Implementability of Timed Controllers

Based on joint works with Karine Altisen, Patricia Bouyer, Martin De Wulf, Laurent Doyen, Jean-François Raskin, Pierre-Alain Reynier, and Stavros Tripakis

Nicolas Markey

Lab. Spécification et Vérification – ENS Cachan & CNRS

September 5, 2007

Controller Synthesis and Implementation

system:

property:

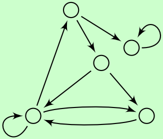
$\Rightarrow$

Controller Synthesis and Implementation

system:

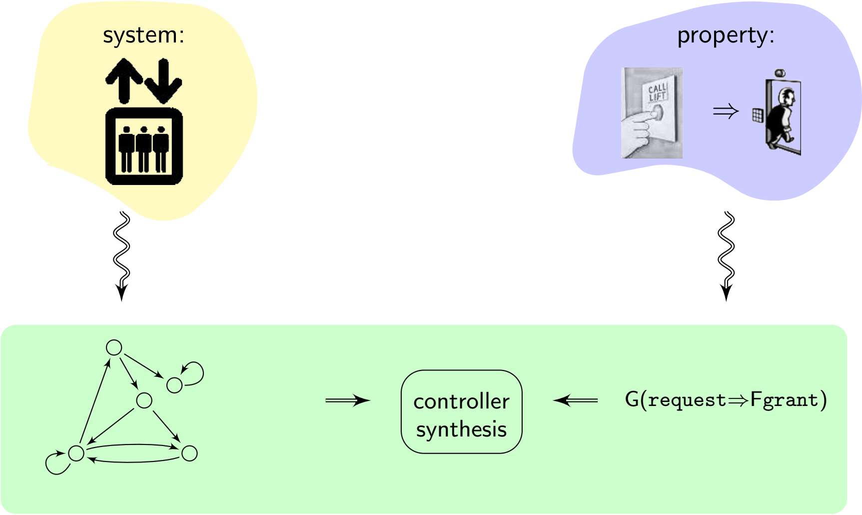
property:

$\Rightarrow$

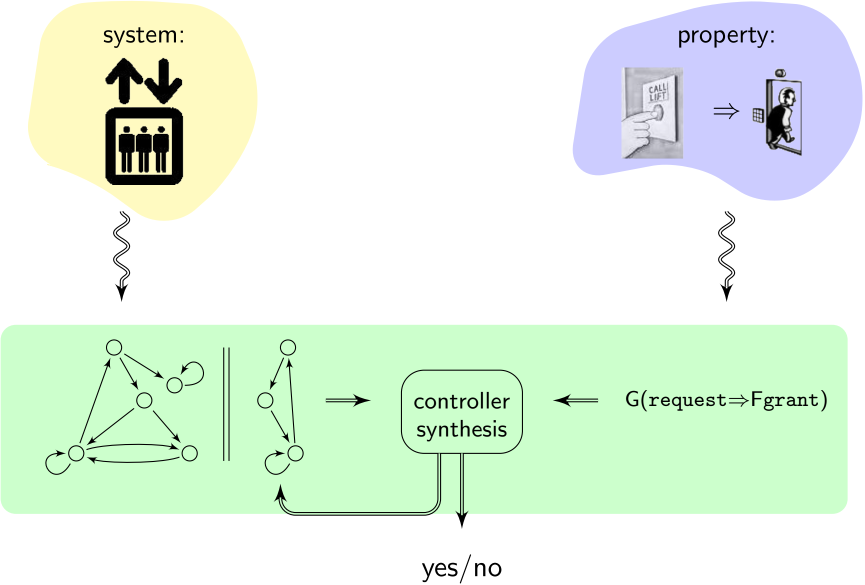


$\mathsf{G}(\mathtt{request} \Rightarrow \mathsf{F}\mathtt{grant})$

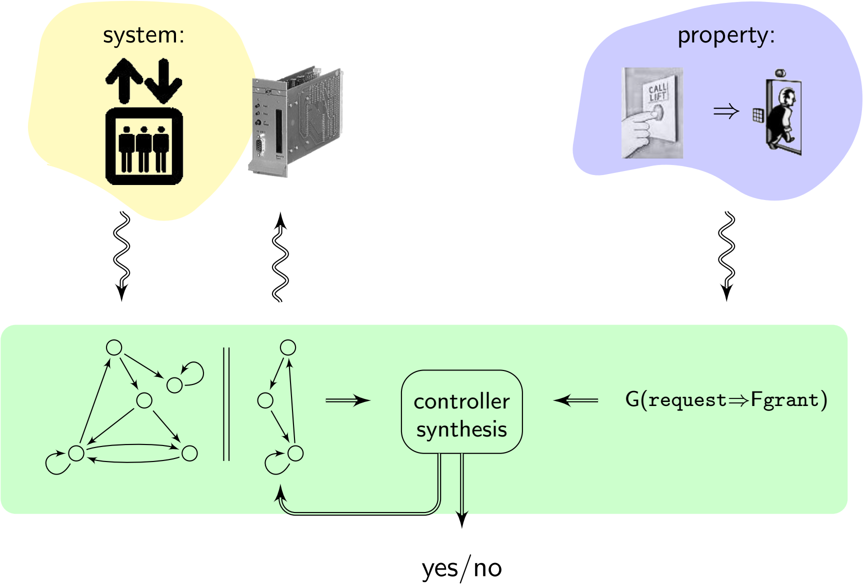
Controller Synthesis and Implementation



Controller Synthesis and Implementation



Controller Synthesis and Implementation



Implementatiblity of Timed Controllers

- The semantics of timed automata is a mathematical idealization:

Implementatiblity of Timed Controllers

- The semantics of timed automata is a mathematical idealization:

  Infinitely punctual : Exact synchronization is required when composing several TAs;

Implementatiblity of Timed Controllers

- The semantics of timed automata is a mathematical idealization:

  Infinitely punctual : Exact synchronization is required when composing several TAs;

  Infinitely precise : Different clocks are assumed to increase at the same rate in both the controler and the system.

Implementatiblity of Timed Controllers

- The semantics of timed automata is a mathematical idealization:

  Infinitely punctual : Exact synchronization is required when composing several TAs;

  Infinitely precise : Different clocks are assumed to increase at the same rate in both the controler and the system.

  Infinitely fast : It may happen, for instance, that a TA will have to perform actions at time n and $n + 1/n$, for all n;

Implementatiblity of Timed Controllers

- The semantics of timed automata is a mathematical idealization:

  Infinitely punctual : Exact synchronization is required when composing several TAs;

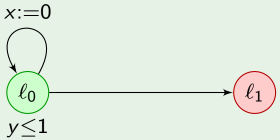
  Infinitely precise : Different clocks are assumed to increase at the same rate in both the controler and the system.

  Infinitely fast : It may happen, for instance, that a TA will have to perform actions at time n and $n + 1/n$, for all n;

- In practice, a processor is digital and imprecise. Even if we prove that a TA will not enter a set of bad states, its implementations could still lead to bad behaviors.

Implementatiblity of Timed Controllers

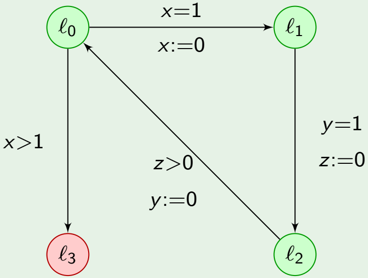
Examples (Zeno behaviors)



- The red state can be avoided;
- But this would require to *prevent* time to elapse.

Implementatiblity of Timed Controllers

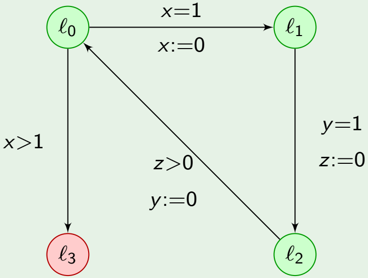
Examples (Cassez *et al.*, 2002)



loc.	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0
x										
y										
z										

Implementatiblity of Timed Controllers

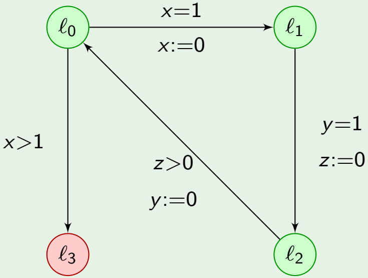
Examples (Cassez *et al.*, 2002)



loc.	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0
x	0									
y	0									
z	0									

Implementatiblity of Timed Controllers

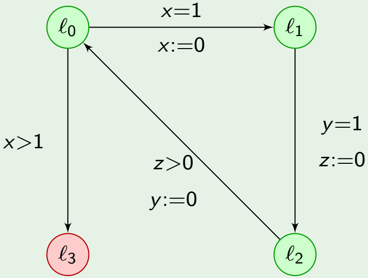
Examples (Cassez *et al.*, 2002)



loc.	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0
x	0	0								
y	0	1								
z	0	1								

Implementatiblity of Timed Controllers

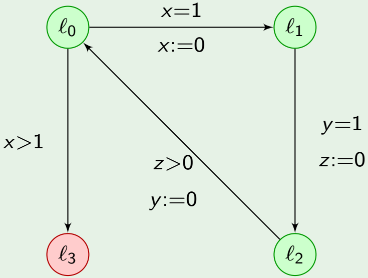
Examples (Cassez *et al.*, 2002)



loc.	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0
x	0	0	0							
y	0	1	1							
z	0	1	0							

Implementatiblity of Timed Controllers

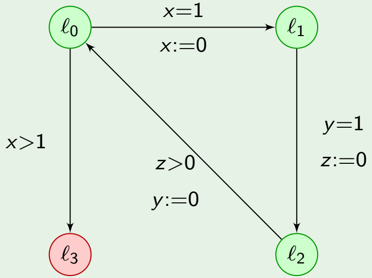
Examples (Cassez *et al.*, 2002)



loc.	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0
x	0	0	0	ϵ_1						
y	0	1	1	0						
z	0	1	0	ϵ_1						

Implementatiblity of Timed Controllers

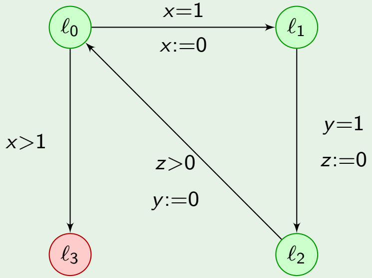
Examples (Cassez *et al.*, 2002)



loc.	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0
x	0	0	0	ϵ_1	0					
y	0	1	1	0	$1-\epsilon_1$					
z	0	1	0	ϵ_1	1					

Implementatiblity of Timed Controllers

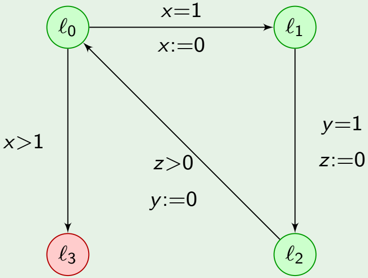
Examples (Cassez *et al.*, 2002)



loc.	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0
x	0	0	0	ϵ_1	0	ϵ_1				
y	0	1	1	0	$1-\epsilon_1$	1				
z	0	1	0	ϵ_1	1	0				

Implementatiblity of Timed Controllers

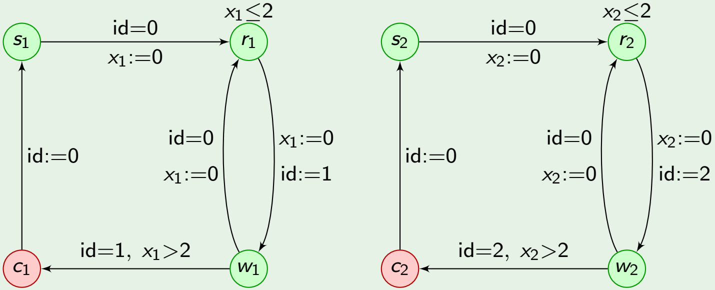
Examples (Cassez *et al.*, 2002)



loc.	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0	ℓ_1	ℓ_2	ℓ_0
x	0	0	0	ϵ_1	0	ϵ_1	$\epsilon_1 + \epsilon_2$	...		
y	0	1	1	0	$1 - \epsilon_1$	1	0	...		
z	0	1	0	ϵ_1	1	0	ϵ_2	...		

