Robustness and Implementability of Timed Systems

Ocan Sankur

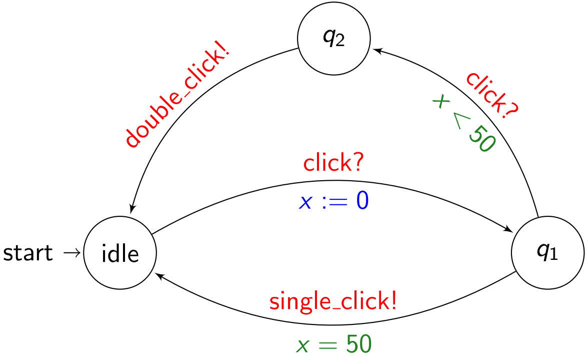
LSV, CNRS & ENS de Cachan

Based on joint work with Patricia Bouyer, Kim Larsen, Nicolas Markey, Claus Thrane

LABRI, 12 Janvier 2012

Timed Automata: **Exact** Semantics

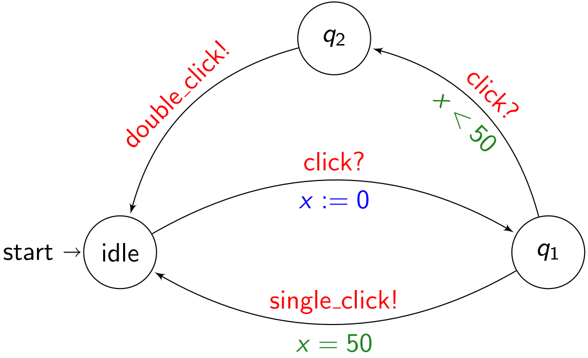
Timed automata = Finite automata + Analog clocks. [Alur and Dill 1994]



- Clocks cannot be stopped, all grow at the same rate.
- An edge is activated when its clock constraint holds.
- A clock can be reset by a transition.

Timed Automata: **Exact** Semantics

Timed automata = Finite automata + Analog clocks. [Alur and Dill 1994]



Runs of a timed automaton: $\mathcal{A}$

$$(\text{idle}, x = 0) \xrightarrow{23.7} (\text{idle}, x = 23.7) \xrightarrow{\textit{click?}} (q_1, x = 0) \xrightarrow{10} (q_1, x = 10)$$
$$\xrightarrow{\textit{click?}} (q_2, x = 10) \xrightarrow{\textit{double\_click}} (\text{idle}, x = 10) \ldots$$

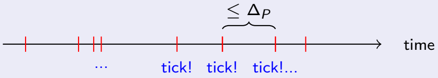
Timed Automata: **Program** Semantics

The semantics of timed automata is idealistic:

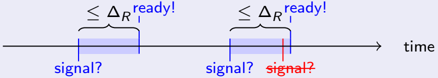
- No minimum delay between actions, $\xrightarrow{a}\xrightarrow{0.00001}\xrightarrow{b}$.
- clocks are infinitely precise. "$1 \leq x \leq 3$".

Real-world systems have

- **digital clocks** updated regularly:



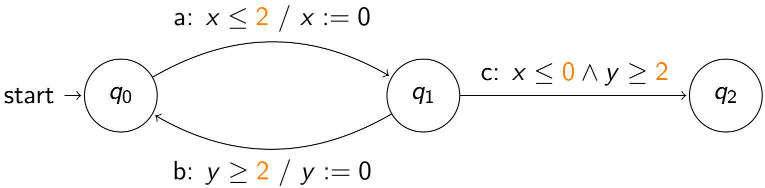
- nonzero **reaction time**:



Program semantics studied by [De Wulf, Doyen and Raskin 2004].

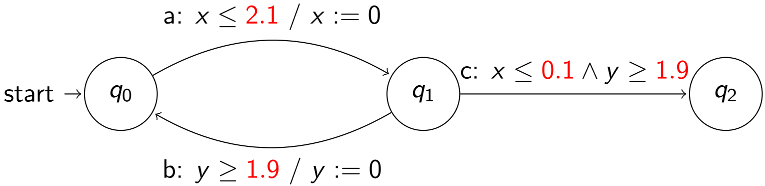
Timed Automata: **Enlarged** Semantics

Clock imprecisions can be modelled by **enlarging** the clock constraints.
Consider the timed automaton $\mathcal{A}$:



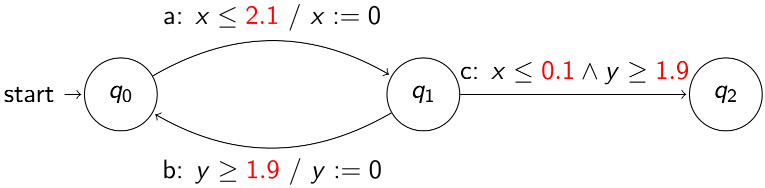
Timed Automata: **Enlarged** Semantics

Clock imprecisions can be modelled by **enlarging** the clock constraints.
For $\Delta = 0.1$, $\mathcal{A}_\Delta$ is defined by,



Timed Automata: **Enlarged** Semantics

Clock imprecisions can be modelled by **enlarging** the clock constraints.
For $\Delta = 0.1$, $\mathcal{A}_\Delta$ is defined by,



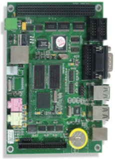
Relation between semantics

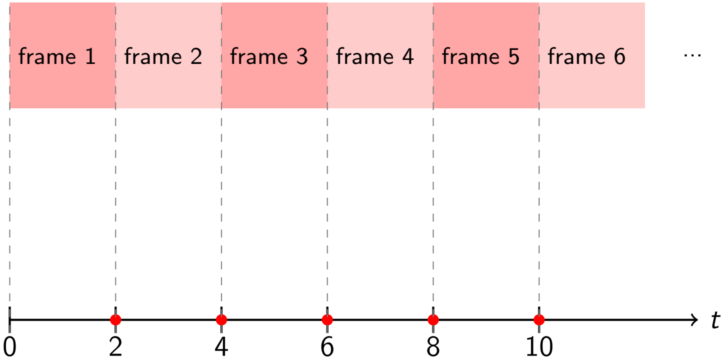
$$\mathcal{A} \quad \sqsubseteq \quad \text{program}(\mathcal{A}_\Delta) \quad \sqsubseteq \quad \mathcal{A}_{2\Delta}$$

for some $\Delta > 0$, [De Wulf, Doyen, Raskin 2004] & [S., Bouyer, Markey 2011].

