Sokendai Lectures

Tokyo, Japan

物理情報システムのための形式手法

Timed model checking – Part 2

Timed automata

Étienne André

Etienne.Andre (à) univ-paris13.fr

Version: May 20, 2019 (slides with empty spaces (Web version))

Partie 2: Timed model checking – Plan

1. Timed automata
2. Specifying with timed temporal logics
3. Specifying with observers
4. Decidability
5. Timed automata in practice
6. Beyond timed automata...

Outline

Beyond finite state automata

Finite State Automata give a simple syntax and a formal semantics to model qualitative aspects of systems

- Executions, sequence of actions
- Modular definitions (parallelism)
- Powerful checking (reachability, safety, liveness...)

Beyond finite state automata

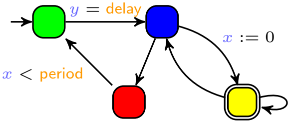
Finite State Automata give a simple syntax and a formal semantics to model qualitative aspects of systems

- Executions, sequence of actions
- Modular definitions (parallelism)
- Powerful checking (reachability, safety, liveness…)

But what about quantitative aspects:

- Time ("the airbag always eventually inflates, but maybe 10 seconds after the crash")
- Temperature ("the alarm always eventually ring, but maybe when the temperature is above 75 degrees")

Model checking timed concurrent systems

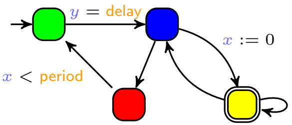


A timed model of the system

 is unreachable

A property to be satisfied

Model checking timed concurrent systems



A timed model of the system

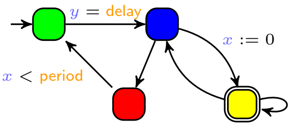
$\overset{?}{\models}$

 is unreachable

A property to be satisfied

- Question: does the model of the system satisfy the property?

Model checking timed concurrent systems



A timed model of the system

$\stackrel{?}{\models}$

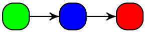
 is unreachable

A property to be satisfied

- Question: does the model of the system satisfy the property?

Yes

No



Counterexample

Formalisms

Many formalisms were proposed to model and verify timed systems

- time(d) Petri nets [Merlin, 1974]
- timed automata [Alur and Dill, 1994]
- timed process algebras [Sun et al., 2009b]
- etc.

Formalisms

Many formalisms were proposed to model and verify timed systems

- time(d) Petri nets [Merlin, 1974]
- timed automata [Alur and Dill, 1994]
- timed process algebras [Sun et al., 2009b]
- etc.

We use here timed automata

See [Bérard et al., 2005, Srba, 2008, Bérard et al., 2013] for a comparison between timed Petri nets and timed automata

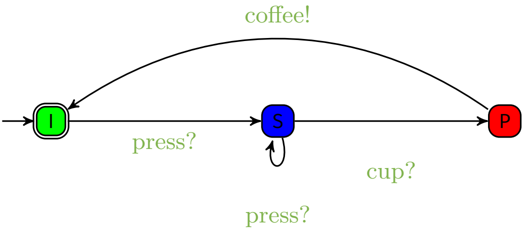
Outline

Timed automaton (TA)

- Finite state automaton (sets of locations)

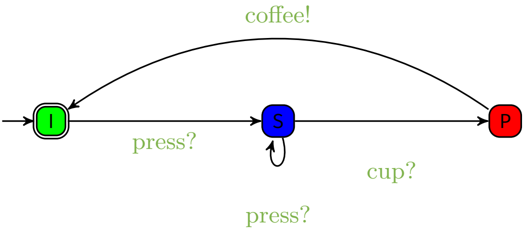
Timed automaton (TA)

- Finite state automaton (sets of locations and actions)



Timed automaton (TA)

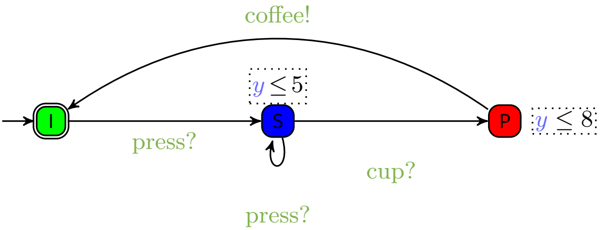
- Finite state automaton (sets of locations and actions) augmented with a set x of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate



Timed automaton (TA)

- Finite state automaton (sets of locations and actions) augmented with a set x of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants

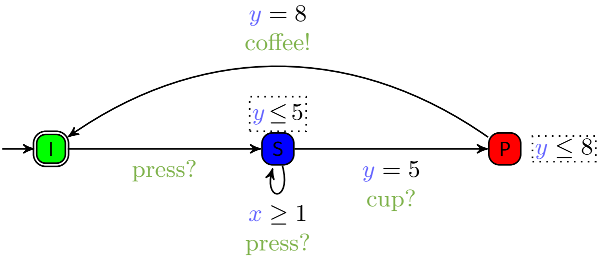
- Features
  - Location invariant: property to be verified to stay at a location



Timed automaton (TA)

- Finite state automaton (sets of locations and actions) augmented with a set x of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants and guards

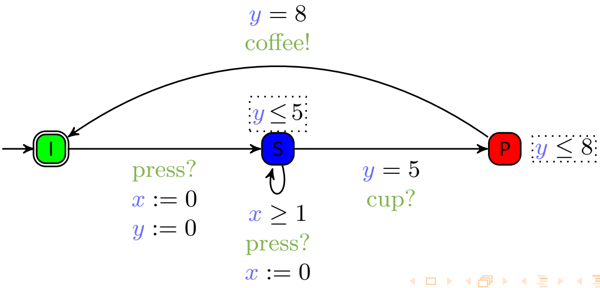
- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition



Timed automaton (TA)

- Finite state automaton (sets of locations and actions) augmented with a set x of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants and guards

- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition
  - Clock reset: some of the clocks can be set to 0 at each transition



Formal definition of timed automata

Definition (Timed automaton)

A timed automaton (TA) $\mathcal{A}$ is a 7-tuple of the form $\mathcal{A} = (L, \Sigma, \ell_0, L_F, X, I, E)$, where

- L is a finite set of locations,
- $\ell_0 \in L$ is the initial location,
- $L_F \subseteq L$ is the set of accepting (or final) locations,
- Σ is a finite set of actions,
- X is a set of clocks,
- I is the invariant, assigning to every $\ell \in L$ a clock constraint $I(\ell)$, and
- E is a step (or "transition") relation consisting of elements of the form $e = (\ell, g, a, R, \ell')$, also denoted by $\ell \xrightarrow{g,a,R} \ell'$, where $\ell, \ell' \in L$, $a \in \Sigma$, $R \subseteq X$ is a set of clock variables to be reset by the step, and g (the step guard) is a clock constraint.

Clock constraints

Definition (clock constraint)

A clock constraint is a conjunction of atomic constraints

Clock constraints

Definition (clock constraint)

A clock constraint is a conjunction of atomic constraints

What is an atomic constraint?

Clock constraints

Definition (clock constraint)

A clock constraint is a conjunction of atomic constraints

What is an atomic constraint?

Various definitions in the literature:

- Originally [Alur and Dill, 1994]: $x \in [c_1, c_2]$ with $c_1 \in \mathbb{N}$ and $c_2 \in \mathbb{N} \cup \{\infty\}$
- Comparing clock values (diagonal constraints) $x_1 - x_2 \bowtie c$
  - $\bowtie \in \{<, \leq, =, \geq, >\}$

For now, we assume the following syntax:

- $x \bowtie c$, with $x \in X$ and $c \in \mathbb{N}$

Exercise 1

Draw the TA $\mathcal{A} = (L, \Sigma, l_1, \{l_2\}, X, I, E)$ such that

- $L = \{l_1, l_2, l_3, l_4\}$,
- $\Sigma = \{a_1, a_2, a_3\}$,
- $X = \{x_1, x_2\}$,
- $I(l_1) = x_1 \leq 3$, and $I(l_3) = x_2 \geq 2$,
- $E = \{(l_1, x_1 \geq 2, a_1, \{x_1\}, l_2),$
  $(l_1, x_2 \leq 1, a_2, \emptyset, l_3),$
  $(l_2, x_2 = 1, a_3, \{x_2\}, l_2),$
  $(l_2, \texttt{true}, a_1, \emptyset, l_3),$
  $(l_3, \texttt{true}, a_2, \{x_1, x_2\}, l_4),$
  $(l_4, x_2 > 2, a_3, \emptyset, l_3)\}$

Exercise 1

Draw the TA $\mathcal{A} = (L, \Sigma, l_1, \{l_2\}, X, I, E)$
such that

- $L = \{l_1, l_2, l_3, l_4\}$,
- $\Sigma = \{a_1, a_2, a_3\}$,
- $X = \{x_1, x_2\}$,
- $I(l_1) = x_1 \leq 3$, and $I(l_3) = x_2 \geq 2$,
- $E = \{(l_1, x_1 \geq 2, a_1, \{x_1\}, l_2),$
  $(l_1, x_2 \leq 1, a_2, \emptyset, l_3),$
  $(l_2, x_2 = 1, a_3, \{x_2\}, l_2),$
  $(l_2, \texttt{true}, a_1, \emptyset, l_3),$
  $(l_3, \texttt{true}, a_2, \{x_1, x_2\}, l_4),$
  $(l_4, x_2 > 2, a_3, \emptyset, l_3)\}$

Exercise 2

Give the formal TA corresponding to the timed coffee machine.

Exercise 2

Give the formal TA corresponding to the timed coffee machine.

Parallel composition of timed automata (1/2)

Just as finite-state automata, timed automata can be composed through parallel composition using synchronization actions

$\mathcal{A}_1 = (L_1, \Sigma_1, (\ell_0)_1, (L_F)_1, X_1, I_1, E_1)$
$\mathcal{A}_2 = (L_2, \Sigma_2, (\ell_0)_2, (L_F)_2, X_2, I_2, E_2)$

Then we define $\mathcal{A}_1 \parallel \mathcal{A}_2$ as

Parallel composition of timed automata (1/2)

Just as finite-state automata, timed automata can be composed through parallel composition using synchronization actions

$$\mathcal{A}_1 = (L_1, \Sigma_1, (\ell_0)_1, (L_F)_1, X_1, I_1, E_1)$$
$$\mathcal{A}_2 = (L_2, \Sigma_2, (\ell_0)_2, (L_F)_2, X_2, I_2, E_2)$$

Then we define $\mathcal{A}_1 \parallel \mathcal{A}_2$ as

Parallel composition of timed automata (1/2)

Just as finite-state automata, timed automata can be composed through parallel composition using synchronization actions

$$\mathcal{A}_1 = (L_1, \Sigma_1, (\ell_0)_1, (L_F)_1, X_1, I_1, E_1)$$
$$\mathcal{A}_2 = (L_2, \Sigma_2, (\ell_0)_2, (L_F)_2, X_2, I_2, E_2)$$

Then we define $\mathcal{A}_1 \parallel \mathcal{A}_2$ as

Parallel composition of timed automata (1/2)

Just as finite-state automata, timed automata can be composed through parallel composition using synchronization actions

$\mathcal{A}_1 = (L_1, \Sigma_1, (\ell_0)_1, (L_F)_1, X_1, I_1, E_1)$
$\mathcal{A}_2 = (L_2, \Sigma_2, (\ell_0)_2, (L_F)_2, X_2, I_2, E_2)$

Then we define $\mathcal{A}_1 \parallel \mathcal{A}_2$ as

Parallel composition of timed automata (1/2)

