Implementation and Robustness of Timed Automata

Joint work with Martin De Wulf, Laurent Doyen and Jean-François Raskin

Nicolas Markey

January 12, 2005

Implementatiblity of a Timed Automaton

- "Implementing" a TA assumes perfect hardware:

  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 a Timed Automaton

- "Implementing" a TA assumes perfect hardware:

  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 a Timed Automaton

- "Implementing" a TA assumes perfect hardware:

  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 a Timed Automaton

- “Implementing” a TA assumes perfect hardware:

  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 a Timed Automaton

- "Implementing" a TA assumes perfect hardware:

  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.

From Implementability to Robustness

[DDR04] defines a semantics for Timed Automata, called "AASAP", by enlarging guards on transitions by a small tolerance Δ:

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

They prove that if the AASAP semantics of the TA does not enter bad states, then there exists an (digital) implementation of the TA that still avoids bad states.

We end up with the following problem:

Problem

Given a Timed Automaton $\mathcal{A}$ and a set of zones B, can we decide the existence of Δ s.t. $Reach([\![A]\!]^{\Delta}) \cap B = \varnothing$?

From Implementability to Robustness

[DDR04] defines a semantics for Timed Automata, called "AASAP", by enlarging guards on transitions by a small tolerance Δ:

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

They prove that if the AASAP semantics of the TA does not enter bad states, then there exists an (digital) implementation of the TA that still avoids bad states.

We end up with the following problem:

Problem

Given a Timed Automaton $\mathcal{A}$ and a set of zones B, can we decide the existence of Δ s.t. $Reach([\![A]\!]^{\Delta}) \cap B = \varnothing$?

Related work – Robustness

Our problem is a robustness question, since it requires that $Reach([\![A]\!]) \cap B = \varnothing$, and that the result still holds for "neighbor" trajectories.

Robustness has already been addressed by Henzinger *et al.* in [GHJ97].

They study different notions of neighborhood, but in any case, two neighbor trajectories will recognize the same untimed word. Their work is thus completely different from ours, since we are interested in the extra behaviors induced by adding neighbor trajectories.

Related work – Robustness

Our problem is a robustness question, since it requires that $Reach(\llbracket A \rrbracket) \cap B = \varnothing$, and that the result still holds for “neighbor” trajectories.

Robustness has already been addressed by Henzinger *et al.* in [GHJ97].
They study different notions of neighborhood, but in any case, two neighbor trajectories will recognize the same untimed word. Their work is thus completely different from ours, since we are interested in the extra behaviors induced by adding neighbor trajectories.

Related work – Robustness

[Pur98] also defines and studies another notion of robustness. In this case, guards are exacts but clocks may evolve at different rates within $[1-\varepsilon, 1+\varepsilon]$.

[Pur98] provides an algorithm to compute the set

$$\bigcap_{\varepsilon>0} \text{Reach}(\llbracket\mathcal{A}\rrbracket^{\varepsilon})$$

in the case where $\mathcal{A}$ only has "progress cycles", *i.e.* cycles along which each clock is reset at least once.

In the sequel, we write

$$R_{\Delta}(\mathcal{A}) = \bigcap_{\Delta>0} \text{Reach}(\llbracket\mathcal{A}\rrbracket^{\Delta}) \qquad R_{\varepsilon}(\mathcal{A}) = \bigcap_{\varepsilon>0} \text{Reach}(\llbracket\mathcal{A}\rrbracket^{\varepsilon})$$

$$R_{\Delta,\varepsilon}(\mathcal{A}) = \bigcap_{\substack{\Delta>0\\ \varepsilon>0}} \text{Reach}(\llbracket\mathcal{A}\rrbracket^{\Delta,\varepsilon})$$

Related work – Robustness

[Pur98] also defines and studies another notion of robustness. In this case, guards are exacts but clocks may evolve at different rates within $[1-\varepsilon, 1+\varepsilon]$.

[Pur98] provides an algorithm to compute the set

$$\bigcap_{\varepsilon>0} \text{Reach}(\llbracket \mathcal{A} \rrbracket^{\varepsilon})$$

in the case where $\mathcal{A}$ only has "progress cycles", *i.e.* cycles along which each clock is reset at least once.

In the sequel, we write

$$R_{\Delta}(\mathcal{A}) = \bigcap_{\Delta>0} \text{Reach}(\llbracket \mathcal{A} \rrbracket^{\Delta}) \qquad R_{\varepsilon}(\mathcal{A}) = \bigcap_{\varepsilon>0} \text{Reach}(\llbracket \mathcal{A} \rrbracket^{\varepsilon})$$

$$R_{\Delta,\varepsilon}(\mathcal{A}) = \bigcap_{\substack{\Delta>0 \\ \varepsilon>0}} \text{Reach}(\llbracket \mathcal{A} \rrbracket^{\Delta,\varepsilon})$$

Our contribution

We adapt (and reprove) [Pur98] algorithm to our notion of robustness. This solves the problem since we have the following result:

Lemma

For any Timed Automata $\mathcal{A}$ and for any set of zones B,

$$R_\Delta(\mathcal{A}) \cap B = \varnothing \quad \textit{iff} \quad \exists \Delta > 0.\ \mathit{Reach}(\llbracket \mathcal{A} \rrbracket^\Delta) \cap B = \varnothing.$$

We also (hope to) extend our algorithm to the natural bi-dimensional enlargement (with both Δ and ε), and to the general case (with no restriction on cycles). We claim the following results:

$$\text{Reach}(\llbracket A \rrbracket) \subseteq R_\varepsilon(\mathcal{A}) = R_\Delta(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A})$$

if $\mathcal{A}$ has only progress cycles, and

$$\text{Reach}(\llbracket A \rrbracket) \subseteq R_\varepsilon(\mathcal{A}) \subseteq R_\Delta(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A})$$

in the general case.

Our contribution

We adapt (and reprove) [Pur98] algorithm to our notion of robustness. This solves the problem since we have the following result:

Lemma

For any Timed Automata $\mathcal{A}$ and for any set of zones B,

$$R_\Delta(\mathcal{A}) \cap B = \varnothing \quad \textit{iff} \quad \exists \Delta > 0.\ \mathit{Reach}([\![\mathcal{A}]\!]^\Delta) \cap B = \varnothing.$$

We also (hope to) extend our algorithm to the natural bi-dimensional enlargement (with both Δ and ε), and to the general case (with no restriction on cycles). We claim the following results:

$$\mathrm{Reach}([\![A]\!]) \subseteq R_\varepsilon(\mathcal{A}) = R_\Delta(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A})$$

if $\mathcal{A}$ has only progress cycles, and

$$\mathrm{Reach}([\![A]\!]) \subseteq R_\varepsilon(\mathcal{A}) \subseteq R_\Delta(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A})$$

in the general case.

