal{A}_{-\mathbf{k}\delta}$ for some finite automaton $\mathcal{F}$)
  [shrinkability w.r.t. untimed simulation]
- $\mathcal{A}_{-\mathbf{k}\delta}$ is non-blocking whenever $\mathcal{A}$ is non-blocking
  [shrinkability w.r.t. non-blockingness]

Parameterized shrunk semantics – Algorithmics

The (parameterized) shrinkability problem

Find parameters $\mathbf{k}$ and δ such that:

- $\mathcal{A} \sqsubseteq_{\texttt{t.a.}} \mathcal{A}_{-\mathbf{k}\delta}$ (or $\mathcal{F} \sqsubseteq_{\texttt{t.a.}} \mathcal{A}_{-\mathbf{k}\delta}$ for some finite automaton $\mathcal{F}$)
  [shrinkability w.r.t. untimed simulation]
- $\mathcal{A}_{-\mathbf{k}\delta}$ is non-blocking whenever $\mathcal{A}$ is non-blocking
  [shrinkability w.r.t. non-blockingness]

Theorem

Parameterized shrinkability can be decided (in exponential time).

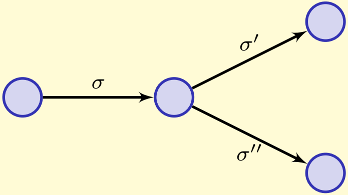
- Challenge: take care of the accumulation of perturbations
- Technical tools: parameterized shrunk DBM, max-plus equations
- Tool Shrinktech developed by Ocan Sankur [San13]

  http://www.lsv.ens-cachan.fr/Software/shrinktech/

[San13] Sankur. Shrinktech: A tool for the robustness analysis of timed automata *(CAV'13)*.

The case of non-blockingness

Non-blockingness

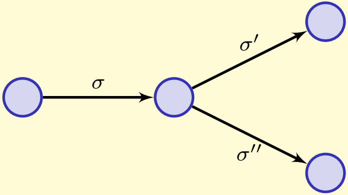


$$\begin{cases} G_\sigma \text{ guard of } \sigma \\ R_\sigma \text{ reset of } \sigma \end{cases}$$

Whenever σ is taken, either σ' or σ'' is eventually firable.

The case of non-blockingness

Non-blockingness



$\begin{cases} G_\sigma \text{ guard of } \sigma \\ R_\sigma \text{ reset of } \sigma \end{cases}$

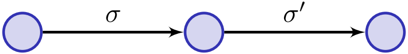
Whenever σ is taken, either σ' or σ'' is eventually firable.

Fix-point characterization

Let G_σ denote the **guards** of the timed automaton. It is non-blocking iff,

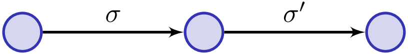
$$\forall \sigma, \quad [\![G_\sigma]\!] \subseteq \bigcup_{l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3} \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!])).$$

Technical tools: shrunk DBMs...

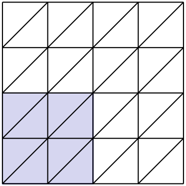


$$[\![G_\sigma]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!]))$$

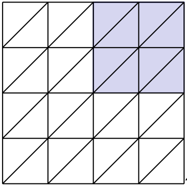
Technical tools: shrunk DBMs...



$$[\![G_\sigma]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!]))$$

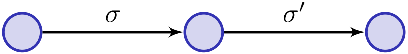


$\subseteq \mathsf{Unreset}_y \left(\mathsf{Pre}_{\mathsf{time}} \left(\right. \right.$

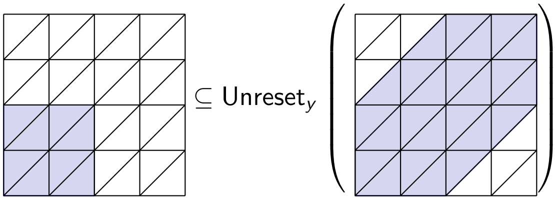


$\left. \left. \right) \right)$

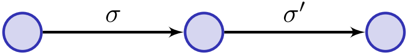
Technical tools: shrunk DBMs...



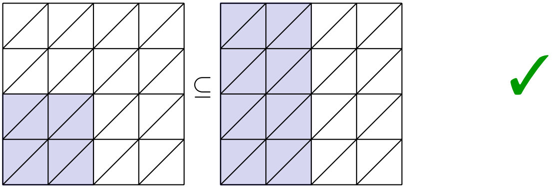
$$[\![G_\sigma]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!]))$$



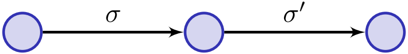
Technical tools: shrunk DBMs...



$$[\![G_\sigma]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!]))$$



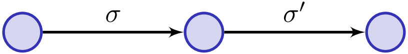
Technical tools: shrunk DBMs...



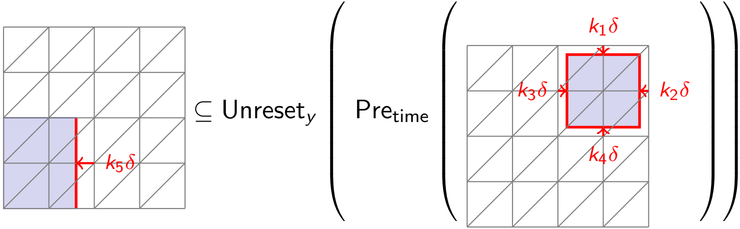
$$[\![\langle G_\sigma \rangle_{-\vec{k}\delta}]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![\langle G_{\sigma'} \rangle_{-\vec{k}\delta}]\!])) \qquad ?$$

Determine $\vec{k}$

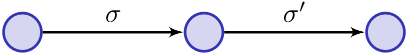
Technical tools: shrunk DBMs...



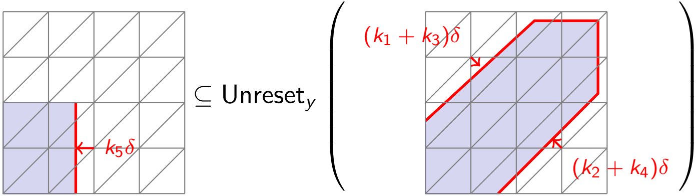
$$[\![\langle G_\sigma\rangle_{-\vec{k}\delta}]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![\langle G_{\sigma'}\rangle_{-\vec{k}\delta}]\!])) \qquad ?$$



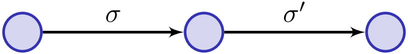
Technical tools: shrunk DBMs...



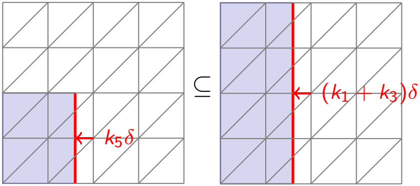
$$[\![\langle G_\sigma \rangle_{-\vec{k}\delta}]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![\langle G_{\sigma'} \rangle_{-\vec{k}\delta}]\!])) \qquad ?$$



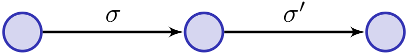
Technical tools: shrunk DBMs...



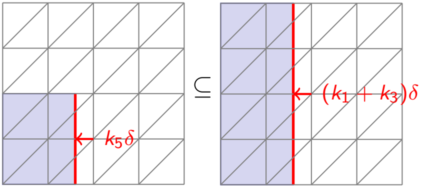
$$[\![\langle G_\sigma \rangle_{-\vec{k}\delta}]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![\langle G_{\sigma'} \rangle_{-\vec{k}\delta}]\!])) \qquad ?$$



Technical tools: shrunk DBMs...and max-plus equations



$$[\![\langle G_\sigma \rangle_{-\vec{k}\delta}]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![\langle G_{\sigma'} \rangle_{-\vec{k}\delta}]\!])) \qquad ?$$



Then, $\vec{k}$ should satisfy

$$k_5 \geq k_1 + k_3 \qquad \text{that is,} \qquad k_5 = \max(k_5, k_1 + k_3)$$

In this case, the above inclusion equation holds for small enough δ's

$$[\![\langle G_\sigma \rangle_{-\vec{k}\delta}]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![\langle G_{\sigma'} \rangle_{-\vec{k}\delta}]\!]))$$
$$\Leftrightarrow$$
$$k_5 = \max(k_5, k_1 + k_3).$$

$$[\![\langle G_\sigma\rangle_{-\vec{k}\delta}]\!] \subseteq \mathrm{Unreset}_{R_\sigma}(\mathrm{Pre}_{\mathrm{time}}([\![\langle G_{\sigma'}\rangle_{-\vec{k}\delta}]\!]))$$
$$\Leftrightarrow$$
$$k_5 = \max(k_5, k_1 + k_3).$$

Key Theorem

Let $\vec{M} = f(\vec{M})$ be a **fixpoint equation on zones**, and $\vec{M}$ a solution.
f uses $\mathrm{Pre}_{\mathrm{time}}()$, $\cap$, $\mathrm{Unreset}_{\cdot}()$.
For any $\vec{k} \in \mathbb{N}^n_{>0}$,

$$\langle\vec{M}\rangle_{-\vec{k}\delta} = f(\langle\vec{M}\rangle_{-\vec{k}\delta}) \qquad \forall \text{ small } \delta > 0$$
$$\Leftrightarrow$$
$$\vec{k} = \varphi(\vec{k}),$$

where φ is a **max-plus expression**.

$$[\![\langle G_\sigma \rangle_{-\vec{k}\delta}]\!] \subseteq \mathrm{Unreset}_{R_\sigma}(\mathrm{Pre}_{\mathrm{time}}([\![\langle G_{\sigma'} \rangle_{-\vec{k}\delta}]\!]))$$
$$\Leftrightarrow$$
$$k_5 = \max(k_5, k_1 + k_3).$$

Key Theorem

Let $\vec{M} = f(\vec{M})$ be a **fixpoint equation on zones**, and $\vec{M}$ a solution.

f uses $\mathrm{Pre}_{\mathrm{time}}()$, $\cap$, $\mathrm{Unreset}_{\cdot}()$.