Just as finite-state automata, timed automata can be composed through parallel composition using synchronization actions

$$\mathcal{A}_1 = (L_1, \Sigma_1, (\ell_0)_1, (L_F)_1, X_1, I_1, E_1)$$
$$\mathcal{A}_2 = (L_2, \Sigma_2, (\ell_0)_2, (L_F)_2, X_2, I_2, E_2)$$

Then we define $\mathcal{A}_1 \parallel \mathcal{A}_2$ as

Parallel composition of timed automata (1/2)

Just as finite-state automata, timed automata can be composed through parallel composition using synchronization actions

$\mathcal{A}_1 = (L_1, \Sigma_1, (\ell_0)_1, (L_F)_1, X_1, I_1, E_1)$
$\mathcal{A}_2 = (L_2, \Sigma_2, (\ell_0)_2, (L_F)_2, X_2, I_2, E_2)$

Then we define $\mathcal{A}_1 \parallel \mathcal{A}_2$ as

Parallel composition of timed automata (1/2)

Just as finite-state automata, timed automata can be composed through parallel composition using synchronization actions

$\mathcal{A}_1 = (L_1, \Sigma_1, (\ell_0)_1, (L_F)_1, X_1, I_1, E_1)$
$\mathcal{A}_2 = (L_2, \Sigma_2, (\ell_0)_2, (L_F)_2, X_2, I_2, E_2)$

Then we define $\mathcal{A}_1 \parallel \mathcal{A}_2$ as

Parallel composition of timed automata (1/2)

Just as finite-state automata, timed automata can be composed through parallel composition using synchronization actions

$$\mathcal{A}_1 = (L_1, \Sigma_1, (\ell_0)_1, (L_F)_1, X_1, I_1, E_1)$$
$$\mathcal{A}_2 = (L_2, \Sigma_2, (\ell_0)_2, (L_F)_2, X_2, I_2, E_2)$$

Then we define $\mathcal{A}_1 \parallel \mathcal{A}_2$ as

Parallel composition of timed automata (2/2)

Parallel composition of timed automata (2/2)

Parallel composition of timed automata (2/2)

Parallel composition of timed automata (2/2)

Parallel composition of timed automata (2/2)

Parallel composition of timed automata (2/2)

Outline

Concrete runs of timed automata

- Concrete state of a TA: pair (ℓ, w), where
  - ℓ is a location,
  - w is a valuation of each clock

Example: ( , $\binom{x=1.2}{y=3.7}$)

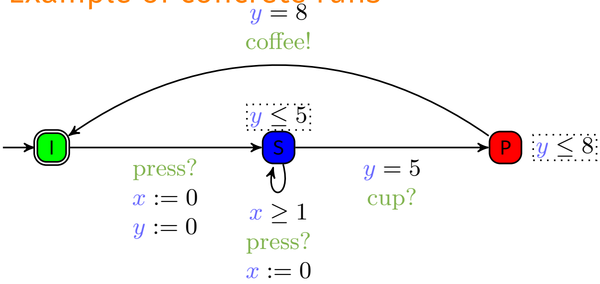
Concrete runs of timed automata

- Concrete state of a TA: pair (ℓ, w), where
  - ℓ is a location,
  - w is a valuation of each clock

  Example: (, $\binom{x=1.2}{y=3.7}$)

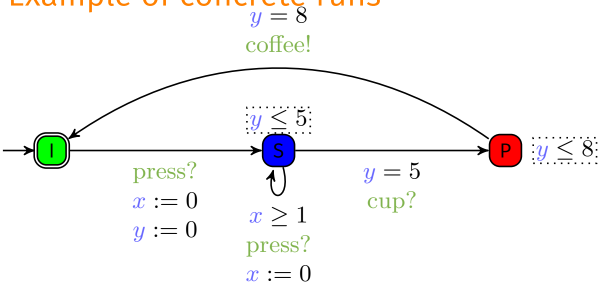
- Concrete run: alternating sequence of concrete states and actions or time elapse

Example of concrete runs



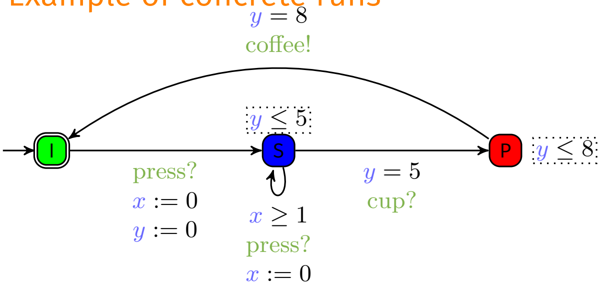
- Possible concrete runs for the coffee machine

Example of concrete runs



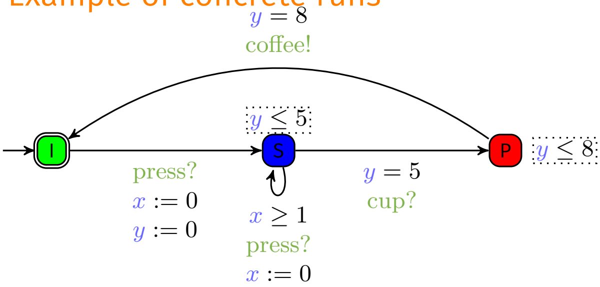
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



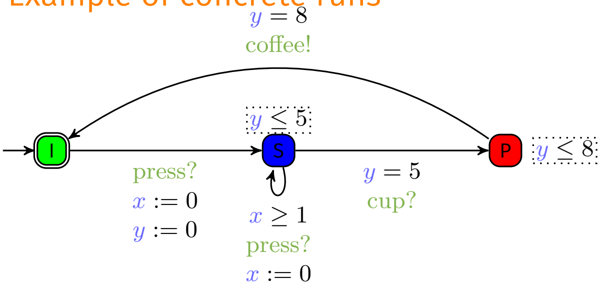
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



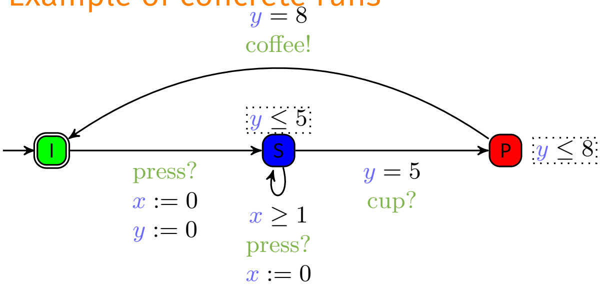
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



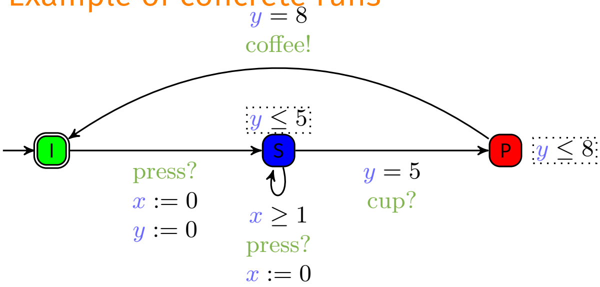
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



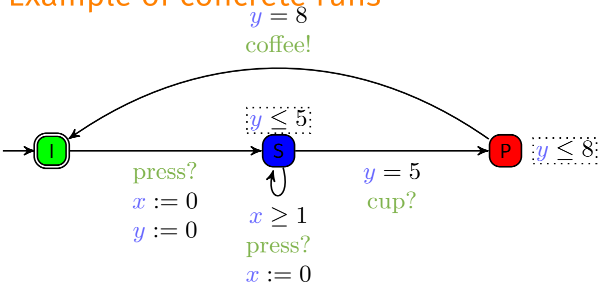
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



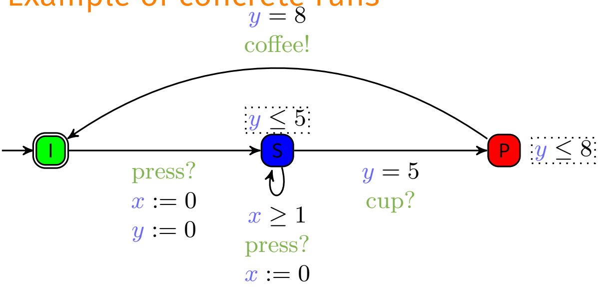
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



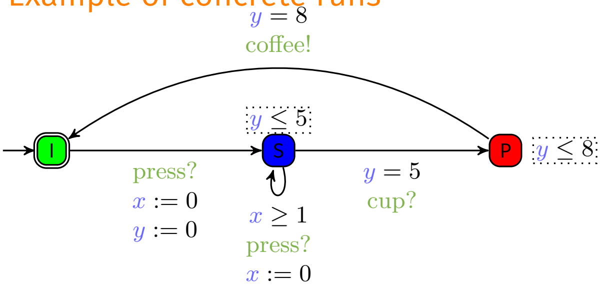
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



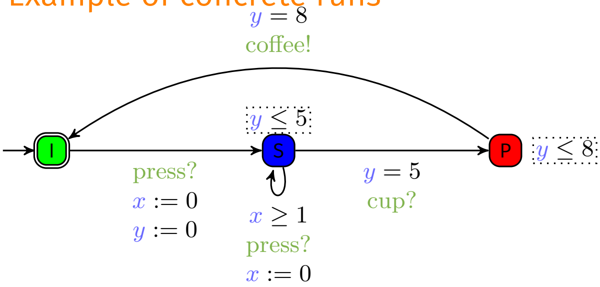
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



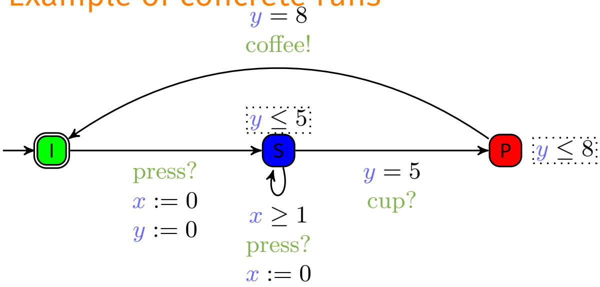
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



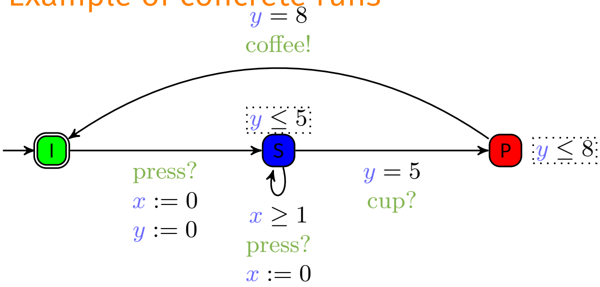
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



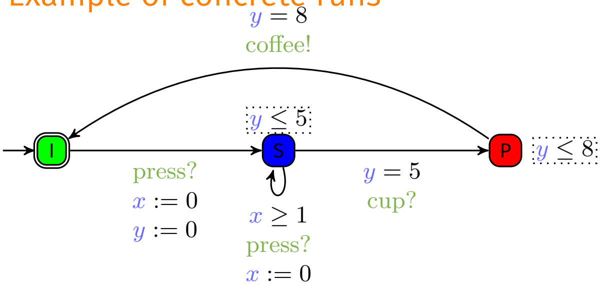
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



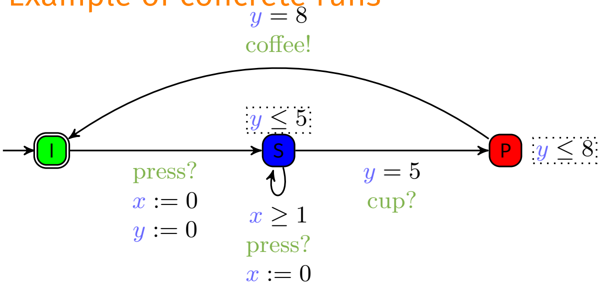
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



