Université Paris 13
Institut Galilée
Master 2 PLS

SITH – Partie 2.1
Systèmes temporisés

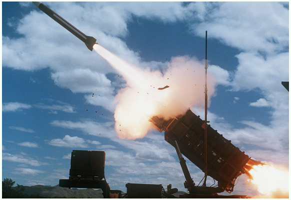
Étienne André
Etienne.Andre (à) univ-paris13.fr

Version : January 26, 2018 (diapositives à trous pour Web)

Partie 2.1: SITH – Plan

Context: Verifying complex timed systems

- Need for early bug detection
  - Bugs discovered when final testing: expensive
  - ⇝ Need for a thorough specification and verification phase



The Therac-25 radiation therapy machine (1/2)

- Radiation therapy machine used in the 1980s
- Involved in accidents between 1985 and 1987, in which patients were given massive overdoses of radiation
  - Approximately 100 times the intended dose!
  - Numerous causes, including race condition

The Therac-25 radiation therapy machine (1/2)

- Radiation therapy machine used in the 1980s
- Involved in accidents between 1985 and 1987, in which patients were given massive overdoses of radiation
  - Approximately 100 times the intended dose!
  - Numerous causes, including race condition

"The failure only occurred when a particular nonstandard sequence of keystrokes was entered on the VT-100 terminal which controlled the PDP-11 computer: an `X` *to (erroneously) select 25MV photon mode followed by ↑,* `E` *to (correctly) select 25 MeV Electron mode, then* `Enter`*, all within eight seconds."*

The Therac-25 radiation therapy machine (2/2)

The testing engineers could obviously not detect this strange (and quick!) sequence leading to the failure.

The Therac-25 radiation therapy machine (2/2)

The testing engineers could obviously not detect this strange (and quick!) sequence leading to the failure.

Limits of testing

This case illustrates the difficulty of bug detection without formal methods.

Bugs can be difficult to find

. . . and can have dramatic consequences for critical systems:

- health-related devices
- aeronautics and aerospace transportation
- smart homes and smart cities
- military devices
- etc.

Bugs can be difficult to find

...and can have dramatic consequences for critical systems:

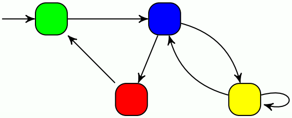
- health-related devices
- aeronautics and aerospace transportation
- smart homes and smart cities
- military devices
- etc.

Hence, high need for formal verification

Plan: Finite-State Automata

Model checking concurrent systems

- Use formal methods [Baier and Katoen, 2008]



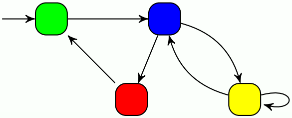
A model of the system

 is unreachable

A property to be satisfied

Model checking concurrent systems

- Use formal methods [Baier and Katoen, 2008]



A model of the system

$\overset{?}{\models}$

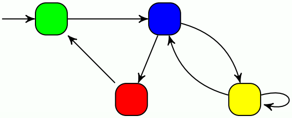
 is unreachable

A property to be satisfied

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

Model checking concurrent systems

- Use formal methods [Baier and Katoen, 2008]



A model of the system

$\overset{?}{\models}$

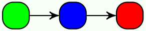
 is unreachable

A property to be satisfied

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

Yes

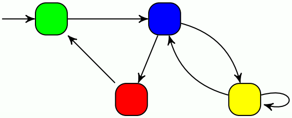
No



Counterexample

Model checking concurrent systems

- Use formal methods [Baier and Katoen, 2008]



$\overset{?}{\models}$

 is unreachable

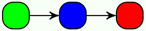
A model of the system

A property to be satisfied

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

Yes

No



Counterexample

Turing award (2007) to Edmund M. Clarke, Allen Emerson and Joseph Sifakis

Transition systems

Définition (Transition system)

A *transition system (TS)* is a tuple $TS = (S, \Sigma, S_I, S_F, \Rightarrow)$, where

- S is a set of states;
- Σ is an alphabet of events;
- $S_I \subseteq S$ is a set of initial states;
- $S_F \subseteq S$ is a set of final (or accepting) states; and,
- $\Rightarrow : S \times \Sigma \to 2^S$ is a transition relation.

Usually, we write $s_1 \stackrel{a}{\Longrightarrow} s_2$ when $(s_1, a, s_2) \in \Rightarrow$.

Finite-state automata

Définition (Finite automaton)

A *Finite automaton (FA)* $FA = (L, \Sigma, l_I, L_F, \rightarrow)$ is a tuple where

- L is a finite set of locations;
- Σ is a finite set of actions;
- $l_I \in L$ is the initial location;
- $L_F \subseteq L$ is a set of final (or accepting) locations;
- $\rightarrow : L \times \Sigma \rightarrow 2^L$ is a transition relation.

Usually, we write $l_1 \xrightarrow{a} l_2$ when $(l_1, a, l_2) \in \rightarrow$.

