Sokendai Lectures
Tokyo, Japan
物理情報システムのための形式手法

Timed model checking – Part 1

Finite-state automata

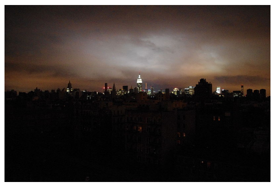
Étienne André
Etienne.Andre@univ-paris13.fr

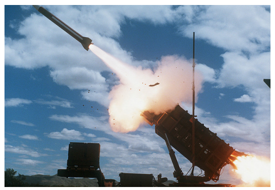
Version: April 27, 2019 (slides with empty spaces)

Partie 1: Untimed model checking – 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 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.

Limits of testing

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

Bugs can be difficult to find

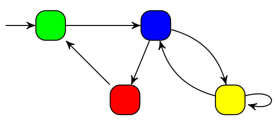
…and can have dramatic consequences for critical systems:

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

Hence, high need for formal verification

Model checking concurrent systems

- Use formal methods [Baier and Katoen, 2008]



A model of the system

$\overset{?}{\models}$

 is unreachable

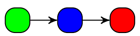
A property to be satisfied

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

Yes

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

No



Counterexample

Transition systems

Definition (Transition system)

A transition system (TS) is a tuple $\mathcal{TS} = (S, \Sigma, S_0, S_F, \Rightarrow)$, 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,
- $\Rightarrow : S \times \Sigma \rightarrow 2^S$ 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

Definition (Finite automaton)

A Finite automaton (FA) $FA = (L, \Sigma, \ell_0, L_F, E)$ is a tuple where

- L is a finite set of locations;
- Σ is a finite set of actions;
- $\ell_0 \in L$ is the initial location;
- $L_F \subseteq L$ is a set of final (or accepting) locations;
- $E : L \times \Sigma \to L$ is a transition relation.

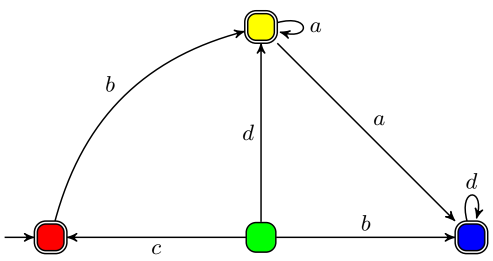
Usually, we write $l_1 \xrightarrow{a} l_2$ when $(l_1, a, l_2) \in E$.

Example 1

$FA = (L, \Sigma, \ell_0, L_F, E)$, with

- $L = \{l_1, l_2, l_3\}$
- $\Sigma = \{a, b, c, d\}$
- $\ell_0 = l_1$
- $L_F = \{l_2\}$
- $E = \{(l_1, a, l_1), (l_1, b, l_2), (l_2, c, l_1), (l_2, d, l_2), (l_3, b, l_2)\}$

Example 2



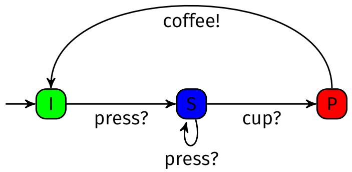
Semantics of finite automata

Definition (Semantics of finite automata)

Let $FA = (L, \Sigma, \ell_0, L_F, \Rightarrow)$ be a Finite Automaton.
The semantics of FA is the transition system $\mathcal{TS} = (S, \Sigma, S_0, S_F, \Rightarrow)$, with

- $S = L$;
- Σ the same;
- $S_0 = \{\ell_0\}$;
- $S_F = L_F$; and,
- $\Rightarrow = E$.

A coffee machine $\mathcal{A}_C$



- I Waiting
- S Adding sugar
- P Preparing coffee

- Examples 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 $\mathcal{A}_{D1}$ 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 cups to wash, and once 5 cups are placed into the washing machine, then the machine washes all cups.