1. Introduction
   - Motivations
   - From Implementability to Robustness
   - Our contributions
2. The algorithm
   - An intuition
   - The algorithm
   - Proof of the algorithm
3. Extensions
   - With both Δ and ε
   - With no constraint on cycles
4. Conclusions

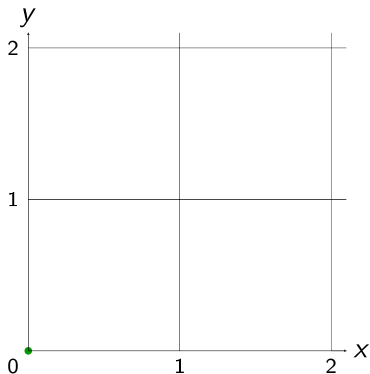
1 Introduction
- Motivations
- From Implementability to Robustness
- Our contributions

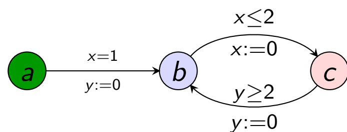
2 The algorithm
- An intuition
- The algorithm
- Proof of the algorithm

3 Extensions
- With both Δ and ε
- With no constraint on cycles

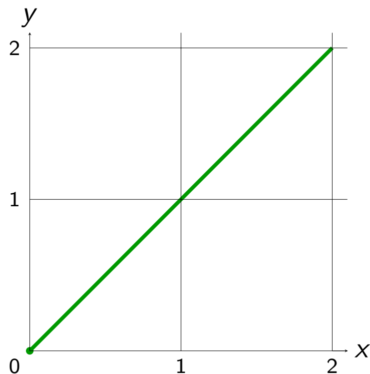
4 Conclusions

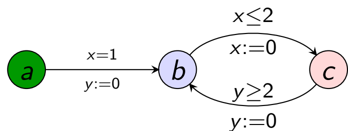
An example: Standard semantics



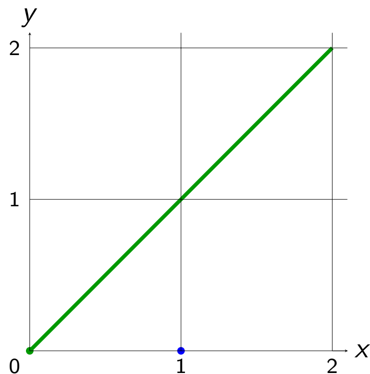


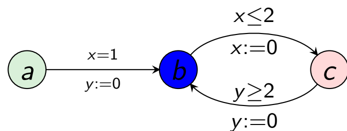
An example: Standard semantics



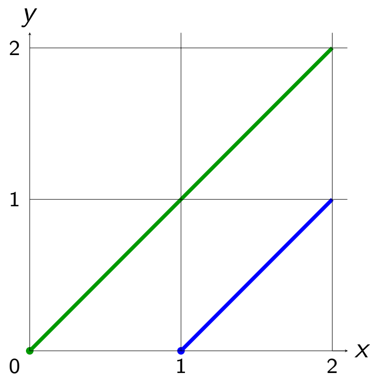


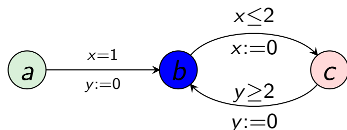
An example: Standard semantics



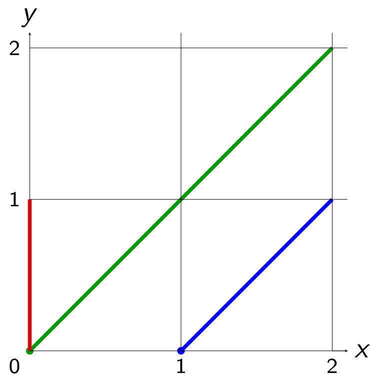


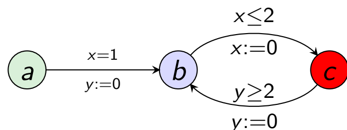
An example: Standard semantics



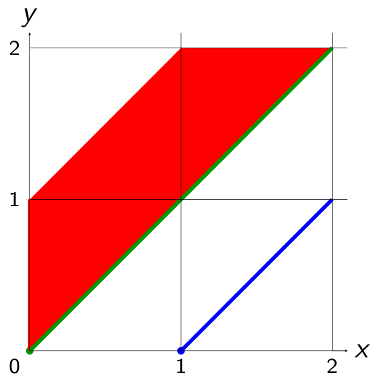


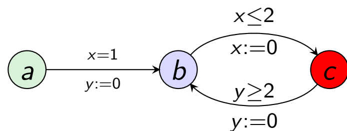
An example: Standard semantics



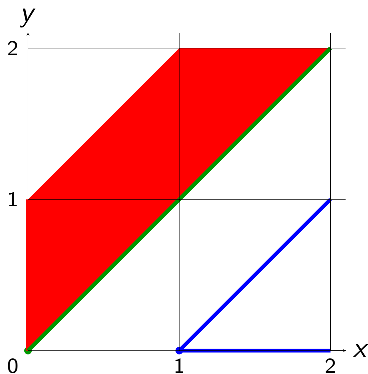


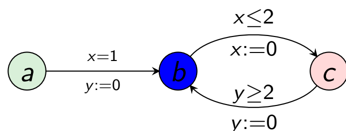
An example: Standard semantics



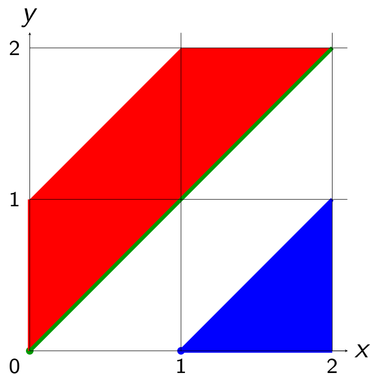


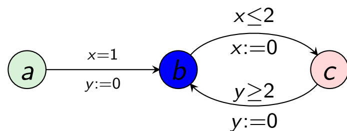
An example: Standard semantics



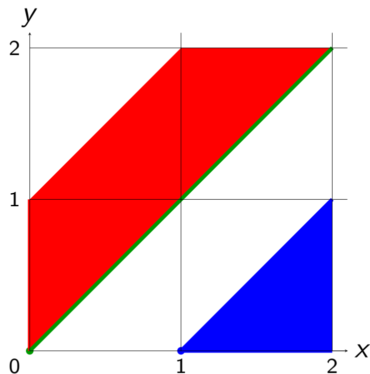


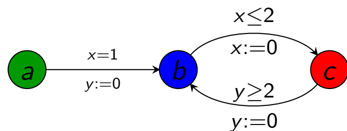
An example: Standard semantics



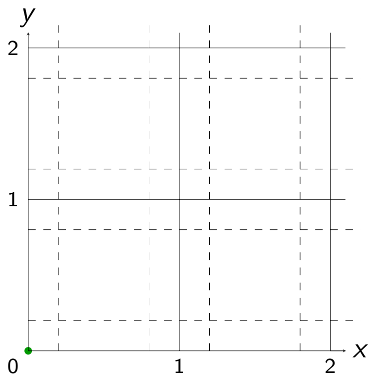


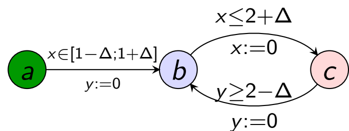
An example: Standard semantics



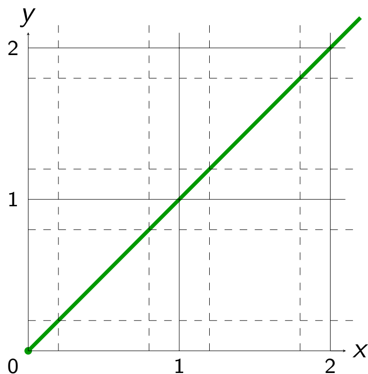


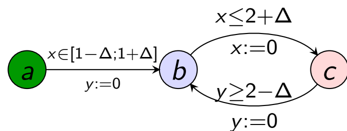
An example with $\Delta > 0$



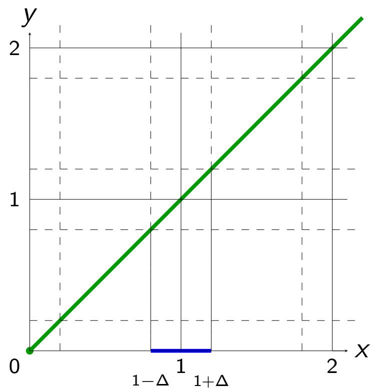


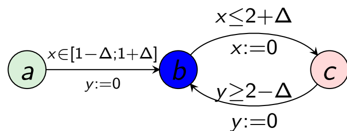