Example 1

$FA = (L, \Sigma, l_I, L_F, \rightarrow)$, with

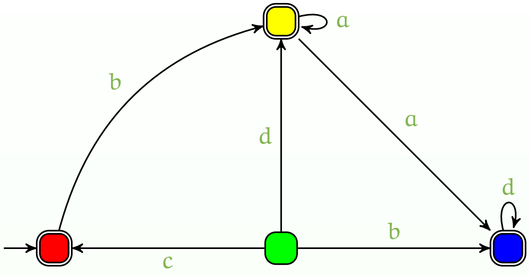
- $L = \{l_1, l_2, l_3\}$
- $\Sigma = \{a, b, c, d\}$
- $l_I = l_1$
- $L_F = \{l_2\}$
- $\rightarrow = \{(l_1, a, l_1), (l_1, b, l_2), (l_2, c, l_1), (l_2, d, l_2), (l_3, b, l_2)\}$

Example 1

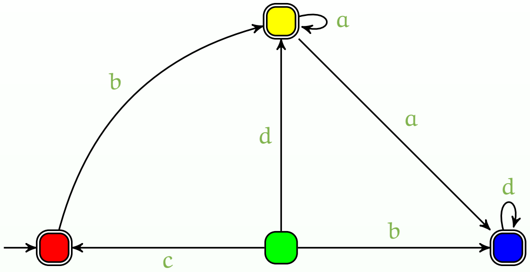
$FA = (L, \Sigma, l_I, L_F, \rightarrow)$, with

- $L = \{l_1, l_2, l_3\}$
- $\Sigma = \{a, b, c, d\}$
- $l_I = l_1$
- $L_F = \{l_2\}$
- $\rightarrow = \{(l_1, a, l_1), (l_1, b, l_2), (l_2, c, l_1), (l_2, d, l_2), (l_3, b, l_2)\}$

Example 2



Example 2



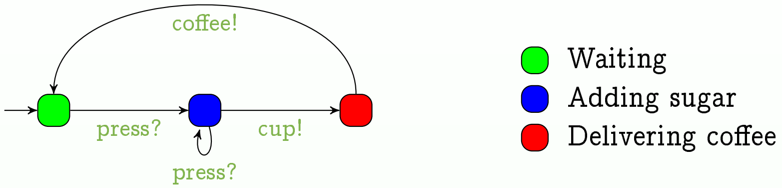
Semantics of finite automata

Définition (Semantics of finite automata)

Let $FA = (L, \Sigma, l_I, L_F, \Rightarrow)$ be a Finite Automaton.
The semantics of FA is the transition system $TS = (S, \Sigma, S_I, S_F, \Rightarrow)$, with

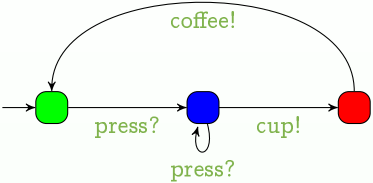
- $S = L$;
- Σ the same;
- $S_I = \{l_I\}$;
- $S_F = L_F$; and,
- $\Rightarrow = \rightarrow$.

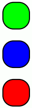
A coffee machine $\mathcal{A}_C$



- Example of runs
  - Coffee with no sugar

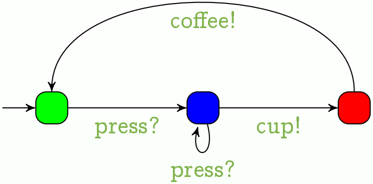
A coffee machine $\mathcal{A}_C$

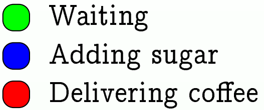




- Example of runs
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

A coffee machine $\mathcal{A}_C$





- Example of runs
  - Coffee with no sugar
  - Coffee with 2 doses of sugar
  - And so on

A coffee drinker (1/2)

- Specify a coffee drinker automaton $\mathcal{A}_{\mathrm{D1}}$ that performs forever the following actions:
  1. press the button once
  2. place the cup
  3. wait for the coffee
  4. drink the coffee
  5. put the cup to the washing machine

A coffee drinker (2/2)

- Specify a coffee drinker automaton $\mathcal{A}_{D2}$ that works just as $\mathcal{A}_{D1}$ except that (s)he can nondeterministically ask for 0, 1 or 2 doses of sugar.

A washing machine

- Specify a washing machine automaton $\mathcal{A}_W$ that accepts up to 5 cups, and washes all cups when the machine is full.

Systems as components

Often, a complex system is made of components or modules

Components can interact with each other:

- using strong synchronization
- using shared variables
- using one-to-one synchronization
- in an interleaving manner

Here, we show that FAs can be composed easily using strong synchronization on actions.

Composition of finite automata

$FA_1 = (L_1, \Sigma_1, (l_I)_1, (L_F)_1, \rightarrow_1)$
$FA_2 = (L_2, \Sigma_2, (l_I)_2, (L_F)_2, \rightarrow_2)$

Then we define $FA_1 \parallel FA_2$ as

Composition of finite automata: Example 1

Draw the automaton composed of the automata $\mathcal{A}_{\mathrm{C}} \parallel \mathcal{A}_{\mathrm{D1}}$

Composition of finite automata: Example 2

Draw the automaton composed of the automata $\mathcal{A}_{\text{C}} \parallel \mathcal{A}_{\text{D2}}$

Composition of finite automata: Example 3

Start to draw the automaton composed of the automata $\mathcal{A}_{\mathrm{C}} \parallel \mathcal{A}_{\mathrm{D2}} \parallel \mathcal{A}_{W}$. What do you notice?

Temporal logics

Modal logics expressing timing information over a set of atomic propositions, and can be used to formally verify a model.