For any $\vec{k} \in \mathbb{N}^n_{>0}$,

$$\langle \vec{M} \rangle_{-\vec{k}\delta} = f(\langle \vec{M} \rangle_{-\vec{k}\delta}) \qquad \forall \text{ small } \delta > 0$$
$$\Leftrightarrow$$
$$\vec{k} = \varphi(\vec{k}),$$

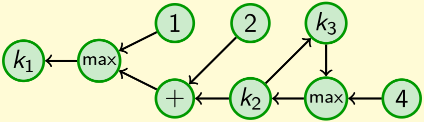
where φ is a **max-plus expression**.

$\rightsquigarrow$ Max-plus algebra: the above fixpoint equations can be solved in polynomial time

Solving max-plus equations

Max-plus graph

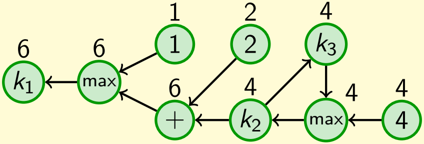
$k_1 \geq \max(1, 2 + k_2) \ \wedge \ k_3 \geq k_2 \ \wedge \ k_2 \geq \max(4, k_3)$



Solving max-plus equations

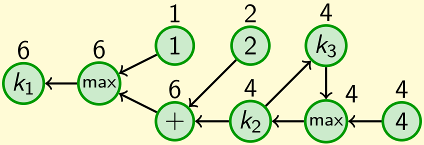
Max-plus graph

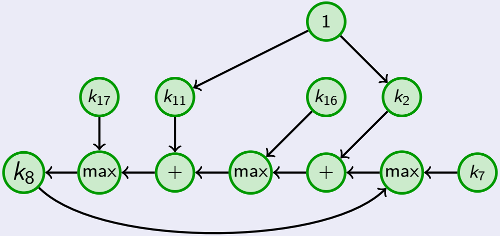
$k_1 \geq \max(1, 2 + k_2) \ \wedge \ k_3 \geq k_2 \ \wedge \ k_2 \geq \max(4, k_3)$



Solving max-plus equations

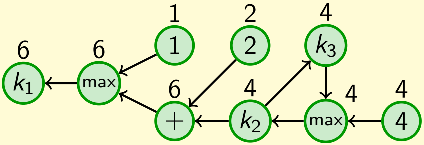
Max-plus graph

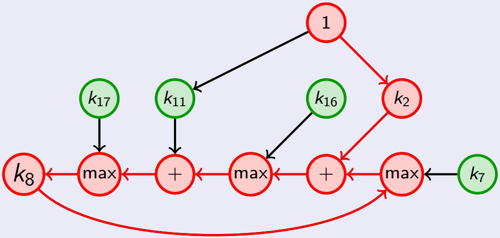




Solving max-plus equations

Max-plus graph





$\rightsquigarrow$ No solution!

Summary of shrinkability

Deciding shrinkability

Apply theorem to following fix-point equations:

- Non-blockingness:

$$\forall\sigma,\quad [\![G_\sigma]\!] \subseteq \bigcup_{l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3} \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!])).$$

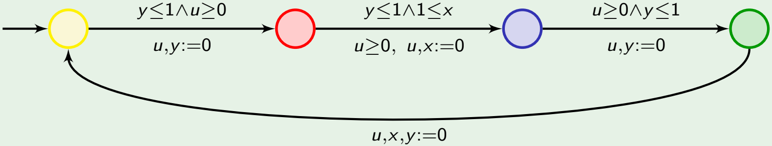
  (Do technical work to remove the union)

- Time-abstract simulation ($\mathcal{A} \sqsubseteq_{\text{t.a.}} \mathcal{A}_{-\delta\vec{k}}$):

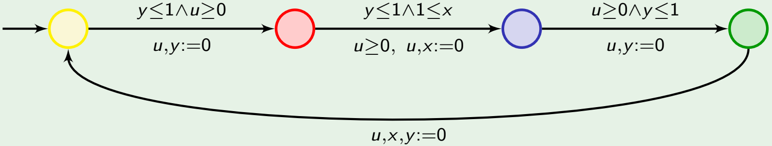
$$[\![M_{l,r}]\!] = \bigcap_{\sigma\in\Sigma} \bigcap_{(l,r)\xrightarrow{\sigma}(l',r')} \mathsf{Pre}_{\mathsf{time}}(\mathsf{Unreset}_{R_\sigma}([\![M_{l',r'}]\!]) \cap [\![G_\sigma]\!]),$$

  where $M_{l,r}$ is the time-abstract simulator set of the region (l, r).

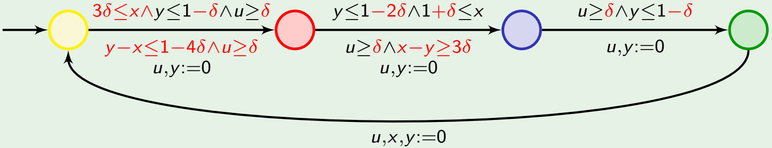
Example


$y \leq 1 \wedge u \geq 0$
$u,y{:=}0$
$y \leq 1 \wedge 1 \leq x$
$u \geq 0,\ u,x{:=}0$
$u \geq 0 \wedge y \leq 1$
$u,y{:=}0$
$u,x,y{:=}0$

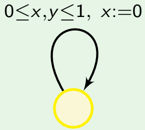
Example



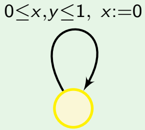
The largest shrunk automaton which is correct w.r.t. untimed simulation and non-blockingness (for all $\delta \in [0, \frac{1}{4}]$) is:



Counter-example



Counter-example

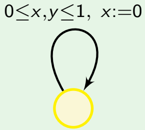


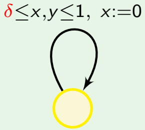
There is no shrunk automaton which is correct w.r.t. non-blockingness.
Indeed, the max-plus equations we obtain are:

$$\begin{cases} \dots \\ k_8 = \max(k_{17}, k_{11} + \max(k_{16}, k_2 + \max(k_7, k_8))) \\ k_{11} = \max(1, k_{11}) \end{cases}$$

which has no solution!
(remember the max-plus graph with no solution)

Counter-example





is blocking

There is no shrunk automaton which is correct w.r.t. non-blockingness. Indeed, the max-plus equations we obtain are:

$$\begin{cases} \dots \\ k_8 = \max(k_{17}, k_{11} + \max(k_{16}, k_2 + \max(k_7, k_8))) \\ k_{11} = \max(1, k_{11}) \end{cases}$$

which has no solution!
(remember the max-plus graph with no solution)

Partial conclusion

- We have presented three methods for verifying robust correctness, hence correct implementation

Partial conclusion

- We have presented three methods for verifying robust correctness, hence correct implementation
- Same complexities as standard model-checking!

Partial conclusion

- We have presented three methods for verifying robust correctness, hence correct implementation
- Same complexities as standard model-checking!
- Technical tools:
  - Extended region automaton
  - Shrunk DBMs
  - And also characterization of reachability relations in timed automata (hidden in this presentation)

Partial conclusion

- We have presented three methods for verifying robust correctness, hence correct implementation
- Same complexities as standard model-checking!
- Technical tools:
  - Extended region automaton
  - Shrunk DBMs
  - And also characterization of reachability relations in timed automata (hidden in this presentation)
- What is missing:
  - A symbolic approach
  - A tool support
    - Shinktech is a prototype for the shrinking approach

      `http://www.lsv.ens-cachan.fr/Software/shrinktech/`
  - Stochastic approach (see later)

Outline

1. Introduction

2. Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking

3. Robust realisability and control
   - Excess semantics
   - Strict semantics

4. Conclusion

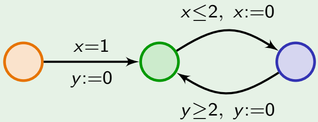
Robust realisability

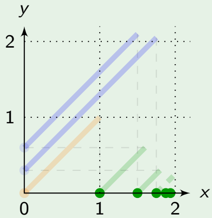
Here, a strategy in a timed automaton is a way to resolve (time and action) non-determinism

Robust realisability

Here, a strategy in a timed automaton is a way to resolve (time and action) non-determinism

Example



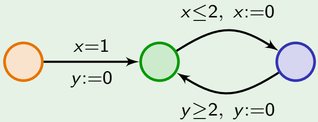


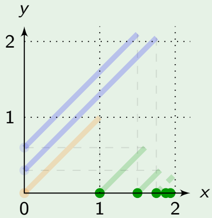
Strategy: in location ◯ with value x, delay $\frac{2-x}{2}$