An example with $\Delta > 0$



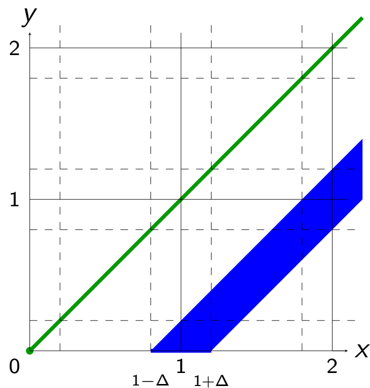


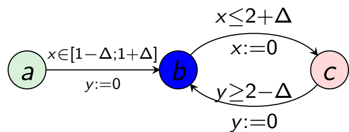
An example with $\Delta > 0$



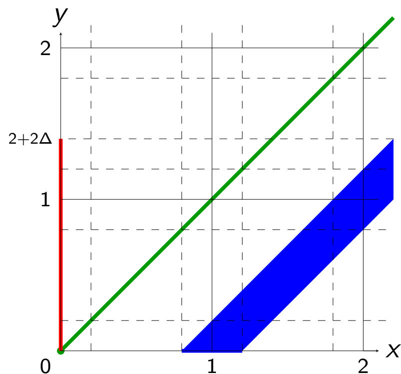


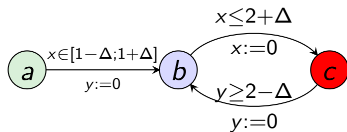
An example with $\Delta > 0$



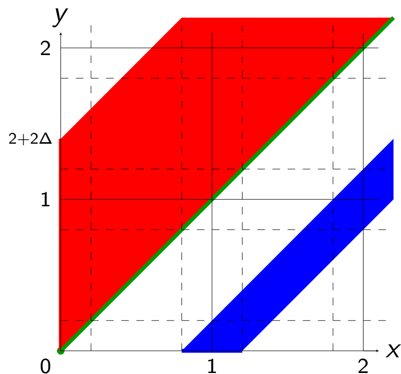


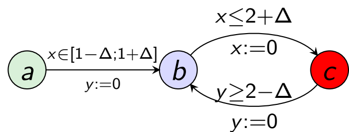
An example with $\Delta > 0$



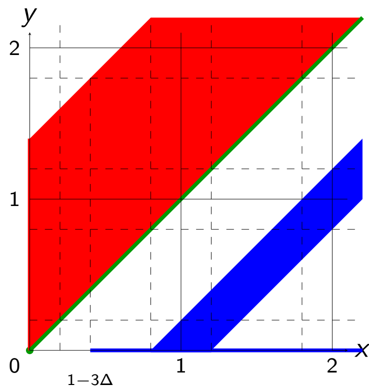


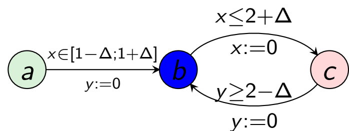
An example with $\Delta > 0$



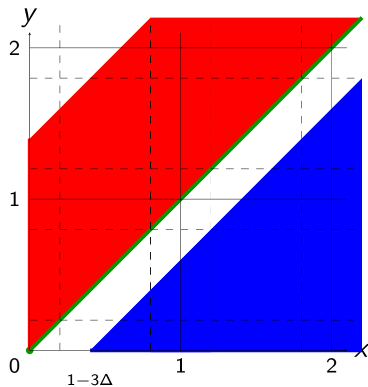


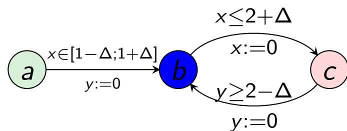
An example with $\Delta > 0$



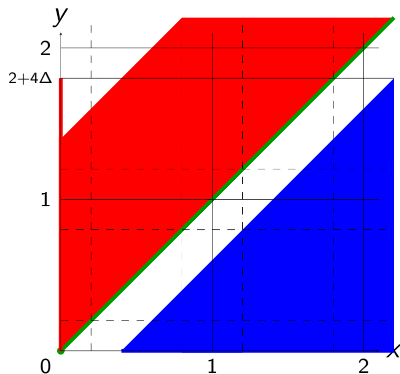


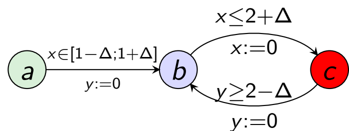
An example with $\Delta > 0$



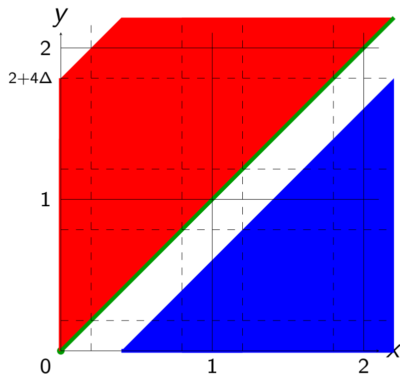


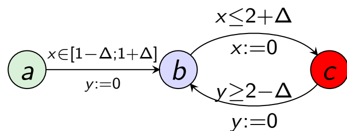
An example with $\Delta > 0$



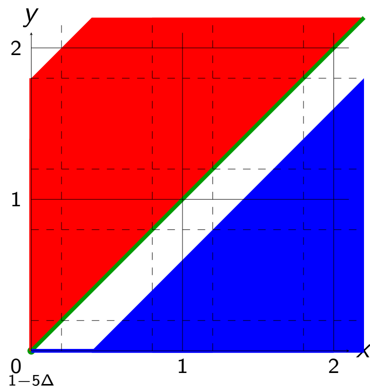


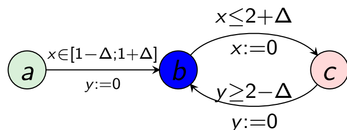
An example with $\Delta > 0$



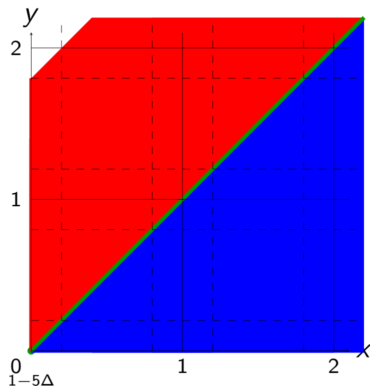


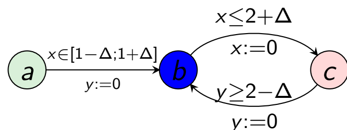
An example with $\Delta > 0$



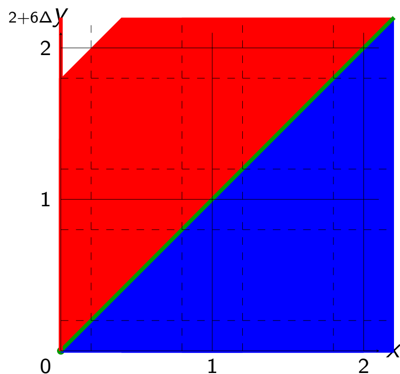


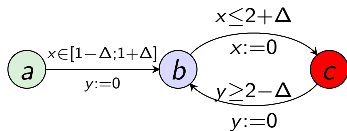
An example with $\Delta > 0$



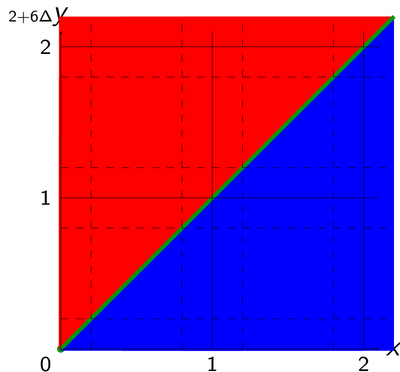


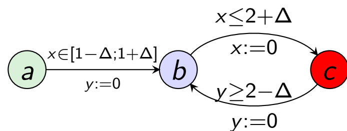
An example with $\Delta > 0$



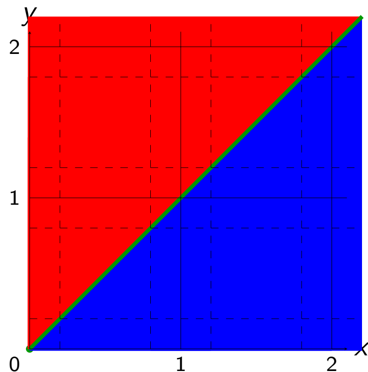


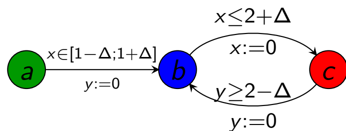
An example with $\Delta > 0$



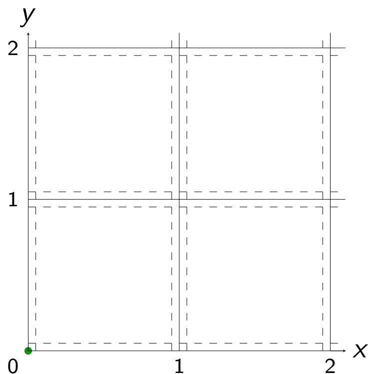


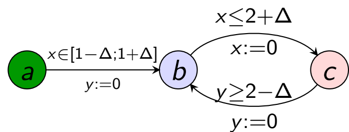
An example with $\Delta > 0$



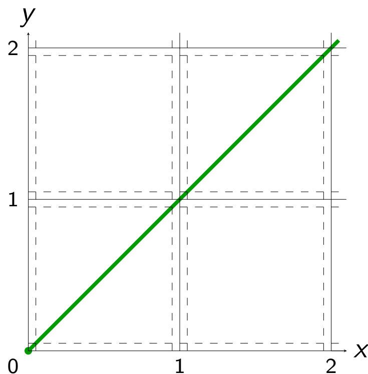


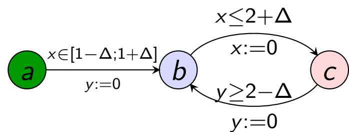
An example with Δ very small



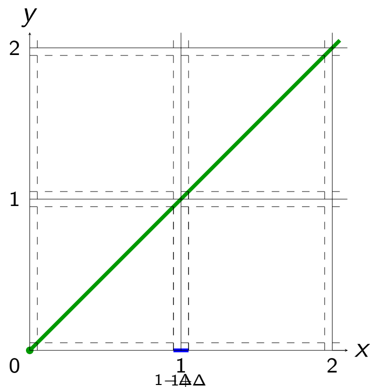


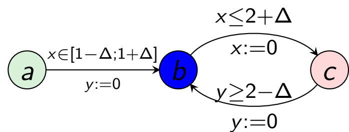
An example with Δ very small



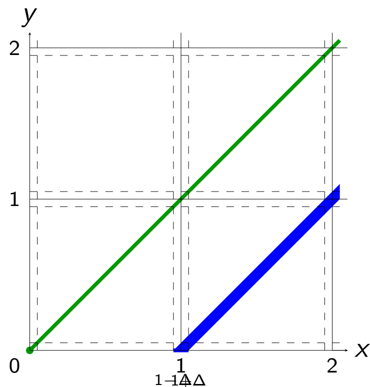


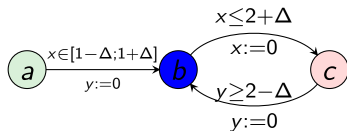