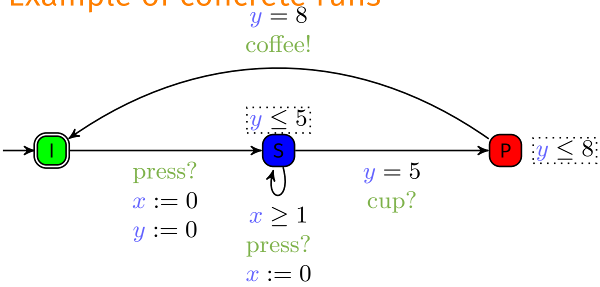
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



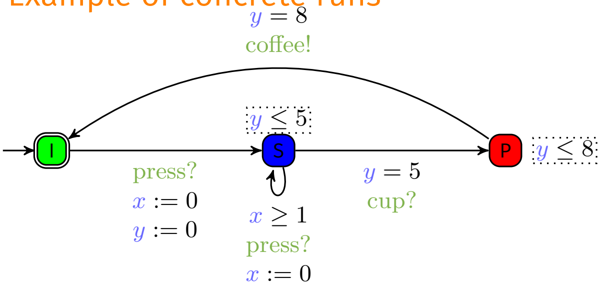
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Timed transition systems

Definition (Timed transition system)

A timed transition system (TTS) is a tuple $\mathcal{TTS} = (S, \Sigma, S_0, S_F, \rightarrow)$, where

- S is a set of states;
- Σ is an alphabet of events;
- $S_0 \subseteq S$ is a set of initial states;
- $S_F \subseteq S$ is a set of final (or accepting) states; and,
- $\rightarrow : S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \rightarrow 2^S$ is a transition relation.

We write $s_1 \xrightarrow{a} s_2$ when $(s_1, a, s_2) \in \rightarrow$.

Concrete semantics of timed automata: definition

Definition (Concrete semantics of a TA)

Given a TA $\mathcal{A} = (\Sigma, L, \ell_0, L_F, X, I, E)$, the concrete semantics of $\mathcal{A}$ is given by the timed transition system $(S, E, S_0, S_F, \rightarrow)$, with

- $S = \{(\ell, w) \in L \times \mathbb{R}_{\geq 0}^{|X|} \mid$

Concrete semantics of timed automata: definition

Definition (Concrete semantics of a TA)

Given a TA $\mathcal{A} = (\Sigma, L, \ell_0, L_F, X, I, E)$, the concrete semantics of $\mathcal{A}$ is given by the timed transition system $(S, E, S_0, S_F, \rightarrow)$, with

- $S = \{(\ell, w) \in L \times \mathbb{R}_{\geq 0}^{|X|} \mid$,
- $S_0 =$

Concrete semantics of timed automata: definition

Definition (Concrete semantics of a TA)

Given a TA $\mathcal{A} = (\Sigma, L, \ell_0, L_F, X, I, E)$, the concrete semantics of $\mathcal{A}$ is given by the timed transition system $(S, E, S_0, S_F, \rightarrow)$, with

- $S = \{(\ell, w) \in L \times \mathbb{R}_{\geq 0}^{|X|} \mid$,
- $S_0 =$ (with $\vec{0} \models I(\ell_0)$), and
- $S_F = \{(\ell, w) \in$

Concrete semantics of timed automata: definition

Definition (Concrete semantics of a TA)

Given a TA $\mathcal{A} = (\Sigma, L, \ell_0, L_F, X, I, E)$, the concrete semantics of $\mathcal{A}$ is given by the timed transition system $(S, E, S_0, S_F, \rightarrow)$, with

- $S = \{(\ell, w) \in L \times \mathbb{R}_{\geq 0}^{|X|} \mid$,
- $S_0 =$ (with $\vec{0} \models I(\ell_0)$), and
- $S_F = \{(\ell, w) \in$,
- $\rightarrow$ consists of the discrete and (continuous) delay transition relations:
  - discrete transitions: $(\ell, w) \xrightarrow{e} (\ell', w')$, if $(\ell, w), (\ell', w') \in S$, there exists $e = (\ell, g, a, R, \ell') \in E$, $w' = w[R]$, and $w \models g$.
  - delay transitions: $(\ell, w) \xrightarrow{d} (\ell, w + d)$, with $d \in \mathbb{R}_{\geq 0}$, if $\forall d' \in [0, d], (\ell, w + d') \in S$.

Notation:

$$w[R](x) = \Bigg\{$$

Concrete semantics of timed automata: definition

Definition (Concrete semantics of a TA)

Given a TA $\mathcal{A} = (\Sigma, L, \ell_0, L_F, X, I, E)$, the concrete semantics of $\mathcal{A}$ is given by the timed transition system $(S, E, S_0, S_F, \rightarrow)$, with

- $S = \{(\ell, w) \in L \times \mathbb{R}_{\geq 0}^{|X|} \mid$,
- $S_0 =$ (with $\vec{0} \models I(\ell_0)$), and
- $S_F = \{(\ell, w) \in$,
- $\rightarrow$ consists of the discrete and (continuous) delay transition relations:
  - discrete transitions: $(\ell, w) \xrightarrow{e} (\ell', w')$, if $(\ell, w), (\ell', w') \in S$, there exists $e = (\ell, g, a, R, \ell') \in E$, $w' = w[R]$, and $w \models g$.
  - delay transitions: $(\ell, w) \xrightarrow{d} (\ell, w + d)$, with $d \in \mathbb{R}_{\geq 0}$, if $\forall d' \in [0, d], (\ell, w + d') \in S$.

Notation:

$$w[R](x) = \begin{cases} & \text{if } x \in R \\ & \end{cases}$$

Concrete semantics of timed automata: definition

Definition (Concrete semantics of a TA)

Given a TA $\mathcal{A} = (\Sigma, L, \ell_0, L_F, X, I, E)$, the concrete semantics of $\mathcal{A}$ is given by the timed transition system $(S, E, S_0, S_F, \rightarrow)$, with

- $S = \{(\ell, w) \in L \times \mathbb{R}_{\geq 0}^{|X|} \mid$,
- $S_0 =$ (with $\vec{0} \models I(\ell_0)$), and
- $S_F = \{(\ell, w) \in$,
- $\rightarrow$ consists of the discrete and (continuous) delay transition relations:
  - discrete transitions: $(\ell, w) \xrightarrow{e} (\ell', w')$, if $(\ell, w), (\ell', w') \in S$, there exists $e = (\ell, g, a, R, \ell') \in E$, $w' = w[R]$, and $w \models g$.
  - delay transitions: $(\ell, w) \xrightarrow{d} (\ell, w + d)$, with $d \in \mathbb{R}_{\geq 0}$, if $\forall d' \in [0, d], (\ell, w + d') \in S$.

Notation:

$$w[R](x) = \begin{cases} & \text{if } x \in R \\ & \text{otherwise} \end{cases}$$

Concrete semantics of timed automata: definition (cont.)

We write $(\ell, w) \overset{(d,e)}{\mapsto} (\ell', w')$ or $((\ell, w), (d, e), (\ell', w')) \in \mapsto$ for a combination of a delay and discrete transitions if

$$\exists w'' : (\ell, w) \xrightarrow{d} (\ell, w'') \xrightarrow{e} (\ell', w')$$

Concrete semantics of timed automata: definition (cont.)

We write $(\ell, w) \overset{(d,e)}{\mapsto} (\ell', w')$ or $((\ell, w), (d, e), (\ell', w')) \in \mapsto$ for a combination of a delay and discrete transitions if

$$\exists w'' : (\ell, w) \xrightarrow{d} (\ell, w'') \xrightarrow{e} (\ell', w')$$

Some remarks on the semantics of timed automata:

- Is $\mathcal{TTS}$ finite?

Concrete semantics of timed automata: definition (cont.)

We write $(\ell, w) \overset{(d,e)}{\mapsto} (\ell', w')$ or $((\ell, w), (d, e), (\ell', w')) \in \mapsto$ for a combination of a delay and discrete transitions if

$$\exists w'' : (\ell, w) \xrightarrow{d} (\ell, w'') \xrightarrow{e} (\ell', w')$$

Some remarks on the semantics of timed automata:

- Is $\mathcal{TTS}$ finite?
- Is $\mathcal{TTS}$ finitely branching?

Concrete semantics of timed automata: definition (cont.)

We write $(\ell, w) \overset{(d,e)}{\mapsto} (\ell', w')$ or $((\ell, w), (d, e), (\ell', w')) \in \mapsto$ for a combination of a delay and discrete transitions if

$$\exists w'' : (\ell, w) \xrightarrow{d} (\ell, w'') \xrightarrow{e} (\ell', w')$$

Some remarks on the semantics of timed automata:

- Is $\mathcal{TTS}$ finite?
- Is $\mathcal{TTS}$ finitely branching?

Timed words

Definition (timed word)

A timed word over an alphabet of actions Σ is a possibly infinite sequence of the form $(a_0, d_0)(a_1, d_1)\cdots$ such that, for all integer $i \geq 0$, $a_i \in \Sigma$ and $d_i \leq d_{i+1}$.

Timed words

Definition (timed word)

A timed word over an alphabet of actions Σ is a possibly infinite sequence of the form $(a_0, d_0)(a_1, d_1)\cdots$ such that, for all integer $i \geq 0$, $a_i \in \Sigma$ and $d_i \leq d_{i+1}$.

Definition (timed word associated with a concrete run)

Given a concrete run ρ $(l_0, w_0)(d_0, e_0)(l_1, w_1)\cdots(d_i, e_i)(l_i, w_i)\cdots$, the timed word associated with ρ is

$$(\mathsf{Act}(e_0), d_0)(\mathsf{Act}(e_1),$$

Timed words

Definition (timed word)

A timed word over an alphabet of actions Σ is a possibly infinite sequence of the form $(a_0, d_0)(a_1, d_1)\cdots$ such that, for all integer $i \geq 0$, $a_i \in \Sigma$ and $d_i \leq d_{i+1}$.