Robust realisability

Here, a strategy in a timed automaton is a way to resolve (time and action) non-determinism

Example





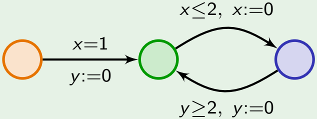
Strategy: in location ◯ with value x, delay $\frac{2-x}{2}$

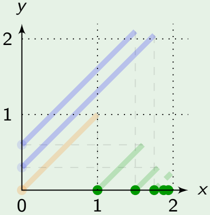
- This strategy requires infinite precision

Robust realisability

Here, a strategy in a timed automaton is a way to resolve (time and action) non-determinism

Example





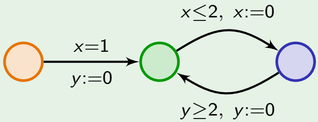
Strategy: in location  with value x, delay $\frac{2-x}{2}$

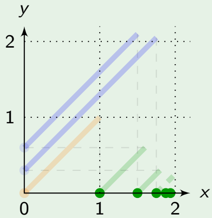
- This strategy requires infinite precision
- In practice, when x is close to 2, no additional delay is supported: the run is theoretically infinite, but it is actually blocking

Robust realisability

Here, a strategy in a timed automaton is a way to resolve (time and action) non-determinism

Example





Strategy: in location ◯ with value x, delay $\frac{2-x}{2}$

- This strategy requires infinite precision
- In practice, when x is close to 2, no additional delay is supported: the run is theoretically infinite, but it is actually blocking
- And that is unavoidable

Robust realisability

Here, a strategy in a timed automaton is a way to resolve (time and action) non-determinism

Idea of robust realisability

Synthesize strategies that realise some property, even under perturbations: strategies should adapt to previous imprecisions

$\rightsquigarrow$ develop a theory of robust strategies that tolerate errors/imprecisions and avoid convergence

Game semantics of a timed automaton

Game semantics $\mathcal{G}_\delta(\mathcal{A})$ of timed automaton $\mathcal{A}$...

... between Controller and Perturbator:

- from (ℓ, v), Controller suggests a delay $d \geq \delta$ and a next edge $e = (\ell \xrightarrow{g,Y} \ell')$ that is available after delay d
- Perturbator then chooses a perturbation $\epsilon \in [-\delta; +\delta]$
- Next state is $(\ell', (v + d + \epsilon)[Y \leftarrow 0])$

Game semantics of a timed automaton

Game semantics $\mathcal{G}_\delta(\mathcal{A})$ of timed automaton $\mathcal{A}$...

... between Controller and Perturbator:

- from (ℓ, v), Controller suggests a delay $d \geq \delta$ and a next edge $e = (\ell \xrightarrow{g,Y} \ell')$ that is available after delay d
- Perturbator then chooses a perturbation $\epsilon \in [-\delta; +\delta]$
- Next state is $(\ell', (v + d + \epsilon)[Y \leftarrow 0])$

Note: when $\delta = 0$, this is the standard semantics of timed automata.

Game semantics of a timed automaton

Game semantics $\mathcal{G}_\delta(\mathcal{A})$ of timed automaton $\mathcal{A}$...

... between Controller and Perturbator:

- from (ℓ, v), Controller suggests a delay $d \geq \delta$ and a next edge $e = (\ell \xrightarrow{g,Y} \ell')$ that is available after delay d
- Perturbator then chooses a perturbation $\epsilon \in [-\delta; +\delta]$
- Next state is $(\ell', (v + d + \epsilon)[Y \leftarrow 0])$

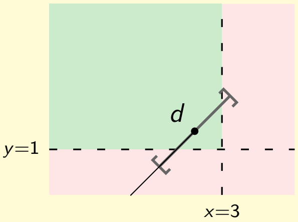
Note: when $\delta = 0$, this is the standard semantics of timed automata.

A δ-robust strategy for Controller is then a strategy that satisfies the expected property, whatever plays Perturbator.

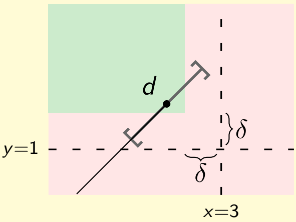
Two possible semantics

Consider a transition with guard $x \leq 3 \wedge y \geq 1$:

excess semantics



strict semantics



Outline

The excess game semantics

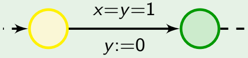
Constraints may not be satisfied after the perturbation
only $v + d$ should satisfy g

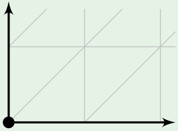
[BMS12] Bouyer, Markey, Sankur. Robust reachability in timed automata: A game-based approach *(ICALP'12)*.

The excess game semantics

Constraints may not be satisfied after the perturbation
only $v + d$ should satisfy g

Example



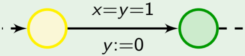


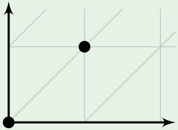
[BMS12] Bouyer, Markey, Sankur. Robust reachability in timed automata: A game-based approach *(ICALP'12)*.

The excess game semantics

Constraints may not be satisfied after the perturbation
only $v + d$ should satisfy g

Example



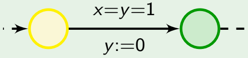


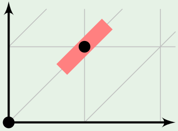
[BMS12] Bouyer, Markey, Sankur. Robust reachability in timed automata: A game-based approach *(ICALP'12)*.

The excess game semantics

Constraints may not be satisfied after the perturbation
only $v + d$ should satisfy g

Example



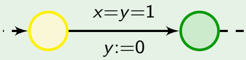


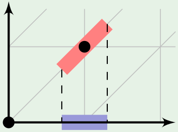
[BMS12] Bouyer, Markey, Sankur. Robust reachability in timed automata: A game-based approach *(ICALP'12)*.

The excess game semantics

Constraints may not be satisfied after the perturbation
only $v + d$ should satisfy g

Example



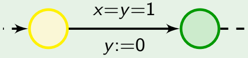


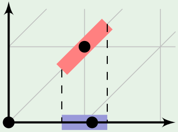
[BMS12] Bouyer, Markey, Sankur. Robust reachability in timed automata: A game-based approach *(ICALP'12)*.

The excess game semantics

Constraints may not be satisfied after the perturbation
only $v + d$ should satisfy g

Example



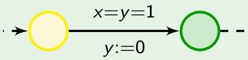


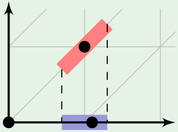
[BMS12] Bouyer, Markey, Sankur. Robust reachability in timed automata: A game-based approach *(ICALP'12)*.

The excess game semantics

Constraints may not be satisfied after the perturbation
only $v + d$ should satisfy g

Example





$\rightsquigarrow$ Allows simple design of constraints, ensures divergence of time, avoids convergence phenomena

[BMS12] Bouyer, Markey, Sankur. Robust reachability in timed automata: A game-based approach *(ICALP'12)*.

The excess game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a δ-robust strategy that achieves a given goal.

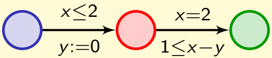
The excess game semantics – Algorithmics

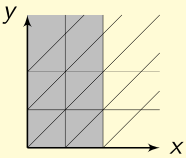
The (parameterized) synthesis problem

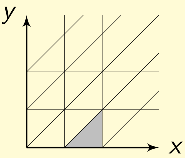
Synthesize $\delta > 0$ and a δ-robust strategy that achieves a given goal.

Two challenges

1. Accumulation of perturbations:







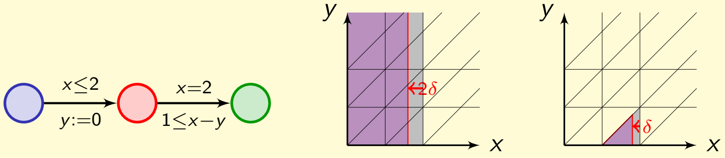
The excess game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a δ-robust strategy that achieves a given goal.

Two challenges

1. Accumulation of perturbations:



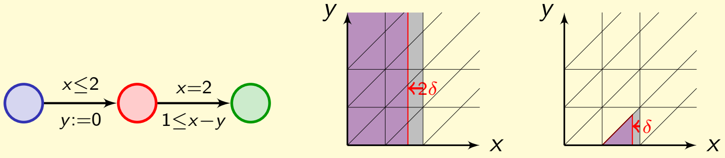
The excess game semantics – Algorithmics

The (parameterized) synthesis problem

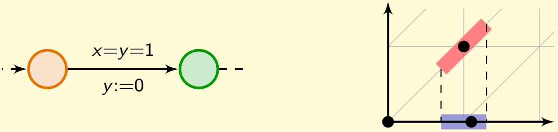
Synthesize $\delta > 0$ and a δ-robust strategy that achieves a given goal.

Two challenges

1. Accumulation of perturbations:



2. New regions become reachable



The excess game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a δ-robust strategy that achieves a given goal.