An example with Δ very small



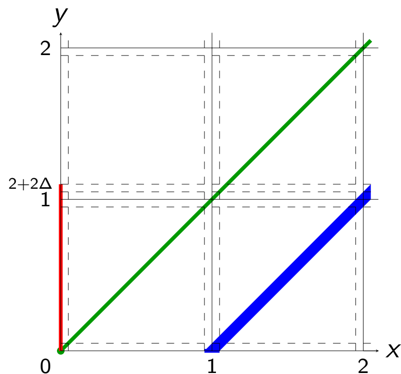


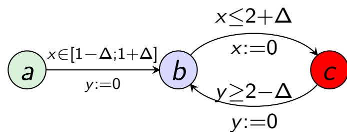
An example with Δ very small



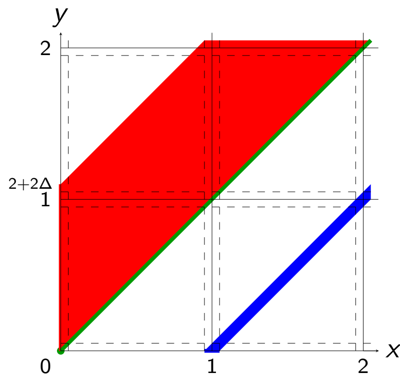


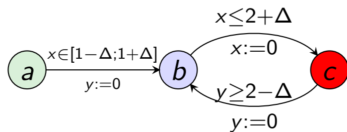
An example with Δ very small



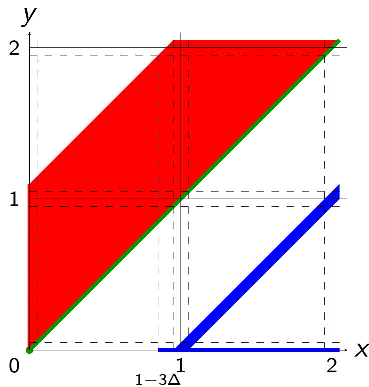


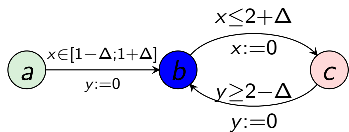
An example with Δ very small



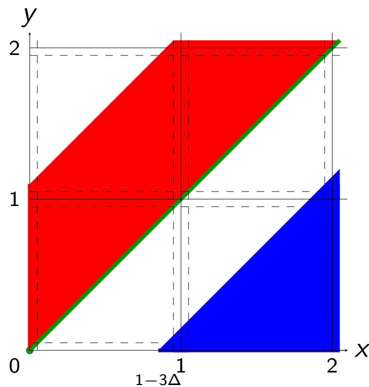


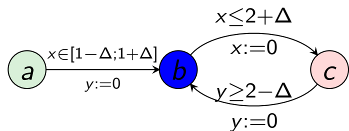
An example with Δ very small



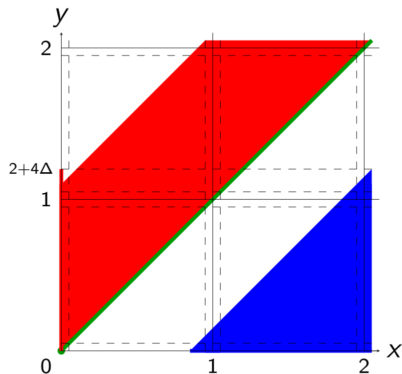


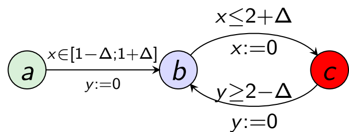
An example with Δ very small



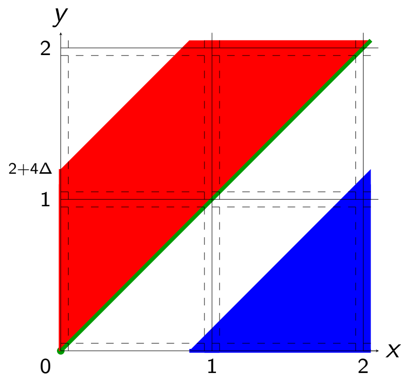


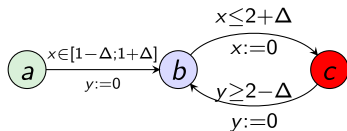
An example with Δ very small



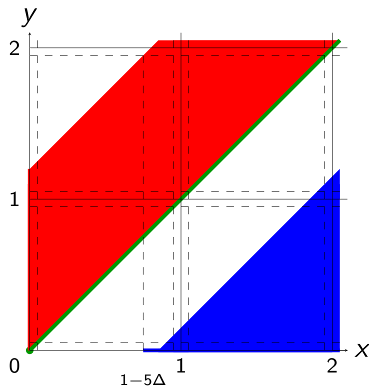


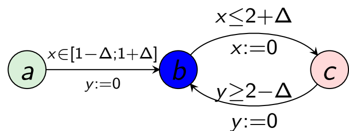
An example with Δ very small



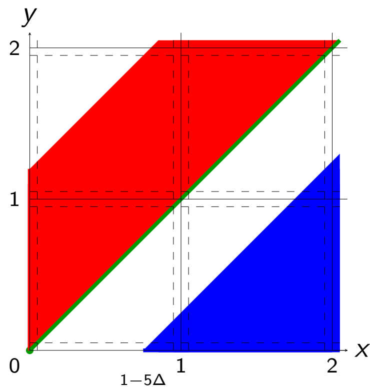


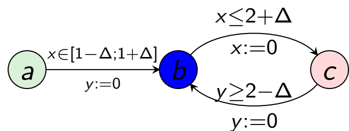
An example with Δ very small



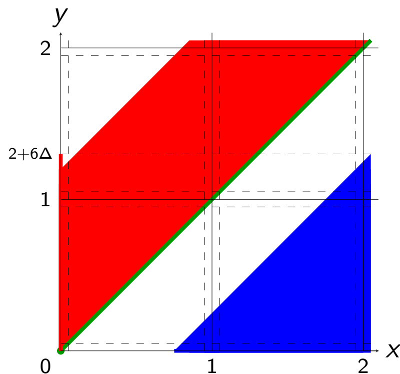


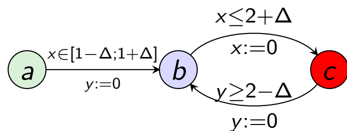
An example with Δ very small



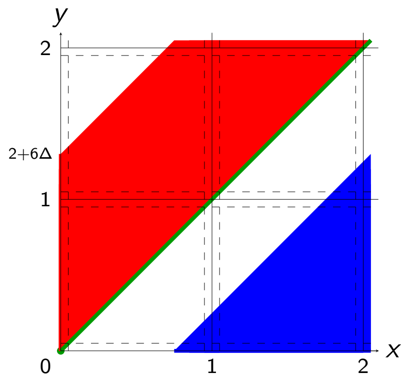


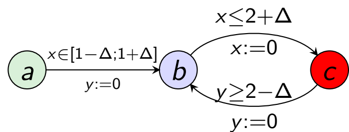
An example with Δ very small



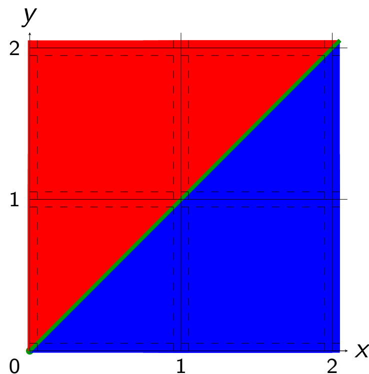


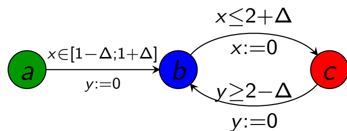
An example with Δ very small



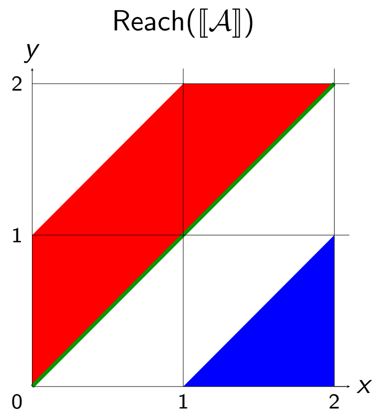


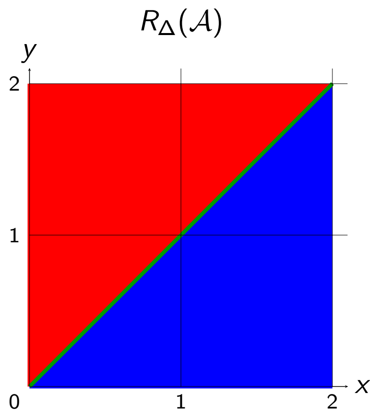
An example with Δ very small

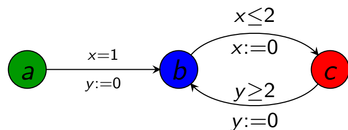




Difference between $[\![\mathcal{A}]\!]$ and $R_\Delta(\mathcal{A})$







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

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

```
1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q_0)];
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);
```

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