Some temporal logics:

- LTL (Linear Temporal Logic) [Pnueli, 1977]
- CTL (Computation Tree Logic) [Clarke and Emerson, 1982]
- MITL
- CTL*
- μ-calculus

Temporal logics

Modal logics expressing timing information over a set of atomic propositions, and can be used to formally verify a model.

Some temporal logics:

- LTL (Linear Temporal Logic) [Pnueli, 1977]
- CTL (Computation Tree Logic) [Clarke and Emerson, 1982]
- MITL
- CTL*
- μ-calculus

Warning

Temporal logics express the ordering between events over time, but do not (in general) contain timed information.

LTL (Linear Temporal Logic) [Pnueli, 1977]

LTL expresses formulas about the future of one path, using a set of atomic propositions AP

Minimal syntax:

$$\varphi ::= p \in AP \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \varphi$$

Explanation and additional operators:

$p \in AP$	atomic proposition	
X	Next	"at the next step"
U	Until	
F	Finally (eventually)	"now or sometime later"
G	Globally	"now and anytime later"
R	Release	
W	Weak until	

LTL: Examples

Express in LTL the following properties:

- "The plane will never crash" (safety property)

LTL: Examples

Express in LTL the following properties:

- "The plane will never crash" (safety property)

LTL: Examples

Express in LTL the following properties:

- "The plane will never crash" (safety property)
- "I will eventually get a job" (liveness property)

LTL: Examples

Express in LTL the following properties:

- “The plane will never crash” (safety property)
- “I will eventually get a job” (liveness property)

LTL: Examples

Express in LTL the following properties:

- "The plane will never crash" (safety property)
- "I will eventually get a job" (liveness property)
- "Every time I ask a question, the teacher will eventually answer me" (liveness property)

LTL: Examples

Express in LTL the following properties:

- "The plane will never crash" (safety property)
- "I will eventually get a job" (liveness property)
- "Every time I ask a question, the teacher will eventually answer me" (liveness property)

LTL: Examples

Express in LTL the following properties:

- "The plane will never crash" (safety property)
- "I will eventually get a job" (liveness property)
- "Every time I ask a question, the teacher will eventually answer me" (liveness property)
- "If I ask for food infinitely often, then I will get food infinitely often" (strong fairness property)

LTL: Examples

Express in LTL the following properties:

- “The plane will never crash” (safety property)
- “I will eventually get a job” (liveness property)
- “Every time I ask a question, the teacher will eventually answer me” (liveness property)
- “If I ask for food infinitely often, then I will get food infinitely often” (strong fairness property)

CTL (Computation Tree Logic) [Clarke and Emerson, 1982]

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

Quantifiers over paths:

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

Quantifiers over states:

$$\psi ::= \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi$$

Explanation:

E	Exists	"along some of the future paths"
A	ForAll	"along all the future paths"

CTL: More on quantifiers

A path quantifier must always be followed by a state quantifier.

Some useful combinations:

CTL: More on quantifiers

A path quantifier must always be followed by a state quantifier.

Some useful combinations:

"there exists a path for which the next state is. . . "

CTL: More on quantifiers

A path quantifier must always be followed by a state quantifier.

Some useful combinations:

"there exists a path for which the next state is..."

"for all possible paths, the next state is..."

CTL: More on quantifiers

A path quantifier must always be followed by a state quantifier.

Some useful combinations:

"there exists a path for which the next state is. . . "
"for all possible paths, the next state is. . . "
"it is possible that eventually. . . "

CTL: More on quantifiers

A path quantifier must always be followed by a state quantifier.

Some useful combinations:

- "there exists a path for which the next state is…"
- "for all possible paths, the next state is…"
- "it is possible that eventually…"
- "in any case, eventually…"

CTL: More on quantifiers

A path quantifier must always be followed by a state quantifier.

Some useful combinations:

“there exists a path for which the next state is. . . ”

“for all possible paths, the next state is. . . ”

“it is possible that eventually. . . ”

“in any case, eventually. . . ”

“in any case, for all states. . . ”

CTL: More on quantifiers

A path quantifier must always be followed by a state quantifier.

Some useful combinations:

"there exists a path for which the next state is. . . "

"for all possible paths, the next state is. . . "

"it is possible that eventually. . . "

"in any case, eventually. . . "

"in any case, for all states. . . "

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)
- "I may eventually get a job" (reachability property)

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)
- "I may eventually get a job" (reachability property)

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)
- "I may eventually get a job" (reachability property)
- "I may love you for the rest of my life"

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)
- "I may eventually get a job" (reachability property)
- "I may love you for the rest of my life"

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)
- "I may eventually get a job" (reachability property)
- "I may love you for the rest of my life"
- "It can always happen that suddenly I discover formal methods and then I may use them for the rest of time"

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)
- "I may eventually get a job" (reachability property)
- "I may love you for the rest of my life"
- "It can always happen that suddenly I discover formal methods and then I may use them for the rest of time"

CTL: Back to the coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

- "After the button is pressed, a coffee is always eventually delivered."

CTL: Back to the coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

- "After the button is pressed, a coffee is always eventually delivered."

CTL: Back to the coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

- "After the button is pressed, a coffee is always eventually delivered."