Implementatiblity of Timed Controllers

Examples (Fischer's Mutual Exclusion Protocol)



- It can be proved that this protocol enforces mutual exclusion in the critical (red) state.
- Any imprecise implementation will fail to fulfil that property.

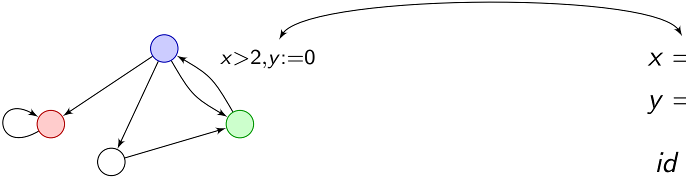
Outline of the talk

1. Introduction
2. Modeling the execution platform [Altisen & Tripakis, 2005]
3. A semantical approach [De Wulf et al., 2004]
4. Conclusions

Outline of the talk

Modeling the execution platform [Altisen & Tripakis, 2005]

$\mathcal{A}$ Env

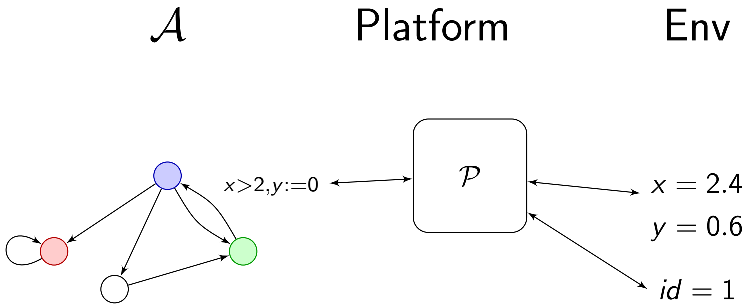


$x = 2.4$

$y = 0.6$

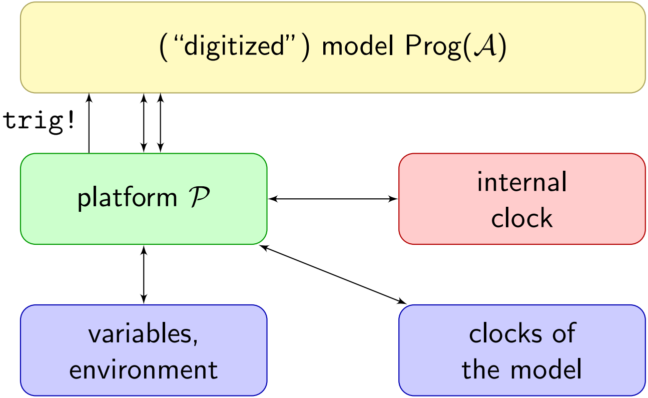
$id = 1$

Modeling the execution platform [Altisen & Tripakis, 2005]



- The automaton $\mathcal{A}$ is now a discrete automaton, using input variables given by the platform;
- The automaton $\mathcal{P}$ is a timed automaton that triggers $\mathcal{A}$ (modeling a digital CPU), and sends input variables to $\mathcal{A}$ depending on the values of the variables in Env;

Modeling the execution platform [Altisen & Tripakis, 2005]



Modeling the execution platform [Altisen & Tripakis, 2005]

1. Transforming $\mathcal{A}$ into Prog($\mathcal{A}$).

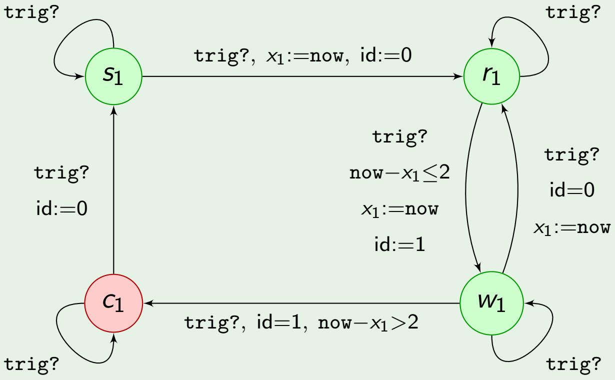
- `trig!` is an input event allowing $\mathcal{A}$ to perform one step;
- the value of a clock is the difference between the current value of the internal clock (`now`) and the date at which the clock was last reset:

$$\text{“}x > 2\text{”} \quad \text{becomes} \quad \text{“}\mathtt{now} - x > 2\text{”}$$
$$\text{“}x := 0\text{”} \quad \text{becomes} \quad \text{“}x := \mathtt{now}\text{”}$$

Modeling the execution platform [Altisen & Tripakis, 2005]

1. Transforming $\mathcal{A}$ into Prog($\mathcal{A}$).

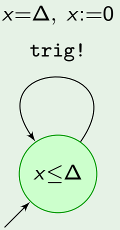
Example (Fischer's Mutual Exclusion Protocol)



Modeling the execution platform [Altisen & Tripakis, 2005]

1. Transforming $\mathcal{A}$ into Prog($\mathcal{A}$).
2. Modeling the digital CPU.

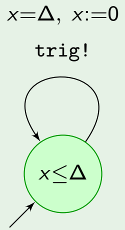
Examples

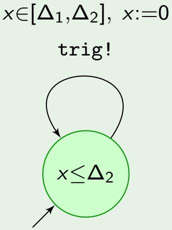


Modeling the execution platform [Altisen & Tripakis, 2005]

1. Transforming $\mathcal{A}$ into Prog($\mathcal{A}$).
2. Modeling the digital CPU.

Examples

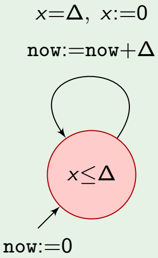




Modeling the execution platform [Altisen & Tripakis, 2005]

1. Transforming $\mathcal{A}$ into Prog($\mathcal{A}$).
2. Modeling the digital CPU.
3. Modeling the global clock.

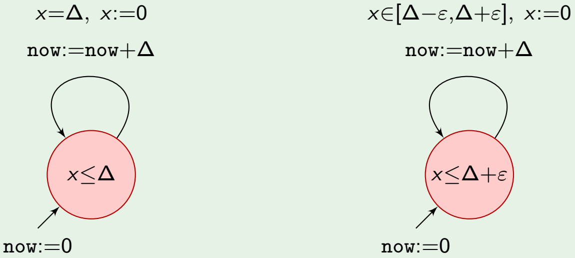
Examples



Modeling the execution platform [Altisen & Tripakis, 2005]

1. Transforming $\mathcal{A}$ into Prog($\mathcal{A}$).
2. Modeling the digital CPU.
3. Modeling the global clock.

Examples



Modeling the execution platform [Altisen & Tripakis, 2005]

1. Transforming $\mathcal{A}$ into Prog($\mathcal{A}$).
2. Modeling the digital CPU.
3. Modeling the global clock.
4. Modeling the input/output variables.
   - delays for reading variables...
   - lock mechanism for writing variables...

Modeling the execution platform [Altisen & Tripakis, 2005]

1. Transforming $\mathcal{A}$ into Prog($\mathcal{A}$).

2. Modeling the digital CPU.

3. Modeling the global clock.

4. Modeling the input/output variables.

5. Classical verification techniques on the product of those automata.

Pros and cons of this approach

- Pros:
  - Very expressive: the platform can be described with many details;
  - Relies on classical techniques: the verification step is applied on standard timed automata. Existing tools can be used.

Pros and cons of this approach

- Pros:
  - Very expressive: the platform can be described with many details;
  - Relies on classical techniques: the verification step is applied on standard timed automata. Existing tools can be used.

- Cons:
  - Formal meaning?: if the model satisfies some property, what does it *really* mean?
  - Faster is better?: we expect that a program proved to be implementable on a given platform remains implementable on a faster platform. This property fails to hold with this modeling.

Outline of the talk

1. Introduction
2. Modeling the execution platform [Altisen & Tripakis, 2005]
3. A semantical approach [De Wulf et al., 2004]
4. Conclusions

A semantical approach [De Wulf *et al.*, 2004]

1. "Implementation" Semantics

We consider a simple model of a platform, that repeatedly executes the following actions:

- store the value of the global clock;
- compute guards;
- fire one of the enabled transitions.

We assume that

- one such loop takes at most Δ_P t.u. to execute;
- the global clock is updated every Δ_L t.u.

$\leadsto$ We write $[\![\mathcal{A}]\!]^{\mathsf{Impl}}_{\Delta_P,\Delta_L}$ for the set of executions of a timed automaton $\mathcal{A}$ under this semantics.

A semantical approach [De Wulf *et al.*, 2004]

1. "Implementation" Semantics

2. Enlarged Semantics

We define the enlarged semantics for timed automata, by enlarging guards on transitions by a small tolerance Δ:

$$\text{If } [\![g]\!] = [a;b], \text{ then } [\![g]\!]_{\Delta}^{\mathsf{AASAP}} = [a-\Delta, b+\Delta].$$

$\leadsto$ We write $[\![\mathcal{A}]\!]_{\Delta}^{\mathsf{AASAP}}$ for the set of executions of a timed automaton $\mathcal{A}$ under this semantics.

A semantical approach [De Wulf *et al.*, 2004]

1. "Implementation" Semantics

2. Enlarged Semantics
   We define the enlarged semantics for timed automata, by enlarging guards on transitions by a small tolerance Δ:

$$\text{If } [\![g]\!] = [a;b], \text{ then } [\![g]\!]_{\Delta}^{\text{AASAP}} = [a-\Delta, b+\Delta].$$

$\rightsquigarrow$ We write $[\![\mathcal{A}]\!]_{\Delta}^{\text{AASAP}}$ for the set of executions of a timed automaton $\mathcal{A}$ under this semantics.

Theorem ([DDR04])

If $\Delta > 3\Delta_L + 4\Delta_P$, then $[\![\mathcal{A}]\!]_{\Delta_P,\Delta_L}^{Impl} \subseteq [\![\mathcal{A}]\!]_{\Delta}^{AASAP}$.

A semantical approach [De Wulf *et al.*, 2004]

We focus on safety properties for the implementation semantics: we want to ensure that an implementation will avoid bad states.