Theorem

The parameterized synthesis problem for reachability properties is decidable and EXPTIME-complete. Furthermore, uniform winning strategies (w.r.t. δ) can be computed.

The excess game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a δ-robust strategy that achieves a given goal.

Theorem

The parameterized synthesis problem for reachability properties is decidable and EXPTIME-complete. Furthermore, uniform winning strategies (w.r.t. δ) can be computed.

- Technical tool: a region-based refined game abstraction, shrunk DBMs

The excess game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a δ-robust strategy that achieves a given goal.

Theorem

The parameterized synthesis problem for reachability properties is decidable and EXPTIME-complete. Furthermore, uniform winning strategies (w.r.t. δ) can be computed.

- Technical tool: a region-based refined game abstraction, shrunk DBMs
- ☺ Extends to two-player games (i.e. to real control problems)
- ☹ Only valid for reachability properties

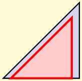
The excess game semantics – Algorithm overview

1. (Forward) Construct an equivalent finite turn-based game $\mathbf{F}(\mathcal{A})$ (based on regions)
2. Solve it
3. (Backward) Construct winning states in $\mathcal{G}_\delta(\mathcal{A})$, and deduce δ_0

The excess game semantics – Algorithm overview

1. (Forward) Construct an equivalent finite turn-based game $\mathbf{F}(\mathcal{A})$ (based on regions)
2. Solve it
3. (Backward) Construct winning states in $\mathcal{G}_\delta(\mathcal{A})$, and deduce δ_0

Winning states will be described by shrinkings of regions:



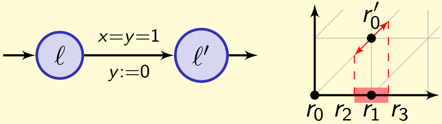
$r - \delta P$

One can win from a region r in $\mathbf{F}(\mathcal{A})$

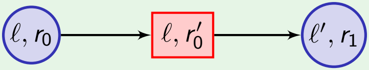
$\Updownarrow$

one can win from a shrinking of r in $\mathcal{G}_\delta(\mathcal{A})$

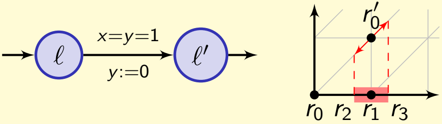
Construction of the finite turn-based game



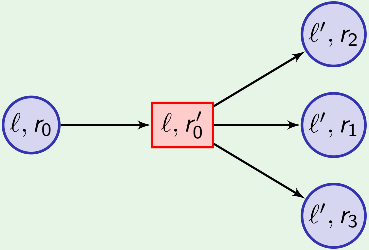
region automaton:



Construction of the finite turn-based game

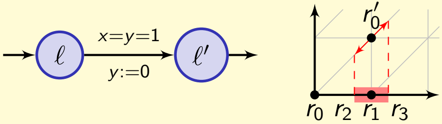


Extended region automaton:

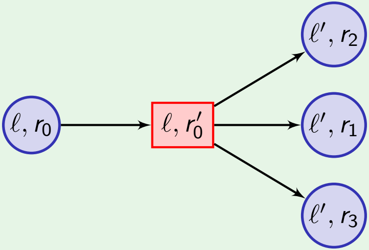


Idea: We win from *some* shrinking of r_0, if, and only if we win from *some* shrinkings of r_1, r_2, r_3.

Construction of the finite turn-based game

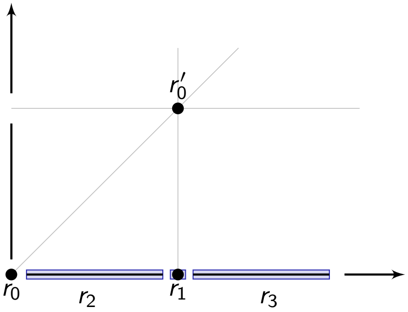


Extended region automaton:

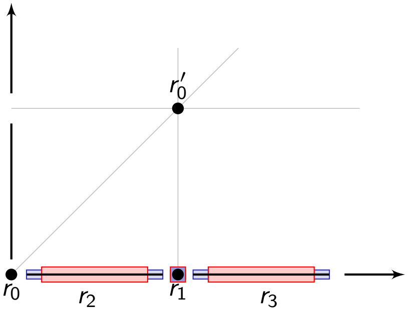


Idea: ~~We win from *some* shrinking of r_0, if, and only if we win from *some* shrinkings of r_1, r_2, r_3.~~

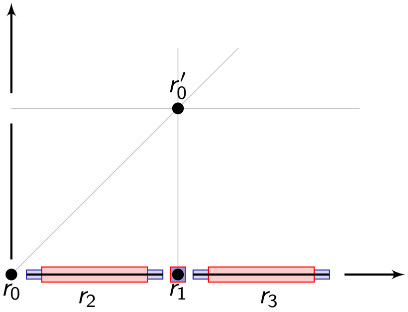
Assume that we win from **some** shrinkings of r_1, r_2, r_3.



Assume that we win from **some** shrinkings of r_1, r_2, r_3.

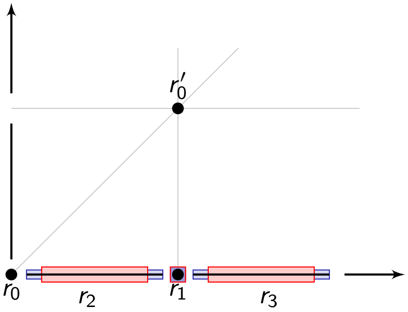


Assume that we win from **some** shrinkings of r_1, r_2, r_3.



Can these be combined to a winning strategy from r_0?

Assume that we win from **some** shrinkings of r_1, r_2, r_3.

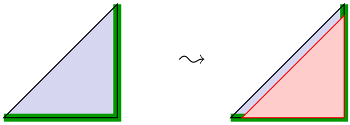


Can these be combined to a winning strategy from r_0?

No: we don't have a strategy for valuations around r_1.

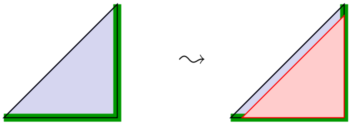
Solution: Look for a shrinking of some regions with constraints.

A constrained region is a region with some marked facets.
A shrinking of a constrained region does not shrink from marked facets.

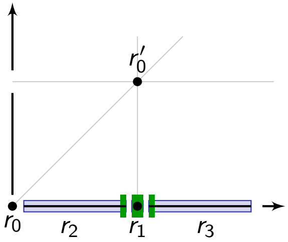


Solution: Look for a shrinking of some regions with constraints.

A constrained region is a region with some marked facets.
A shrinking of a constrained region does not shrink from marked facets.

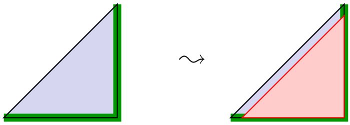


We win from r_0 iff we win from constrained shrinkings of r_1, r_2, r_3.

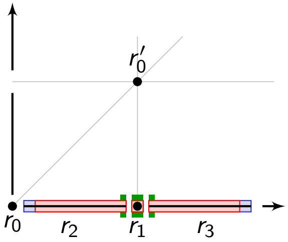


Solution: Look for a shrinking of some regions with constraints.

A constrained region is a region with some marked facets.
A shrinking of a constrained region does not shrink from marked facets.

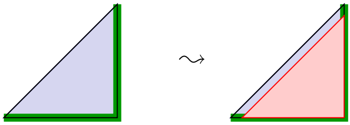


We win from r_0 iff we win from constrained shrinkings of r_1, r_2, r_3.

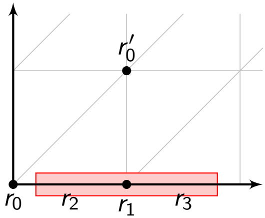


Solution: Look for a shrinking of some regions with constraints.

A constrained region is a region with some marked facets.
A shrinking of a constrained region does not shrink from marked facets.

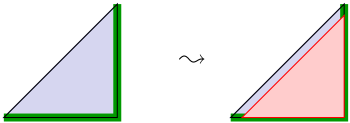


In fact,

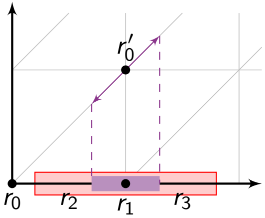


Solution: Look for a shrinking of some regions with constraints.

A constrained region is a region with some marked facets.
A shrinking of a constrained region does not shrink from marked facets.



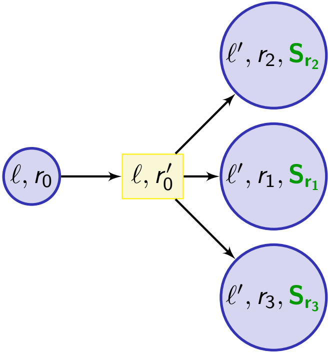
In fact,



OK, we have a strategy for all the points in the violet area.

Finite game $\mathbf{F}(\mathcal{A})$

Shrinking constraint for region r is represented by a boolean matrix $\mathbf{S_r}$.