Definition (timed word associated with a concrete run)

Given a concrete run ρ $(l_0, w_0)(d_0, e_0)(l_1, w_1)\cdots(d_i, e_i)(l_i, w_i)\cdots$, the timed word associated with ρ is

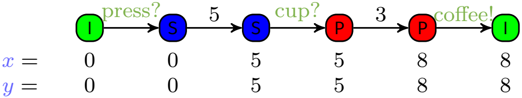
$$(\mathsf{Act}(e_0), d_0)(\mathsf{Act}(e_1),$$

Notation: $\mathsf{Act}(e_i)$ denotes the action of edge e_i

Timed words: exercise

Give the (formal) run and the associated timed words associated with the two example runs of the coffee machine:

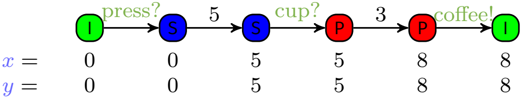
- Coffee with no sugar



Timed words: exercise

Give the (formal) run and the associated timed words associated with the two example runs of the coffee machine:

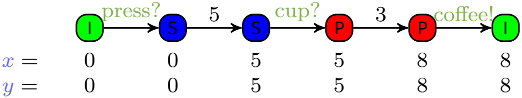
- Coffee with no sugar



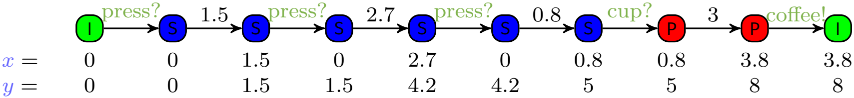
Timed words: exercise

Give the (formal) run and the associated timed words associated with the two example runs of the coffee machine:

- Coffee with no sugar



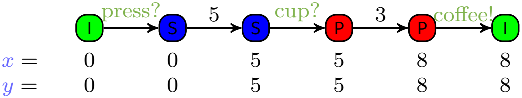
- Coffee with 2 doses of sugar



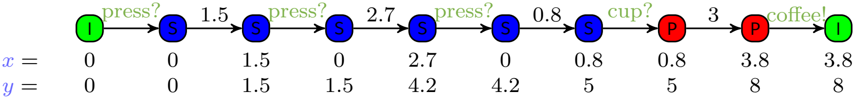
Timed words: exercise

Give the (formal) run and the associated timed words associated with the two example runs of the coffee machine:

- Coffee with no sugar



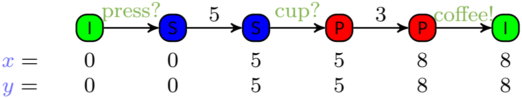
- Coffee with 2 doses of sugar



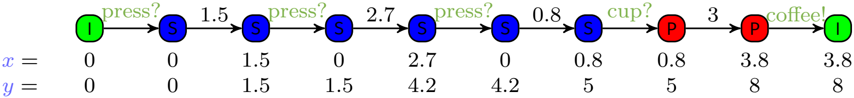
Timed words: exercise

Give the (formal) run and the associated timed words associated with the two example runs of the coffee machine:

- Coffee with no sugar



- Coffee with 2 doses of sugar



Timed language

Definition (timed language)

Given a TA $\mathcal{A}$, the timed language of $\mathcal{A}$ is the set of timed words associated with the runs of $\mathcal{A}$ ending in a location

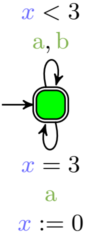
Timed language

Definition (timed language)

Given a TA $\mathcal{A}$, the timed language of $\mathcal{A}$ is the set of timed words associated with the runs of $\mathcal{A}$ ending in a location

Timed language: Example 1

Give the timed language of the following automaton [Alur and Dill, 1994]



Timed language: Example 2

Give the timed language of the following automaton

Timed language: Example 3

Give the timed language of the coffee machine

Accepting locations?

Timed automata may or may not be equipped with accepting locations

Often, timed automata with no accepting locations are called timed safety automata [Henzinger et al., 1994]

In that case the timed language can be defined as:

- All possible timed words read by the automaton
- All possible maximal timed words read by the automaton
  - Maximal: infinite or that cannot be extended
- All possible infinite timed words read by the automaton

Accepting locations?

Timed automata may or may not be equipped with accepting locations

Often, timed automata with no accepting locations are called timed safety automata [Henzinger et al., 1994]

In that case the timed language can be defined as:

- All possible timed words read by the automaton
- All possible maximal timed words read by the automaton
    - Maximal: infinite or that cannot be extended
- All possible infinite timed words read by the automaton

Theorem

The expressive power of timed safety automata is strictly less than timed automata with accepting locations *[Henzinger et al., 1995]*

Deadlocks and timelocks

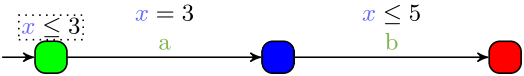
Timed automata can be subject to two annoying behaviors:

- Deadlock: similar to finite state automata
  - Can be a problem of

Deadlocks and timelocks

Timed automata can be subject to two annoying behaviors:

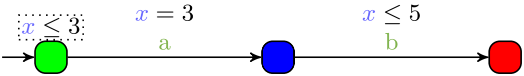
- Deadlock: similar to finite state automata
  - Can be a problem of



Deadlocks and timelocks

Timed automata can be subject to two annoying behaviors:

- Deadlock: similar to finite state automata
  - Can be a problem of

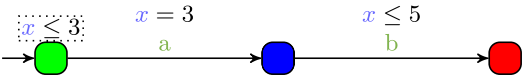


- Timelock: coming from the timed nature of TAs
  - Can

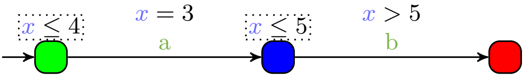
Deadlocks and timelocks

Timed automata can be subject to two annoying behaviors:

- Deadlock: similar to finite state automata
  - Can be a problem of



- Timelock: coming from the timed nature of TAs
  - Can



The Zeno problem (1/2)

Definition (Zeno run)

A run is Zeno if it contains an infinite number of actions in finite time.

The Zeno problem (1/2)

Definition (Zeno run)

A run is Zeno if it contains an infinite number of actions in finite time.

- Example of TA containing at least one Zeno run

The Zeno problem (1/2)

Definition (Zeno run)

A run is Zeno if it contains an infinite number of actions in finite time.

- Example of TA containing at least one Zeno run
- Example of TA containing only non-Zeno runs

The Zeno problem (1/2)

Definition (Zeno run)

A run is Zeno if it contains an infinite number of actions in finite time.

- Example of TA containing at least one Zeno run

- Example of TA containing only non-Zeno runs

The Zeno problem (2/2)

Problem (Zeno runs)

An infinite number of actions in finite time is impossible in practice

- *Processors have finite precision*

Zeno runs must be pruned when performing model checking

The Zeno problem (2/2)

Problem (Zeno runs)

An infinite number of actions in finite time is impossible in practice

- *Processors have finite precision*

Zeno runs must be pruned when performing model checking

Some solutions:

- Transform the TA (with an additional clock)
  [Tripakis, 1999, Tripakis et al., 2005, Bowman and Gómez, 2006, Gómez and Bowman, 2007]
- Transform the zone graph [Herbreteau et al., 2012]
- Consider a different but closely related formalism [Sun et al., 2013]
- Transform the TA on-the-fly [Wang et al., 2015]

Outline

Example: Railroad gate controller [Alur et al., 1993b]

Design three timed automata in parallel:

1. The train: once it is approaching (action $\mathrm{approach}$), it will come in (action in) after at least 5 time units, then go out (action out) and finally exit (action exit) after at most 6 time units
2. The gate: upon reception of a lower signal, starts to lower; once it is down, and upon reception of a raise signal, the gate raises again; the time to lower and to raise the gate is an interval $[1, 3]$
3. The controller: once a train approaches (action $\mathrm{approach}$), it triggers the lower signal within $[2, 3]$ time units; then, once the train exits (action exit), it triggers the raise signal again within $[2, 4]$ time units

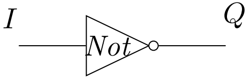
All TAs are cyclic, i. e., repeat the same behavior forever.

Example: Railroad gate controller (train)

Example: Railroad gate controller (gate)

Example: Railroad gate controller (controller)

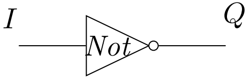
Example: A hardware gate



The output Q reacts to the change of the input I (actions $I^{\uparrow}$ and $I^{\downarrow}$) after a delay $[5, 9]$

[Chevallier et al., 2009]

Example: A hardware gate



The output Q reacts to the change of the input I (actions $I^{\uparrow}$ and $I^{\downarrow}$) after a delay $[5, 9]$

[Chevallier et al., 2009]

Example: A nuclear power plant

Design a PTA modeling a nuclear power plant:

- At first, the plant is in normal mode.
- Suddenly, it may start to heat (action $\mathrm{startHeating}$).
- At that point, a timer is set; after p_2 time units, the timer will trigger an alarm (action alarm).
- Then, p_3 time units later, a watering system (action $\mathrm{watering}$) starts.
- This watering system lasts for at most p_4 time units, after which the plant is cool again (action cool) and goes back to the normal mode.
- However, p_1 time units after the plant starts to heat, the plant may explode at any time (action boom)—unless of course it is cool again.

Example: A nuclear power plant (solution)

Example: A real-time system

Design a (network of) timed automata modeling the following components:

1. a periodic task T_1 of period 5 with offset 2, best and worst case execution times in $[3, 4]$
2. a sporadic task T_2 of minimum interarrival time 20, best and worst case execution times in $[1, 2]$
3. a non-preemptive scheduler with fixed priority

Example: A real-time system (solution)

Outline

Timed temporal logics

- Specify properties on the order and the delays between events
- No X operator because

Timed temporal logics

- Specify properties on the order and the delays between events
- No X operator because

TCTL (Timed CTL) [Alur et al., 1993a]

TCTL expresses formulas on the order and the time between the future events for some or for all paths, using a set of atomic propositions AP

- Timed extension of CTL

Quantifiers over paths:

$$\varphi ::= p \in AP \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi$$

Quantifiers over states:

$$\psi ::= \varphi \mathsf{U}_I \varphi$$

I is an interval of the form $[a, b]$, $[a, b)$, $(a, b]$, (a, b), $[a, \infty)$, or (a, ∞), where $a, b \in \mathbb{N}$

Semantics of TCTL: discrete vs. continuous

Two semantics:

- Continuous semantics: signals

Semantics of TCTL: discrete vs. continuous

Two semantics:

- Continuous semantics: signals

- Discrete (point-wise) semantics: timed words

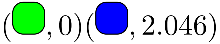
(, 0)

Semantics of TCTL: discrete vs. continuous

Two semantics:

- Continuous semantics: signals

- Discrete (point-wise) semantics: timed words

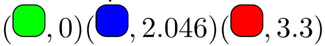


Semantics of TCTL: discrete vs. continuous

Two semantics:

- Continuous semantics: signals

- Discrete (point-wise) semantics: timed words

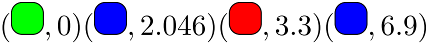


Semantics of TCTL: discrete vs. continuous

Two semantics:

- Continuous semantics: signals

- Discrete (point-wise) semantics: timed words



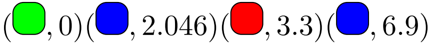
Are they equivalent?

Semantics of TCTL: discrete vs. continuous

Two semantics:

- Continuous semantics: signals

- Discrete (point-wise) semantics: timed words



Are they equivalent?