- $\rightsquigarrow$ $\text{Reach}_{\Delta}(\mathcal{A})$ is the set of reachable states under the AASAP semantics.

$$\Delta_1 \leq \Delta_2 \Rightarrow \text{Reach}_{\Delta_1}(\mathcal{A}) \subseteq \text{Reach}_{\Delta_2}(\mathcal{A})$$

- $\rightsquigarrow$ $R(\mathcal{A}) = \bigcap_{\Delta>0} \text{Reach}_{\Delta}(\mathcal{A})$ is the set of reachable states under the AASAP semantics for any $\Delta > 0$.

A semantical approach [De Wulf *et al.*, 2004]

We focus on safety properties for the implementation semantics: we want to ensure that an implementation will avoid bad states.

$\rightsquigarrow$ $\mathsf{Reach}_\Delta(\mathcal{A})$ is the set of reachable states under the AASAP semantics.

$$\Delta_1 \leq \Delta_2 \Rightarrow \mathsf{Reach}_{\Delta_1}(\mathcal{A}) \subseteq \mathsf{Reach}_{\Delta_2}(\mathcal{A})$$

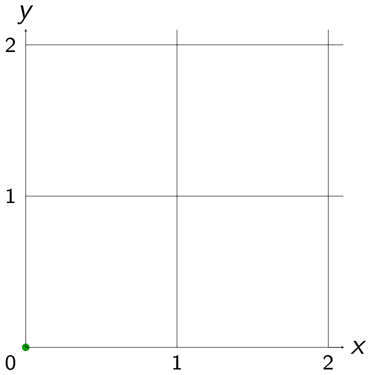
$\rightsquigarrow$ $R(\mathcal{A}) = \bigcap_{\Delta>0} \mathsf{Reach}_\Delta(\mathcal{A})$ is the set of reachable states under the AASAP semantics for any $\Delta > 0$.

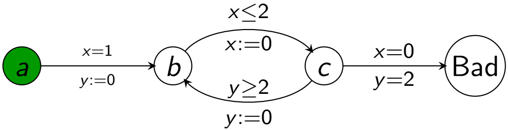
Lemma

For any timed automata $\mathcal{A}$ and for any set of zones B,

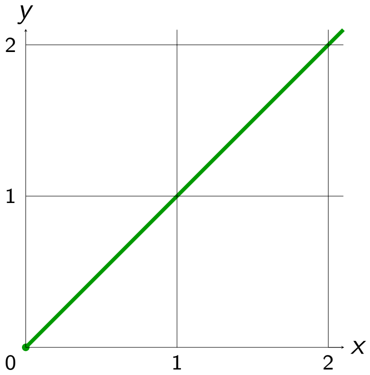
$$R(\mathcal{A}) \cap B = \varnothing \quad \textit{iff} \quad \exists \Delta > 0.\ \mathit{Reach}_\Delta(\mathcal{A}) \cap B = \varnothing.$$