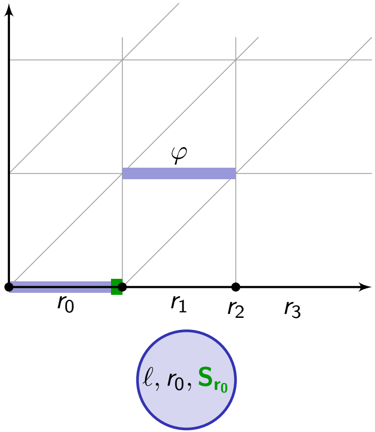
Theorem

Controller wins in $\mathcal{G}_\delta(\mathcal{A})$ for all $\delta \in [0, \delta_0]$ for some $\delta_0 > 0$

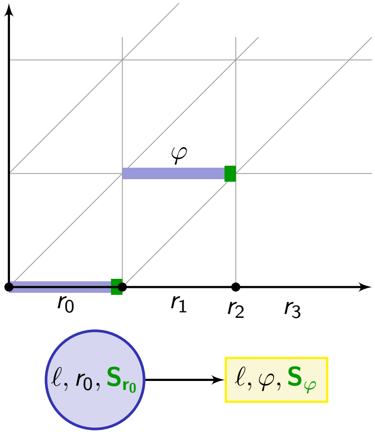
$\Updownarrow$

Controller wins in $\mathbf{F}(\mathcal{A})$.

Details on the definition of $\mathbf{F}(\mathcal{A})$



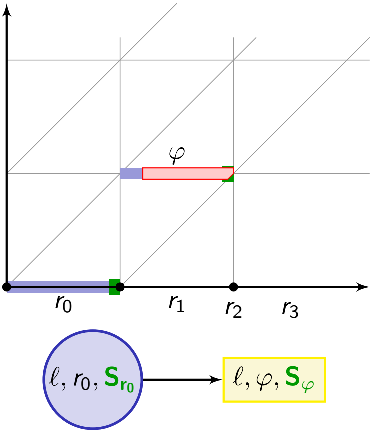
Details on the definition of $\mathbf{F}(\mathcal{A})$



$\mathbf{S}_\varphi$ is defined such that:

Controller wins from *some* shrinking of $(\varphi, \mathbf{S}_\varphi)$ **iff**
Controller wins from *some* shrinking of $(r_0, \mathbf{S_{r_0}})$.

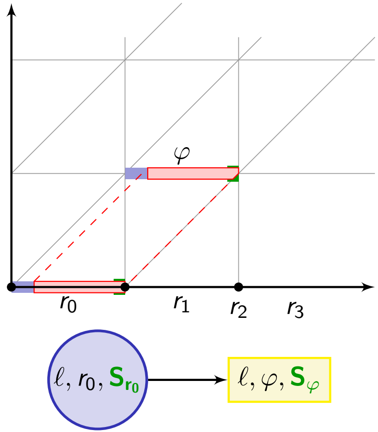
Details on the definition of $\mathbf{F}(\mathcal{A})$



$\mathbf{S}_\varphi$ is defined such that:

Controller wins from *some* shrinking of $(\varphi, \mathbf{S}_\varphi)$ **iff**
Controller wins from *some* shrinking of $(r_0, \mathbf{S}_{\mathbf{r_0}})$.

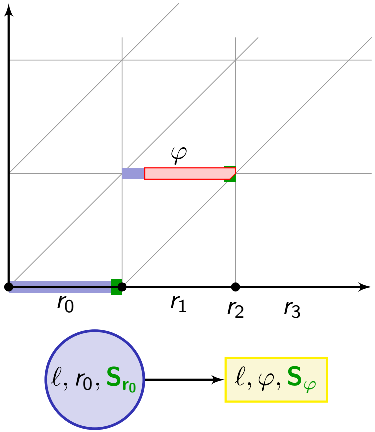
Details on the definition of $\mathbf{F}(\mathcal{A})$



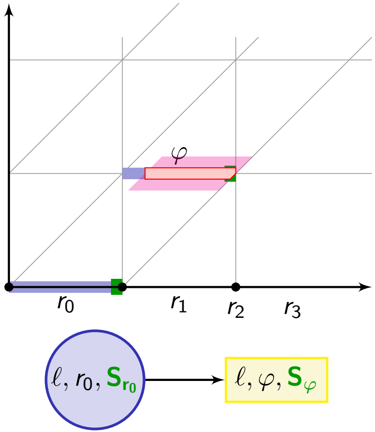
$\mathbf{S}_\varphi$ is defined such that:

Controller wins from *some* shrinking of $(\varphi, \mathbf{S}_\varphi)$ **iff**
Controller wins from *some* shrinking of $(r_0, \mathbf{S_{r_0}})$.

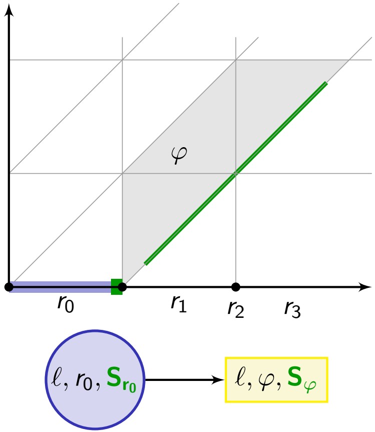
Details on the definition of $\mathbf{F}(\mathcal{A})$



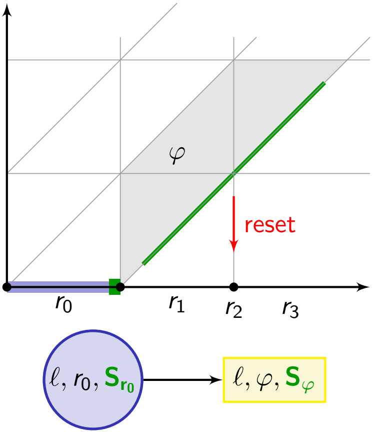
Details on the definition of $\mathbf{F}(\mathcal{A})$



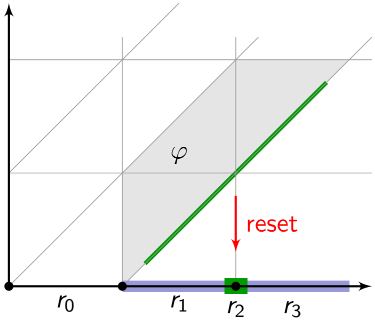
Details on the definition of $\mathbf{F}(\mathcal{A})$

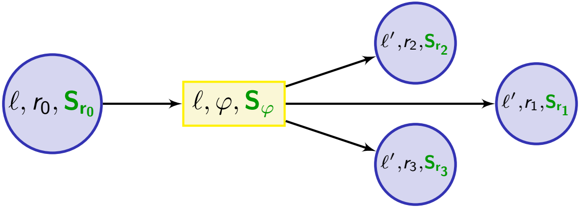


Details on the definition of $\mathbf{F}(\mathcal{A})$

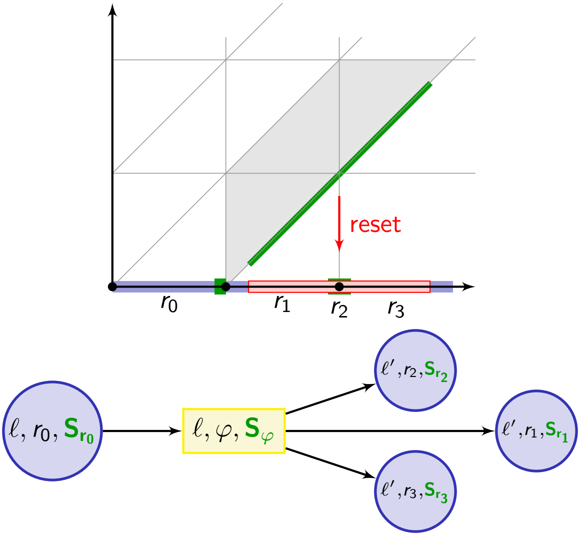


Details on the definition of $\mathbf{F}(\mathcal{A})$

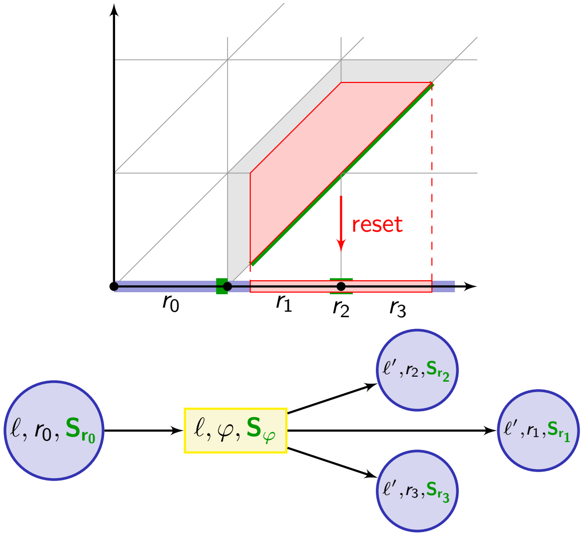




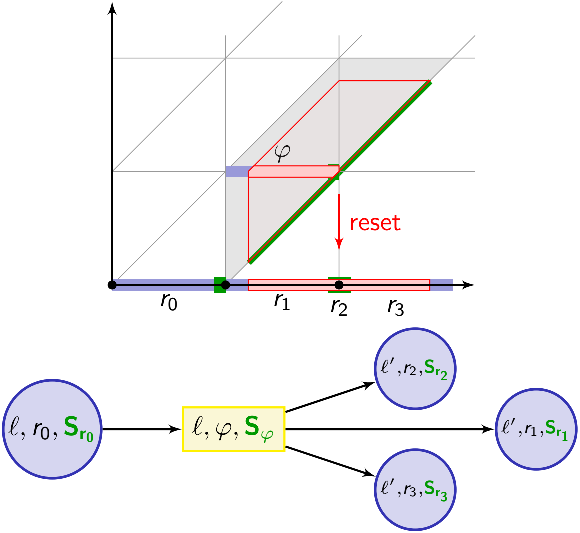
Constructing a winning strategy from $\mathbf{F}(\mathcal{A})$



