Robust Model Checking of Timed Automata Using Pumping in Channel Machines

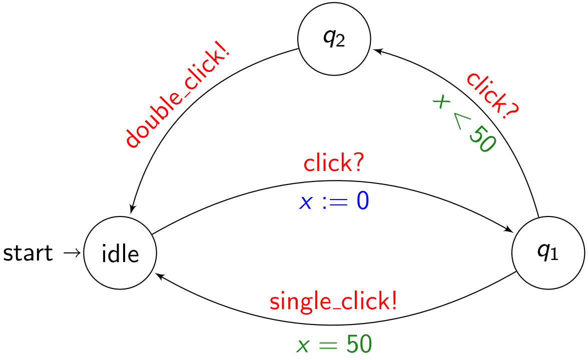
Patricia Bouyer, Nicolas Markey, **Ocan Sankur**

LSV, CNRS & Ecole Normale Supérieure de Cachan

FORMATS 2011

Abstract Model: Timed Automata (TA)

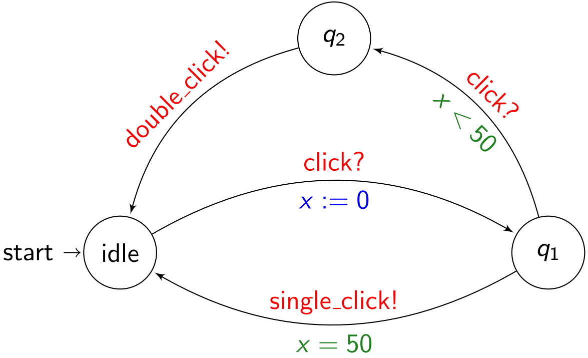
Timed automata = Finite automata + Analog clocks. [Alur and Dill 1994]



- 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.

Abstract Model: Timed Automata (TA)

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



Runs of a timed automaton

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

Robustness Issues in Timed Automata

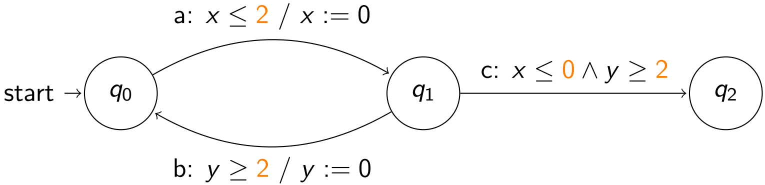
The semantics of timed automata is idealistic:

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

But real world systems have finite frequency, digital clocks...

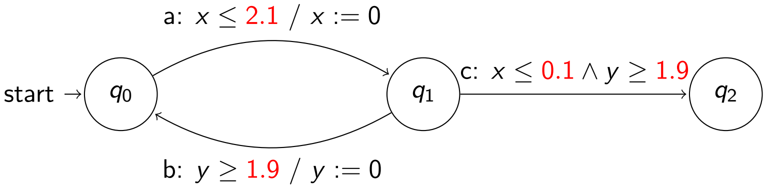
Enlargement

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



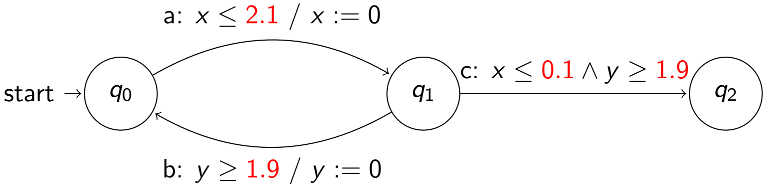
Enlargement

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



Enlargement

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

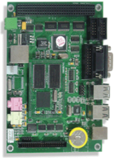


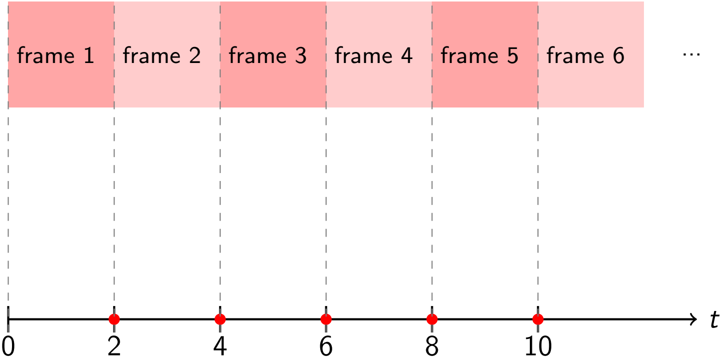
Corresponds to a micro-processor executing $\mathcal{A}$ as a program