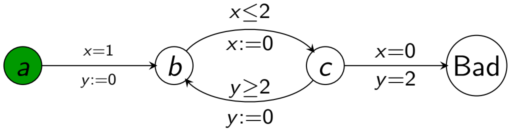
An example: Standard semantics



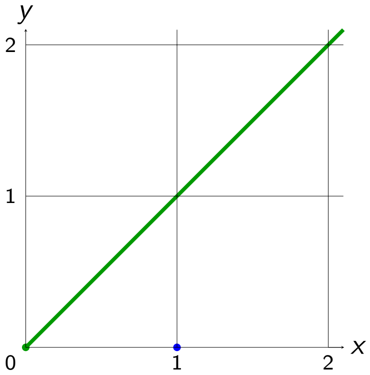


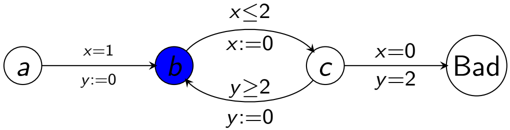
An example: Standard semantics



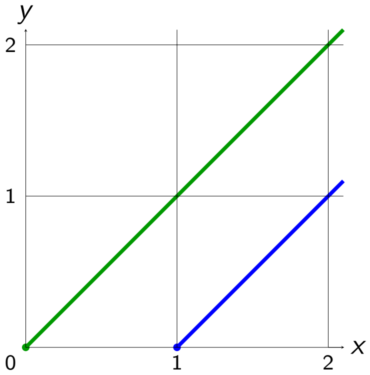


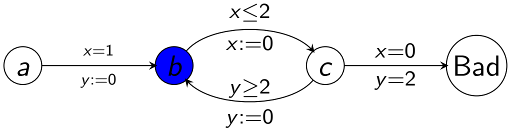
An example: Standard semantics



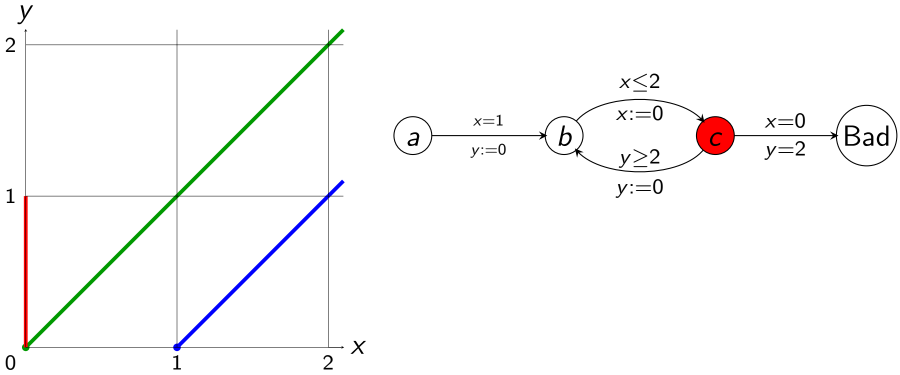


An example: Standard semantics

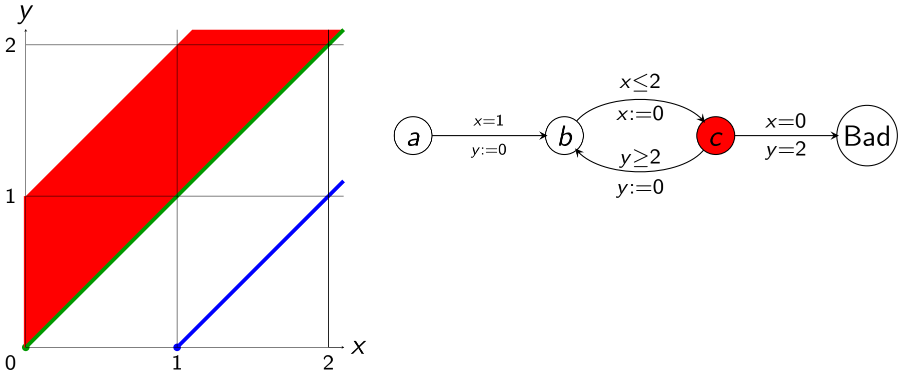




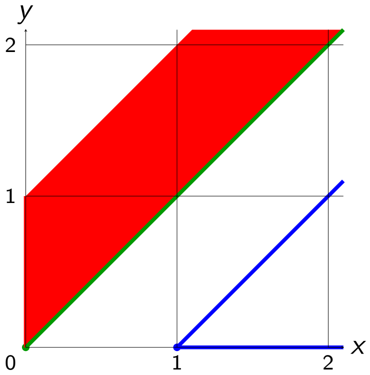
An example: Standard semantics

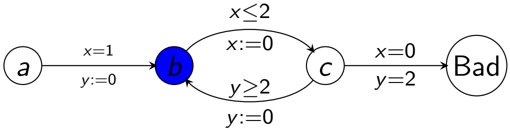


An example: Standard semantics

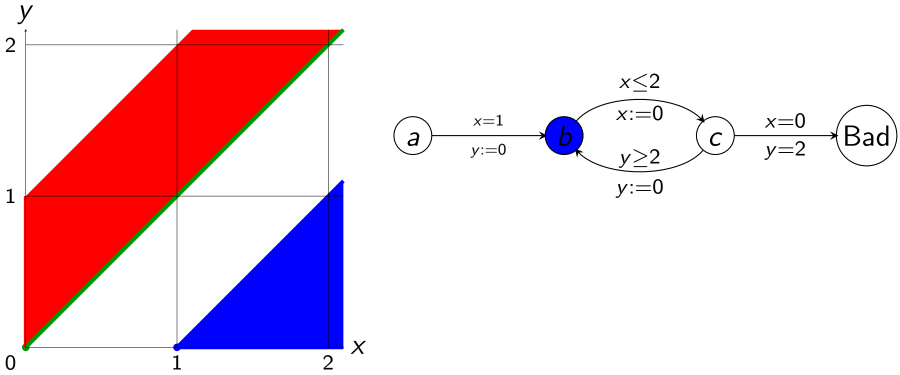


An example: Standard semantics

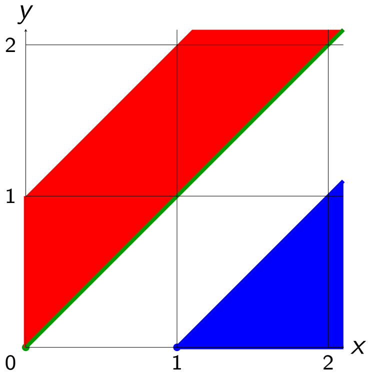


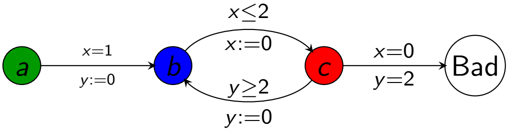


An example: Standard semantics

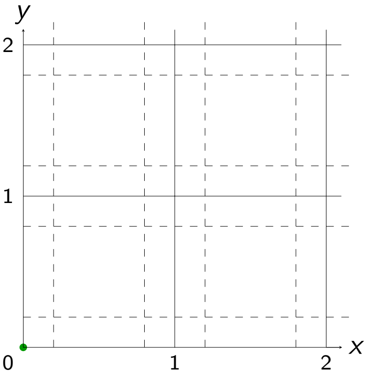


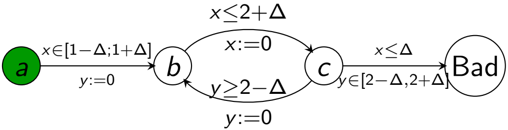
An example: Standard semantics



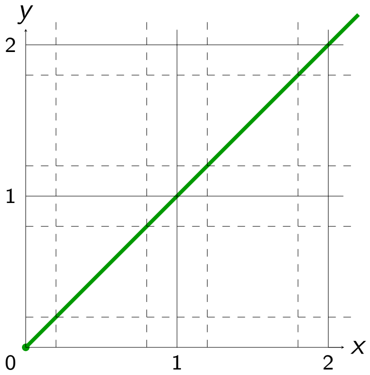


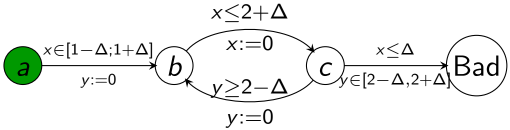
An example with $\Delta > 0$



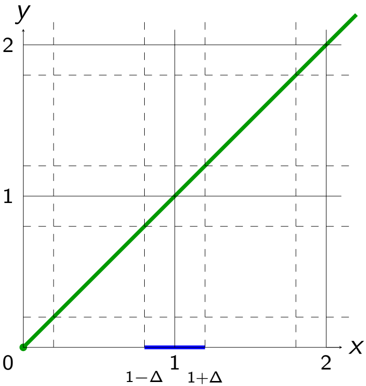


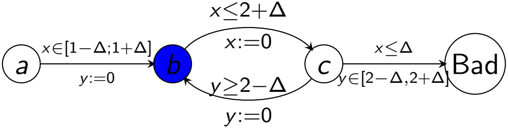
An example with $\Delta > 0$



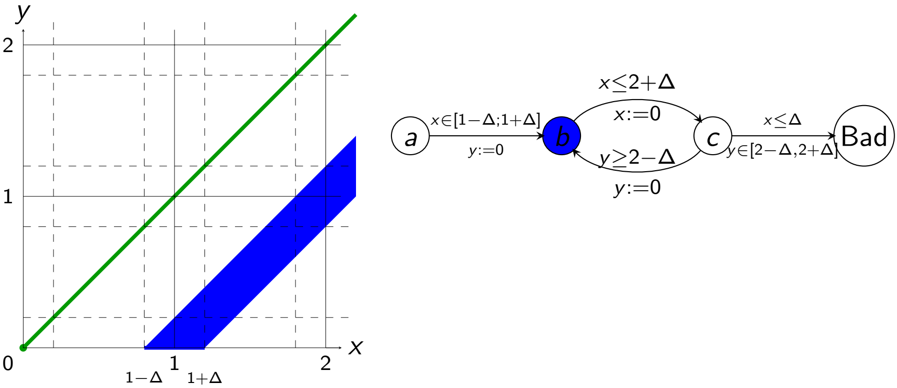


An example with $\Delta > 0$

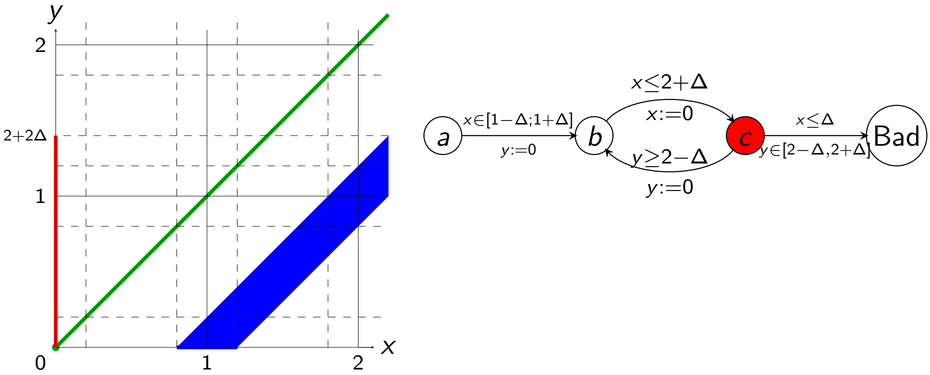




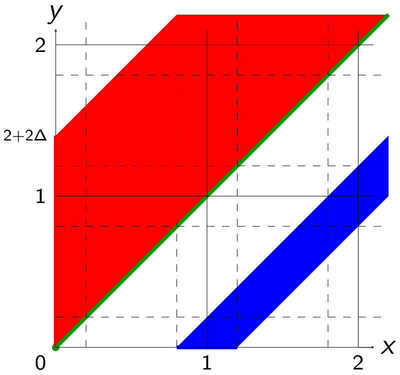
An example with $\Delta > 0$

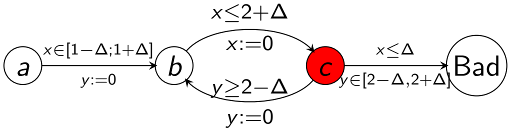


An example with $\Delta > 0$

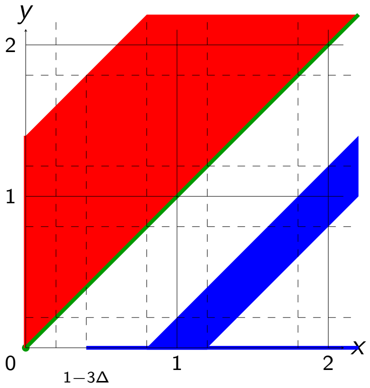


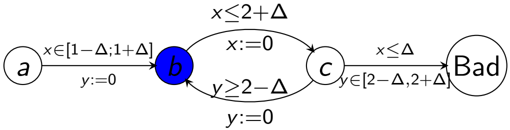
An example with $\Delta > 0$



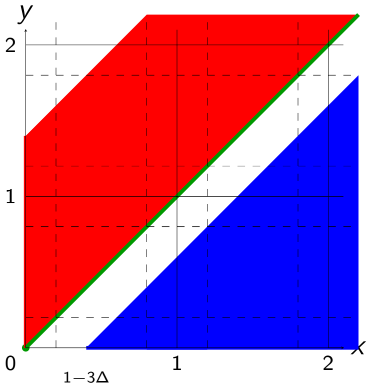


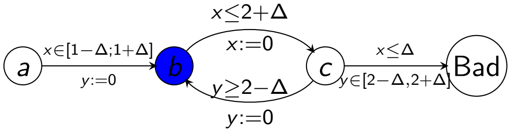
An example with $\Delta > 0$



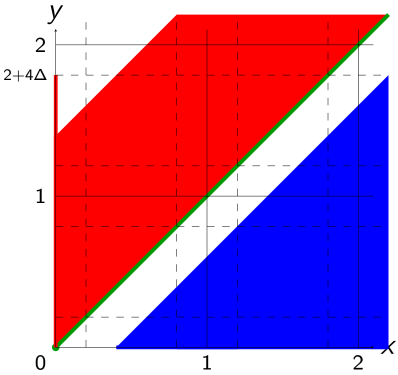


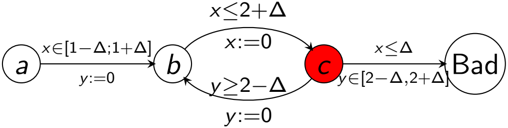
An example with $\Delta > 0$



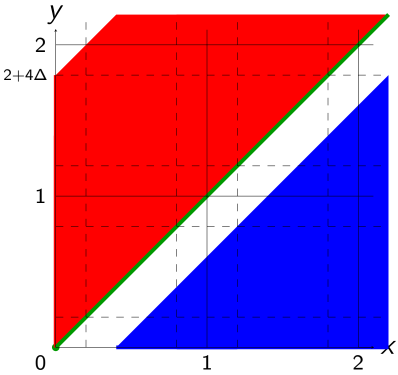


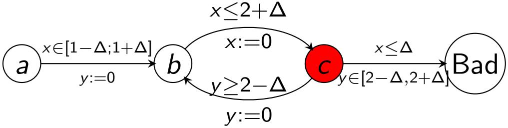
An example with $\Delta > 0$



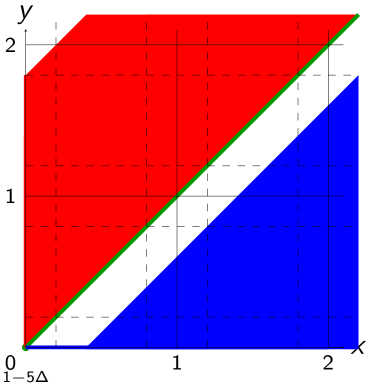


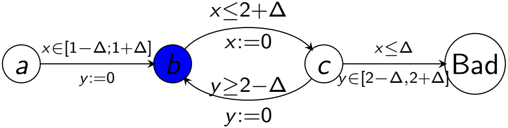
An example with $\Delta > 0$



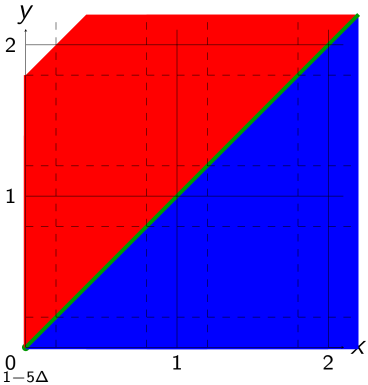


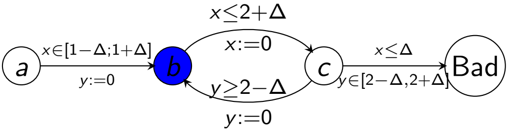
An example with $\Delta > 0$



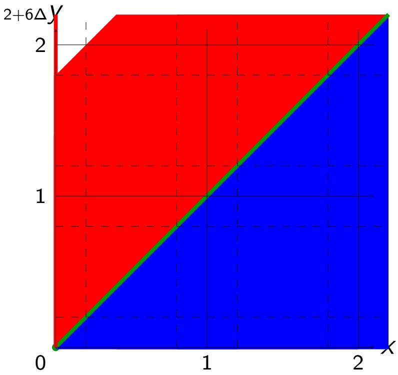


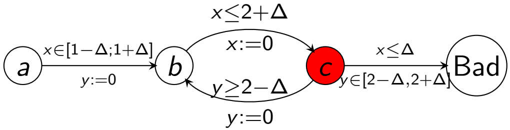
An example with $\Delta > 0$



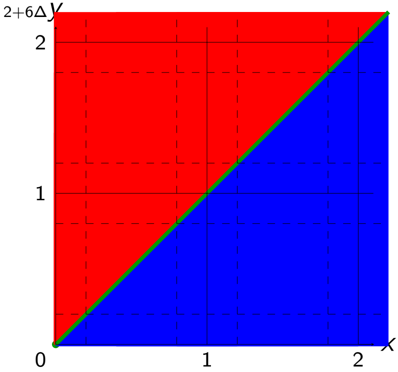


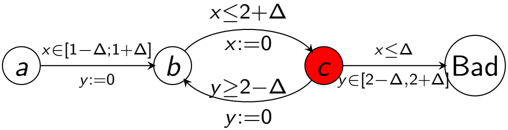
An example with $\Delta > 0$



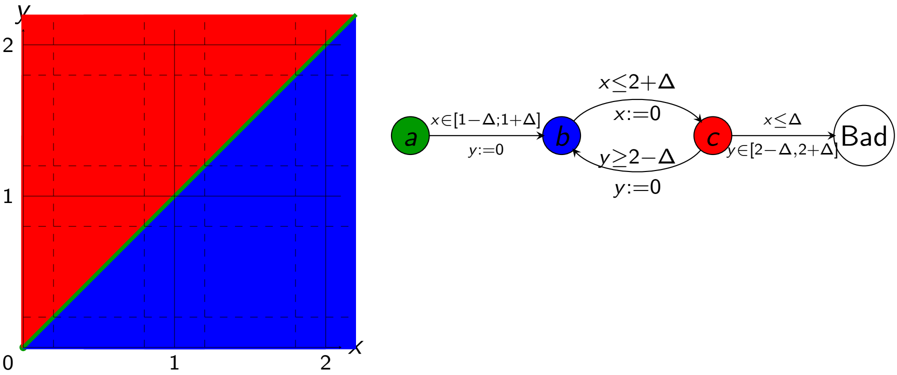


An example with $\Delta > 0$

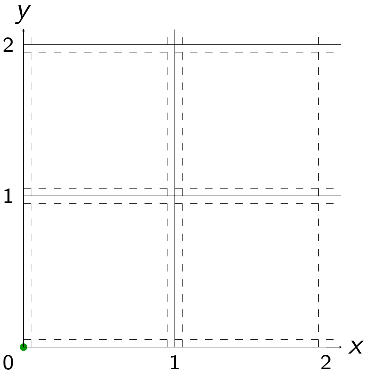


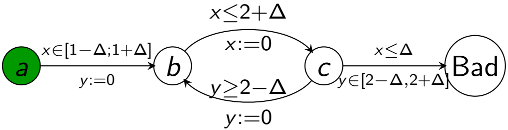


An example with $\Delta > 0$

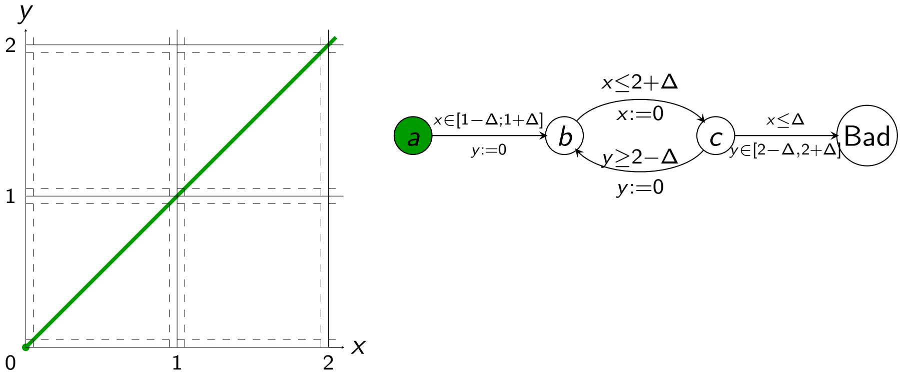


An example with Δ very small

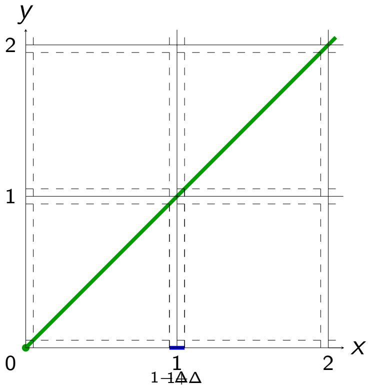


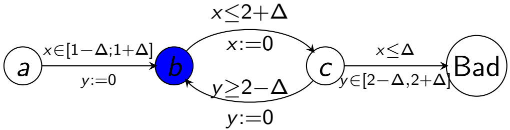


An example with Δ very small

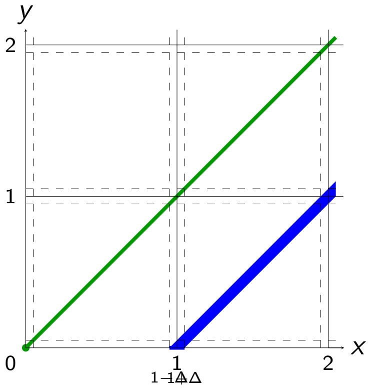


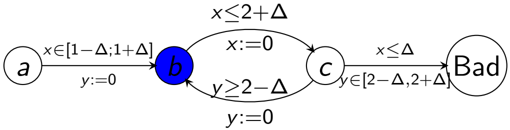
An example with Δ very small



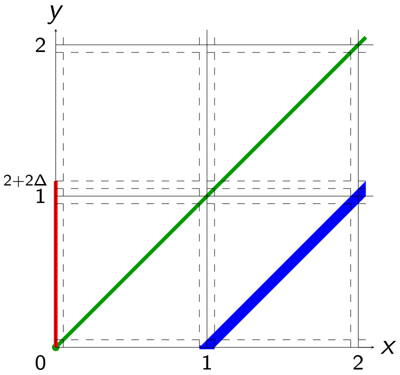