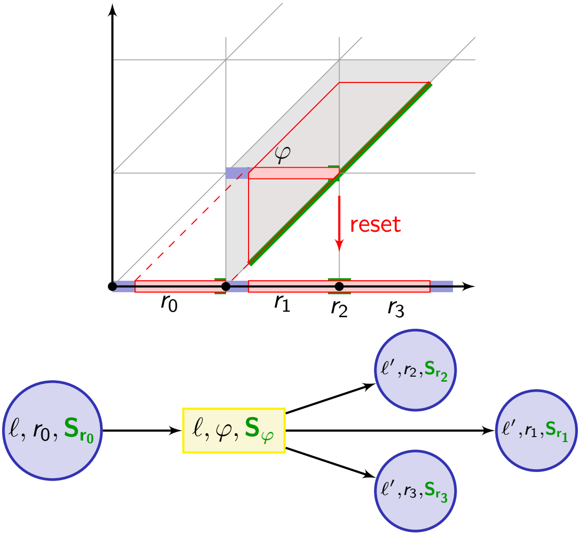
Constructing a winning strategy from $\mathbf{F}(\mathcal{A})$



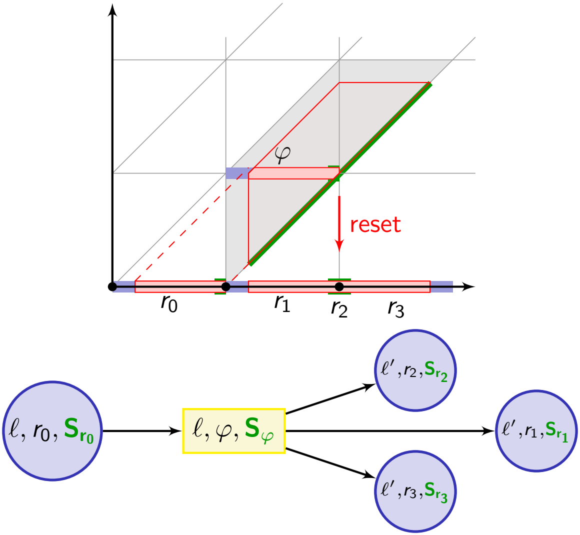
Constructing a winning strategy from $\mathbf{F}(\mathcal{A})$



Constructing a winning strategy from $\mathbf{F}(\mathcal{A})$



Constructing a winning strategy from $\mathbf{F}(\mathcal{A})$



- Each step of the backward propagation gives an upper bound on δ.

EXPTIME-hardness

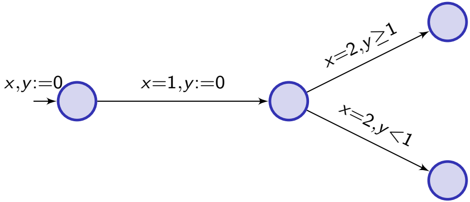
Usual semantics in timed automata can encode reachability in linearly bounded Turing machines (PSPACE-complete).

Robust semantics in timed automata can encode reachability in alternating linearly bounded Turing machines (EXPTIME-complete).

EXPTIME-hardness

Usual semantics in timed automata can encode reachability in linearly bounded Turing machines (PSPACE-complete).

Robust semantics in timed automata can encode reachability in alternating linearly bounded Turing machines (EXPTIME-complete).



Perturbator has a strategy to choose between any of the two branches.

- Top branch: make the first transition earlier
- Bottom branch: delay the first transition

Outline

The strict game semantics

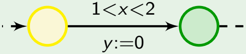
Constraints have to be satisfied after the perturbation:
$v + d + \epsilon$ should satisfy g for every $\epsilon \in [-\delta; +\delta]$

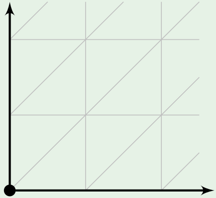
[SBMR13] Sankur, Bouyer, Markey, Reynier. Robust Controller Synthesis in Timed Automata *(CONCUR'13)*.

The strict game semantics

Constraints have to be satisfied after the perturbation:
$v + d + \epsilon$ should satisfy g for every $\epsilon \in [-\delta; +\delta]$

Example

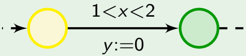


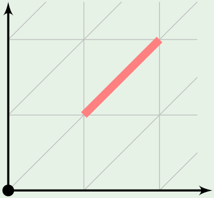


The strict game semantics

Constraints have to be satisfied after the perturbation:
$v + d + \epsilon$ should satisfy g for every $\epsilon \in [-\delta; +\delta]$

Example



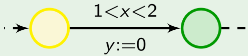


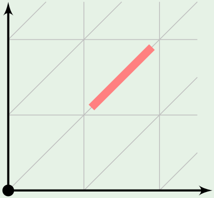
[SBMR13] Sankur, Bouyer, Markey, Reynier. Robust Controller Synthesis in Timed Automata *(CONCUR'13)*.

The strict game semantics

Constraints have to be satisfied after the perturbation:
$v + d + \epsilon$ should satisfy g for every $\epsilon \in [-\delta; +\delta]$

Example



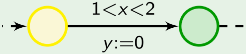


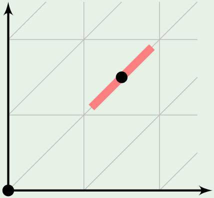
[SBMR13] Sankur, Bouyer, Markey, Reynier. Robust Controller Synthesis in Timed Automata *(CONCUR'13)*.

The strict game semantics

Constraints have to be satisfied after the perturbation:
$v + d + \epsilon$ should satisfy g for every $\epsilon \in [-\delta; +\delta]$

Example



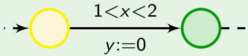


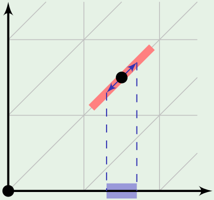
[SBMR13] Sankur, Bouyer, Markey, Reynier. Robust Controller Synthesis in Timed Automata *(CONCUR'13)*.

The strict game semantics

Constraints have to be satisfied after the perturbation:
$v + d + \epsilon$ should satisfy g for every $\epsilon \in [-\delta; +\delta]$

Example



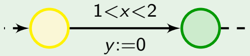


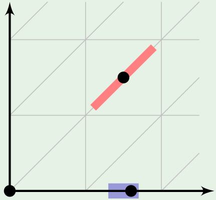
[SBMR13] Sankur, Bouyer, Markey, Reynier. Robust Controller Synthesis in Timed Automata *(CONCUR'13)*.

The strict game semantics

Constraints have to be satisfied after the perturbation:
$v + d + \epsilon$ should satisfy g for every $\epsilon \in [-\delta; +\delta]$

Example



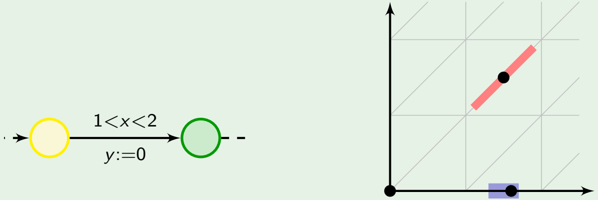


[SBMR13] Sankur, Bouyer, Markey, Reynier. Robust Controller Synthesis in Timed Automata *(CONCUR'13)*.

The strict game semantics

Constraints have to be satisfied after the perturbation:
$v + d + \epsilon$ should satisfy g for every $\epsilon \in [-\delta; +\delta]$

Example



$\rightsquigarrow$ Strongly ensures timing constraints, ensures divergence of time, prevents converging phenomena

[SBMR13] Sankur, Bouyer, Markey, Reynier. Robust Controller Synthesis in Timed Automata *(CONCUR'13)*.

The strict game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a δ-robust strategy that achieves a given goal.

The strict game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a δ-robust strategy that achieves a given goal.

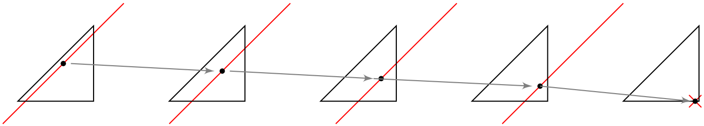
Theorem

The synthesis problem for Büchi properties is decidable and PSPACE-complete. Furthermore, δ is at most doubly-exponential, and uniform winning strategies (w.r.t. δ) can be computed.

The problem consists in finding cycles that do not become blocked.