Continuous semantics of TCTL

$s \models p$	iff	p holds at the current position
$s \models \neg p$	iff	p does not hold at the current position
$s \models \varphi \land \psi$	iff	$s \models \varphi \land s \models \psi$
$s \models \varphi \lor \psi$	iff	$s \models \varphi \lor s \models \psi$
$s \models \mathsf{E}\psi\mathsf{U}_I\varphi$	iff	there exists a future path and $t \in I$ for which ψ holds until t and φ holds at t
$s \models \mathsf{A}\psi\mathsf{U}_I\varphi$	iff	for all future paths, there exists $t \in I$ for which ψ holds until t and φ holds at t

Illustrating TCTL operators

Discrete semantics of TCTL [Bouyer et al., 2017]

Informal description of the U (the rest is similar):

$s \models \mathsf{E}\psi\mathsf{U}_I\varphi$ iff there exists $n > 0$ such that φ holds from point n (with the time of point n within I) and for each $0 < m < n$, ψ holds at point m

Note: strict version of the U, considered in [Bouyer et al., 2017] (not necessarily standard)

Semantics of TCTL: discrete vs. continuous (example)

Exhibit a word and a TCTL formula for which:

1. the formula holds under the continuous but not the discrete semantics
2. the formula holds under the discrete but not the continuous semantics

Semantics of TCTL: discrete vs. continuous (example)

Exhibit a word and a TCTL formula for which:

1. the formula holds under the continuous but not the discrete semantics
2. the formula holds under the discrete but not the continuous semantics

An example TA:

Semantics of TCTL: discrete vs. continuous (example)

Exhibit a word and a TCTL formula for which:

1. the formula holds under the continuous but not the discrete semantics
2. the formula holds under the discrete but not the continuous semantics

An example TA:

A concrete run ω (continuous semantics):

Semantics of TCTL: discrete vs. continuous (example)

Exhibit a word and a TCTL formula for which:

1. the formula holds under the continuous but not the discrete semantics
2. the formula holds under the discrete but not the continuous semantics

An example TA:

A concrete run ω (continuous semantics):

Equivalent run ρ in the discrete semantics:

Semantics of TCTL: discrete vs. continuous (example)

Exhibit a word and a TCTL formula for which:

1. the formula holds under the continuous but not the discrete semantics
2. the formula holds under the discrete but not the continuous semantics

An example TA:

A concrete run ω (continuous semantics):

Equivalent run ρ in the discrete semantics:

Semantics of TCTL: discrete vs. continuous (example)

Exhibit a word and a TCTL formula for which:

1. the formula holds under the continuous but not the discrete semantics
2. the formula holds under the discrete but not the continuous semantics

An example TA:

A concrete run ω (continuous semantics):

Equivalent run ρ in the discrete semantics:

Semantics of TCTL: discrete vs. continuous (example)

Exhibit a word and a TCTL formula for which:

1. the formula holds under the continuous but not the discrete semantics
2. the formula holds under the discrete but not the continuous semantics

An example TA:

A concrete run ω (continuous semantics):

Equivalent run ρ in the discrete semantics:

Semantics of TCTL: discrete vs. continuous (example)

Exhibit a word and a TCTL formula for which:

1. the formula holds under the continuous but not the discrete semantics
2. the formula holds under the discrete but not the continuous semantics

An example TA:

A concrete run ω (continuous semantics):

Equivalent run ρ in the discrete semantics:

Semantics of TCTL: discrete vs. continuous (example)

Exhibit a word and a TCTL formula for which:

1. the formula holds under the continuous but not the discrete semantics
2. the formula holds under the discrete but not the continuous semantics

An example TA:

A concrete run ω (continuous semantics):

Equivalent run ρ in the discrete semantics:

Semantics of TCTL: discrete vs. continuous (example)

Exhibit a word and a TCTL formula for which:

1. the formula holds under the continuous but not the discrete semantics
2. the formula holds under the discrete but not the continuous semantics

An example TA:

A concrete run ω (continuous semantics):

Equivalent run ρ in the discrete semantics:

TCTL: Examples

- “Whatever happens, the plane will never crash in the next 10 minutes”

TCTL: Examples

- “Whatever happens, the plane will never crash in the next 10 minutes”

- “I may get a job within one year”

TCTL: Examples

- “Whatever happens, the plane will never crash in the next 10 minutes”
- “I may get a job within one year”
- “I am sure to get a job within one year”

TCTL: Examples

- "Whatever happens, the plane will never crash in the next 10 minutes"
- "I may get a job within one year"
- "I am sure to get a job within one year"
- "Whenever a fire breaks, it is sure that the alarm will start ringing at least 5 seconds and at most 10 seconds later"

TCTL: Examples

- "Whatever happens, the plane will never crash in the next 10 minutes"
- "I may get a job within one year"
- "I am sure to get a job within one year"
- "Whenever a fire breaks, it is sure that the alarm will start ringing at least 5 seconds and at most 10 seconds later"
- "Whatever happens, I will love you for 2 years after we marry"

TCTL: Examples

- "Whatever happens, the plane will never crash in the next 10 minutes"
- "I may get a job within one year"
- "I am sure to get a job within one year"
- "Whenever a fire breaks, it is sure that the alarm will start ringing at least 5 seconds and at most 10 seconds later"
- "Whatever happens, I will love you for 2 years after we marry"

TCTL: Examples (coffee machine)

- “Whenever the button is pressed, a coffee is necessarily eventually delivered within 10 units of time.”

TCTL: Examples (coffee machine)

- "Whenever the button is pressed, a coffee is necessarily eventually delivered within 10 units of time."

TCTL: Examples (coffee machine)

- “Whenever the button is pressed, a coffee is necessarily eventually delivered within 10 units of time.”

TCTL: Examples (coffee machine)

- “Whenever the button is pressed, a coffee is necessarily eventually delivered within 10 units of time.”

- “It must never happen that the button can be pressed twice within 1 unit of time.”

TCTL: Examples (coffee machine)

- “Whenever the button is pressed, a coffee is necessarily eventually delivered within 10 units of time.”

- “It must never happen that the button can be pressed twice within 1 unit of time.”

TCTL: Examples (coffee machine)

- "Whenever the button is pressed, a coffee is necessarily eventually delivered within 10 units of time."

- "It must never happen that the button can be pressed twice within 1 unit of time."

TCTL: Examples (coffee machine)

- “Whenever the button is pressed, a coffee is necessarily eventually delivered within 10 units of time.”

- “It must never happen that the button can be pressed twice within 1 unit of time.”

- “It must never happen that the button can be pressed twice within a time strictly less than 1 unit of time.”

TCTL: Examples (coffee machine)

- “Whenever the button is pressed, a coffee is necessarily eventually delivered within 10 units of time.”

- “It must never happen that the button can be pressed twice within 1 unit of time.”

- “It must never happen that the button can be pressed twice within a time strictly less than 1 unit of time.”

TCTL: Examples (coffee machine)

- "Whenever the button is pressed, a coffee is necessarily eventually delivered within 10 units of time."
- "It must never happen that the button can be pressed twice within 1 unit of time."
- "It must never happen that the button can be pressed twice within a time strictly less than 1 unit of time."

Other timed temporal logics

- MTL: linear time [Koymans, 1990]
  - Can be seen as a timed extension of LTL (just as TCTL is a timed extension of CTL)
  - Variant: MITL [Alur et al., 1996]
    - Variant of MTL disallowing punctuality
- STL: to reason about signals [Maler and Nickovic, 2004]
- etc.

See, e. g., [Bouyer et al., 2017] for a partial survey

Outline

Observers for timed automata

Observers (both untimed and timed) can be used for timed automata

Just as for FA:

- A TA observer is an automaton that observes the system behavior
- It synchronizes with other automata's actions
- It can read the clocks of the system, and/or feature its own clock(s)
- It must be non-blocking
  - Pay attention to timelocks or deadlocks!
- Its location(s) give an indication on the system property

Then verifying the property reduces to a reachability condition on the observer (in parallel with the system)

The expressive power of observers for timed automata has been studied in [Aceto et al., 1998, Aceto et al., 2003]

Exercise: An observer for the coffee machine

Design an observer for the coffee machine verifying that it must never happen that the button can be pressed twice within a time strictly less than 1 unit of time.

Outline

What is decidability?

Definition

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

What is decidability?

Definition

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

"given three integers, is one of them the product of the other two?"

"given a context-free grammar, does it generate all strings?"

"given a Turing machine, will it eventually halt?"

"given a timed automaton, does there exist a run from the initial state to a given location ℓ?"

What is decidability?

Definition

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

"given three integers, is one of them the product of the other two?"

"given a context-free grammar, does it generate all strings?"

"given a Turing machine, will it eventually halt?"

"given a timed automaton, does there exist a run from the initial state to a given location ℓ?"

What is decidability?

Definition

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

"given three integers, is one of them the product of the other two?"

"given a context-free grammar, does it generate all strings?"

"given a Turing machine, will it eventually halt?"

"given a timed automaton, does there exist a run from the initial state to a given location ℓ?"

What is decidability?

Definition

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

“given three integers, is one of them the product of the other two?”

“given a context-free grammar, does it generate all strings?”

“given a Turing machine, will it eventually halt?”

“given a timed automaton, does there exist a run from the initial state to a given location ℓ?”

What is decidability?

Definition

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

"given three integers, is one of them the product of the other two?"

"given a context-free grammar, does it generate all strings?"

"given a Turing machine, will it eventually halt?"

"given a timed automaton, does there exist a run from the initial state to a given location ℓ?"

Why studying decidability?

If a decision problem is undecidable, it is hopeless to look for algorithms yielding exact solutions (because that is impossible)

Why studying decidability?

If a decision problem is undecidable, it is hopeless to look for algorithms yielding exact solutions (because that is impossible)

However, one can:

- design semi-algorithms: if the algorithm halts, then its result is correct
- design algorithms yielding over- or under-approximations

Problem: an infinite concrete semantics

- Time is dense: transitions can be taken anytime
  - Infinite number of timed runs
  - Infinite number of states
  - Infinitely branching structure

Problem: an infinite concrete semantics

- Time is dense: transitions can be taken anytime
  - Infinite number of timed runs
  - Infinite number of states
  - Infinitely branching structure
  - Model checking needs a finite structure!

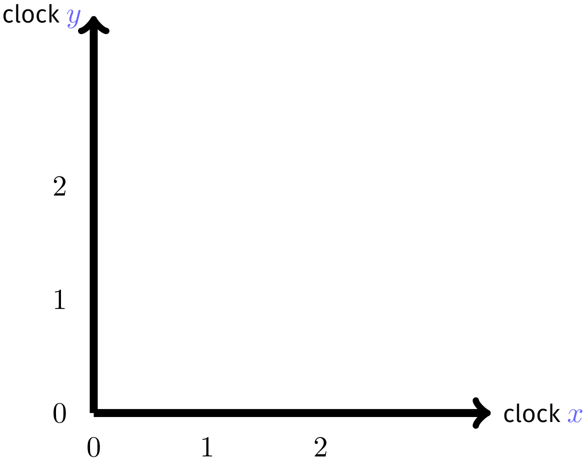
Outline

Dense time

- A first remark: Some runs are equivalent
  - Taking the press? action at $t = 1.5$ or $t = 1.57$ is equivalent w.r.t. the possible actions

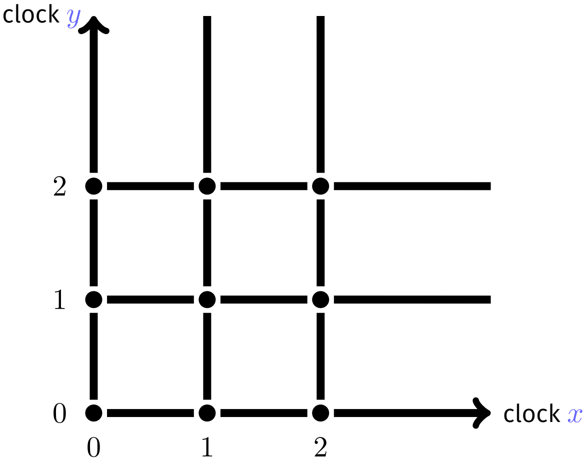
- Idea: reason with abstractions
  - Region automaton [Alur and Dill, 1994], and zone automaton
  - Example: in location , all clock values in the following zone are equivalent
$$y \leq 5 \land y - x \geq 4$$
  - This abstraction is finite

Regions



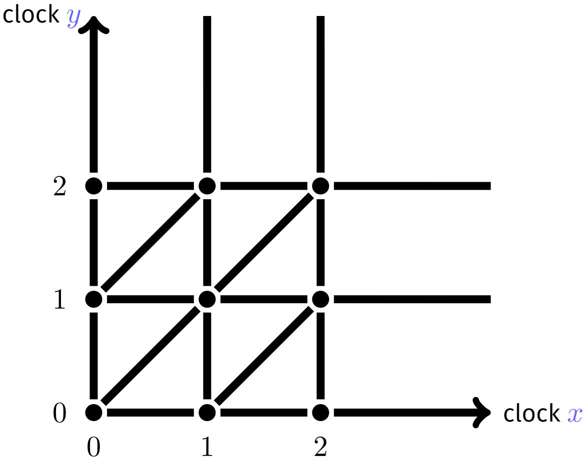
Inspired by a similar LaTeX illustration by Patricia Bouyer