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

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q_0)];
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);
```

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

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

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q_0)];
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);
```

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

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

1. build the region graph G of $\mathcal{A}$;
2. compute $\mathtt{SCC}(G)$ = the set of strongly connected components of G;
3. $J := [(q_0)]$;
4. $J := \mathtt{Reach}(G, J)$;
5. while $\exists\ S \in \mathtt{SCC}(G).\ S \not\subseteq J$ and $S \cap J \neq \varnothing$,
   $J := J \cup S$;
   $J := \mathtt{Reach}(G, J)$;
6. return(J);

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

Input: A Timed Automaton $\mathcal{A}$
Output: The set $R_\Delta(\mathcal{A})$

1. build the region graph G of $\mathcal{A}$;
2. compute $\mathtt{SCC}(G)$ = the set of strongly connected components of G;
3. $J := [(q_0)]$;
4. $J := \mathtt{Reach}(G, J)$;
5. while $\exists\ S \in \mathtt{SCC}(G).\ S \not\subseteq J$ and $S \cap J \neq \varnothing$,
   $J := J \cup S$;
   $J := \mathtt{Reach}(G, J)$;
6. return(J);

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

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

1. build the region graph G of $\mathcal{A}$;
2. compute $\mathtt{SCC}(G)$ = the set of strongly connected components of G;
3. $J := [(q_0)]$;
4. $J := \mathtt{Reach}(G, J)$;
5. while $\exists\ S \in \mathtt{SCC}(G).\ S \not\subseteq J$ and $S \cap J \neq \varnothing$,
   $J := J \cup S$;
   $J := \mathtt{Reach}(G, J)$;
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. □

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.