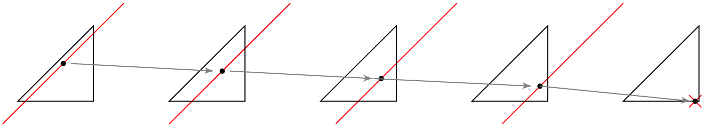
The problem consists in finding cycles that do not become blocked.

- A converging phenomena:

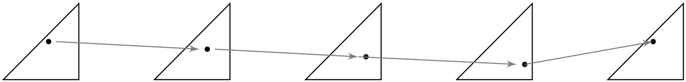


The problem consists in finding cycles that do not become blocked.

- A converging phenomena:



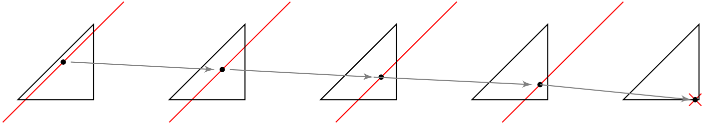
- No convergence:



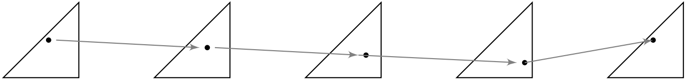
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

The problem consists in finding cycles that do not become blocked.

- A converging phenomena:



- No convergence:

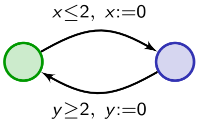


Tools for solving the synthesis problem

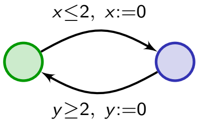
- Orbit graphs, forgetful cycles [AB11]
- Forgetful orbit graph $\Leftrightarrow$ no convergence phenomena
  $\leadsto$ strong relation with thick automata.

[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

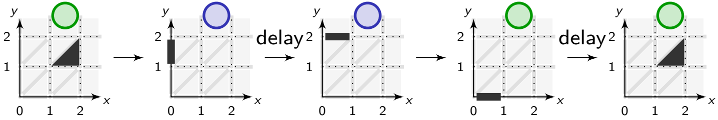
Technical tool: the (folded) orbit graph



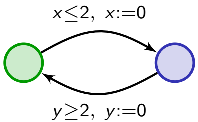
Technical tool: the (folded) orbit graph



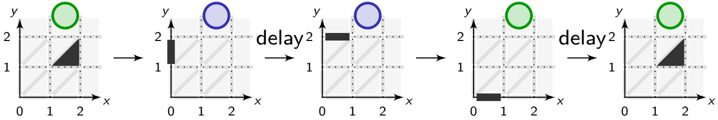
A region cycle:



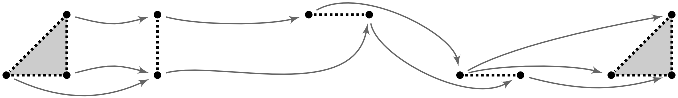
Technical tool: the (folded) orbit graph



A region cycle:

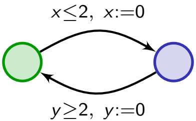


The corresponding orbit graph:

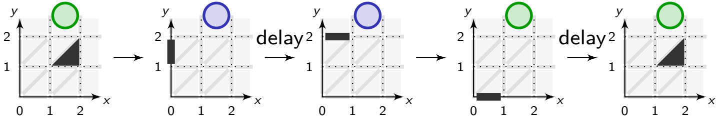


$\rightsquigarrow$ stores the reachability relation between vertices of the regions

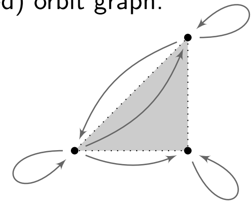
Technical tool: the (folded) orbit graph



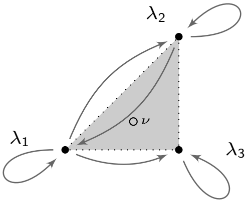
A region cycle:



The corresponding (folded) orbit graph:



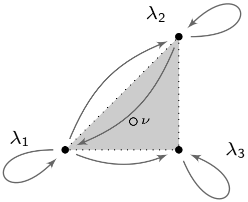
Understanding the folded orbit graph



$\nu = \vec{\lambda} \cdot \vec{v}$ (convex combination of the vertices)

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

Understanding the folded orbit graph



$\nu = \vec{\lambda} \cdot \vec{v}$ (convex combination of the vertices)

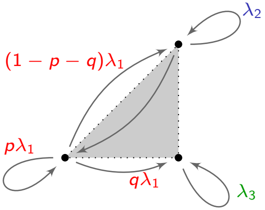
Reachability relation [Pur00]

Given a region cycle ρ, and valuation $\nu = \vec{\lambda} \cdot \vec{v}$,

$\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda}' \vec{v} \quad \Leftrightarrow$ $\vec{\lambda}'$ is computed by distributing each λ_v to its successors following a probability distribution

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

Understanding the folded orbit graph



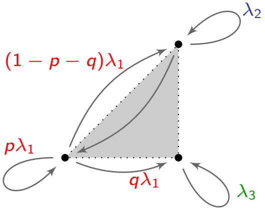
Reachability relation [Pur00]

Given a region cycle ρ, and valuation $\nu = \vec{\lambda} \cdot \vec{v}$,

$\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda'}\vec{v} \quad \Leftrightarrow$ $\vec{\lambda'}$ is computed by distributing each λ_v to its successors following a probability distribution

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

Understanding the folded orbit graph



$$\begin{cases} \lambda_1' = p\lambda_1 \\ \\ \end{cases}$$

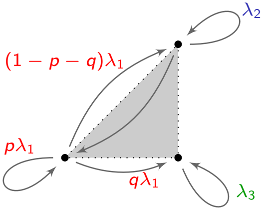
Reachability relation [Pur00]

Given a region cycle ρ, and valuation $\nu = \vec{\lambda} \cdot \vec{v}$,

$\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda'}\vec{v} \quad \Leftrightarrow$ $\vec{\lambda'}$ is computed by distributing each λ_v to its successors following a probability distribution

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

Understanding the folded orbit graph



$$\begin{cases} \lambda_1' = p\lambda_1 \\ \lambda_2' = (1-p-q)\lambda_1 + \lambda_2 \\ \end{cases}$$

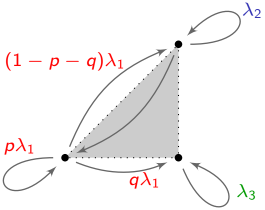
Reachability relation [Pur00]

Given a region cycle ρ, and valuation $\nu = \vec{\lambda} \cdot \vec{v}$,

$\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda}' \vec{v} \quad \Leftrightarrow$ $\vec{\lambda}'$ is computed by distributing each λ_v to its successors following a probability distribution

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010).*
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11).*

Understanding the folded orbit graph



$$\begin{cases} \lambda_1' = p\lambda_1 \\ \lambda_2' = (1-p-q)\lambda_1 + \lambda_2 \\ \lambda_3' = q\lambda_1 + \lambda_3 \end{cases}$$

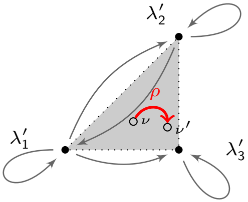
Reachability relation [Pur00]

Given a region cycle ρ, and valuation $\nu = \vec{\lambda} \cdot \vec{v}$,

$\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda}' \vec{v} \quad \Leftrightarrow$ $\vec{\lambda}'$ is computed by distributing each λ_v to its successors following a probability distribution

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

Understanding the folded orbit graph



$$\begin{cases} \lambda'_1 = p\lambda_1 \\ \lambda'_2 = (1-p-q)\lambda_1 + \lambda_2 \\ \lambda'_3 = q\lambda_1 + \lambda_3 \end{cases}$$

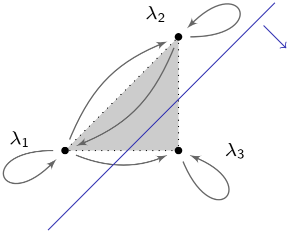
Reachability relation [Pur00]

Given a region cycle ρ, and valuation $\nu = \vec{\lambda} \cdot \vec{v}$,

$\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda'}\vec{v} \quad \Leftrightarrow$ $\vec{\lambda'}$ is computed by distributing each λ_v to its successors following a probability distribution

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

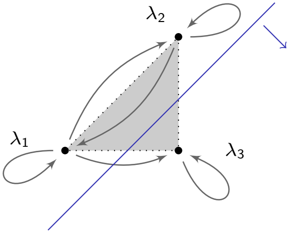
Understanding the folded orbit graph



$\lambda_1 + \lambda_2$ is non-increasing and λ_3 is non-decreasing

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

Understanding the folded orbit graph

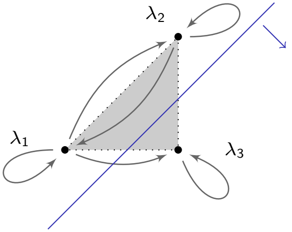


Generalization

- The reachability relation along one cycle is complete iff its folded orbit graph is complete. [Pur00]

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

Understanding the folded orbit graph

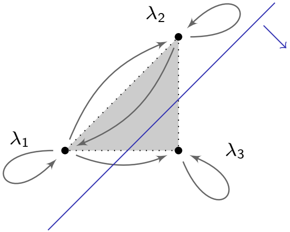


Generalization

- The reachability relation along one cycle is complete iff its folded orbit graph is complete. [Pur00]
- If the folded orbit graph is connected but not strongly connected, then there is some convergence phenomenon in the direction of the hyperplane $\sum_{v \in I} \lambda_v$ [AB11]

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

Understanding the folded orbit graph

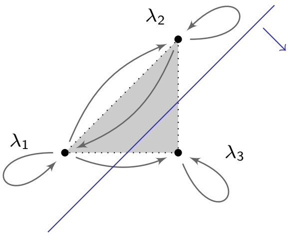


Classification of cycles

A cycle is aperiodic if all its iterations are strongly connected.