Regions



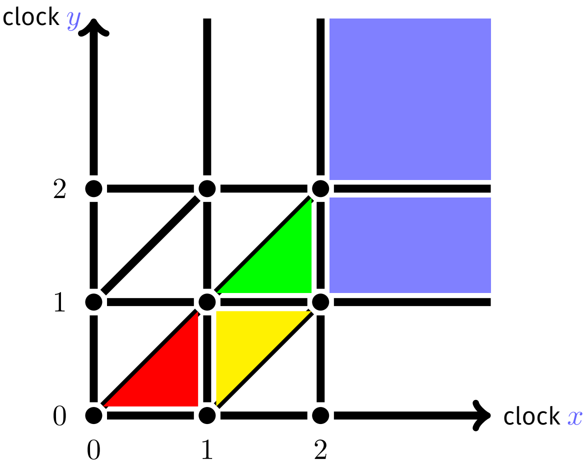
Inspired by a similar LaTeX illustration by Patricia Bouyer

Regions



Inspired by a similar LaTeX illustration by Patricia Bouyer

Regions



Inspired by a similar LaTeX illustration by Patricia Bouyer

Region graph construction

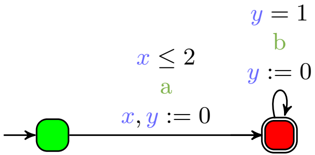
Two successors:

- time-elapsing
- clock reset

(see white board for the graph construction)

Region graph construction: exercise

Construct the region graph of the following TA:



On the region graph finiteness

Is the region graph of TAs finite?

On the region graph finiteness

Is the region graph of TAs finite?

- Example with two clocks x, y:

On the region graph finiteness

Is the region graph of TAs finite?

- Example with two clocks x, y:

Solution: k-extrapolation

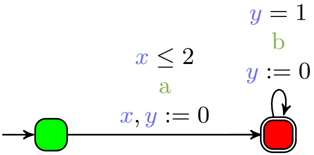
- Idea: "all integer (resp. rational) clock valuations above the greatest constant k of the TA are equivalent" [Alur and Dill, 1994]

With this additional technicality, there is a finite number of regions in a TA

Extrapolation: illustration

Extrapolation: exercise

Construct the region graph (with the k-extrapolation) of the following TA:



Outline

Zone construction for timed automata

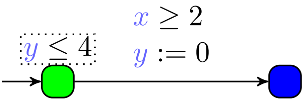
- Objective: group all concrete states reachable by the same sequence of discrete actions
- Symbolic state: a location ℓ and a (infinite) set of states Z
- For timed automata, Z can be represented by a convex polyhedron with a special form called zone, with constraints

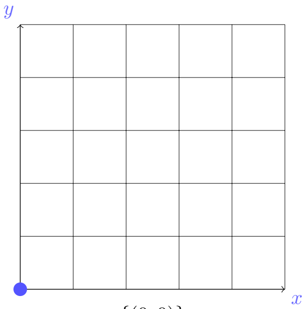
$$-d_{0i} \leq x_i \leq d_{i0} \text{ and } x_i - x_j \leq d_{ij}$$

- Computation of successive reachable symbolic states can be performed symbolically with polyhedral operations: for edge $e = (\ell, a, g, R, \ell')$:

$$\mathsf{Succ}\big((\ell, Z), e\big) = \Big(\ell', \big((Z \cap g)[R] \cap I(\ell')\big)^{\nearrow} \cap I(\ell')\Big)$$

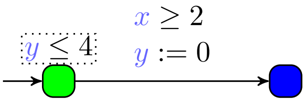
Zone construction for timed automata: Example

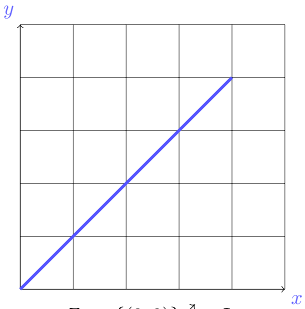




$\{(0, 0)\}$

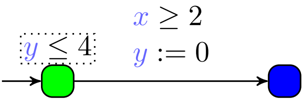
Zone construction for timed automata: Example

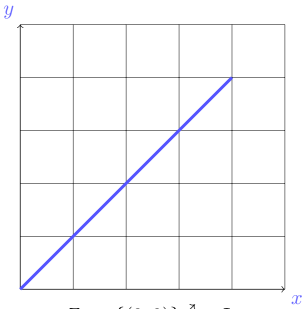




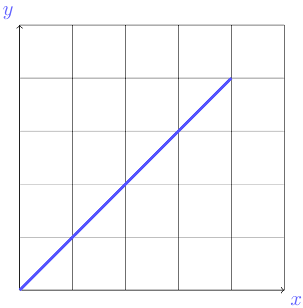
$Z_0 = \{(0,0)\}^{\nearrow} \cap I$

Zone construction for timed automata: Example



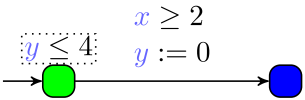


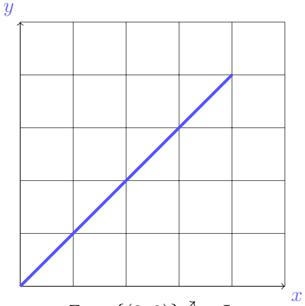
$Z_0 = \{(0,0)\}^{\nearrow} \cap I$



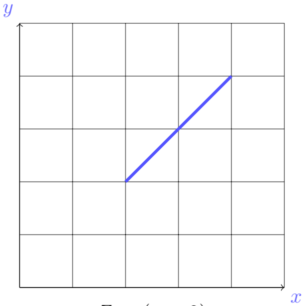
Z_0

Zone construction for timed automata: Example



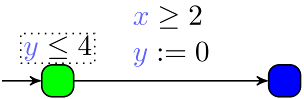


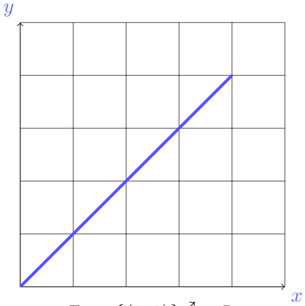
$Z_0 = \{(0,0)\}^{\nearrow} \cap I$



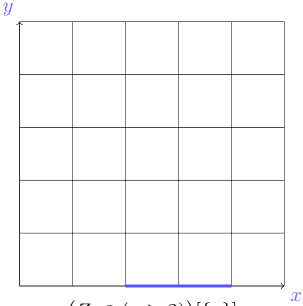
$Z_0 \cap (x \geq 2)$

Zone construction for timed automata: Example



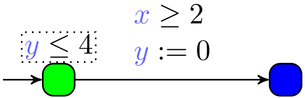


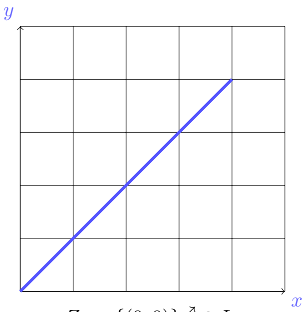
$Z_0 = \{(0,0)\}^{\nearrow} \cap I$



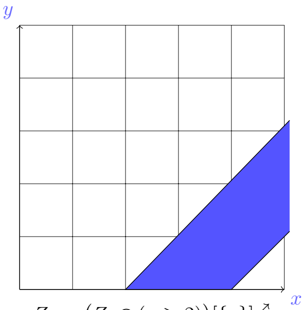
$\big(Z_0 \cap (x \geq 2)\big)[\{y\}]$

Zone construction for timed automata: Example





$Z_0 = \{(0,0)\}^{\nearrow} \cap I$



$Z_1 = \big(Z_0 \cap (x \geq 2)\big)[\{y\}]^{\nearrow}$

TikZ animation based on a LaTeX code by Didier Lime

Zone graph of timed automata

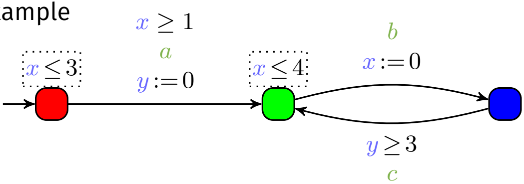
- Abstract state of a TA: pair (ℓ, C), where
  - ℓ is a location, and C is a constraint on the clocks ("zone")

Zone graph of timed automata

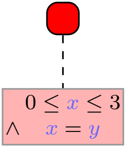
- Abstract state of a TA: pair (ℓ, C), where
  - ℓ is a location, and C is a constraint on the clocks ("zone")
- Abstract run: alternating sequence of abstract states and actions

Zone graph of timed automata

- Abstract state of a TA: pair (ℓ, C), where
  - ℓ is a location, and C is a constraint on the clocks ("zone")
- Abstract run: alternating sequence of abstract states and actions
- Example

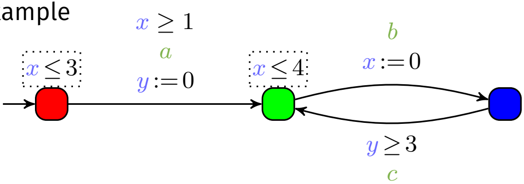


  - Possible abstract run from the zone graph of this TA

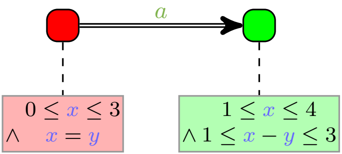


Zone graph of timed automata

- Abstract state of a TA: pair (ℓ, C), where
  - ℓ is a location, and C is a constraint on the clocks ("zone")
- Abstract run: alternating sequence of abstract states and actions
- Example

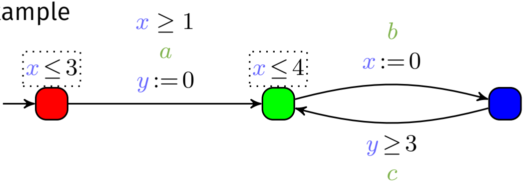


  - Possible abstract run from the zone graph of this TA

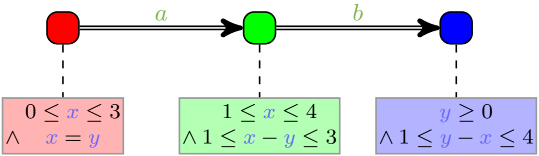


Zone graph of timed automata

- Abstract state of a TA: pair (ℓ, C), where
  - ℓ is a location, and C is a constraint on the clocks ("zone")
- Abstract run: alternating sequence of abstract states and actions
- Example

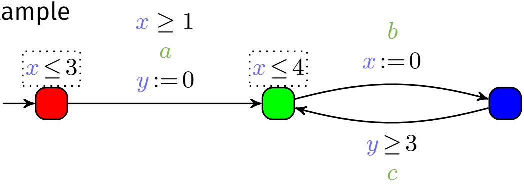


  - Possible abstract run from the zone graph of this TA

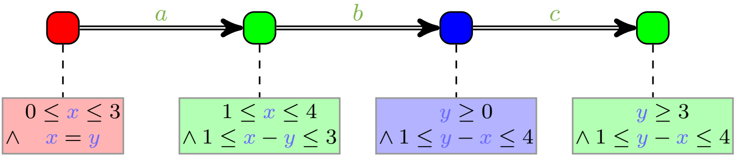


Zone graph of timed automata

- Abstract state of a TA: pair (ℓ, C), where
  - ℓ is a location, and C is a constraint on the clocks ("zone")
- Abstract run: alternating sequence of abstract states and actions
- Example



  - Possible abstract run from the zone graph of this TA



On the zone graph finiteness

Is the zone graph of TAs finite?