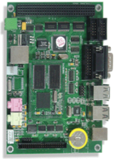
"Implementations can have more behaviours than the exact semantics".

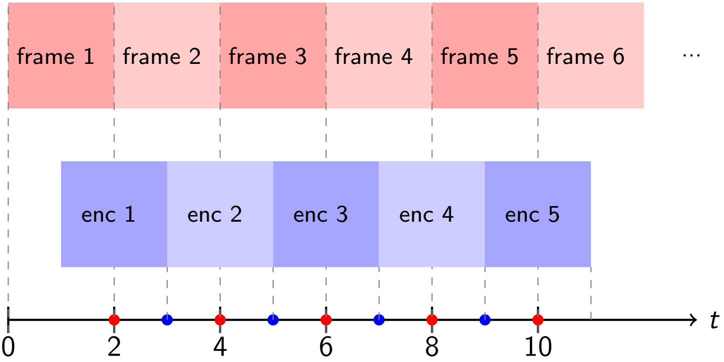
A **Non-Robust** Timed System



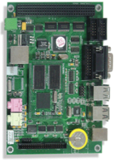


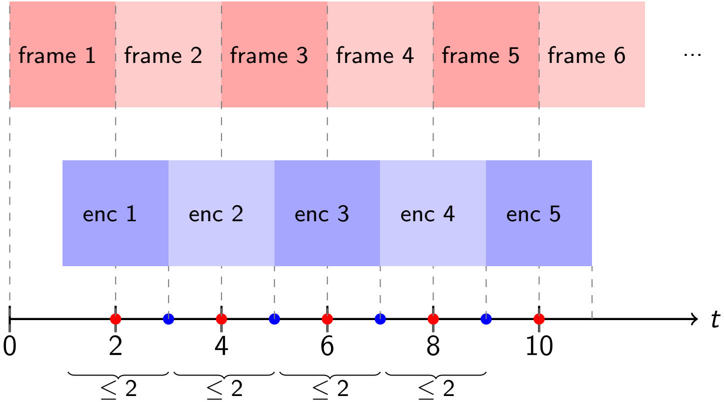
A **Non-Robust** Timed System



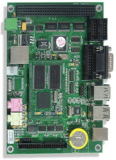


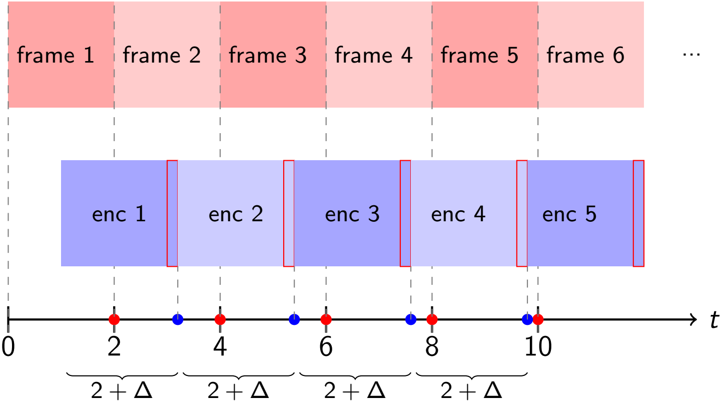
A **Non-Robust** Timed System



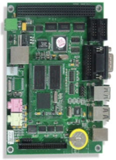


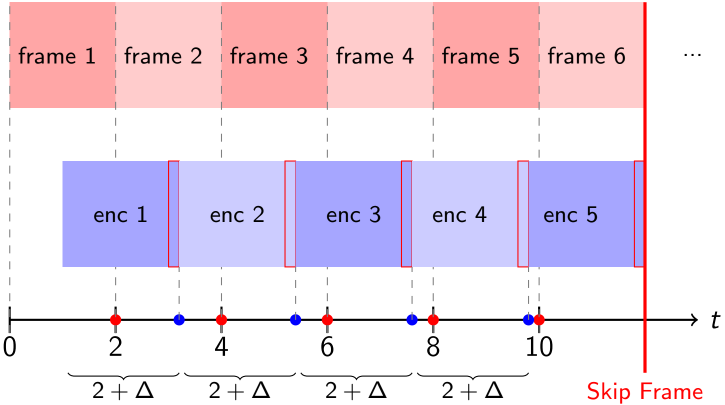
A **Non-Robust** Timed System



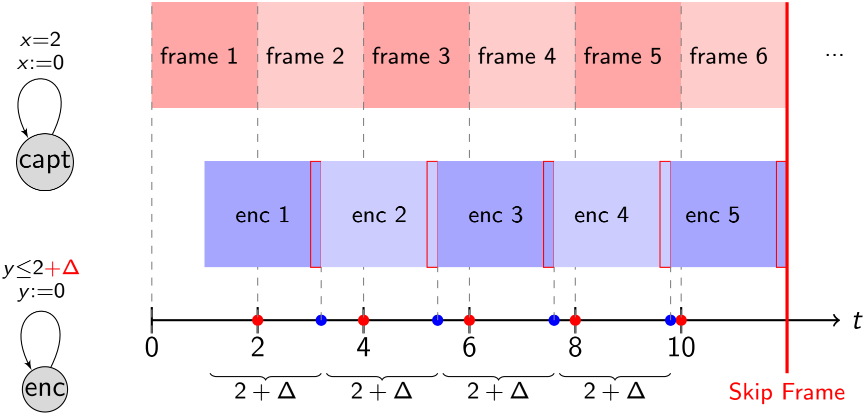


A **Non-Robust** Timed System





A **Non-Robust** Timed System



First Approach

Decide the existence of a bound on Δ under which the automaton satisfies some property.