CTL: Back to the coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

- "After the button is pressed, a coffee is always eventually delivered."
- "After the button is pressed, there exists an execution such that a coffee is eventually delivered."

CTL: Back to the coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

- "After the button is pressed, a coffee is always eventually delivered."

- "After the button is pressed, there exists an execution such that a coffee is eventually delivered."

CTL: Back to the coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

- "After the button is pressed, a coffee is always eventually delivered."
- "After the button is pressed, there exists an execution such that a coffee is eventually delivered."
- "Once the cup is delivered, coffee will come next."

CTL: Back to the coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

- "After the button is pressed, a coffee is always eventually delivered."
- "After the button is pressed, there exists an execution such that a coffee is eventually delivered."
- "Once the cup is delivered, coffee will come next."

CTL: Back to the coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

- "After the button is pressed, a coffee is always eventually delivered."
- "After the button is pressed, there exists an execution such that a coffee is eventually delivered."
- "Once the cup is delivered, coffee will come next."
- "It is possible to get a coffee with 2 doses of sugar."

CTL: Back to the coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

- "After the button is pressed, a coffee is always eventually delivered."
- "After the button is pressed, there exists an execution such that a coffee is eventually delivered."
- "Once the cup is delivered, coffee will come next."
- "It is possible to get a coffee with 2 doses of sugar."

The reachability problem

The reachability problem

Given FA, given a given location $\mathfrak{l}$, does there exist a path from an initial location of FA leading to $\mathfrak{l}$?

Applications:

- Is there an execution of the therapy machine leading to the delivery of high radiations?
- Can the coffee machine deliver a coffee with five doses of sugar?

Forward reachability

Let S be the set of all reachable states.
Given a subset $S' \subseteq S$ of states, which states of S are reachable from S' in just one step?

Forward reachability

Let S be the set of all reachable states.
Given a subset $S' \subseteq S$ of states, which states of S are reachable from S' in just one step?

Définition (Post)

Given a set $S' \subseteq S$ of states, we define *Post* as:

$$Post(S') = \{s \in S \mid$$

Forward reachability

Let S be the set of all reachable states.
Given a subset $S' \subseteq S$ of states, which states of S are reachable from S' in just one step?

Définition (Post)

Given a set $S' \subseteq S$ of states, we define $Post$ as:

$$Post(S') = \{s \in S \mid$$

By extension, we write $Post^*(S')$ for the set of all states reachable from states of S'.

Forward reachability: Algorithm

Algorithm *isReachable*(TS, S_I, S_F)

input : Set S_I of initial states, set S_F of final states

output : `true` if S_F is reachable from S_I, `false` otherwise

$S \leftarrow S_I$; $i \leftarrow 0$;

Forward reachability: Algorithm

Algorithm $isReachable(TS, S_I, S_F)$

input : Set S_I of initial states, set S_F of final states

output : `true` if S_F is reachable from S_I, `false` otherwise

```
S ← S_I ; i ← 0 ;
repeat
    if S ∩ S_F ≠ ∅ then
        ;
```

Forward reachability: Algorithm

Algorithm *isReachable*(TS, S_I, S_F)

input : Set S_I of initial states, set S_F of final states

output : `true` if S_F is reachable from S_I, `false` otherwise

```
S ← S_I ; i ← 0 ;
repeat
    if S ∩ S_F ≠ ∅ then
        ;
    S ←  ;
    i ← i + 1 ;
```

Forward reachability: Algorithm

Algorithm *isReachable*(TS, S_I, S_F)

input : Set S_I of initial states, set S_F of final states

output : `true` if S_F is reachable from S_I, `false` otherwise

```
S ← S_I ; i ← 0 ;
repeat
    if S ∩ S_F ≠ ∅ then
        ;
    S ←          ;
    i ← i + 1 ;
until            ;
```

Forward reachability: Algorithm

Algorithm *isReachable*(TS, S_I, S_F)

input : Set S_I of initial states, set S_F of final states

output : `true` if S_F is reachable from S_I, `false` otherwise

$S \leftarrow S_I$; $i \leftarrow 0$;
repeat
  if $S \cap S_F \neq \emptyset$ **then**
    ;
  $S \leftarrow$;
  $i \leftarrow i + 1$;
until ;
return ;

Forward reachability: Applications

Verifying properties using observers

An observer is an automaton that observes the system behavior

- It synchronizes with other automata's actions
- It must be non-blocking (see example on the white board)
- 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)

Observers for the coffee machine (1/2)

Design an observer for the coffee machine and the drinker verifying that it is possible to order a coffee with at least one dose of sugar.
(. . . and check the validity of the property)

Observers for the coffee machine (2/2)

Design an observer for the coffee machine and the drinker verifying that whenever the coffee comes, the cup was not put to the washing machine before.

(. . . and check the validity of the property)

Plan: Timed Automata

Beyond finite state automata

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

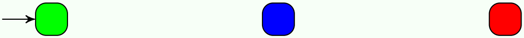
- Executions, sequence of actions
- Modular definitions (parallelism)
- Powerful checking (reachability, safety, liveness. . .)