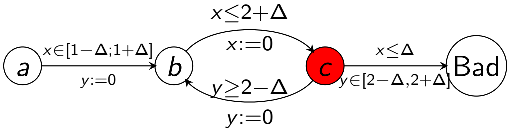
An example with Δ very small



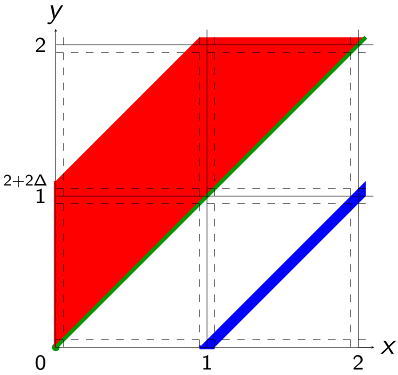


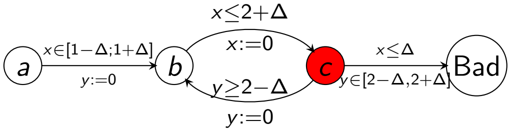
An example with Δ very small



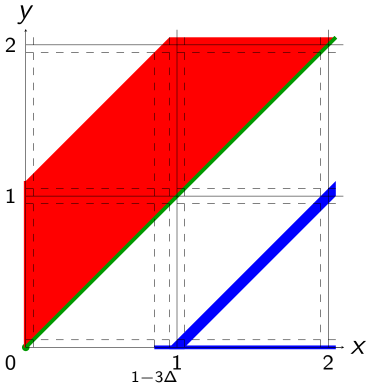


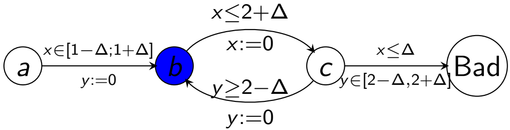
An example with Δ very small



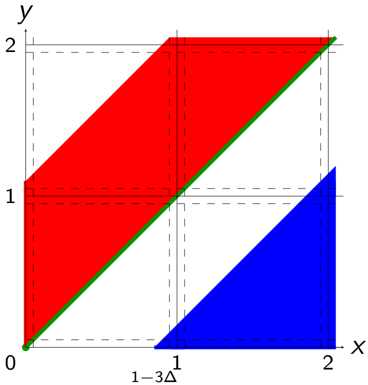


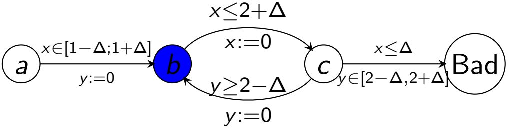
An example with Δ very small



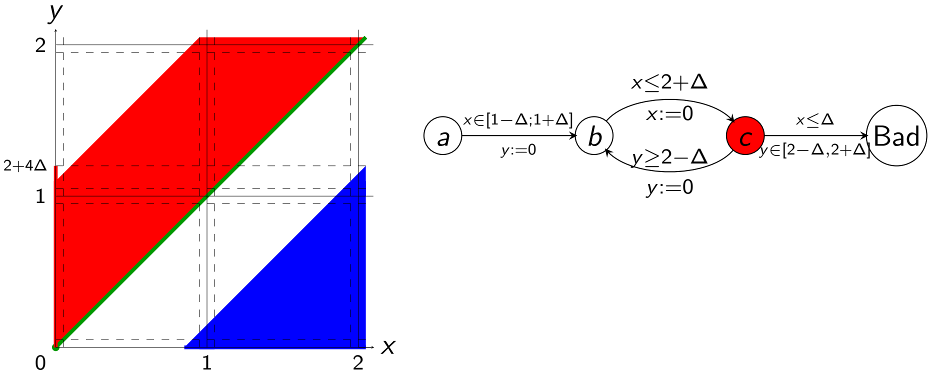


An example with Δ very small

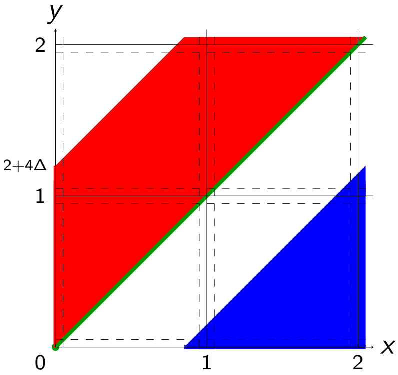


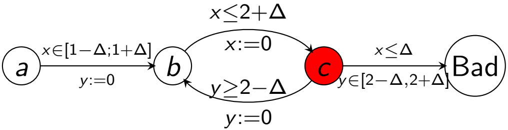


An example with Δ very small

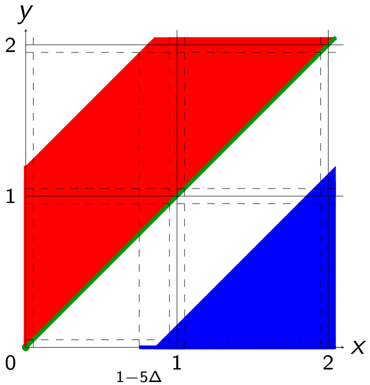


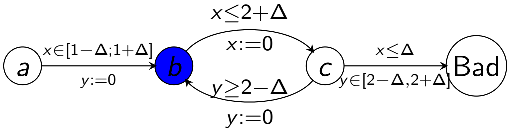
An example with Δ very small



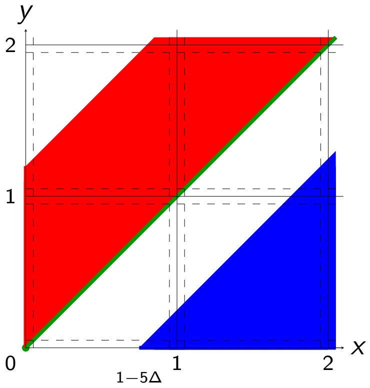


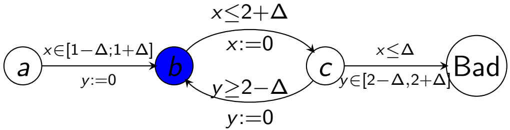
An example with Δ very small



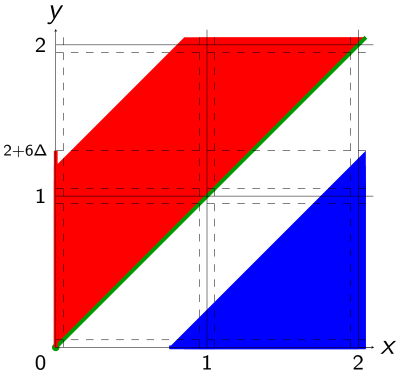


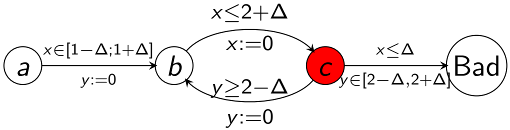
An example with Δ very small



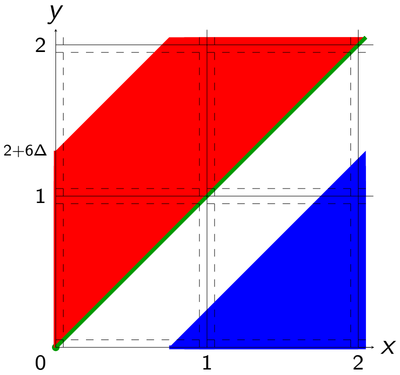


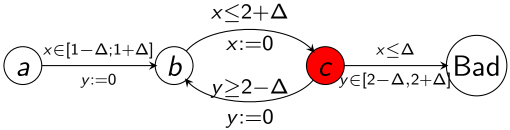
An example with Δ very small



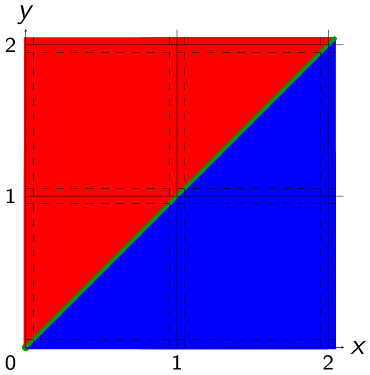


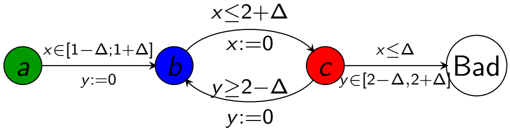
An example with Δ very small



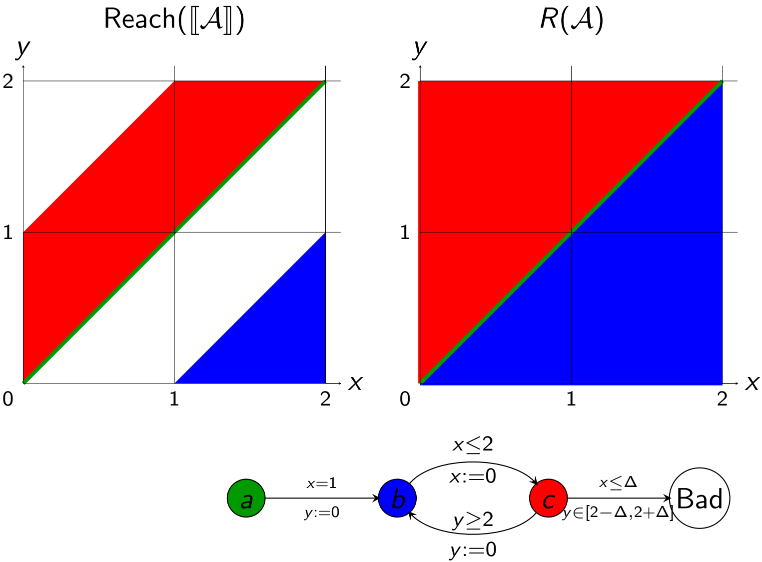


An example with Δ very small





Difference between $[\![\mathcal{A}]\!]$ and $R(\mathcal{A})$



An algorithm for computing $R(\mathcal{A})$

`Input: A Timed Automaton` $\mathcal{A}$
`Output: The set` $R(\mathcal{A})$

An algorithm for computing $R(\mathcal{A})$

```
Input: A Timed Automaton 𝒜
Output: The set R(𝒜)

1. build the region graph G of 𝒜;
```

An algorithm for computing $R(\mathcal{A})$

```
Input: A Timed Automaton 𝒜
Output: The set R(𝒜)

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
```

An algorithm for computing $R(\mathcal{A})$

```
Input: A Timed Automaton 𝒜
Output: The set R(𝒜)

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q0)];



6. return(J);
```

An algorithm for computing $R(\mathcal{A})$

```
Input: A Timed Automaton 𝒜
Output: The set R(𝒜)

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q0)];
4. J := Reach(G, J);


6. return(J);
```

An algorithm for computing $R(\mathcal{A})$

```
Input: A Timed Automaton 𝒜
Output: The set R(𝒜)

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q0)];
4. J := Reach(G, J);
5. while ∃ S ∈ SCC(G). S ⊈ J and S ∩ J ≠ ∅,
        J := J ∪ S;
        J := Reach(G, J);
6. return(J);
```

$J \subseteq R_\Delta(\mathcal{A})$

Lemma

Let $\mathcal{A}$ be a TA with n clocks, $\Delta \in \mathbb{Q}^{>0}$, and $\delta = \Delta/n$. Let u be a valuation s.t. there exists a trajectory $\pi[0, T]$ in $[\![\mathcal{A}]\!]$ with $\pi(0) = \pi(T) = u$. Let $v \in [u] \cap B(u, \delta)$. Then there exists a trajectory from u to v in $[\![\mathcal{A}]\!]^\Delta$.