On the zone graph finiteness

Is the zone graph of TAs finite?

- Example:

On the zone graph finiteness

Is the zone graph of TAs finite?

- Example:

Solution: k-extrapolation

- Idea: "all clock valuations above the greatest constant k of the TA are equivalent" [Bengtsson and Yi, 2003]
- Can we do more efficient?
  - L/U-abstractions [Behrmann et al., 2006]
  - Lazy abstractions [Herbreteau et al., 2013]

With this additional technicality, there is a finite number of reachable zones in a TA

More on zones

- Symbolic states can be efficiently computed using Difference Bound Matrices (DBMs)
- $isReachable$ can be applied to the abstract semantics of timed automata (the underlying finite transition system)
- The zone graph is theoretically larger than the region graph but practically smaller
  - On-the-fly construction
  - Various optimization techniques

Outline

Decision problems for timed automata

The finiteness of the region automaton allows us to check properties

- ☺ Reachability of a location (PSPACE-complete) [Alur and Dill, 1994]
- ☺ Liveness (Büchi conditions)
- ☺ TCTL model-checking [Alur and Dill, 1994]

Some problems impossible to check using the zone graph (but still decidable)

- ☺ non-Zenoness emptiness check [Gómez and Bowman, 2007]

Some undecidable problems

- ☹ universality of the timed language [Alur and Dill, 1994]
- ☹ timed language inclusion [Alur and Dill, 1994]
  - Some decidable subclasses

    [Alur and Dill, 1994, Ouaknine and Worrell, 2003, Ouaknine and Worrell, 2004]

    [Abdulla et al., 2008, Bertrand et al., 2011]

Syntactic variants of timed automata

Variants of the syntax with consequences on the decidability

- Can we use diagonal constraints ("$x - y$")? [Bouyer, 2003]
- Can we reset clocks to constants $\neq 0$? [Bouyer et al., 2004]
- Can we reset clocks to other clocks? [Bouyer et al., 2004]
- Can we reset clocks to unknown constants? [André et al., 2019]
- Can we stop the elapsing of some clocks? [Cassez and Larsen, 2000]

Further challenges

- Controller synthesis [Sankur et al., 2013, Bacci et al., 2018]
  - Game theory
- Timed language inclusion (using TA as a specification language)
  - Decidable subclasses [Ouaknine and Worrell, 2003, Ouaknine and Worrell, 2004]
  - Practical algorithms [Wang et al., 2017]
- Robustness [De Wulf et al., 2004, Bouyer et al., 2013, Bacci et al., 2018]
- Distributed algorithms [Laarman et al., 2013, Zhang et al., 2016]

Further challenges

- Controller synthesis [Sankur et al., 2013, Bacci et al., 2018]
  - Game theory
- Timed language inclusion (using TA as a specification language)
  - Decidable subclasses [Ouaknine and Worrell, 2003, Ouaknine and Worrell, 2004]
  - Practical algorithms [Wang et al., 2017]
- Robustness [De Wulf et al., 2004, Bouyer et al., 2013, Bacci et al., 2018]
- Distributed algorithms [Laarman et al., 2013, Zhang et al., 2016]

Still a very active research field!

Outline

Software supporting timed automata

Timed automata have been successfully used since the 1990s

Tools for modeling and verifying models specified using TA

- HyTech (also hybrid, parametric timed automata) [Henzinger et al., 1997]
- Kronos [Yovine, 1997]
- TReX (also parametric timed automata) [Annichini et al., 2001]
- Uppaal [Larsen et al., 1997]
- Roméo (parametric time Petri nets) [Lime et al., 2009]
- PAT (also other formalisms) [Sun et al., 2009a]
- IMITATOR (also parametric timed automata) [André et al., 2012]

Some case studies and application domains

- Scheduling and real-time systems

  [Fehnker, 1999, Abdeddaïm and Maler, 2001, Adbeddaïm et al., 2006, Abdeddaïm and Masson, 2012]

- Protocols
  - Bounded retransmission protocol [D'Argenio et al., 1997]
  - Audio-video protocol [Havelund et al., 1997]
  - Fast Reservation Protocol [Tripakis and Yovine, 1998]
  - IEEE 1394a root contention protocol [Simons and Stoelinga, 2001]

- Hardware circuits

  [Bozga et al., 2002, Chevallier et al., 2009]

- Health and biology [Schivo et al., 2014]

- Monitoring [Waga et al., 2016, Waga et al., 2018]

- Survey on the industrial use of UPPAAL [Larsen et al., 2018]

Outline

What's beyond timed automata...?

- Stopping clocks: stopwatch automata [Cassez and Larsen, 2000]
  - ☹ Undecidable
  - ☺ Interesting application domains
- Adding costs: energy [Behrmann et al., 2001, Alur et al., 2004]
- Enriching TA with tasks [Fersman et al., 2007]
- Adding unknown parameters [Alur et al., 1993b]
- Allowing non-linear clocks: hybrid automata [Henzinger, 1996, Asarin et al., 2012]
- Adding probabilities [Kwiatkowska et al., 2002]
  - Statistical model checking [Legay et al., 2010]

Towards a parametrization…

- Challenge 1: systems incompletely specified
  - Some delays may not be known yet, or may change
- Challenge 2: Robustness [Markey, 2011]
  - What happens if 8 is implemented with 7.99?
  - Can I really get a coffee with 5 doses of sugar?
- Challenge 3: Optimization of timing constants
  - Up to which value of the delay between two actions press? can I still order a coffee with 3 doses of sugar?
- Challenge 4: Avoiding numerous verifications
  - If one of the timing delays of the model changes, should I model check again the whole system?

Towards a parametrization…

- Challenge 1: systems incompletely specified
  - Some delays may not be known yet, or may change
- Challenge 2: Robustness [Markey, 2011]
  - What happens if 8 is implemented with 7.99?
  - Can I really get a coffee with 5 doses of sugar?
- Challenge 3: Optimization of timing constants
  - Up to which value of the delay between two actions $\mathrm{press?}$ can I still order a coffee with 3 doses of sugar?
- Challenge 4: Avoiding numerous verifications
  - If one of the timing delays of the model changes, should I model check again the whole system?
- A solution: Parametric analysis
  - Consider that timing constants are unknown (parameters)
  - Find good values for the parameters s.t. the system behaves well

Source and references

General references

- Timed Automata: Semantics, Algorithms and Tools [Bengtsson and Yi, 2003]
- Systems and Software Verification [Bérard et al., 2001]
- Principles of Model Checking [Baier and Katoen, 2008]
- Timed temporal logics [Bouyer et al., 2017]

References I

Abdeddaïm, Y. and Maler, O. (2001).
Job-shop scheduling using timed automata.
In Berry, G., Comon, H., and Finkel, A., editors, *CAV*, volume 2102 of *LNCS*, pages 478–492. Springer.

Abdeddaïm, Y. and Masson, D. (2012).
Real-time scheduling of energy harvesting embedded systems with timed automata.
In *RTCSA*, pages 31–40. IEEE Computer Society.

Abdulla, P. A., Deneux, J., Ouaknine, J., Quaas, K., and Worrell, J. (2008).
Universality analysis for one-clock timed automata.
Fundamenta Informaticae, 89(4):419–450.

Aceto, L., Bouyer, P., Burgueño, A., and Larsen, K. G. (2003).
The power of reachability testing for timed automata.
Theoretical Computer Science, 300(1-3):411–475.

Aceto, L., Burgueño, A., and Larsen, K. G. (1998).
Model checking via reachability testing for timed automata.
In Steffen, B., editor, *TACAS*, volume 1384 of *LNCS*, pages 263–280. Springer.

Adbeddaïm, Y., Asarin, E., and Maler, O. (2006).
Scheduling with timed automata.
Theoretical Computer Science, 354(2):272–300.

Alur, R., Courcoubetis, C., and Dill, D. L. (1993a).
Model-checking in dense real-time.
Information and Computation, 104(1):2–34.

References II

Alur, R. and Dill, D. L. (1994).
A theory of timed automata.
Theoretical Computer Science, 126(2):183–235.

Alur, R., Feder, T., and Henzinger, T. A. (1996).
The benefits of relaxing punctuality.
Journal of the ACM, 43(1):116–146.

Alur, R., Henzinger, T. A., and Vardi, M. Y. (1993b).
Parametric real-time reasoning.
In Kosaraju, S. R., Johnson, D. S., and Aggarwal, A., editors, *STOC*, pages 592–601, New York, NY, USA. ACM.

Alur, R., La Torre, S., and Pappas, G. J. (2004).
Optimal paths in weighted timed automata.
Theoretical Computer Science, 318(3):297–322.

André, É., Fribourg, L., Kühne, U., and Soulat, R. (2012).
IMITATOR 2.5: A tool for analyzing robustness in scheduling problems.
In Giannakopoulou, D. and Méry, D., editors, *FM*, volume 7436 of *LNCS*, pages 33–36. Springer.

André, É., Lime, D., and Ramparison, M. (2019).
Parametric updates in parametric timed automata.
In Pérez, J. A. and Yoshida, N., editors, *FORTE*. Springer.
To appear.

References III

André, É. and Soulat, R. (2013).
The Inverse Method.
FOCUS Series in Computer Engineering and Information Technology. ISTE Ltd and John Wiley & Sons Inc. 176 pages.

Annichini, A., Bouajjani, A., and Sighireanu, M. (2001).
TReX: A tool for reachability analysis of complex systems.
In Berry, G., Comon, H., and Finkel, A., editors, *CAV*, volume 2102 of *LNCS*, pages 368–372. Springer.

Asarin, E., Mysore, V., Pnueli, A., and Schneider, G. (2012).
Low dimensional hybrid systems – decidable, undecidable, don't know.
Information and Computation, 211:138–159.

Bacci, G., Bouyer, P., Fahrenberg, U., Larsen, K. G., Markey, N., and Reynier, P. (2018).
Optimal and robust controller synthesis – using energy timed automata with uncertainty.
In Havelund, K., Peleska, J., Roscoe, B., and de Vink, E. P., editors, *FM*, volume 10951 of *LNCS*, pages 203–221. Springer.

Baier, C. and Katoen, J.-P. (2008).
Principles of Model Checking.
MIT Press.

Behrmann, G., Bouyer, P., Larsen, K. G., and Pelánek, R. (2006).
Lower and upper bounds in zone-based abstractions of timed automata.
International Journal on Software Tools for Technology Transfer, 8(3):204–215.

References IV

Behrmann, G., Fehnker, A., Hune, T., Larsen, K. G., Pettersson, P., Romijn, J., and Vaandrager, F. W. (2001).
Minimum-cost reachability for priced timed automata.
In Benedetto, M. D. D. and Sangiovanni-Vincentelli, A. L., editors, *HSCC*, volume 2034 of *LNCS*, pages 147–161. Springer.

Bengtsson, J. and Yi, W. (2003).
Timed automata: Semantics, algorithms and tools.
In Desel, J., Reisig, W., and Rozenberg, G., editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *LNCS*, pages 87–124. Springer.

Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, Ph. (2001).
Systems and Software Verification. Model-Checking, Techniques and Tools.
Springer.