δ corresponds to the *clock error* and *hardware frequency*

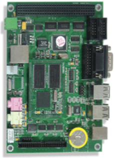
[De Wulf, Doyen, Raskin 2004]

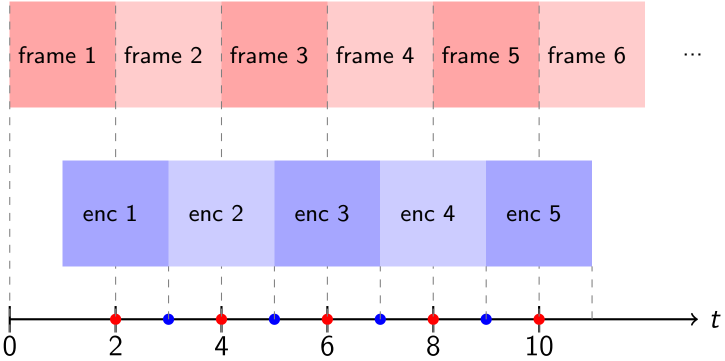
A Non-Robust Timed System in the enlarged semantics



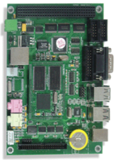


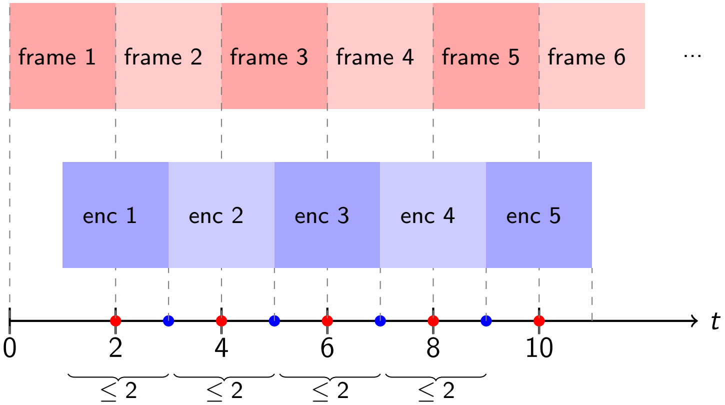
A Non-Robust Timed System in the enlarged semantics



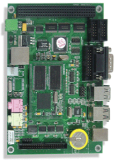


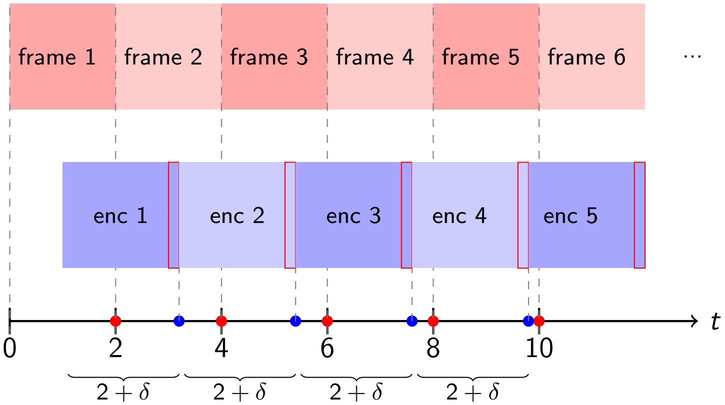
A Non-Robust Timed System in the enlarged semantics