$\rightsquigarrow$ **Parameterized Robust** model-checking

Background

"Enlarged/Program semantics can **add undesired** behaviour to timed automata".

[Puri 1998, De Wulf, Doyen, Markey, Raskin 2004]

Parameterized Robust Model-Checking

Given TA $\mathcal{A}$ and property ϕ, decide if $\exists \Delta > 0$, $\mathcal{A}_\Delta \models \phi$.

Decidable for:

- Safety (PSPACE-c), [Puri '98], [DDMR '04] [Daws, Kordy '06], [Jaubert, Reynier '11]
- LTL (PSPACE-c), [Bouyer, Markey, Reynier 2006], [Bouyer, Markey, S. 2011]
- coFlat-MTL (EXPSPACE-c) [Bouyer, Markey, Reynier 2008]
- Untimed language equivalence (EXPSPACE) $L(\mathcal{A}) = L(\mathcal{A}_\Delta)$ [S. 2011]

Background

"Enlarged/Program semantics can **add undesired** behaviour to timed automata".

[Puri 1998, De Wulf, Doyen, Markey, Raskin 2004]

Parameterized Robust Model-Checking

Given TA $\mathcal{A}$ and property ϕ, decide if $\exists\Delta > 0$, $\mathcal{A}_\Delta \models \phi$.

Decidable for:

- Safety (PSPACE-c), [Puri '98], [DDMR '04] [Daws, Kordy '06], [Jaubert, Reynier '11]
- LTL (PSPACE-c), [Bouyer, Markey, Reynier 2006], [Bouyer, Markey, S. 2011]
- coFlat-MTL (EXPSPACE-c) [Bouyer, Markey, Reynier 2008]
- Untimed language equivalence (EXPSPACE) $L(\mathcal{A}) = L(\mathcal{A}_\Delta)$ [S. 2011]

Param. Robust Model-Checking: ω-regular properties

Theorem (Bouyer, Markey, S. 2011)

Robust model-checking timed automata against ω-regular properties can be reduced to classical model-checking with optimal complexity (PSPACE).

The algorithm: For any $\mathcal{A}$, there exists some (computable) $\Delta_0 > 0$ s.t.

$$\exists \Delta > 0, \mathcal{A}_\Delta \models \phi \quad \Leftrightarrow \quad \mathcal{A}_{\Delta_0} \models \phi.$$

But $\mathcal{A}_{\Delta_0}$ is an ordinary timed automaton

Param. Robust Model-Checking: ω-regular properties

Theorem (Bouyer, Markey, S. 2011)

Robust model-checking timed automata against ω-regular properties can be reduced to classical model-checking with optimal complexity (PSPACE).

The algorithm: For any $\mathcal{A}$, there exists some (computable) $\Delta_0 > 0$ s.t.

$$\exists \Delta > 0, \mathcal{A}_\Delta \models \phi \quad \Leftrightarrow \quad \mathcal{A}_{\Delta_0} \models \phi.$$

But $\mathcal{A}_{\Delta_0}$ is an ordinary timed automaton

- Use your favorite model-checker to check robustness.

Promising preliminary experimental results!

Param. Robust Model-Checking: ω-regular properties

Theorem (Bouyer, Markey, S. 2011)

Robust model-checking timed automata against ω-regular properties can be reduced to classical model-checking with optimal complexity (PSPACE).

The algorithm: For any $\mathcal{A}$, there exists some (computable) $\Delta_0 > 0$ s.t.

$$\exists \Delta > 0, \mathcal{A}_\Delta \models \phi \quad \Leftrightarrow \quad \mathcal{A}_{\Delta_0} \models \phi.$$

- Use your favorite model-checker to check robustness.

N.B. An algorithm for this problem was known before for TAs

1. whose all cycles reset all clocks + bounded clocks,
2. based on a modification of the region construction (one couldn't directly use existing model-checkers).

A Stronger Notion: Untimed Language Preservation

Untimed Language Preservation

Does there exist $\Delta > 0$ s.t. $L_{\mathsf{untime}}(\mathcal{A}_\Delta) = L_{\mathsf{untime}}(\mathcal{A})$.

A Stronger Notion: Untimed Language Preservation

Untimed Language Preservation

Does there exist $\Delta > 0$ s.t. $L_{\text{untime}}(\mathcal{A}_\Delta) = L_{\text{untime}}(\mathcal{A})$.

Theorem (S. 2011)

Untimed language preservation is decidable in EXPSPACE in general, and in PSPACE for a deterministic subclass.

The algorithm: For any $\mathcal{A}$, there exists some $\Delta_0 > 0$ such that

$$\exists \Delta > 0, L_{\text{untime}}(\mathcal{A}_\Delta) = L_{\text{untime}}(\mathcal{A}) \quad \Leftrightarrow \quad L_{\text{untime}}(\mathcal{A}_{\Delta_0}) = L_{\text{untime}}(\mathcal{A}).$$

- Only check whether $L_{\text{untime}}(\mathcal{A}_{\Delta_0}) = L_{\text{untime}}(\mathcal{A})$

N.B. Untimed language universality (thus equiv.) is EXPSPACE-complete [Brenguier, Göller, S. 2011 - unpublished].

Param. Robust Model-checking: Summary

Conclusion

Imprecisions /unexpected delays always add additional behaviour in implementation.

Param. robust model-checking: check whether the additional behaviours are "harmless".

Same theoretical complexity as for model-checking timed automata.

It is still **open** whether one can derive efficient algorithms.

Param. Robust Model-checking: Summary

Conclusion

Imprecisions /unexpected delays always add additional behaviour in implementation.

Param. robust model-checking: check whether the additional behaviours are "harmless".

Same theoretical complexity as for model-checking timed automata.

It is still **open** whether one can derive efficient algorithms.

Next: Prevent additional behaviours to appear in implementation.

Second Approach

Transform a given timed automaton into a robust one.