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")

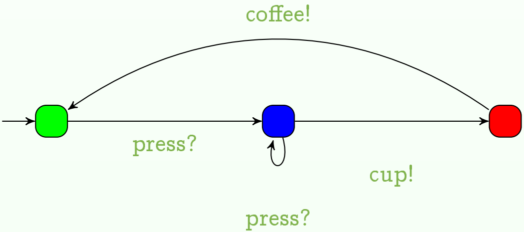
Timed automaton (TA)

- Finite state automaton (sets of locations)



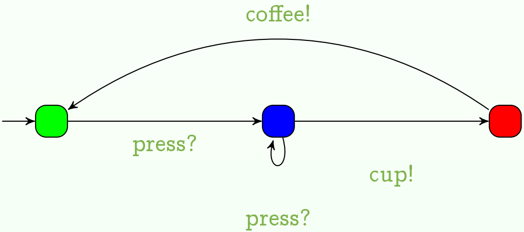
Timed automaton (TA)

- Finite state automaton (sets of locations and actions)



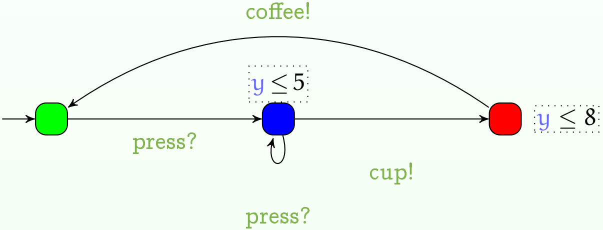
Timed automaton (TA)

- 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



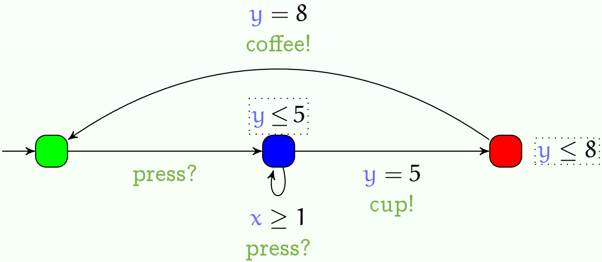
Timed automaton (TA)

- 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)

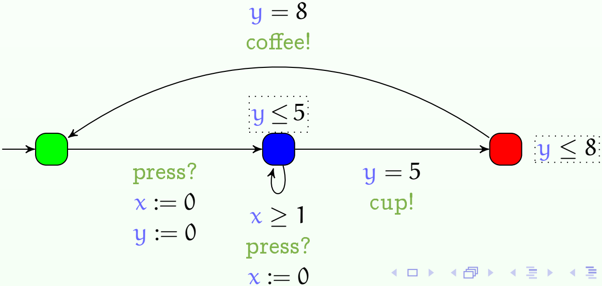
- Finite state automaton (sets of locations and actions) augmented with a set X of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants and guards
- Features
  - Location invariant: property to be verified to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

Définition (Timed automaton)

A *timed automaton (TA)* $\mathcal{A}$ is a 6-tuple of the form $\mathcal{A} = (L, \Sigma, l_I, X, Inv, \rightarrow)$, where

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

Example 1

Draw the TA $\mathcal{A} = (L, \Sigma, l_1, X, Inv, \rightarrow)$ such that

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

Example 2

Give the formal TA corresponding to the timed coffee machine.

Example 2

Give the formal TA corresponding to the timed coffee machine.

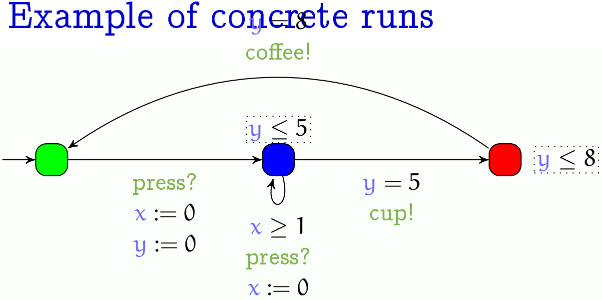
Concrete semantics of timed automata

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

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

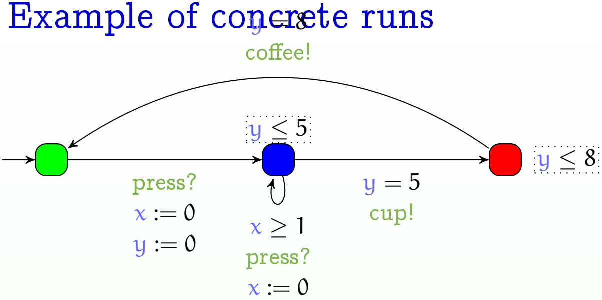
- Concrete run: alternating sequence of concrete states and actions or time elapse

Example of concrete runs



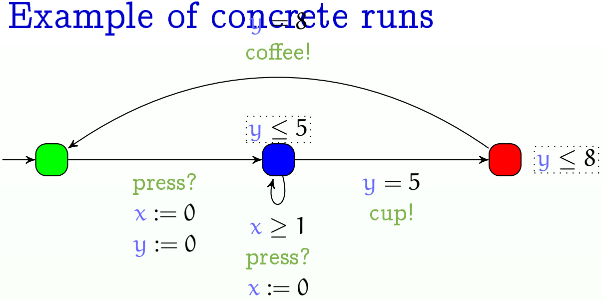
- Possible concrete runs for the coffee machine