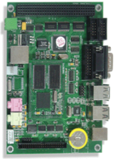


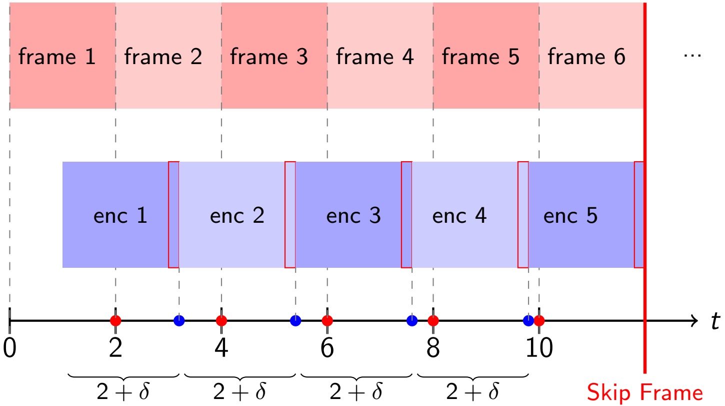
A Non-Robust Timed System in the enlarged semantics



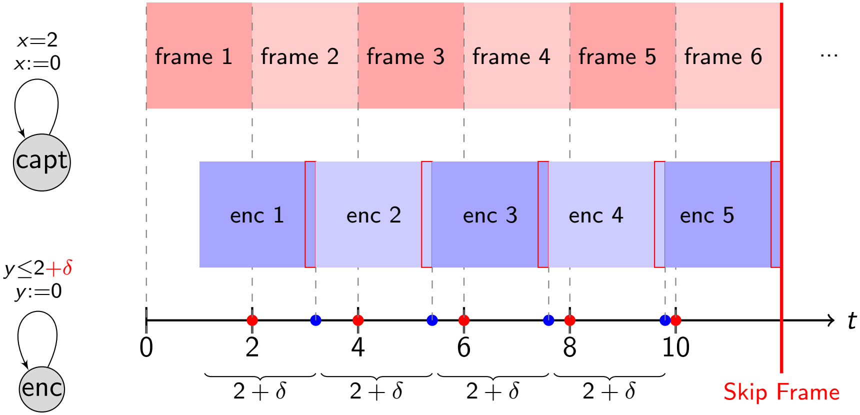


A Non-Robust Timed System in the enlarged semantics





A Non-Robust Timed System in the enlarged semantics



Background: Enlarged semantics

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

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

Robust model-checking

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

Decidable for:

- Safety, [Puri 1998], [DDMR 2004] [Daws, Kordy 2006], [Jaubert, Reynier 2011]
- ω-regular properties (LTL), coFlat-MTL [Bouyer, Markey, Reynier 2006/08].
- Untimed language equivalence $L(\mathcal{A}) = L(\mathcal{A}_\delta)$ [S. 2011]

Only valid for **timed automata with progress cycles**:

along any cycle, all clocks must be reset at least once.

Background: Enlarged semantics

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

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

Robust model-checking

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

Decidable for:

- Safety, [Puri 1998], [DDMR 2004] [Daws, Kordy 2006], [Jaubert, Reynier 2011]
- ω-regular properties (LTL), coFlat-MTL [Bouyer, Markey, Reynier 2006/08].
- Untimed language equivalence $L(\mathcal{A}) = L(\mathcal{A}_\delta)$ [S. 2011]

Only valid for **timed automata with progress cycles**:

along any cycle, all clocks must be reset at least once.

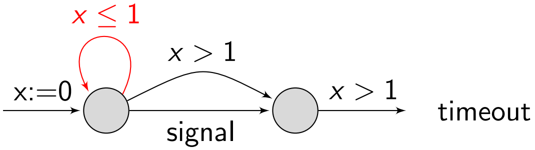
In this work: **New** algorithm for robust model-checking of **general** timed automata

Progress cycles are restrictive

All clocks must be reset in all cycles
"$\Leftrightarrow$"
Cannot measure time spent in a program loop

```
clock x;

x := 0;
while ( x <= 1 ){
   ...
   if ( signal() )
      break;
}
if ( x >= 1 )
   timeout := true;
```



Results

Theorem

For any ω-regular property ϕ, and any timed automaton $\mathcal{A}$, there exists $\delta_0 > 0$ such that

$$\exists \delta > 0, \mathcal{A}_\delta \models \phi \qquad \Leftrightarrow \qquad \mathcal{A}_{\delta_0} \models \phi.$$

Algorithm: Model-check $\mathcal{A}_{\delta_0}$ which is a timed automaton.