Robust implementation /refinement

Approximate implementation

Preliminary definition: Two states are ϵ-**bisimilar** if there is a bisimulation in which delays differ by at most ϵ. — denoted by $\sim_\epsilon$

Approximate implementation

Preliminary definition: Two states are ϵ-**bisimilar** if there is a bisimulation in which delays differ by at most ϵ. — denoted by $\sim_\epsilon$

Theorem [Bouyer, Larsen, Markey, S., Thrane 2011]

Given any timed automaton $\mathcal{A}$, any $\epsilon > 0$, one can compute $\mathcal{A}'$ such that

- $\mathcal{A} \sim_0 \mathcal{A}'$,
- $\mathcal{A}' \sim_\epsilon \mathcal{A}'_\Delta$ for all $0 \leq \Delta < O(\epsilon)$,

We get $\mathcal{A} \sim_\epsilon \mathcal{A}'_\Delta$.

In practice: Design / model-check $\mathcal{A}$, then "compile to" $\mathcal{A}'$.

Approximate implementation

Preliminary definition: Two states are ϵ-**bisimilar** if there is a bisimulation in which delays differ by at most ϵ. — denoted by $\sim_\epsilon$

Theorem 2 [Bouyer, Larsen, Markey, S., Thrane 2011]

Given any timed automaton $\mathcal{A}$, any $\epsilon > 0$, one can compute $\mathcal{A}'$ such that

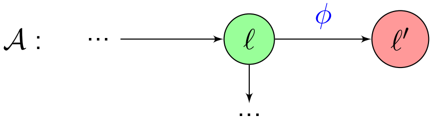
- $\mathcal{A} \sim_0 \mathcal{A}'$,
- Same locations reachable in $\mathcal{A}'$ and $\mathcal{A}'_\Delta$ for all $0 \leq \Delta < O(\epsilon)$,

We get $\mathcal{A}$ is safe $\Rightarrow \mathcal{A}'$ is safe.

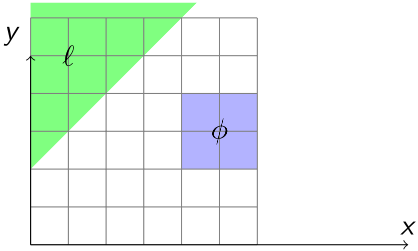
In practice: Design / model-check $\mathcal{A}$, then "compile to" $\mathcal{A}'$.

Approximate implementation: Safety

Consider a timed automaton $\mathcal{A}$ with clocks x, y,
such that location ℓ' is not reachable:

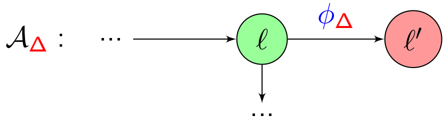


Consider the **reachable states** in ℓ:

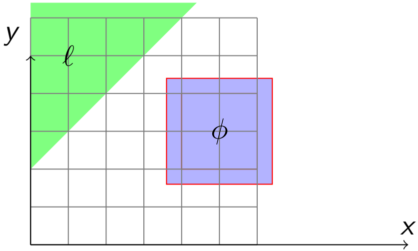


Approximate implementation: Safety

Consider a timed automaton $\mathcal{A}$ with clocks x, y,
such that location ℓ' is not reachable:

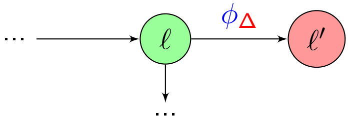


Consider the **reachable states** in ℓ:

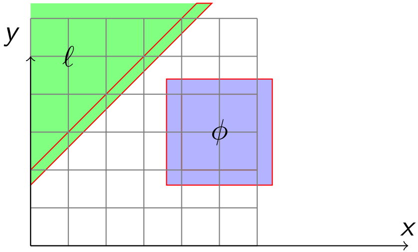


Approximate implementation: Safety

Consider a timed automaton $\mathcal{A}$ with clocks x, y,
such that location ℓ' is not reachable:

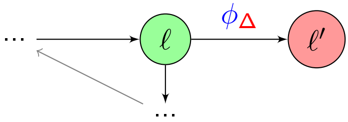


Consider the **reachable states** in ℓ:

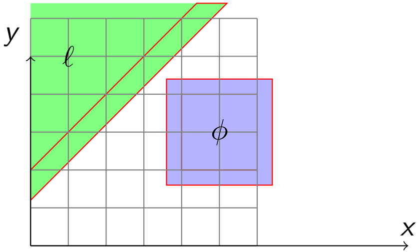


Approximate implementation: Safety

Consider a timed automaton $\mathcal{A}$ with clocks x, y,
such that location ℓ' is not reachable:

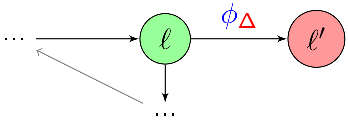


Consider the **reachable states** in ℓ:

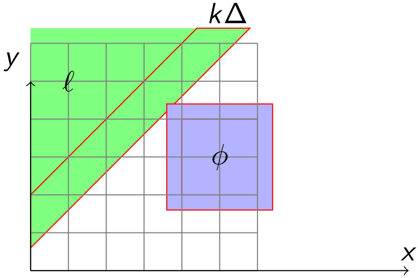


Approximate implementation: Safety

Consider a timed automaton $\mathcal{A}$ with clocks x, y,
such that location ℓ' is not reachable:

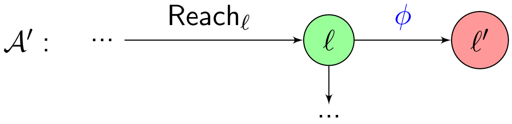


Consider the **reachable states** in ℓ: ℓ' reachable

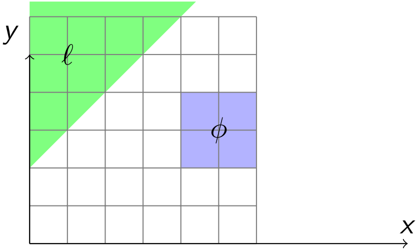


Approximate implementation: Safety

Define $\mathcal{A}'$ as follows:

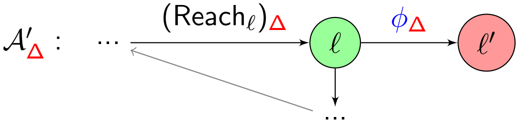


Reachable states in ℓ:

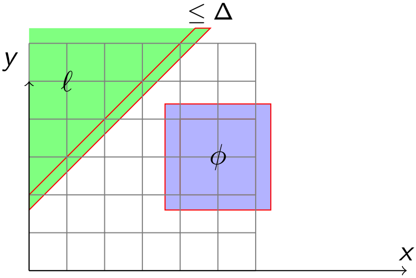


Approximate implementation: Safety

Define $\mathcal{A}'$ as follows:

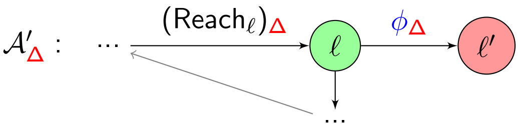


Reachable states in ℓ: ℓ' **not** reachable in $\mathcal{A}'_\Delta$.