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

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.

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 $d(J, y) < \alpha$.

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

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 $d(J, y) < \alpha$.

1. Introduction
   - Motivations
   - From Implementability to Robustness
   - Our contributions
2. The algorithm
   - An intuition
   - The algorithm
   - Proof of the algorithm
3. Extensions
   - With both Δ and ε
   - With no constraint on cycles
4. Conclusions

Extension with both Δ and ε

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

Theorem

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

We can easily adapt our proofs to the case where both Δ-imprecision and ε-drift are involved.

Extension with both Δ and ε

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

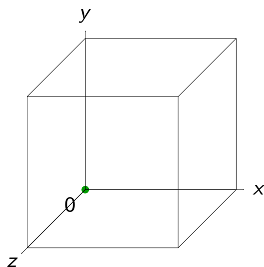
Theorem

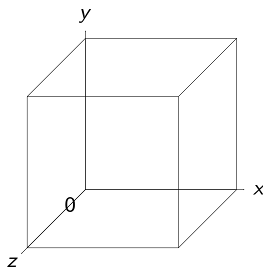
$$R_{\Delta}(\mathcal{A}) = R_{\varepsilon}(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

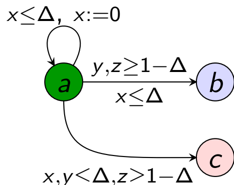
We can easily adapt our proofs to the case where both Δ-imprecision and ε-drift are involved.