- Algorithm in PSPACE (optimal).
- Robust model-checking reduced to classical model-checking for timed automata. (One can use any model-checker for timed automata).
- Valid for **all** timed automata.

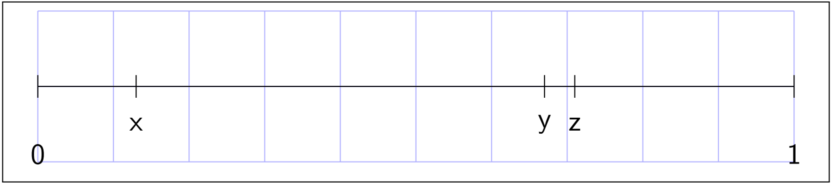
Technically, proof based on encoding by channel machines [BMOW07].
$\rightarrow$ Channel machines finely capture behaviour of timed automata.

Encoding by Channel Machines

Goal: Capture the behaviour of $\mathcal{A}_\delta$ by $\mathcal{C}_{\mathcal{A}}(N)$ a **finite state machine with a FIFO channel** with parameter N.

Encoding by Channel Machines

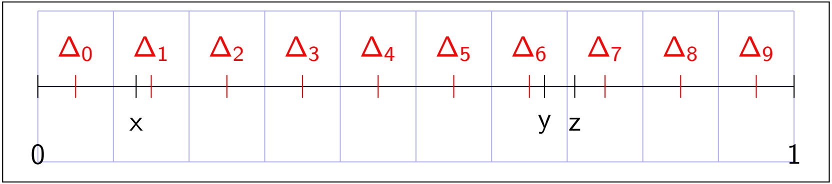
Goal: Capture the behaviour of $\mathcal{A}_\delta$ by $\mathcal{C}_\mathcal{A}(N)$ a **finite state machine with a FIFO channel** with parameter N.



Consider a state of $\mathcal{A}$ (where $\lfloor x \rfloor = 1, \lfloor y \rfloor = 2, \lfloor z \rfloor = 0$).

Encoding by Channel Machines

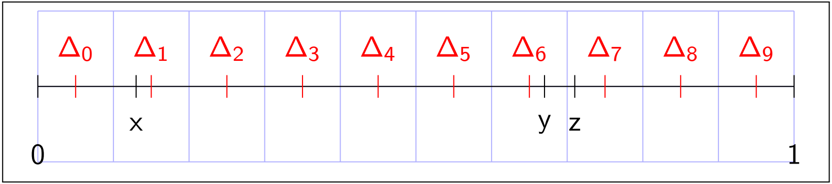
Goal: Capture the behaviour of $\mathcal{A}_\delta$ by $\mathcal{C}_\mathcal{A}(N)$ a **finite state machine with a FIFO channel** with parameter N.



Add N new clocks that are regularly distributed in $[0,1]$ and that have values mod 1.

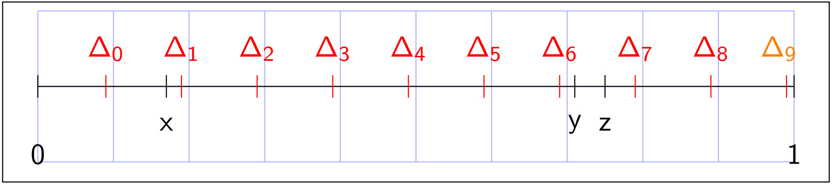
$\mathcal{C}_\mathcal{A}(N)$ encodes the **regions** of the states of $\mathcal{A} + \{\Delta_0, \ldots, \Delta_{N-1}\}$ using a *discrete state* and a *channel*.