Approximate implementation: Safety

Define $\mathcal{A}'$ as follows:



Reachable states in ℓ: ℓ' **not** reachable in $\mathcal{A}'_\Delta$.

No cheating

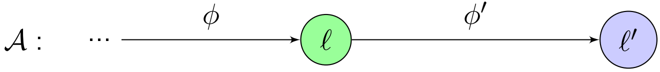
We do not **remove** the edge $\ell \xrightarrow{\phi} \ell'$.

Ready simulation: $\mathcal{A}'_\Delta \sqsubseteq^{\mathsf{Bad}} \mathcal{A}_\Delta$.

"Any run of $\mathcal{A}'_\Delta$ can be imitated in $\mathcal{A}_\Delta$ without enabling **bad** transitions."

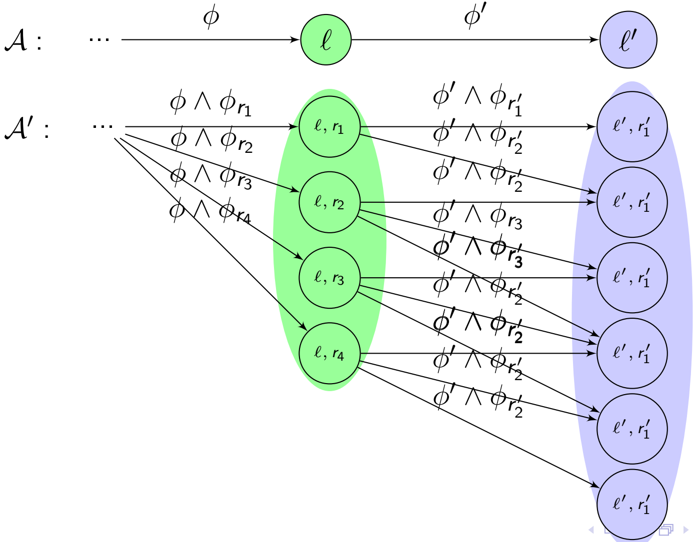
Approximate implementation: Bisimulation

Constructing $\mathcal{A}'$ s.t. $\mathcal{A}' \sim_\epsilon \mathcal{A}'_\Delta$: split locations to regions



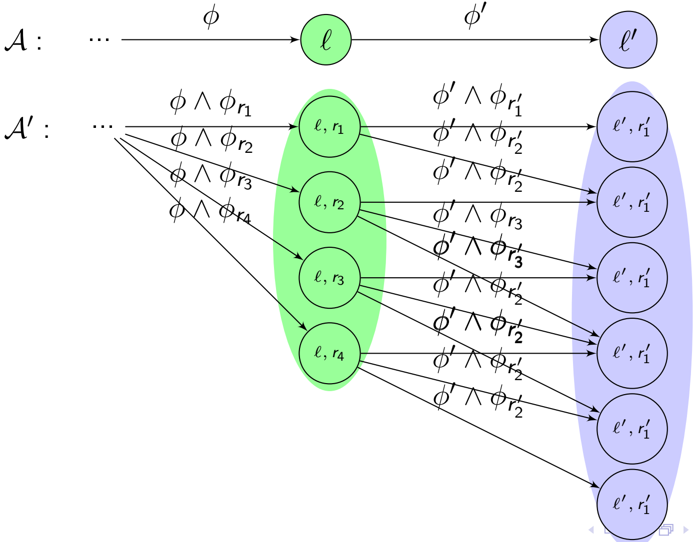
Approximate implementation: Bisimulation

Constructing $\mathcal{A}'$ s.t. $\mathcal{A}' \sim_\epsilon \mathcal{A}'_\Delta$: split locations to regions



Approximate implementation: Bisimulation

Constructing $\mathcal{A}'$ s.t. $\mathcal{A}' \sim_\epsilon \mathcal{A}'_\Delta$: split locations to bisimulation classes

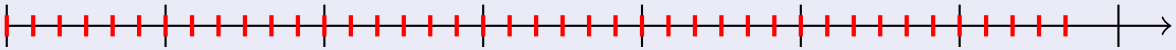


Approximate implementation: Summary

Pros

1. One can choose arbitrarily small ϵ,
2. Works for all timed automata,
3. We preserve time-abstract behaviour + approximate timings.
4. Same result for the semantics under **sampling**:

$$\mathrm{Sampled}_{\frac{1}{n}}(\mathcal{A}) \sqsubseteq \mathcal{A}.$$



We construct $\mathcal{A}'$ such that $\mathrm{Sampled}_{\frac{1}{n}}(\mathcal{A}') \sim_{\epsilon} \mathcal{A}$.

Approximate implementation: Summary

Pros

1. One can choose arbitrarily small ϵ,
2. Works for all timed automata,
3. We preserve time-abstract behaviour + approximate timings.
4. Same result for the semantics under **sampling**:

Cons

1. Size blow-up – although safety construction could do well in practice,
2. Timings are not strictly preserved (but only upto ϵ)
   We still allow additional behaviours.
3. Not clear whether the behaviour is preserved in the **program semantics**.

Next: "Strong" implementation of (1) same size, (2) with strict timings, (3) behaviour is preserved in the program semantics.