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

Understanding the folded orbit graph



Classification of cycles

A cycle is aperiodic if all its iterations are strongly connected. Then:

- aperiodic cycle: no convergence phenomenon
  (some iterate is complete)
- non-aperiodic cycle: convergence phenomenon
  (convergence phenomenon from the non strongly connected iterate)

[Pur00] Puri. Dynamical properties of timed automata *(Discrete Event Dynamic Systems, 2010)*.
[AB11] Asarin, Basset. Thin and thick timed regular languages *(FORMATS'11)*.

Back to robustness

Characterization

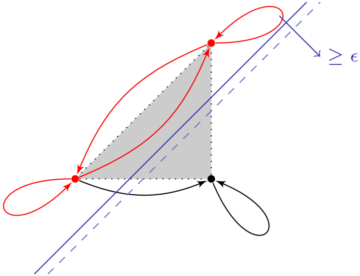
There exists $\delta > 0$ such that Controller has a δ-robust strategy ensuring a Büchi condition in $\mathcal{G}_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.

Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a δ-robust strategy ensuring a Büchi condition in $\mathcal{G}_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$,

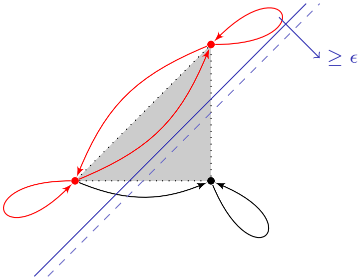


Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a δ-robust strategy ensuring a Büchi condition in $\mathcal{G}_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$, hence he can enforce convergence, and Controller gets stuck



Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a δ-robust strategy ensuring a Büchi condition in $\mathcal{G}_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.

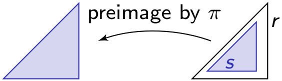
- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$,
- Aperiodic cycle π: Controller can target the middle of the regions and stay far from the borders.

Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a δ-robust strategy ensuring a Büchi condition in $\mathcal{G}_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$,
- Aperiodic cycle π: Controller can target the middle of the regions and stay far from the borders.

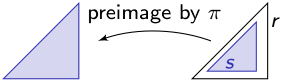


Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a δ-robust strategy ensuring a Büchi condition in $\mathcal{G}_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$,
- Aperiodic cycle π: Controller can target the middle of the regions and stay far from the borders.



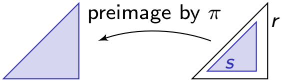
  - Remember shrunk DBMs: preimage of s by π under δ-perturbations is $r - \delta Q$ (Q fixed) for small δ's
    $\rightsquigarrow$ from $r - \delta Q$, Controller has a strategy to ensure s

Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a δ-robust strategy ensuring a Büchi condition in $\mathcal{G}_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$,
- Aperiodic cycle π: Controller can target the middle of the regions and stay far from the borders.



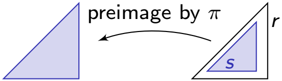
  - Remember shrunk DBMs: preimage of s by π under δ-perturbations is $r - \delta Q$ (Q fixed) for small δ's
    $\rightsquigarrow$ from $r - \delta Q$, Controller has a strategy to ensure s
  - Property of s: $s \subseteq r - \delta Q$ for small δ's
    $\rightsquigarrow$ we can repeat the above strategy

Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a δ-robust strategy ensuring a Büchi condition in $\mathcal{G}_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$,
- Aperiodic cycle π: Controller can target the middle of the regions and stay far from the borders.



- Remember shrunk DBMs: preimage of s by π under δ-perturbations is $r - \delta Q$ (Q fixed) for small δ's
  $\rightsquigarrow$ from $r - \delta Q$, Controller has a strategy to ensure s
- Property of s: $s \subseteq r - \delta Q$ for small δ's
  $\rightsquigarrow$ we can repeat the above strategy

$\Rightarrow$ Robust strategy: enforce s at each cycle

Going further [ORS14]

Extension to two-player games

- New rules: Controller chooses a delay and an action, and Perturbator perturbs the delay and resolves the non-determinism, if any
- Robustness under strict semantics can be solved in this case as well (EXPTIME)

[ORS14] Oualhadj, Reynier, Sankur. Probabilistic robust timed games *(CONCUR'14)*.

Going further [ORS14]

Extension to two-player games

- New rules: Controller chooses a delay and an action, and Perturbator perturbs the delay and resolves the non-determinism, if any
- Robustness under strict semantics can be solved in this case as well (EXPTIME)

Beyond worst-case robustness

- Assume perturbations are randomized!
  (uniform distributions over $[d - \delta; d + \delta]$)

[ORS14] Oualhadj, Reynier, Sankur. Probabilistic robust timed games *(CONCUR'14)*.

Going further [ORS14]

Extension to two-player games

- New rules: Controller chooses a delay and an action, and Perturbator perturbs the delay and resolves the non-determinism, if any
- Robustness under strict semantics can be solved in this case as well (EXPTIME)

Beyond worst-case robustness

- Assume perturbations are randomized!
  (uniform distributions over $[d - \delta; d + \delta]$)
- Existence of an almost-sure winning strategy for Controller can be decided in EXPTIME. Furthermore there is a dichotomy:
  - either Controller wins almost-surely
  - or Perturbator wins almost-surely

[ORS14] Oualhadj, Reynier, Sankur. Probabilistic robust timed games *(CONCUR'14)*.

Partial conclusion

- We have presented a possible approach to the robust realizability and control problems
  - There are two natural semantics (excess or strict)
  - Interesting relation between non-convergent cycles and robust cycles
  - Interesting complexities as well!

Partial conclusion

- We have presented a possible approach to the robust realizability and control problems
  - There are two natural semantics (excess or strict)
  - Interesting relation between non-convergent cycles and robust cycles
  - Interesting complexities as well!

- Teachnical tools:
  - Regions
  - Shrunk DBMs
  - Orbit graphs

Partial conclusion

- We have presented a possible approach to the robust realizability and control problems
  - There are two natural semantics (excess or strict)
  - Interesting relation between non-convergent cycles and robust cycles
  - Interesting complexities as well!
- Teachnical tools:
  - Regions
  - Shrunk DBMs
  - Orbit graphs
- What is missing:
  - A symbolic approach
  - A tool support
  - Stochastic approach at the beginning only

Outline

Conclusion

- Timed automata: a nice mathematical model for real-time systems with interesting decidability properties and algorithmics solutions.

Conclusion

- Timed automata: a nice mathematical model for real-time systems with interesting decidability properties and algorithmics solutions.
- Not always easy to transfer correctness proven in this model to real behaviours of the system.

Conclusion

- Timed automata: a nice mathematical model for real-time systems with interesting decidability properties and algorithmics solutions.
- Not always easy to transfer correctness proven in this model to real behaviours of the system.
- We have shown several frameworks for robustness that can be used to ensure correctness in the real-world.

Conclusion

- Timed automata: a nice mathematical model for real-time systems with interesting decidability properties and algorithmics solutions.
- Not always easy to transfer correctness proven in this model to real behaviours of the system.
- We have shown several frameworks for robustness that can be used to ensure correctness in the real-world.
- We have seen several tools that are useful in this context (*eg*, shrunk DBMs, orbit graphs)

Conclusion

- Timed automata: a nice mathematical model for real-time systems with interesting decidability properties and algorithmics solutions.
- Not always easy to transfer correctness proven in this model to real behaviours of the system.
- We have shown several frameworks for robustness that can be used to ensure correctness in the real-world.
- We have seen several tools that are useful in this context (*eg*, shrunk DBMs, orbit graphs)

- Extension of these works to richer models seems unfortunately hard [BMS13]
- Probabilistic perturbations [ORS14]: compute/give lower bound average-time to failure?
- Symbolic algorithms?

Conclusion

- Timed automata: a nice mathematical model for real-time systems with interesting decidability properties and algorithmics solutions.
- Not always easy to transfer correctness proven in this model to real behaviours of the system.
- We have shown several frameworks for robustness that can be used to ensure correctness in the real-world.
- We have seen several tools that are useful in this context (*eg*, shrunk DBMs, orbit graphs)

- Extension of these works to richer models seems unfortunately hard [BMS13]
- Probabilistic perturbations [ORS14]: compute/give lower bound average-time to failure?
- Symbolic algorithms?

- This list of possible approaches is not exhaustive:
  - tube acceptance [GHJ97]
  - sampling approach [KP05,BLM+11]
  - probabilistic approach [BBB+08,BBJM12]
  - ...