Example of concrete runs



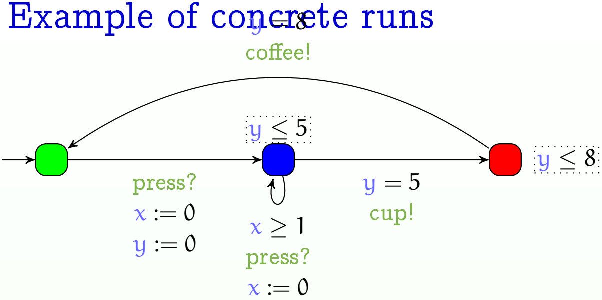
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



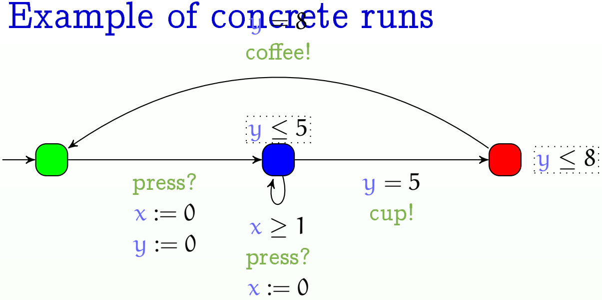
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



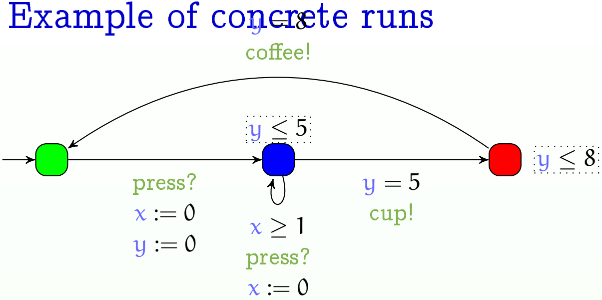
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



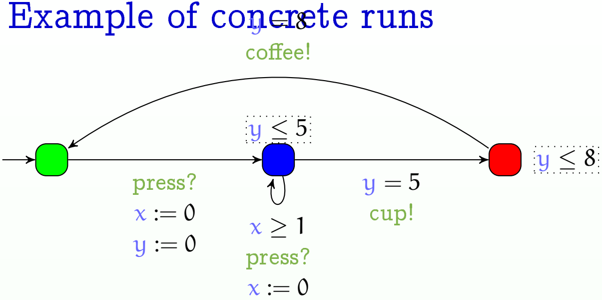
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



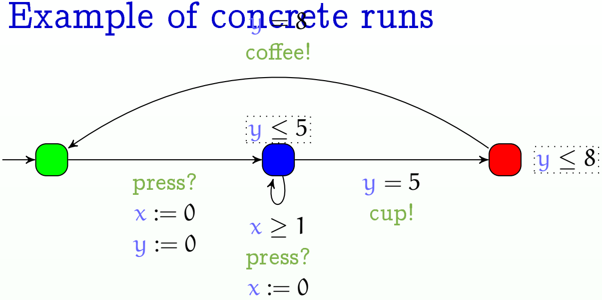
- Possible concrete runs for the coffee machine
    - Coffee with no sugar

Example of concrete runs



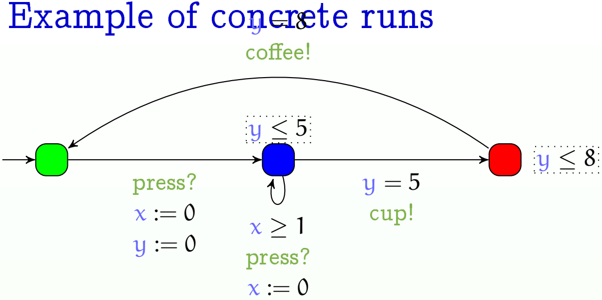
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

Example of concrete runs



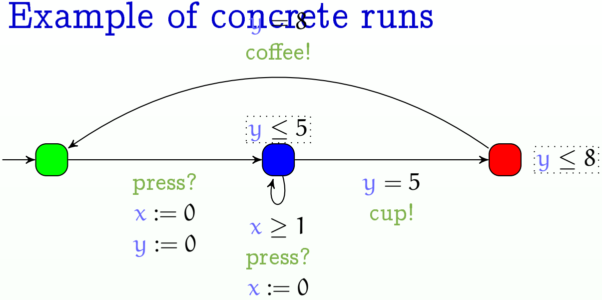
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



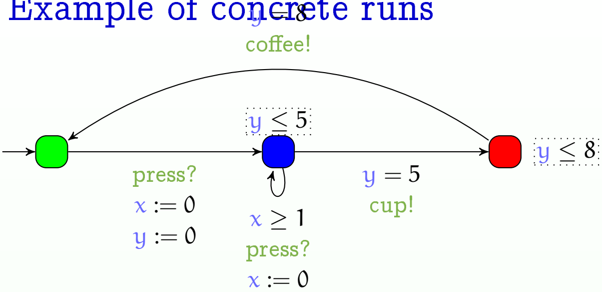
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



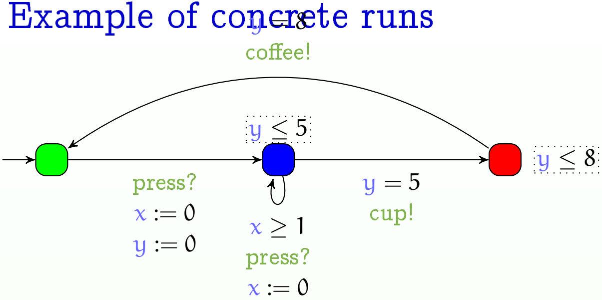
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