Strong Implementation: Shrinking Timed Automata

Abstract model	Real-world behaviour
$\ell \xrightarrow{1 \leq x \leq 2} \ell'$	$\ell \xrightarrow{1-\Delta \leq x \leq 2+\Delta} \ell'$

Strong Implementation: Shrinking Timed Automata

Abstract Model	Real-world behaviour
$\ell \xrightarrow{1 \leq x \leq 2} \ell'$	$\ell \xrightarrow{1-\Delta \leq x \leq 2+\Delta} \ell'$
$\ell \xrightarrow{1+\delta' \leq x \leq 2-\delta} \ell'$	$\ell \xrightarrow{1+\delta'-\Delta \leq x \leq 2-\delta+\Delta} \ell'$

Strong Implementation: Shrinking Timed Automata

Abstract Model	Real-world behaviour
$\ell \xrightarrow{1 \leq x \leq 2} \ell'$	$\ell \xrightarrow{1-\Delta \leq x \leq 2+\Delta} \ell'$
$\ell \xrightarrow{1+\delta' \leq x \leq 2-\delta} \ell'$	$\ell \xrightarrow{1+\delta'-\Delta \leq x \leq 2-\delta+\Delta} \ell'$

$$1 \leq \quad 1 + \delta' - \Delta \leq x \leq 2 - \delta + \Delta \quad \leq 2 \qquad \text{when } \delta, \delta' \geq \Delta.$$

Shrink the clock constraints in the model, to prevent additional behaviour in implementation.

Strong Implementation: Shrinking Timed Automata

Abstract Model	Real-world behaviour
$\ell \xrightarrow{1 \leq x \leq 2} \ell'$	$\ell \xrightarrow{1-\Delta \leq x \leq 2+\Delta} \ell'$
$\ell \xrightarrow{1+\delta' \leq x \leq 2-\delta} \ell'$	$\ell \xrightarrow{1+\delta'-\Delta \leq x \leq 2-\delta+\Delta} \ell'$

$$1 \leq \quad 1+\delta'-\Delta \leq x \leq 2-\delta+\Delta \quad \leq 2 \qquad \text{when } \delta, \delta' \geq \Delta.$$

We consider a separate **shrinking parameter** for each atomic clock constraint: $k_1\delta, k_2\delta, \ldots$ where $\delta > 0$ and $\vec{k} \in \mathbb{N}_{>0}$

Looking for $\vec{\delta} \in \mathbb{Q}^n_{>0}$ $\Leftrightarrow$ looking for $\delta\vec{k}$, where $\delta \in \mathbb{Q}_{>0}$ and $\vec{k} \in \mathbb{N}^n_{>0}$.

The **shrunk automaton** is written $\mathcal{A}_{-\delta\vec{k}}$.

Strong Implementation: Shrinkability of Timed Automata

We have

$$\mathrm{program}(\mathcal{A}_{-\delta\vec{k}+\Delta}) \quad \sqsubseteq \quad \mathcal{A}.$$

for appropriate $0 < 2\Delta < \min \delta\vec{k}$.

- The behaviour of the **real-world system** $\mathrm{program}(\mathcal{A}_{-\delta\vec{k}})$ is included in that of the **abstract model** $\mathcal{A}$.

Strong Implementation: Shrinkability of Timed Automata

We have

$$\text{program}(\mathcal{A}_{-\delta\vec{k}+\Delta}) \quad \sqsubseteq \quad \mathcal{A}.$$

for appropriate $0 < 2\Delta < \min \delta\vec{k}$.

Problem: Shrinkability

Find $\delta\vec{k}$ such that **program**$(\mathcal{A}_{-\delta\vec{k}+\Delta})$ satisfies:

- $\mathcal{A} \sqsubseteq_{\text{t.a.}} \text{program}(\mathcal{A}_{-\delta\vec{k}+\Delta})$,
- and it is non-blocking.

Strong Implementation: Shrinkability of Timed Automata

We have

$$\mathcal{A}_{-\delta\vec{k}} \quad \sqsubseteq \quad \text{program}(\mathcal{A}_{-\delta\vec{k}+\Delta}) \quad \sqsubseteq \quad \mathcal{A}.$$

for appropriate $0 < 2\Delta < \min \delta\vec{k}$.

Problem: Shrinkability

Find $\delta\vec{k}$ such that **program**$(\mathcal{A}_{-\delta\vec{k}+\Delta})$ satisfies:

- $\mathcal{A} \sqsubseteq_{\text{t.a.}} \text{program}(\mathcal{A}_{-\delta\vec{k}+\Delta})$,
- and it is non-blocking.

Strong Implementation: Shrinkability of Timed Automata

We have

$$\mathcal{A} \quad \sqsubseteq_{\text{t.a.}}? \quad \mathcal{A}_{-\delta\vec{k}} \quad \sqsubseteq \quad \mathsf{program}(\mathcal{A}_{-\delta\vec{k}+\Delta}) \quad \sqsubseteq \quad \mathcal{A}.$$

for appropriate $0 < 2\Delta < \min \delta\vec{k}$.

Theorem (Shrinkability) [S., Bouyer, Markey 2011]

One can decide the existence of $\delta\vec{k}$, and compute the "least" solution, for which,

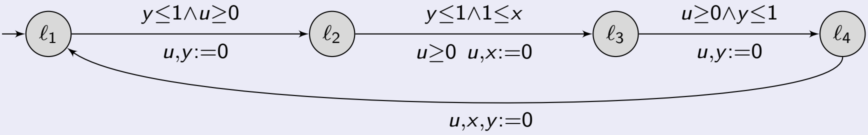
- $\mathcal{A} \quad \sqsubseteq_{\text{t.a.}} \quad \mathcal{A}_{-\delta\vec{k}},$ in **EXPTIME**,
- $\mathcal{A}_{-\delta\vec{k}}$ is non-blocking. in **PSPACE**, and **NP** for bounded-branching

and both at the same time in **EXPTIME**.

$\Rightarrow$ **program**$(\mathcal{A}_{-\delta\vec{k}+\Delta})$ is non-blocking.