Proof: We build the new trajectory by slightly modifying the delay transitions in π. This crucially depends on the fact that all clocks are reset along the cycle. □

$J \subseteq R_\Delta(\mathcal{A})$

Lemma

Let $\mathcal{A}$ be a TA with n clocks, $\Delta \in \mathbb{Q}^{>0}$, and $\delta = \Delta/n$. Let u be a valuation s.t. there exists a trajectory $\pi[0, T]$ in $[\![\mathcal{A}]\!]$ with $\pi(0) = \pi(T) = u$. Let $v \in [u] \cap B(u, \delta)$. Then there exists a trajectory from u to v in $[\![\mathcal{A}]\!]^\Delta$.

Proof: We build the new trajectory by slightly modifying the delay transitions in π. This crucially depends on the fact that all clocks are reset along the cycle. □

Corollary

Let $\mathcal{A}$ be a TA and $p = p_0 p_1 \dots p_k$ be a cycle in the region graph (i.e. $p_k = p_0$). For any $\Delta > 0$ and any $x, y \in p_0$, there exists a trajectory from x to y.

$J \supseteq R_\Delta(\mathcal{A})$

Lemma

Let $\mathcal{A}$ be a TA, $\delta \in \mathbb{R}^{>0}$ and $k \in \mathbb{N}$. There exists $D \in \mathbb{Q}^{>0}$ s.t. for all $\Delta \leq D$, any k-step trajectory $\pi' = (q'_0, t'_0)(q'_1, t'_1) \dots (q'_k, t'_k)$ in $[\![\mathcal{A}]\!]^\Delta$ can be approximated be a k-step trajectory $\pi = (q_0, t_0)(q_1, t_1) \dots (q_k, t_k)$ in $[\![\mathcal{A}]\!]$ with $\|q_i - q'_i\| \leq \delta$ for all i.

The proof involves parametric DBMs.

$J \supseteq R_\Delta(\mathcal{A})$

Lemma

Let $\mathcal{A}$ be a TA, $\delta \in \mathbb{R}^{>0}$ and $k \in \mathbb{N}$. There exists $D \in \mathbb{Q}^{>0}$ s.t. for all $\Delta \leq D$, any k-step trajectory $\pi' = (q'_0, t'_0)(q'_1, t'_1)\ldots(q'_k, t'_k)$ in $[\![\mathcal{A}]\!]^\Delta$ can be approximated be a k-step trajectory $\pi = (q_0, t_0)(q_1, t_1)\ldots(q_k, t_k)$ in $[\![\mathcal{A}]\!]$ with $\|q_i - q'_i\| \leq \delta$ for all i.

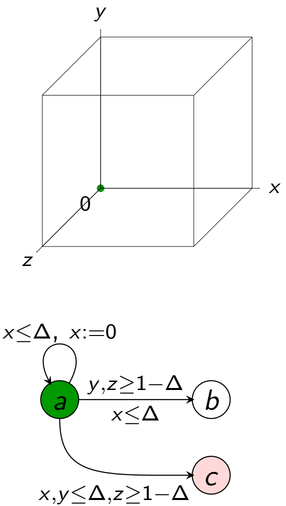
The proof involves parametric DBMs.

Corollary

Let $\mathcal{A}$ be a TA with n clocks and W regions, $\alpha < 1/(2n)$, and $\Delta < \frac{\alpha}{2^{2^W} \cdot (4n+2)}$. Let $x \in J$ and y s.t. there exists a trajectory from x to y in $[\![\mathcal{A}]\!]^\Delta$. Then $d(J, y) < \alpha$.

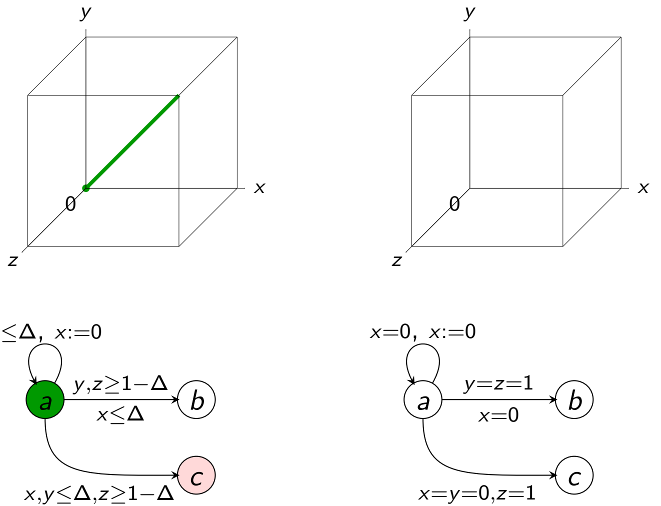
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:



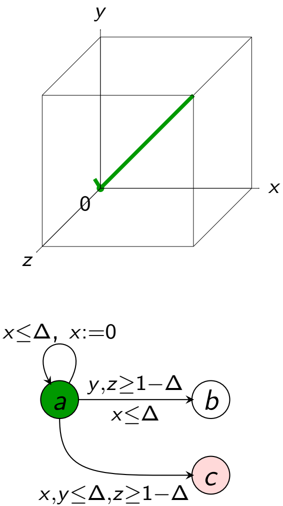
Can we relax the assumption on cycles?

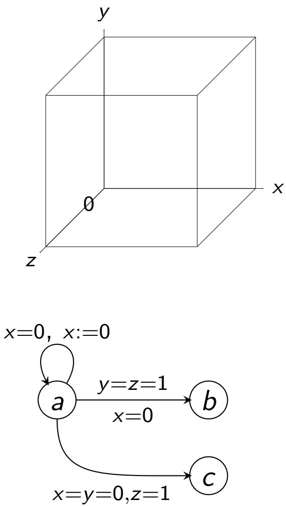
Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:



Can we relax the assumption on cycles?

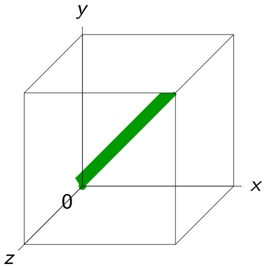
Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:

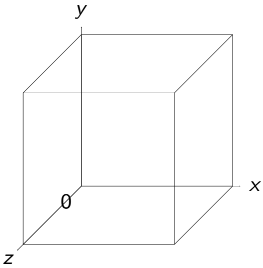


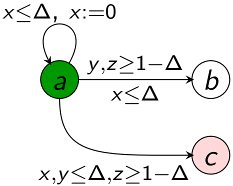


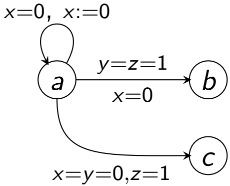
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:



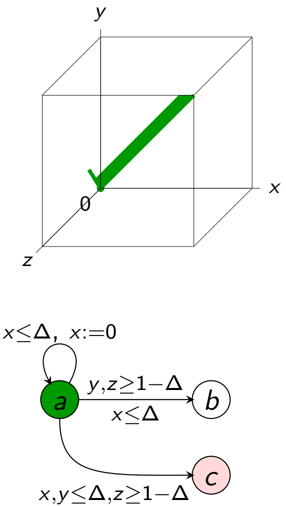


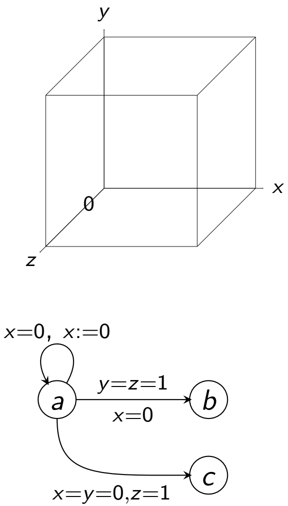




Can we relax the assumption on cycles?