Bérard, B., Cassez, F., Haddad, S., Lime, D., and Roux, O. H. (2005).
Comparison of the expressiveness of timed automata and time Petri nets.
In Pettersson, P. and Yi, W., editors, *FORMATS*, volume 3829 of *LNCS*, pages 211–225. Springer.

Bérard, B., Cassez, F., Haddad, S., Lime, D., and Roux, O. H. (2013).
The expressive power of time Petri nets.
Theoretical Computer Science, 474:1–20.

Bertrand, N., Bouyer, P., Brihaye, T., and Stainer, A. (2011).
Emptiness and universality problems in timed automata with positive frequency.
In Aceto, L., Henzinger, M., and Sgall, J., editors, *ICALP, Part II*, volume 6756 of *LNCS*, pages 246–257. Springer.

References V

Bouyer, P. (2003).
Untameable timed automata!
In Alt, H. and Habib, M., editors, *STACS*, volume 2607 of *LNCS*, pages 620–631. Springer.

Bouyer, P., Dufourd, C., Fleury, E., and Petit, A. (2004).
Updatable timed automata.
Theoretical Computer Science, 321(2-3):291–345.

Bouyer, P., Laroussinie, F., Markey, N., Ouaknine, J., and Worrell, J. (2017).
Timed temporal logics.
In Aceto, L., Bacci, G., Bacci, G., Ingólfsdóttir, A., Legay, A., and Mardare, R., editors, *Models, Algorithms, Logics and Tools*, volume 10460 of *LNCS*, pages 211–230. Springer.

Bouyer, P., Markey, N., and Sankur, O. (2013).
Robustness in timed automata.
In Abdulla, P. A. and Potapov, I., editors, *RP*, volume 8169 of *LNCS*, pages 1–18. Springer.
Invited paper.

Bowman, H. and Gómez, R. (2006).
How to stop time stopping.
Formal Aspects of Computing, 18(4):459–493.

Bozga, M., Hou, J., Maler, O., and Yovine, S. (2002).
Verification of asynchronous circuits using timed automata.
Electronic Notes in Theoretical Computer Science, 65(6):47–59.

References VI

Cassez, F. and Larsen, K. G. (2000).
The impressive power of stopwatches.
In Palamidessi, C., editor, *CONCUR*, volume 1877 of *LNCS*, pages 138–152. Springer.

Chevallier, R., Encrenaz-Tiphène, E., Fribourg, L., and Xu, W. (2009).
Timed verification of the generic architecture of a memory circuit using parametric timed automata.
Formal Methods in System Design, 34(1):59–81.

D'Argenio, P. R., Katoen, J.-P., Ruys, T. C., and Tretmans, J. (1997).
The bounded retransmission protocol must be on time!
In Brinksma, E., editor, *TACAS*, volume 1217 of *LNCS*, pages 416–431. Springer.

De Wulf, M., Doyen, L., Markey, N., and Raskin, J. (2004).
Robustness and implementability of timed automata.
In Lakhnech, Y. and Yovine, S., editors, *FORMATS and FTRTFT*, volume 3253 of *LNCS*, pages 118–133. Springer.

Fehnker, A. (1999).
Scheduling a steel plant with timed automata.
In *RTCSA*, pages 280–286. IEEE Computer Society.

Fersman, E., Krcál, P., Pettersson, P., and Yi, W. (2007).
Task automata: Schedulability, decidability and undecidability.
Information and Computation, 205(8):1149–1172.

Gómez, R. and Bowman, H. (2007).
Efficient detection of Zeno runs in timed automata.
In Raskin, J. and Thiagarajan, P. S., editors, *FORMATS*, volume 4763 of *LNCS*, pages 195–210. Springer.

References VII

Havelund, K., Skou, A., Larsen, K. G., and Lund, K. (1997).
Formal modeling and analysis of an audio/video protocol: an industrial case study using UPPAAL.
In *RTSS*, pages 2–13. IEEE Computer Society.

Henzinger, T. A. (1996).
The theory of hybrid automata.
In Vardi, M. Y. and Clarke, E. M., editors, *LiCS*, pages 278–292. IEEE Computer Society.

Henzinger, T. A., Ho, P.-H., and Wong-Toi, H. (1997).
HyTech: A model checker for hybrid systems.
Software Tools for Technology Transfer, 1:110–122.

Henzinger, T. A., Kopke, P. W., and Wong-Toi, H. (1995).
The expressive power of clocks.
In Fülöp, Z. and Gécseg, F., editors, *ICALP*, volume 944 of *LNCS*, pages 417–428. Springer.

Henzinger, T. A., Nicollin, X., Sifakis, J., and Yovine, S. (1994).
Symbolic model checking for real-time systems.
Information and Computation, 111(2):193–244.

Herbreteau, F., Srivathsan, B., and Walukiewicz, I. (2012).
Efficient emptiness check for timed Büchi automata.
Formal Methods in System Design, 40(2):122–146.

Herbreteau, F., Srivathsan, B., and Walukiewicz, I. (2013).
Lazy abstractions for timed automata.
In Sharygina, N. and Veith, H., editors, *CAV*, volume 8044 of *LNCS*, pages 990–1005. Springer.

References VIII

Koymans, R. (1990).
Specifying real-time properties with metric temporal logic.
Real-Time Systems, 2(4):255–299.

Kwiatkowska, M. Z., Norman, G., Segala, R., and Sproston, J. (2002).
Automatic verification of real-time systems with discrete probability distributions.
Theoretical Computer Science, 282(1):101–150.

Laarman, A., Olesen, M. C., Dalsgaard, A. E., Larsen, K. G., and Van De Pol, J. (2013).
Multi-core emptiness checking of timed Büchi automata using inclusion abstraction.
In Sharygina, N. and Veith, H., editors, *CAV*, volume 8044 of *LNCS*, pages 968–983, Heidelberg, Germany. Springer.

Larsen, K. G., Lorber, F., and Nielsen, B. (2018).
20 years of UPPAAL enabled industrial model-based validation and beyond.
In Margaria, T. and Steffen, B., editors, *ISoLA, Part IV*, volume 11247 of *LNCS*, pages 212–229. Springer.

Larsen, K. G., Pettersson, P., and Yi, W. (1997).
UPPAAL in a nutshell.
International Journal on Software Tools for Technology Transfer, 1(1-2):134–152.

Legay, A., Delahaye, B., and Bensalem, S. (2010).
Statistical model checking: An overview.
In Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G. J., Rosu, G., Sokolsky, O., and Tillmann, N., editors, *RV*, volume 6418 of *LNCS*, pages 122–135. Springer.

References IX

Lime, D., Roux, O. H., Seidner, C., and Traonouez, L.-M. (2009).
Romeo: A parametric model-checker for Petri nets with stopwatches.
In Kowalewski, S. and Philippou, A., editors, *TACAS*, volume 5505 of *LNCS*, pages 54–57. Springer.

Maler, O. and Nickovic, D. (2004).
Monitoring temporal properties of continuous signals.
In Lakhnech, Y. and Yovine, S., editors, *FORMATS and FTRTFT*, volume 3253 of *LNCS*, pages 152–166. Springer.

Markey, N. (2011).
Robustness in real-time systems.
In Bate, I. and Passerone, R., editors, *SIES*, pages 28–34. IEEE Computer Society Press.

Merlin, P. M. (1974).
A study of the recoverability of computing systems.
PhD thesis, University of California, Irvine, CA, USA.

Ouaknine, J. and Worrell, J. (2003).
Universality and language inclusion for open and closed timed automata.
In Maler, O. and Pnueli, A., editors, *HSCC*, volume 2623 of *LNCS*, pages 375–388. Springer.

Ouaknine, J. and Worrell, J. (2004).
On the language inclusion problem for timed automata: Closing a decidability gap.
In *LICS*, pages 54–63. IEEE Computer Society.

Sankur, O., Bouyer, P., Markey, N., and Reynier, P. (2013).
Robust controller synthesis in timed automata.
volume 8052 of *LNCS*, pages 546–560. Springer.

References X

Schivo, S., Scholma, J., Wanders, B., Camacho, R. A. U., van der Vet, P. E., Karperien, M., Langerak, R., van de Pol, J., and Post, J. N. (2014).
Modeling biological pathway dynamics with timed automata.
IEEE Journal of Biomedical and Health Informatics, 18(3):832–839.

Simons, D. P. L. and Stoelinga, M. (2001).
Mechanical verification of the IEEE 1394a root contention protocol using Uppaal2k.
International Journal on Software Tools for Technology Transfer, 3(4):469–485.

Srba, J. (2008).
Comparing the expressiveness of timed automata and timed extensions of Petri nets.
In Cassez, F. and Jard, C., editors, *FORMATS*, volume 5215 of *LNCS*, pages 15–32. Springer.

Sun, J., Liu, Y., Dong, J. S., Liu, Y., Shi, L., and André, É. (2013).
Modeling and verifying hierarchical real-time systems using Stateful Timed CSP.
ACM Transactions on Software Engineering and Methodology, 22(1):3.1–3.29.

Sun, J., Liu, Y., Dong, J. S., and Pang, J. (2009a).
PAT: Towards flexible verification under fairness.
In Bouajjani, A. and Maler, O., editors, *CAV*, volume 5643 of *LNCS*, pages 709–714. Springer.

Sun, J., Liu, Y., Dong, J. S., and Zhang, X. (2009b).
Verifying stateful timed CSP using implicit clocks and zone abstraction.
In Breitman, K. K. and Cavalcanti, A., editors, *ICFEM*, volume 5885 of *LNCS*, pages 581–600. Springer.

References XI

Tripakis, S. (1999).
Verifying progress in timed systems.
In Katoen, J., editor, *ARTS*, volume 1601 of *LNCS*, pages 299–314. Springer.

Tripakis, S. and Yovine, S. (1998).
Verification of the fast reservation protocol with delayed transmission using the tool Kronos.
In *RTAS*, pages 165–170. IEEE Computer Society.

Tripakis, S., Yovine, S., and Bouajjani, A. (2005).
Checking timed büchi automata emptiness efficiently.
Formal Methods in System Design, 26(3):267–292.

Waga, M., Akazaki, T., and Hasuo, I. (2016).
A Boyer-Moore type algorithm for timed pattern matching.
In Fränzle, M. and Markey, N., editors, *FORMATS*, volume 9884 of *LNCS*, pages 121–139. Springer.

Waga, M., Hasuo, I., and Suenaga, K. (2018).
MONAA: A tool for timed pattern matching with automata-based acceleration.
In *MT@CPSWeek*, pages 14–15. IEEE.

Wang, T., Sun, J., Wang, X., Liu, Y., Si, Y., Dong, J. S., Yang, X., and Li, X. (2015).
A systematic study on explicit-state non-Zenoness checking for timed automata.
IEEE Transactions on Software Engineering, 41(1):3–18.

Wang, X., Sun, J., Wang, T., and Qin, S. (2017).
Language inclusion checking of timed automata with non-Zenoness.
IEEE Transactions on Software Engineering, 43(11):995–1008.

References XII

Yovine, S. (1997).
Kronos: A verification tool for real-time systems.
International Journal on Software Tools for Technology Transfer, 1(1-2):123–133.

Zhang, Z., Nielsen, B., and Larsen, K. G. (2016).
Time optimal reachability analysis using swarm verification.
In Ossowski, S., editor, *SAC*, pages 1634–1640. ACM.

License

Source of the graphics

Titre: Clock 256
Auteur: Everaldo Coelho
Source: https://commons.wikimedia.org/wiki/File:Clock_256.png
Licence: GNU LGPL

Title: Smiley green alien big eyes (aaah)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

Title: Smiley green alien big eyes (cry)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

License of this document

Author: Étienne André

(LaTeX source available on demand)