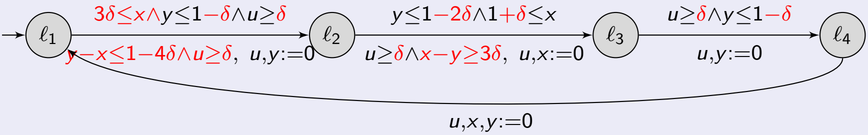
Example of Shrinking

A **shrinkable** automaton



Example of Shrinking

A **shrunk** automaton



$$\mathcal{A} \quad \sqsubseteq_{\text{t.a.}} \quad \mathcal{A}_{-\delta\vec{k}} \quad \sqsubseteq \quad \mathcal{A}.$$

and non-blocking, for **all** $\delta \in [0, \frac{1}{4}]$

Interpretation of Shrinking

Developer's guide to shrinking

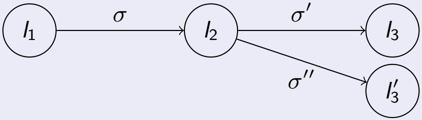
$$\ell_1 \xrightarrow{3+2\delta \leq x \leq 7-4\delta} \ell_2$$

- If the edge is **controllable** by the system, do the action 2δ later than allowed, and 4δ before the deadline.

- If the edge is **uncontrollable** (e.g. execution of task), the guard corresponds to BCET $\leq x \leq$ WCET:
adjust your timing analysis to ensure $3+2\delta \leq x \leq 7-4\delta$.

Non-blocking Timed Automata

Definition: Non-blockingness



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

Fix-point characterization

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

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

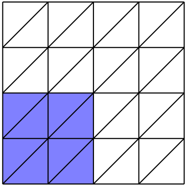
Technique for Computing Shrinking Parameters

$$l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3$$

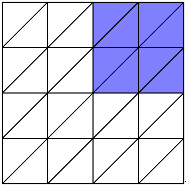
$$[\![G_\sigma]\!] \subseteq \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!])).$$

Technique for Computing Shrinking Parameters

$$l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3$$



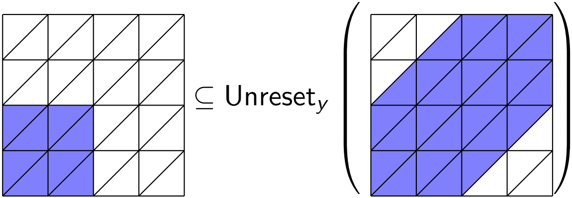
$\subseteq \mathsf{Unreset}_y \left(\mathsf{Pre}_{\mathsf{time}} \left(\right. \right.$



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

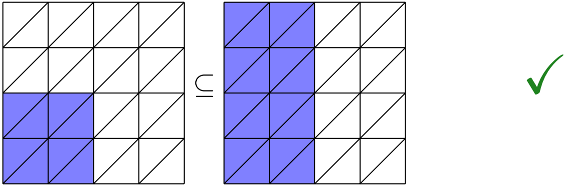
Technique for Computing Shrinking Parameters

$$l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3$$



Technique for Computing Shrinking Parameters

$$l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3$$



Technique for Computing Shrinking Parameters

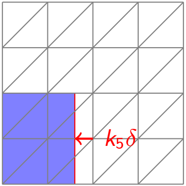
$$l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3$$

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

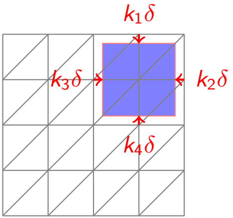
Determine $\vec{k}$

Technique for Computing Shrinking Parameters

$$l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3$$



$\subseteq \mathsf{Unreset}_y\Big(\mathsf{Pre}_{\mathsf{time}}\Big($

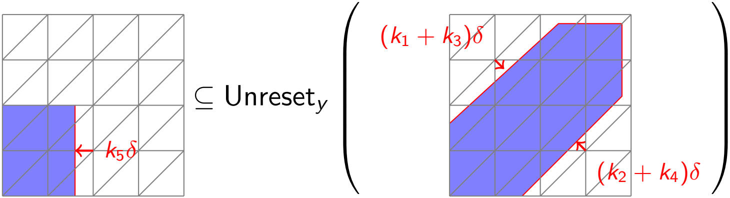


$\Big)\Big)$

for all $\delta < \frac{1}{2}\min_i \frac{1}{k_i}$.

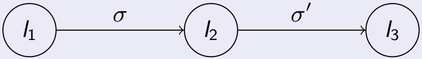
Technique for Computing Shrinking Parameters

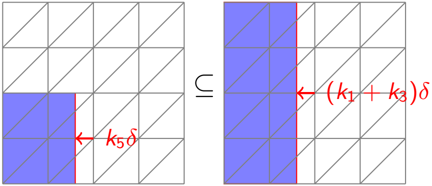
$$l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3$$



for all $\delta < \frac{1}{2} \min\left(\frac{1}{k_1+k_3}, \frac{1}{k_2+k_4}, \min_i \frac{1}{k_i}\right)$.

Technique for Computing Shrinking Parameters

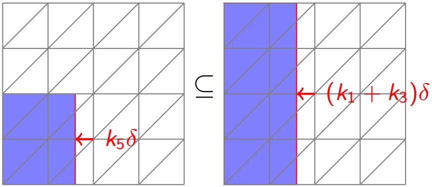




for all $\delta < \frac{1}{2}\min\left(\frac{1}{k_1+k_3}, \frac{1}{k_2+k_4}, \min_i \frac{1}{k_i}\right)$.

Technique for Computing Shrinking Parameters

$$l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3$$



Then, $\vec{k}$ should satisfy

$$k_5 = \max(k_5, k_1 + k_3).$$

$$\text{for all } \delta < \frac{1}{2}\min\left(\frac{1}{k_1+k_3}, \frac{1}{k_2+k_4}, \min_i \frac{1}{k_i}\right).$$

Technique for Computing Shrinking Parameters

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

Technique for Computing Shrinking Parameters

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

In fact,

let f be any operation among **Pretime**, $\cap$, **Unreset**,
and let $M = f(N)$.