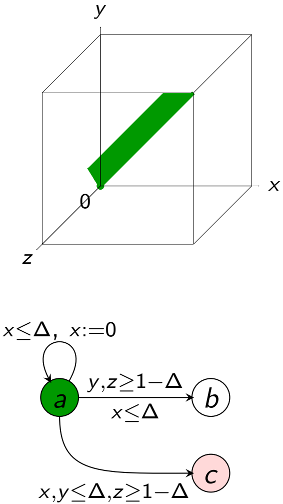
Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:

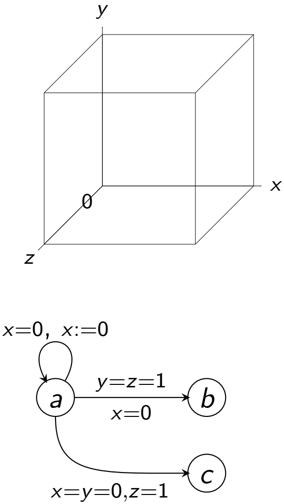




Can we relax the assumption on cycles?

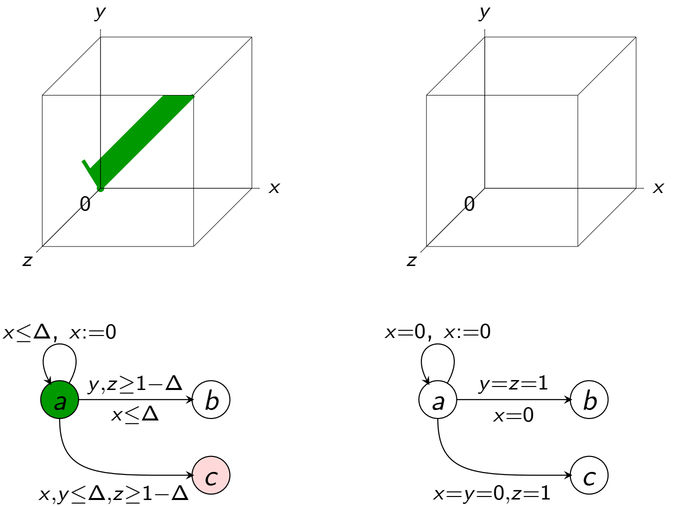
Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:





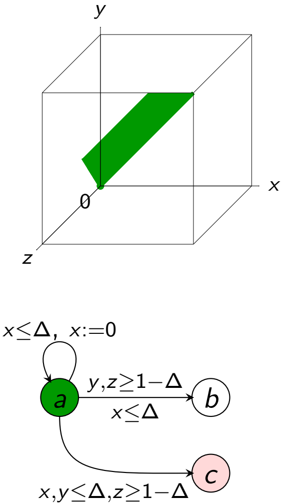
Can we relax the assumption on cycles?

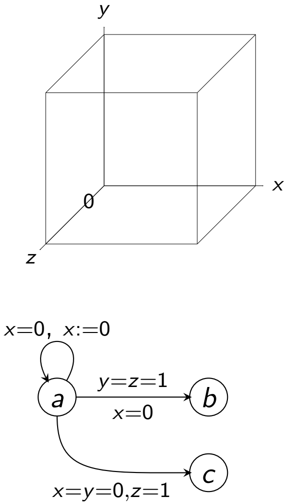
Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:



Can we relax the assumption on cycles?

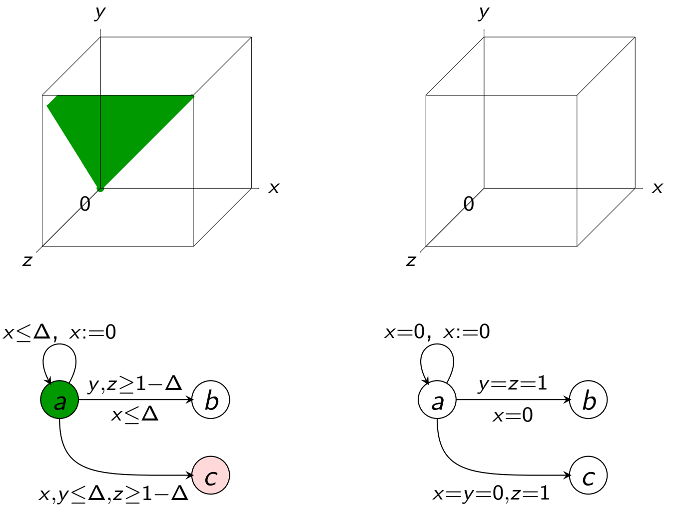
Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:





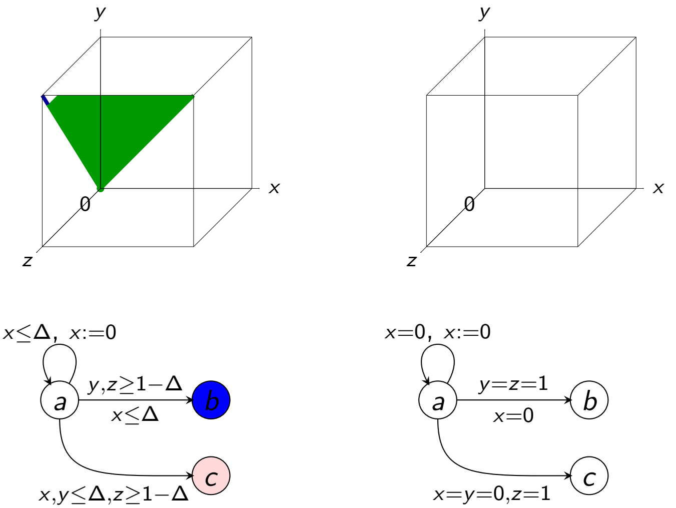
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:



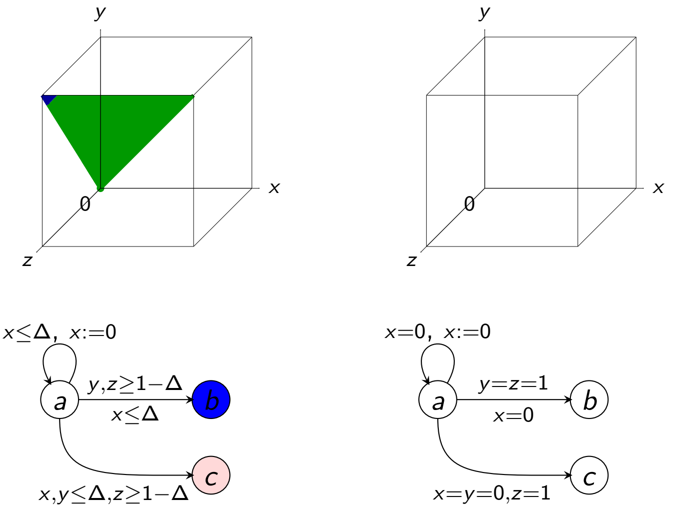
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:



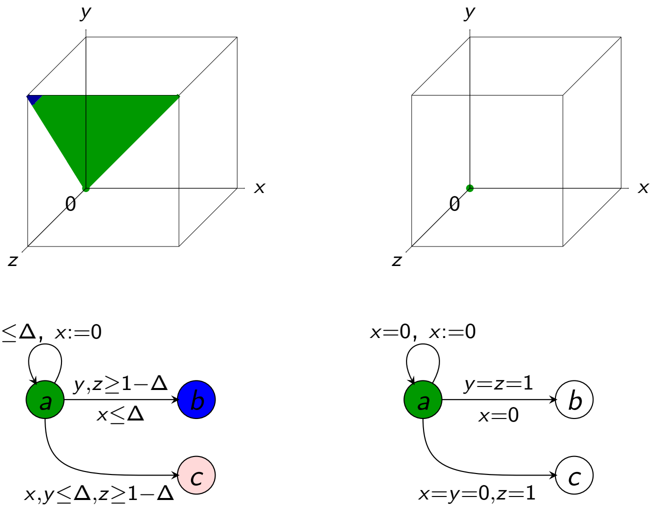
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:



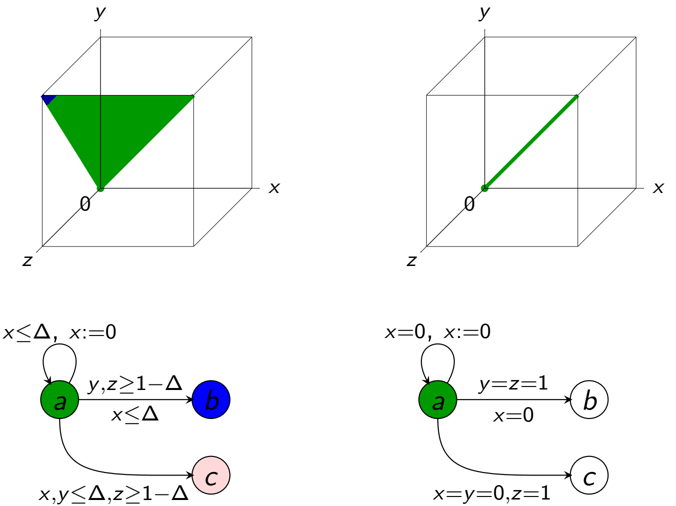
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:



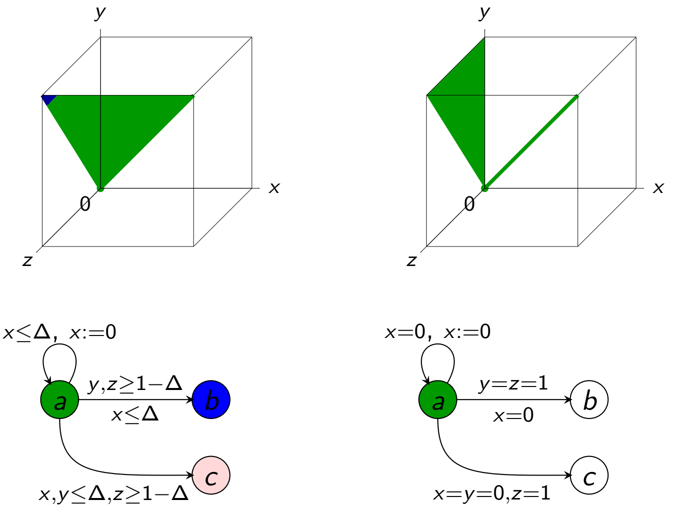
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:



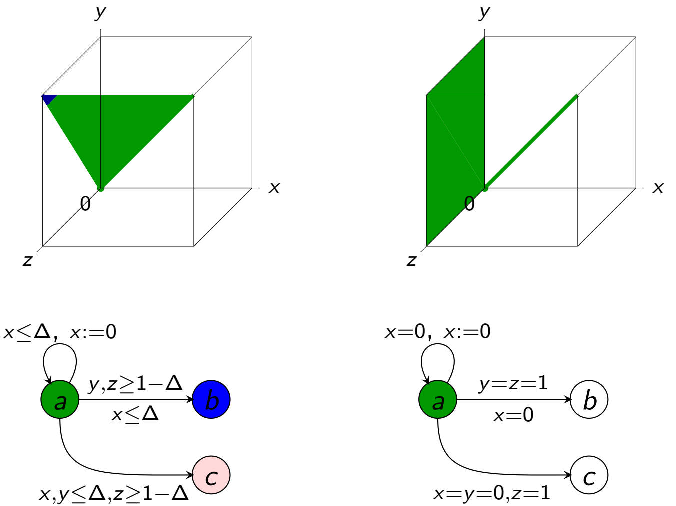
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:



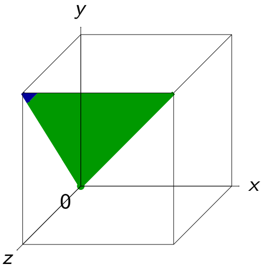
Can we relax the assumption on cycles?