Encoding by Channel Machines



$$\text{head}\rightarrow \underbrace{\Delta x \Delta\Delta\Delta\Delta\Delta\Delta yz\Delta\Delta\Delta}_{\text{channel}} \leftarrow\text{tail} \; \underbrace{(\lfloor x \rfloor = 1, \lfloor y \rfloor = 2, \lfloor z \rfloor = 0)}_{\text{discrete state}}.$$

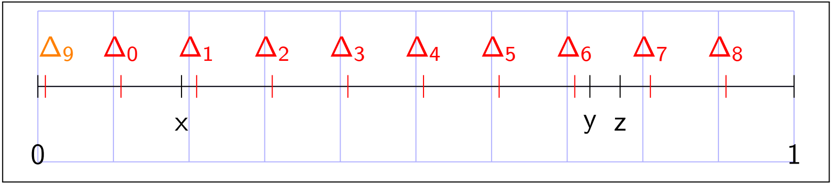
Encoding by Channel Machines



Delay of 0.04 time units

$\mathcal{C}_{\mathcal{A}}(N)$: $\Delta x \Delta\Delta\Delta\Delta\Delta\Delta yz \Delta\Delta\Delta$ ($\lfloor x \rfloor = 1$, $\lfloor y \rfloor = 2$, $\lfloor z \rfloor = 0$).

Encoding by Channel Machines

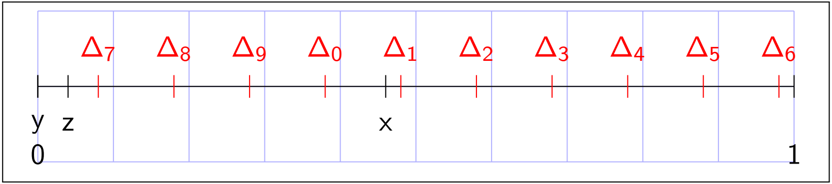


Delay of 0.02 time units

$\mathcal{C}_{\mathcal{A}}(N)$: $\Delta\Delta x\Delta\Delta\Delta\Delta\Delta\Delta yz\Delta\Delta$ ($\lfloor x\rfloor = 1$, $\lfloor y\rfloor = 2$, $\lfloor z\rfloor = 0$).

Rule: When a Δ is read from the channel, write it back into the channel.

Encoding by Channel Machines



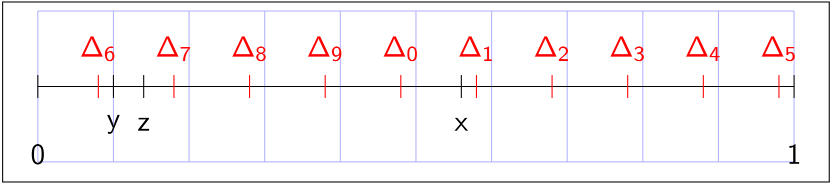
Delay of 0.15 time units

$\mathcal{C}_{\mathcal{A}}(N)$: $yz\Delta\Delta x\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta$ ($\lfloor x \rfloor = 0$, $\lfloor y \rfloor = 3$, $\lfloor z \rfloor = 1$).

Rule: When a clock $y \neq \Delta$ is read from the channel, write it back into the channel and increment its integer part.