Then, for any parameters $\vec{k}$, there exists $\vec{l}$ such that

$$\langle M\rangle_{-\vec{l}\delta} \quad = \quad f(\langle N\rangle_{-\vec{k}\delta}),$$

for all small enough $\delta > 0$,
where $\vec{l}$ can be expressed by a max-plus expression of $\vec{k}$.

Technique for Computing Shrinking Parameters

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

Key Theorem

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

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

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

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

where ϕ is a **max-plus expression**.

Technique for Computing Shrinking Parameters

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

Key Theorem

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

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

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

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

where ϕ is a **max-plus expression**.

- **Max-plus algebra:** We prove that such fixpoint equations can be solved in polynomial time.

Shrinking Algorithm for Non-blockingness

Need to solve:

$$\forall \sigma, \quad G_\sigma \subseteq \bigcup_{(\sigma,\sigma')} \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}(G_{\sigma'})).$$

but theorem doesn't allow union.

Shrinking Algorithm for Non-blockingness

Need to solve:

$$\forall \sigma, \quad G_\sigma = \bigcup_{(\sigma,\sigma')} \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}(G_{\sigma'})) \cap G_\sigma.$$

but theorem doesn't allow union.

Shrinking Algorithm for Non-blockingness

Equivalently, solve:

$$\forall \sigma, \sigma', \qquad G_\sigma = \bigcup_{\sigma,\sigma'} M_{\sigma,\sigma'},$$
$$M_{\sigma,\sigma'} = \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}(G_{\sigma'})) \cap G_\sigma.$$

Shrinking Algorithm for Non-blockingness

$$\forall \sigma, \sigma', \qquad G_\sigma = \bigcup_{\sigma,\sigma'} M_{\sigma,\sigma'},$$
$$M_{\sigma,\sigma'} = \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}(G_{\sigma'})) \cap G_\sigma.$$

Lemma

Consider any equation of the form $G = \bigcup_i M_i$, and $k_1, \ldots, k_n \in \mathbb{N}_{>0}$ such that $\langle G \rangle_{-\delta\vec{k}} = \bigcup_i \langle M_i \rangle_{-\delta\vec{k}}$ for small enough $\delta > 0$. Consider

$$k_{\alpha(1)} \leq k_{\alpha(2)} \leq \ldots \leq k_{\alpha(n)}, \qquad \text{for some perm. } \alpha.$$

Then for any $k'_1, \ldots, k'_n \in \mathbb{N}_{>0}$ with the same ordering, i.e.

$$k'_{\alpha(1)} \leq k'_{\alpha(2)} \leq \ldots \leq k'_{\alpha(n)},$$

we have $\langle G \rangle_{-\delta\vec{k'}} = \bigcup_i \langle M_i \rangle_{-\delta\vec{k'}}$ for small enough $\delta > 0$.

Shrinking Algorithm for Non-blockingness

$$\forall \sigma, \sigma', \qquad G_\sigma = \bigcup_{\sigma,\sigma'} M_{\sigma,\sigma'},$$
$$M_{\sigma,\sigma'} = \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}(G_{\sigma'})) \cap G_\sigma.$$

Lemma

Given $G = \bigcup_i M_i$, whether a vector $\vec{k}$ satisfies

$$\langle G \rangle_{-\delta \vec{k}} = \bigcup_i \langle M_i \rangle_{-\delta \vec{k}} \text{ for small enough } \delta > 0,$$

only depends on the **ordering** of k_i's.

Overall Algorithm:

1) Guess the ordering (polynomially many guesses),
2) Solve above equation augmented with this ordering,
3) Verify union. (in PSPACE or NP if bounded nb of edges per location)

Summary of the Fixpoint Equations

Deciding implementability

Apply theorem to following fix-point equations:

- Non-blockingness:

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

  (Do technical work to remove the union)

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

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

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

Shrinking: Summary

Summary

- Shrinking always ensures Imp $\sqsubseteq$ Spec.
- Shrinking parameters can be **synthesized automatically** so that Spec $\sqsubseteq$ Imp, and Imp non-blocking.
- Complexity: NP, PSPACE, EXPTIME.
- These properties preserved in the **program semantics**.

$\rightarrow$ Can be used to define the implementation or in the timing analysis.

Technics

- Zones + parameters $\leftrightarrow$ max-plus algebra.
- New data structure: **DBMs w/ parameterized max-plus exp.**. Application to other problems: e.g. robust controller synthesis.

Conclusion

Robustness Analysis

"Parameterized robust model-checking has same theoretical complexity as model-checking for timed automata."

safety, LTL, coFlat-MTL, untimed language equiv., ...

Some work on symbolic algorithms e.g. [Daws, Cordy '06], [Jaubert, Reynier '11]

Robust Implementation

Approximate Implementation: All timed automata can be compiled into a larger system, with approximately the same behaviour.

Shrinking: Parameter synthesis for adjusting timing constraints.

Future Work

1. Efficient algorithms:
   - for approximate implementation w.r.t. safety.
   - for shrinkability.
2. Compositionality.
3. Robust reachability in timed games:

   $$\text{Player 1 chooses a delay } d \geq 0 \text{ and an edge } \sigma,$$
   $$\text{Player 2 adds a perturbation: } d + \epsilon \text{ where } \epsilon \in [-\delta, \delta].$$

   Thm: Reachability is EXPTIME-complete.
   Winning sets are described by shrunk zones.

   How about safety and Büchi properties?
4. Similar semantics with probabilistic perturbation.

Future Work

1. Efficient algorithms:
   - for approximate implementation w.r.t. safety.
   - for shrinkability.
2. Compositionality.
3. Robust reachability in timed games:

   Player 1 chooses a delay $d \geq 0$ and an edge σ,
   Player 2 adds a perturbation: $d + \epsilon$ where $\epsilon \in [-\delta, \delta]$.

   Thm: Reachability is EXPTIME-complete.
   Winning sets are described by shrunk zones.

   How about safety and Büchi properties?
4. Similar semantics with probabilistic perturbation.

Thank you!