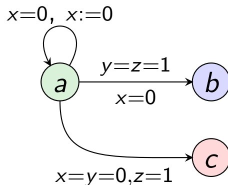
With no constraint on cycles

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



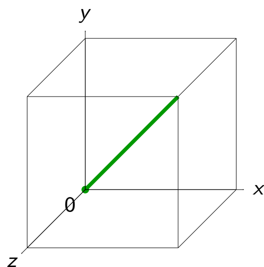


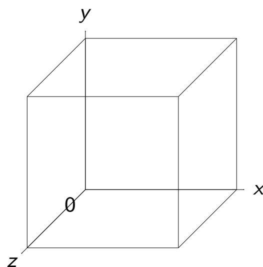


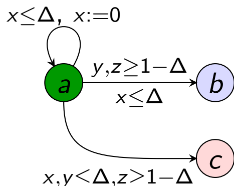


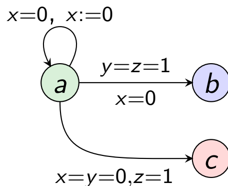
With no constraint on cycles

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



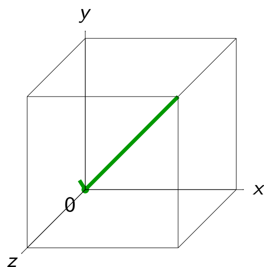


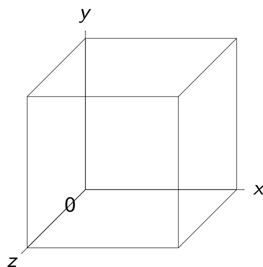


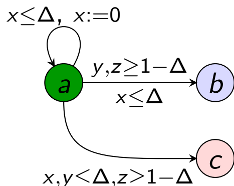


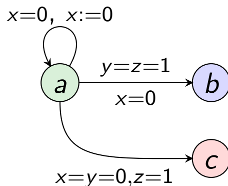
With no constraint on cycles

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



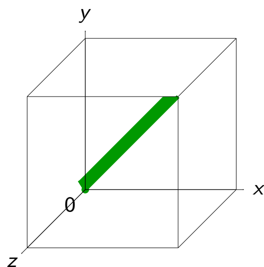


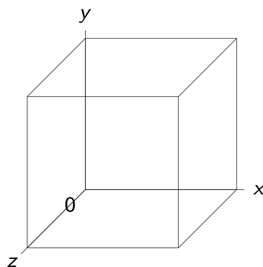


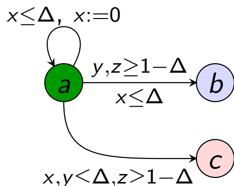


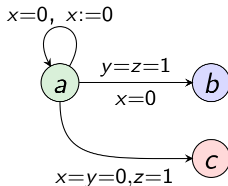
With no constraint on cycles

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



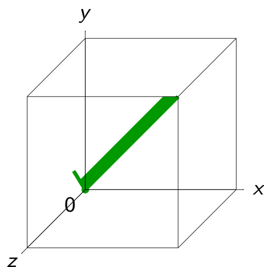


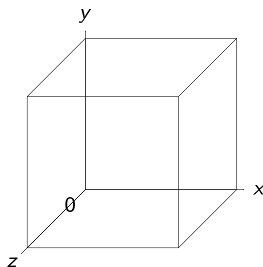


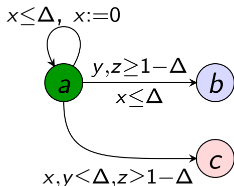


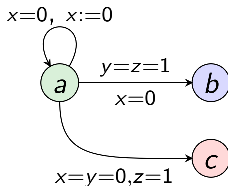
With no constraint on cycles

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



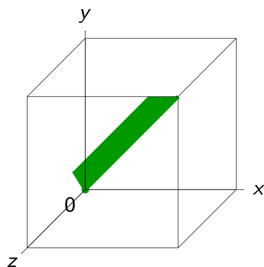


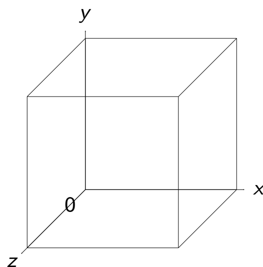


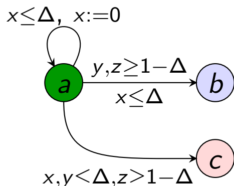


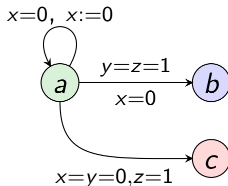
With no constraint on cycles

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



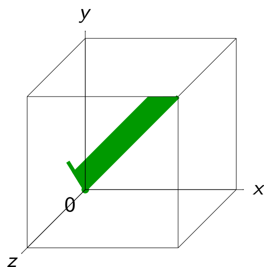


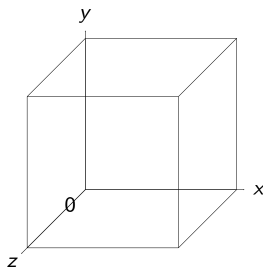


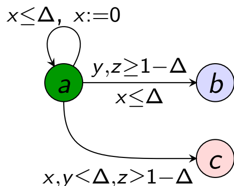


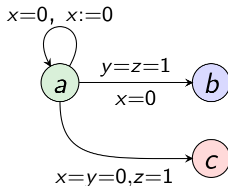
With no constraint on cycles

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



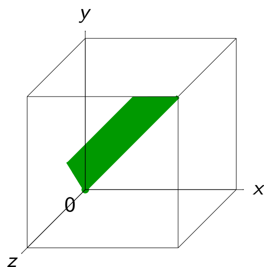


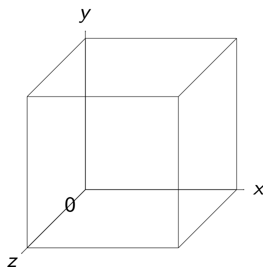


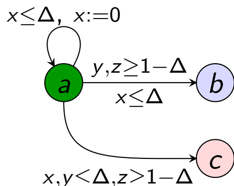


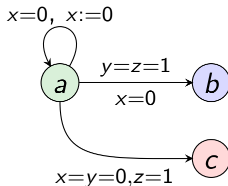
With no constraint on cycles

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



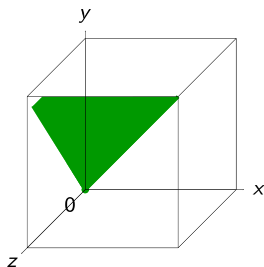


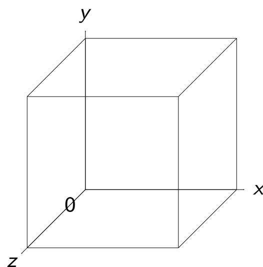


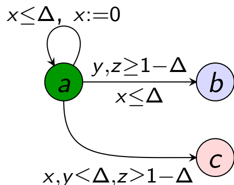


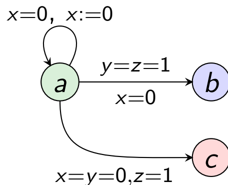
With no constraint on cycles

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



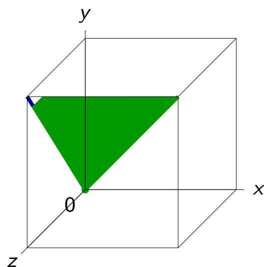


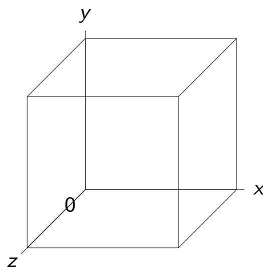


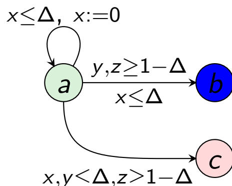


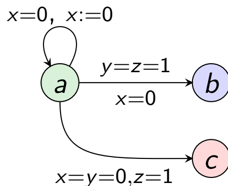
With no constraint on cycles

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



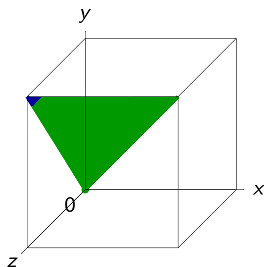


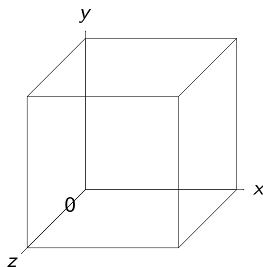


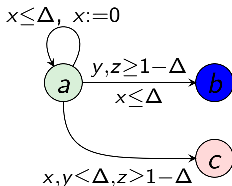


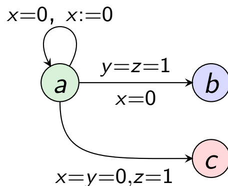
With no constraint on cycles

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



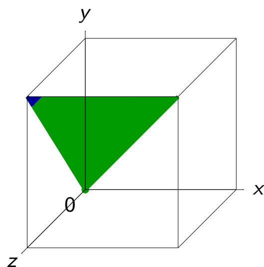


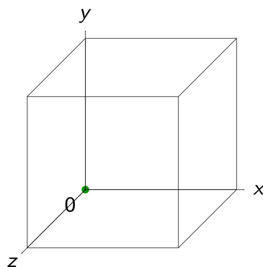


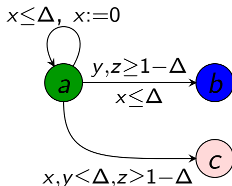


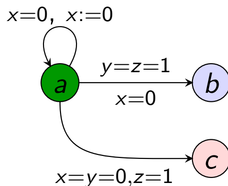
With no constraint on cycles

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



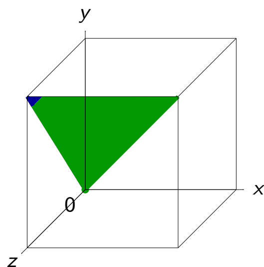


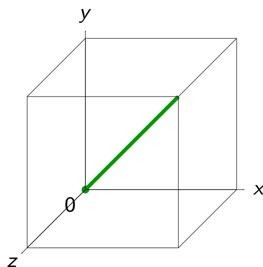


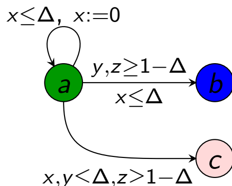


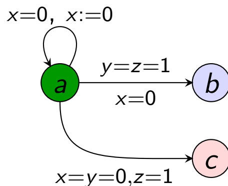
With no constraint on cycles

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



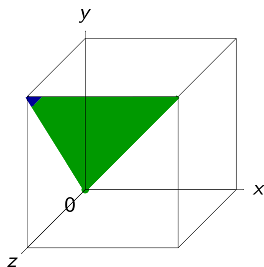


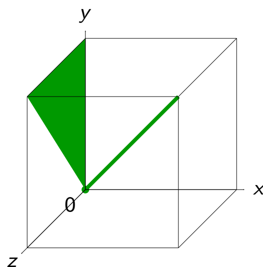


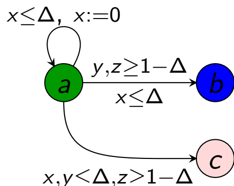


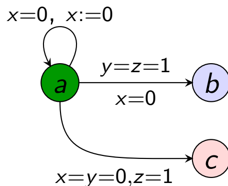
With no constraint on cycles

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



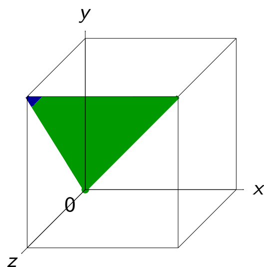


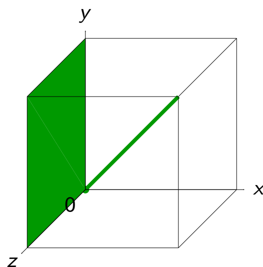


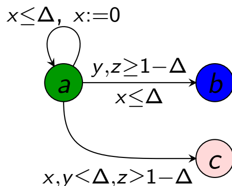


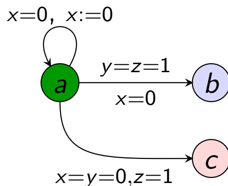
With no constraint on cycles

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



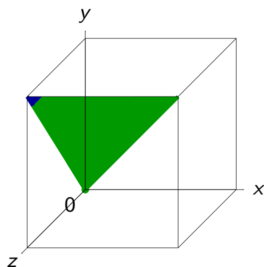


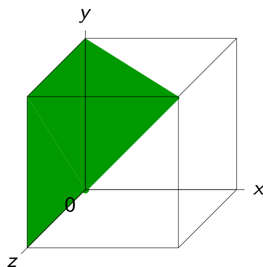


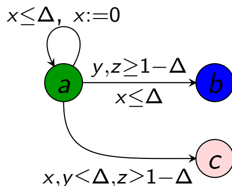


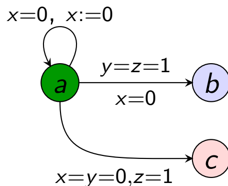
With no constraint on cycles

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



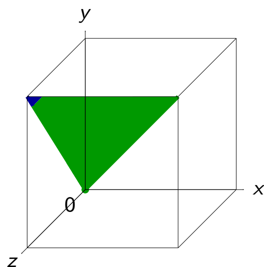


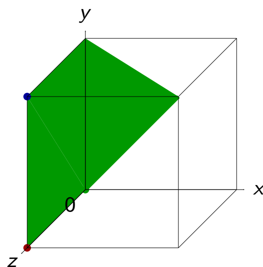


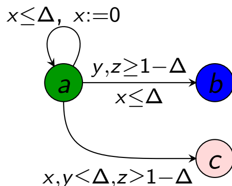


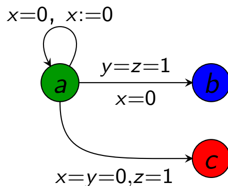
With no constraint on cycles

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









New algorithm for $R_\Delta(\mathcal{A})$ in the general case

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

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q_0)];
4. J := Reach(G, J);
5. while ∃ S ∈ SCC(G). S ⊄ J & S ∩ J ≠ ∅,
       if S is a progress cycle
           then J := J ∪ S;
           else J := J ∪ ((S ∩ J) ↗_{X_S}) ∩ S;
       fi
       J := Reach(G, J);
6. return(J);
```

New algorithm for $R_\Delta(\mathcal{A})$ in the general case

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

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q_0)];
4. J := Reach(G, J);
5. while ∃ S ∈ SCC(G). S ⊄ J & S ∩ J ≠ ∅,
       if S is a progress cycle
           then J := J ∪ S;