Delay transition = Read/Write to the channel

Encoding by Channel Machines



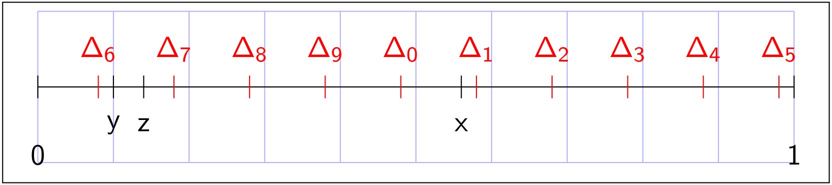
Guard $y \leq k$ is satisfied if

$\lfloor y \rfloor \leq k - 1$ $\rightarrow$ discrete state

or

$$\lfloor y \rfloor = k \text{ and } \underbrace{\Delta y}_{\leq \Delta^1} z\Delta\Delta\Delta\Delta x\Delta\Delta\Delta\Delta\Delta$$

Encoding by Channel Machines



Guard $y \leq k$ is satisfied if

$\lfloor y \rfloor \leq k - 1$ $\rightarrow$ discrete state

or

$$\lfloor y \rfloor = k \text{ and } \underbrace{\Delta y}_{\leq \Delta^1} z\Delta\Delta\Delta\Delta x\Delta\Delta\Delta\Delta\Delta$$

From the encoding, we know that $|y - \lfloor y \rfloor| \leq \frac{2}{N}$.

- Small $\delta \Leftrightarrow$ large N.

[Bouyer, Markey, Reynier 2008]

Main Result restated for channel machines

Theorem

For any ω-regular property ϕ, and any timed automaton $\mathcal{A}$, there exists $N_0 > 0$ such that

$$\exists N > 0, \mathcal{C}_{\mathcal{A}}(N) \models \phi \quad \Rightarrow \quad \mathcal{C}_{\mathcal{A}}(N_0) \models \phi.$$

$$\mathcal{C}_{\mathcal{A}}(N_0) \not\models \phi \quad \Rightarrow \quad \forall N > 0, \mathcal{C}_{\mathcal{A}}(N) \not\models \phi.$$

Main Result restated for channel machines

Theorem

For any ω-regular property ϕ, and any timed automaton $\mathcal{A}$, there exists $N_0 > 0$ such that

$$\exists N > 0, \mathcal{C}_{\mathcal{A}}(N) \models \phi \quad \Rightarrow \quad \mathcal{C}_{\mathcal{A}}(N_0) \models \phi.$$

$$\mathcal{C}_{\mathcal{A}}(N_0) \not\models \phi \quad \Rightarrow \quad \forall N > 0, \mathcal{C}_{\mathcal{A}}(N) \not\models \phi.$$

Proof (two cases): $\forall K \geq 1$

- Any run of $\mathcal{C}_{\mathcal{A}}(N_0)$ can be simulated on $\mathcal{C}_{\mathcal{A}}(N_0 - K)$ (easy)

Main Result restated for channel machines

Theorem

For any ω-regular property ϕ, and any timed automaton $\mathcal{A}$, there exists $N_0 > 0$ such that

$$\exists N > 0, \mathcal{C}_{\mathcal{A}}(N) \models \phi \quad \Rightarrow \quad \mathcal{C}_{\mathcal{A}}(N_0) \models \phi.$$

$$\mathcal{C}_{\mathcal{A}}(N_0) \not\models \phi \quad \Rightarrow \quad \forall N > 0, \mathcal{C}_{\mathcal{A}}(N) \not\models \phi.$$

Proof (two cases): $\forall K \geq 1$

- Any run of $\mathcal{C}_{\mathcal{A}}(N_0)$ can be simulated on $\mathcal{C}_{\mathcal{A}}(N_0 - K)$ (easy)
- Any run of $\mathcal{C}_{\mathcal{A}}(N_0)$ can be *adapted* to $\mathcal{C}_{\mathcal{A}}(N_0 + K)$.
  $\rightarrow$Pumping lemma (main lemma — difficult).

Pumping lemma: simple case by example

Adapting a run of $\mathcal{C}_{\mathcal{A}}(N_0)$ to $\mathcal{C}_{\mathcal{A}}(N_0+1)$.

- Delay transitions.

$\mathcal{C}_{\mathcal{A}}(N)$ $\qquad\qquad$ $\mathcal{C}_{\mathcal{A}}(N+1)$

$\Delta\Delta x\Delta\Delta\Delta y\Delta\Delta$

$\xrightarrow{\text{delay}}$

$\Delta\Delta\Delta\Delta x\Delta\Delta\Delta y$

Pumping lemma: simple case by example

Adapting a run of $\mathcal{C}_{\mathcal{A}}(N_0)$ to $\mathcal{C}_{\mathcal{A}}(N_0+1)$.