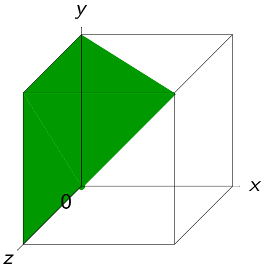
Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:

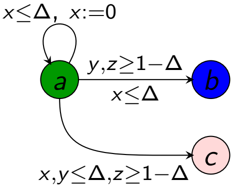


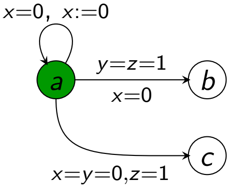
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:



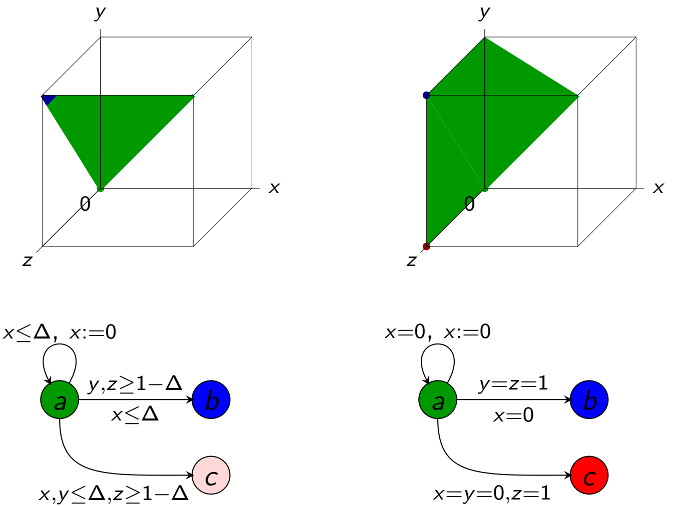






Can we relax the assumption on cycles?

Our algorithm does not work if we relax the "progress-cycle" constraint. For instance:

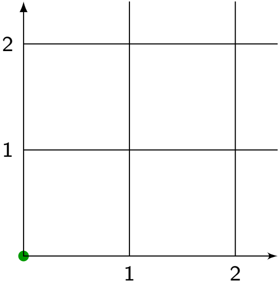


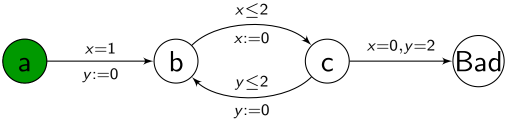
Extension with clock drifts

when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

Extension with clock drifts

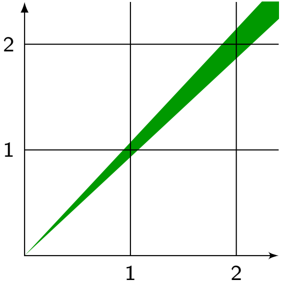
when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

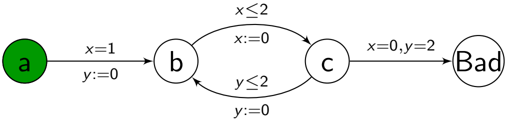




Extension with clock drifts

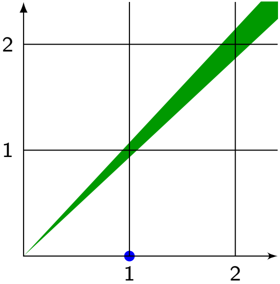
when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

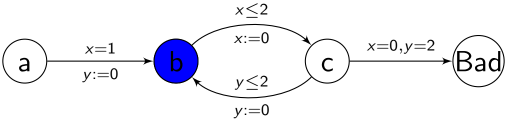




Extension with clock drifts

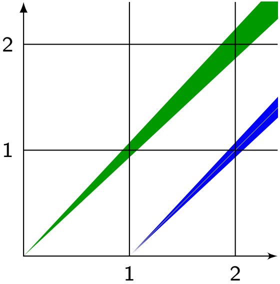
when d time unit elapse, each clock is incremented by some value between $d \times (1-\epsilon)$ and $d \times (1+\epsilon)$.

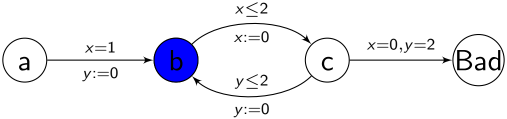




Extension with clock drifts

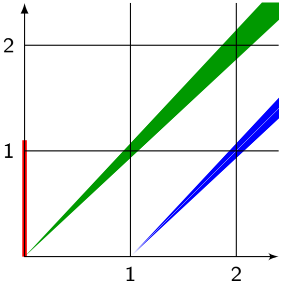
when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

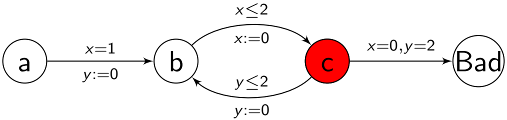




Extension with clock drifts

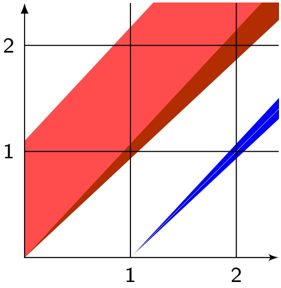
when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

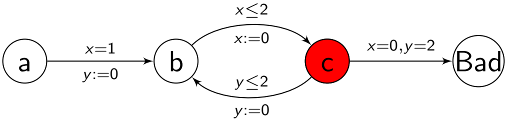




Extension with clock drifts

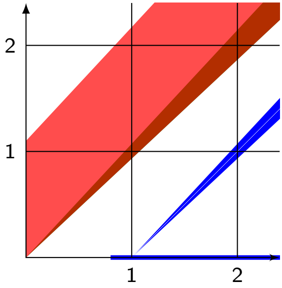
when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

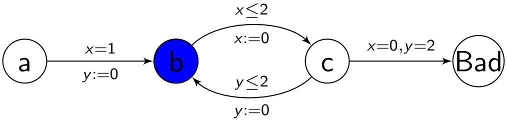




Extension with clock drifts

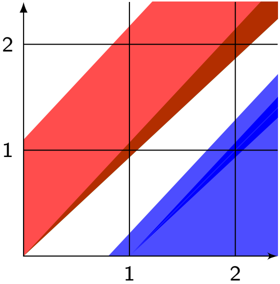
when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

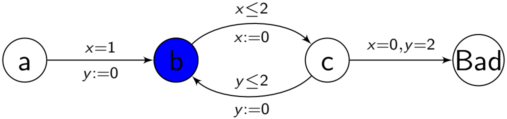




Extension with clock drifts

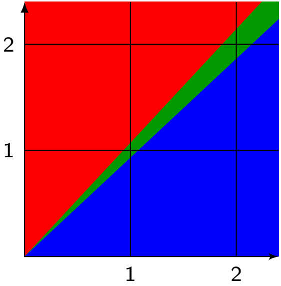
when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

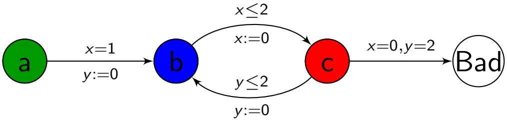




Extension with clock drifts

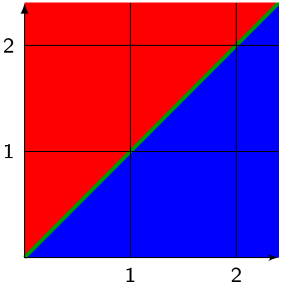
when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

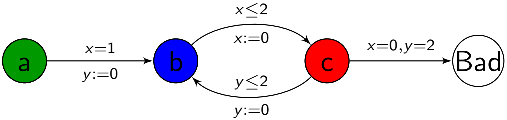




Extension with clock drifts

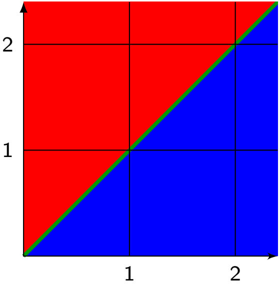
when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

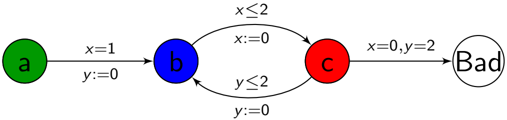




Extension with clock drifts

when d time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.





Since our algorithm is the same as [Pur98]'s, we get the following:

Theorem

$$R_\Delta(\mathcal{A}) = R_\varepsilon(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

Pros and cons of this approach

- Cons:
  - Not very expressive: the platform is very simple, thus not very realistic. Also, we over-approximate the set of executions.
  - New techniques, and much work still needed in order to be applicable;

Pros and cons of this approach

- Cons:
  - Not very expressive: the platform is very simple, thus not very realistic. Also, we over-approximate the set of executions.
  - New techniques, and much work still needed in order to be applicable;

- Pros:
  - Formal approach: we know what we are doing...
  - Reasonnable complexity: “only” PSPACE;
  - Faster is better: the enlarged semantics obviously satisfies this property.

Recent related work

This approach has received much attention in the last 3 years:

- extension to LTL properties [BMR06]:

- Büchi automata techniques;
- Repeated reachability.

Recent related work

This approach has received much attention in the last 3 years:

- extension to LTL properties [BMR06]:

  - Büchi automata techniques;
  - Repeated reachability.

- Extension to timed properties:

  - Different techniques;
  - No restrictions on cycles.

Recent related work

This approach has received much attention in the last 3 years:

- extension to LTL properties [BMR06]:

  - Büchi automata techniques;
  - Repeated reachability.

- Extension to timed properties:

  - Different techniques;
  - No restrictions on cycles.

- adaptations towards symbolic (zone-based) algorithms [DK06,SF07].

Outline of the talk

Conclusions & Future Work

- Implementability is an important problem: the semantics of timed automata is too mathematical;
- Two different approaches:
  - modeling the platform is a very expressive approach that involves only classical techniques;
  - enlarging the semantics is a coarser solution, but has nice theoretical properties.

Conclusions & Future Work

- Implementability is an important problem: the semantics of timed automata is too mathematical;
- Two different approaches:
  - modeling the platform is a very expressive approach that involves only classical techniques;
  - enlarging the semantics is a coarser solution, but has nice theoretical properties.

- Future work:
  - Developpment and implementation of symbolic (zone-based) algorithms;
  - Direct synthesis of robust controllers.