           else J := J ∪ ((S ∩ J) ↗_{X_S}) ∩ S;
       fi
       J := Reach(G, J);
6. return(J);
```

New algorithm for $R_\Delta(\mathcal{A})$ in the general case

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

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q_0)];
4. J := Reach(G, J);
5. while ∃ S ∈ SCC(G). S ⊈ J & S ∩ J ≠ ∅,
       if S is a progress cycle
           then J := J ∪ S;
           else J := J ∪ ((S ∩ J) ↗_{X_S}) ∩ S;
       fi
       J := Reach(G, J);
6. return(J);
```

New algorithm for $R_\Delta(\mathcal{A})$ in the general case

```
Input: A Timed Automaton 𝒜
Output: The set R_Δ(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 & S ∩ J ≠ ∅,
        if S is a progress cycle
            then J := J ∪ S;
            else J := J ∪ ((S ∩ J) ↗_{X_S}) ∩ S;
        fi
        J := Reach(G, J);
6. return(J);
```

New algorithm for $R_\Delta(\mathcal{A})$ in the general case

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

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q_0)];
4. J := Reach(G, J);
5. while ∃ S ∈ SCC(G). S ⊄ J & S ∩ J ≠ ∅,
       if S is a progress cycle
           then J := J ∪ S;
           else J := J ∪ ((S ∩ J) ↗_{X_S}) ∩ S;
       fi
       J := Reach(G, J);
6. return(J);
```

New algorithm for $R_\Delta(\mathcal{A})$ in the general case

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

1. build the region graph G of 𝒜;
2. compute SCC(G) = the set of strongly connected
   components of G;
3. J := [(q_0)];
4. J := Reach(G, J);
5. while ∃ S ∈ SCC(G). S ⊄ J & S ∩ J ≠ ∅,
        if S is a progress cycle
             then J := J ∪ S;
             else J := J ∪ ((S ∩ J) ↗_{X_S}) ∩ S;
        fi
        J := Reach(G, J);
6. return(J);
```

Conclusions & Future Work

- Current work:
  - Prove that our last algorithm is correct.
- Conclusions: Surprising results
  - With only progress cycles:

$$\mathrm{Reach}(\llbracket A \rrbracket) \subseteq R_\varepsilon(\mathcal{A}) = R_\Delta(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

  - In the general case:

$$\mathrm{Reach}(\llbracket A \rrbracket) \subseteq R_\varepsilon(\mathcal{A}) \subseteq R_\Delta(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

  - Still PSPACE-complete.
- Future work:
  - Model checking TA with this semantics,
  - Generation of robust controlers,
  - Language problems for this semantics...

Conclusions & Future Work

- Current work:
  - Prove that our last algorithm is correct.
- Conclusions: Surprising results
  - With only progress cycles:

$$\mathrm{Reach}(\llbracket A \rrbracket) \subseteq R_\varepsilon(\mathcal{A}) = R_\Delta(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

  - In the general case:

$$\mathrm{Reach}(\llbracket A \rrbracket) \subseteq R_\varepsilon(\mathcal{A}) \subseteq R_\Delta(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

  - Still PSPACE-complete.
- Future work:
  - Model checking TA with this semantics,
  - Generation of robust controlers,
  - Language problems for this semantics...

Conclusions & Future Work

- Current work:
  - Prove that our last algorithm is correct.
- Conclusions: Surprising results
  - With only progress cycles:

$$\mathsf{Reach}([\![A]\!]) \subseteq R_{\varepsilon}(\mathcal{A}) = R_{\Delta}(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

  - In the general case:

$$\mathsf{Reach}([\![A]\!]) \subseteq R_{\varepsilon}(\mathcal{A}) \subseteq R_{\Delta}(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

  - Still PSPACE-complete.
- Future work:
  - Model checking TA with this semantics,
  - Generation of robust controlers,
  - Language problems for this semantics...

Conclusions & Future Work

- Current work:
  - Prove that our last algorithm is correct.
- Conclusions: Surprising results
  - With only progress cycles:

$$\text{Reach}(\llbracket A \rrbracket) \subseteq R_{\varepsilon}(\mathcal{A}) = R_{\Delta}(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

  - In the general case:

$$\text{Reach}(\llbracket A \rrbracket) \subseteq R_{\varepsilon}(\mathcal{A}) \subseteq R_{\Delta}(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

  - Still PSPACE-complete.
- Future work:
  - Model checking TA with this semantics,
  - Generation of robust controlers,
  - Language problems for this semantics...

Conclusions & Future Work

- Current work:
  - Prove that our last algorithm is correct.
- Conclusions: Surprising results
  - With only progress cycles:

$$\mathsf{Reach}(\llbracket A \rrbracket) \subseteq R_{\varepsilon}(\mathcal{A}) = R_{\Delta}(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

  - In the general case:

$$\mathsf{Reach}(\llbracket A \rrbracket) \subseteq R_{\varepsilon}(\mathcal{A}) \subseteq R_{\Delta}(\mathcal{A}) = R_{\Delta,\varepsilon}(\mathcal{A}).$$

  - Still PSPACE-complete.
- Future work:
  - Model checking TA with this semantics,
  - Generation of robust controlers,
  - Language problems for this semantics...