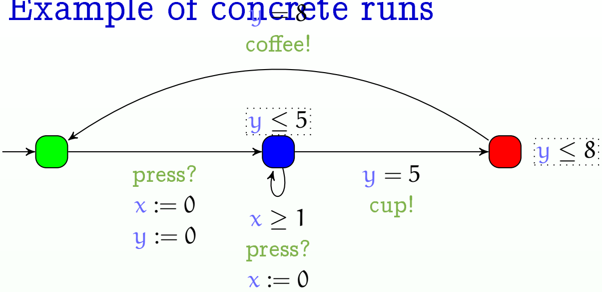
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



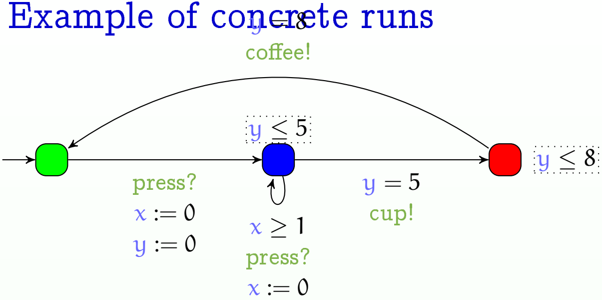
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



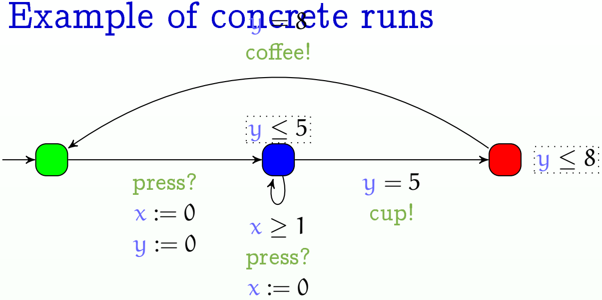
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



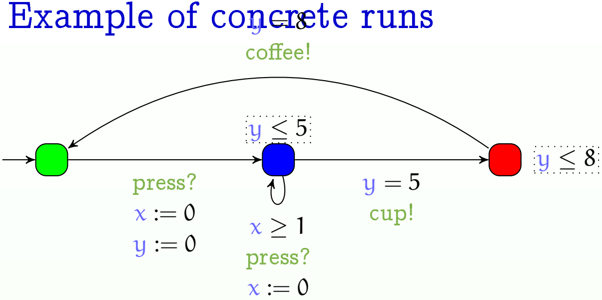
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Example of concrete runs



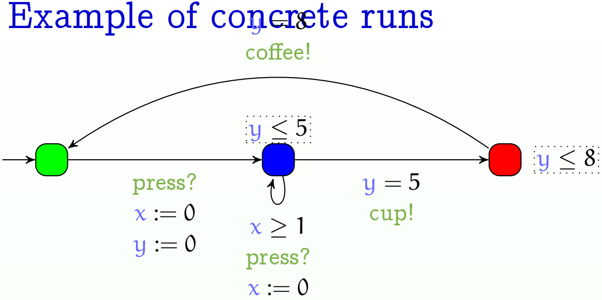
- Possible concrete runs for the coffee machine
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

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

Dense time

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

Abstract semantics of timed automata

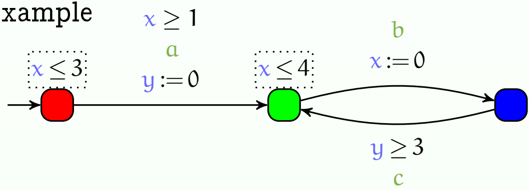
- Abstract state of a TA: pair (l, C), where
  - l is a location, and C is a constraint on the clocks ("zone")

Abstract semantics of timed automata

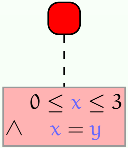
- Abstract state of a TA: pair (l, C), where
  - l is a location, and C is a constraint on the clocks ("zone")
- Abstract run: alternating sequence of abstract states and actions