- Delay transitions.

$$\begin{array}{ll} \mathcal{C}_{\mathcal{A}}(N) & \mathcal{C}_{\mathcal{A}}(N+1) \\ \Delta\Delta x\Delta\Delta\Delta y\Delta\Delta & \Delta\Delta x\Delta\Delta\Delta y\Delta\Delta\Delta \\ \xrightarrow{\text{delay}} & \xrightarrow{\text{delay}} \\ \Delta\Delta\Delta\Delta x\Delta\Delta\Delta y & \Delta\Delta\Delta\Delta\Delta x\Delta\Delta\Delta y \end{array}$$

Pumping lemma: simple case by example

$\boxed{\text{Adapting a run of } \mathcal{C}_{\mathcal{A}}(N_0) \text{ to } \mathcal{C}_{\mathcal{A}}(N_0+1).}$

- Delay transitions.

$\mathcal{C}_{\mathcal{A}}(N)$	$\mathcal{C}_{\mathcal{A}}(N+1)$
$\Delta\Delta x\Delta\Delta\Delta y\Delta\Delta$	$\Delta\Delta x\Delta\Delta\Delta y\Delta\Delta\Delta$
$\xrightarrow{\text{delay}}$	$\xrightarrow{\text{delay}}$
$\Delta\Delta\Delta\Delta x\Delta\Delta\Delta y$	$\Delta\Delta\Delta\Delta\Delta x\Delta\Delta\Delta y$

- Discrete transitions (easy)

$\Delta\Delta\Delta\Delta x\Delta\Delta\Delta y$	$\Delta\Delta\Delta\Delta\Delta x\Delta\Delta\Delta y$
$\xrightarrow{x\leq 1,\ x:=0}$	$\xrightarrow{x\leq 1,\ x:=0}$
$x\Delta\Delta\Delta\Delta\Delta\Delta\Delta y$	$x\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta y$

Pumping lemma: simple case by example

Adapting a run of $\mathcal{C}_\mathcal{A}(N_0)$ to $\mathcal{C}_\mathcal{A}(N_0+1)$.

- Delay transitions.

$$\begin{array}{ll} \mathcal{C}_\mathcal{A}(N) & \mathcal{C}_\mathcal{A}(N+1) \\ \Delta\Delta x\Delta\Delta\Delta y{\color{blue}\Delta\Delta} & \Delta\Delta x\Delta\Delta\Delta y{\color{blue}\Delta\Delta}{\color{red}\Delta} \\ \xrightarrow{\text{delay}} & \xrightarrow{\text{delay}} \\ {\color{blue}\Delta\Delta}\Delta\Delta x\Delta\Delta\Delta y & {\color{blue}\Delta\Delta}{\color{red}\Delta}\Delta\Delta x\Delta\Delta\Delta y \end{array}$$

- Discrete transitions (difficult)

$$\begin{array}{ll} \Delta x\Delta\Delta\Delta y & \Delta{\color{red}\Delta} x\Delta\Delta\Delta y \\ \xrightarrow{x\leq 1,\ x:=0} & {\color{red}\not\rightarrow} \\ x\Delta\Delta\Delta\Delta y & \end{array}$$

Our proofs are based on watching the evolution of the sizes of Δ-blocks.

Conclusion

- Robust model-checking for general timed automata
  $\rightarrow$ reduced to classical model-checking on timed automata.
- Another algorithm based on region automaton construction
  [upcoming tech report]
- Channel machines capture well the robust behaviour of timed automata:
  Similar theorem for safety can be derived from [DDMR08] but with an exponentially larger complexity.
- We need arbitrarily large constants in model-checkers (e.g. Uppaal).

Next

- Robust controllers
- Probabilistic imprecisions instead of worst-case