Abstract semantics 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

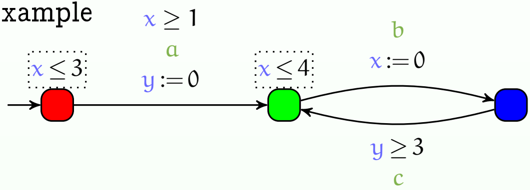


  - Possible abstract run for this TA

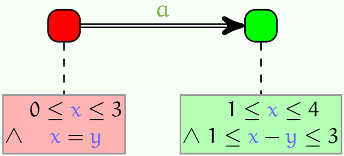


Abstract semantics of timed automata

- 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

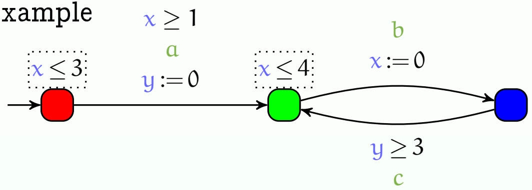


  - Possible abstract run for this TA

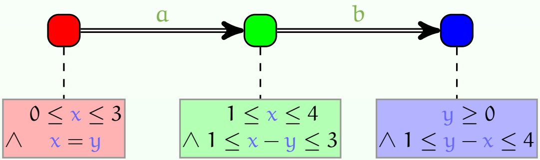


Abstract semantics of timed automata

- 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

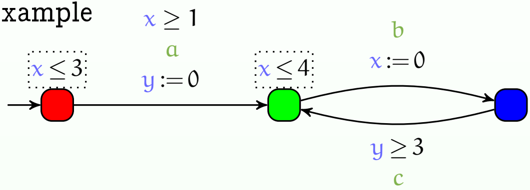


  - Possible abstract run for this TA

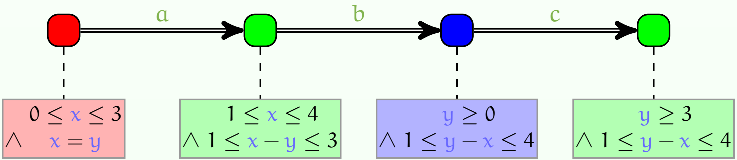


Abstract semantics of timed automata

- 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 for this TA



What is decidability?

Définition

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?

Définition

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 timed automaton, does there exist a run from the initial state to a given location l?"
- "given a context-free grammar, does it generate all strings?"
- "given a Turing machine, will it eventually halt?"

What is decidability?

Définition

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

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

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

What is decidability?

Définition

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

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

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

What is decidability?

Définition

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

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

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

What is decidability?

Définition

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

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

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

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

Decision problems for timed automata

The finiteness of the region automaton allows us to check properties

- ☺ Reachability of a location (PSPACE-complete)
- ☺ Liveness (Büchi conditions)

Some problems impossible to check using the region automaton (but still decidable)

- ☺ non-Zenoness emptiness check

Some undecidable problems (and hence impossible to check in general)

- ☹ universality of the timed language
- ☹ timed language inclusion

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., 2009]
- IMITATOR (also parametric timed automata) [André et al., 2012]

Timed temporal logics

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

Timed temporal logics

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

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

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

Quantifiers over paths:

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

Quantifiers over states:

$$\psi ::= \varphi U_I \varphi$$

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

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 before next year"

TCTL: Examples

- "Whatever happens, the plane will never crash in the next 10 minutes"
- "I may get a job before next 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 before next 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 before next 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."

- "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."

Remarks on timed automata

- Timed automata can be composed just as finite-state automata
- 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)
- Observers (both untimed and timed) can be used for timed automata

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

Exercise: An observer for timed automata

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.

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 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 et références

General References

- Systems and Software Verification (Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, Philippe Schnoebelen), Springer, 2001
- Principles of Model Checking (Christel Baier and Joost-Pieter Katoen), MIT Press, 2008

References I

Aceto, L., Bouyer, P., Burgueño, A., and Larsen, K. G. (1998a).
The power of reachability testing for timed automata.
In Arvind, V. and Ramanujam, R., editors, *FSTTCS'98*, volume 1530 of *Lecture Notes in Computer Science*, pages 245–256. Springer.

Aceto, L., Burgueño, A., and Larsen, K. G. (1998b).
Model checking via reachability testing for timed automata.
In *TACAS'98*, volume 1384 of *Lecture Notes in Computer Science*, pages 263–280. Springer.

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

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

References II

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, *Proceedings of the 18th International Symposium on Formal Methods (FM'12)*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer.

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'01*, volume 2102 of *Lecture Notes in Computer Science*, pages 368–372. Springer.

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

References III

Clarke, E. M. and Emerson, E. A. (1982).
Design and synthesis of synchronization skeletons using branching-time temporal logic.
In *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer.

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.

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.

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, *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, volume 5505 of *LNCS*, pages 54–57. Springer.

References IV

Markey, N. (2011).
Robustness in real-time systems.
In *Proceedings of the 6th IEEE International Symposium on Industrial Embedded Systems (SIES'11)*, pages 28–34, Västerås, Sweden. IEEE Computer Society Press.

Pnueli, A. (1977).
The temporal logic of programs.
In *FOCS*, pages 46–57. IEEE Computer Society.

Sun, J., Liu, Y., Dong, J. S., and Pang, J. (2009).
PAT: Towards flexible verification under fairness.
In *CAV'09*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer.

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

License

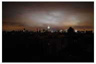
Source of the graphics (1)

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

Source of the graphics (2)



Title: Hurricane Sandy Blackout New York Skyline
Author: David Shankbone
Source: https://commons.wikimedia.org/wiki/File:Hurricane_Sandy_Blackout_New_York_Skyline.JPG
License: CC BY 3.0

Title: Sad mac
Author: Przemub
Source: https://commons.wikimedia.org/wiki/File:Sad_mac.png
License: Public domain

Title: Deepwater Horizon Offshore Drilling Platform on Fire
Author: ideum
Source: https://secure.flickr.com/photos/ideum/4711481781/
License: CC BY-SA 2.0

Title: DA-SC-88-01663
Author: imcomkorea
Source: https://secure.flickr.com/photos/imcomkorea/3017886760/
License: CC BY-NC-ND 2.0

Licence de ce document

Auteur : Étienne André

(Source LaTeX disponible sur demande)