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

$FA_1 = (L_1, \Sigma_1, (\ell_0)_1, (L_F)_1, E_1)$
$FA_2 = (L_2, \Sigma_2, (\ell_0)_2, (L_F)_2, E_2)$

Then we define $FA_1 \parallel FA_2$ as

Composition of finite automata: Example 1

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

Composition of finite automata: Example 2

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

Composition of finite automata: Example 3

Start to draw the automaton composed of the automata $\mathcal{A}_C \parallel \mathcal{A}_{D2} \parallel \mathcal{A}_W$. 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

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:

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

Explanation and additional operators:

$p \in AP$	atomic proposition	
$\mathbf{X}$	Next	“at the next step”
$\mathbf{U}$	Until	“ψ holds until φ holds”
$\mathbf{F}$	Finally (eventually)	“now or sometime later”
$\mathbf{G}$	Globally	“now and anytime later”
$\mathbf{R}$	Release	
$\mathbf{W}$	Weak until	“ψ holds either until φ holds or forever”

Illustrating LTL operators

$\mathbf{X}\varphi$

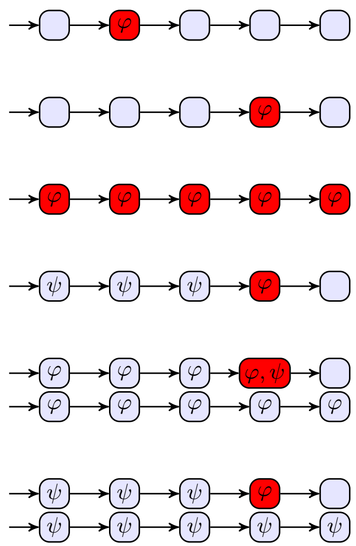
$\mathbf{F}\varphi$

$\mathbf{G}\varphi$

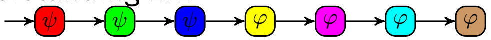
$\psi\mathbf{U}\varphi$

$\psi\mathbf{R}\varphi$

$\psi\mathbf{W}\varphi$



Exercise: Understanding LTL



On which states do the following properties hold?

φ	
$\mathbf{X}\varphi$	
$\mathbf{F}\varphi$	
$\mathbf{F}\psi$	
$\mathbf{G}\varphi$	
$\mathbf{GX}(\varphi \vee \psi)$	
$\mathbf{GF}\varphi$	
$\mathbf{GF}\psi$	
$\psi\mathbf{U}\varphi$	
$\varphi\mathbf{U}\psi$	
$\psi\mathbf{W}\varphi$	
$\varphi\mathbf{W}\psi$	
$\psi\mathbf{R}\varphi$	

Exercise: Specifying with LTL

Express in LTL the following properties:

- “The plane will never crash” (safety property)
- “I will eventually get a job” (liveness property)
- “Every day, I will be alive until the day of my death—unless I am immortal”
- “Every time I ask a question, the teacher will eventually answer me” (fairness 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:

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

Quantifiers over states:

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

Explanation:

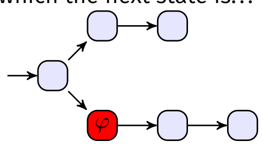
$\mathbf{E}$	Exists	“along some of the future paths”
$\mathbf{A}$	ForAll	“along all the future paths”

Illustrating combined quantifiers (1/2)

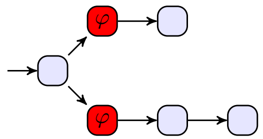
A path quantifier must always be followed by a state quantifier.

Some useful combinations:

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

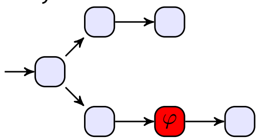


“for all possible paths, the next state is…”

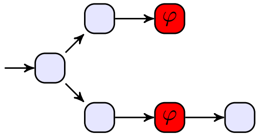


Illustrating combined quantifiers (2/2)

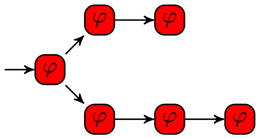
"it is possible that eventually..."



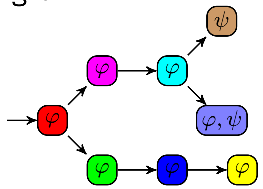
"in any case, eventually..."



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



Exercise: Understanding CTL



On which states do the following properties hold?

$\varphi \implies \psi$
$\mathbf{EX}\varphi$
$\mathbf{AX}\varphi$
$\mathbf{EF}\psi$
$\mathbf{AF}\psi$
$\mathbf{AX}(\varphi \wedge \psi)$
$\mathbf{EG}\varphi$
$\mathbf{AG}\varphi$
$\mathbf{A}\varphi\mathbf{U}\psi$

Exercise: specifying with CTL

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"

Exercise: CTL and 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 ℓ, does there exist a path from an initial location of FA leading to 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?

Definition (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'.

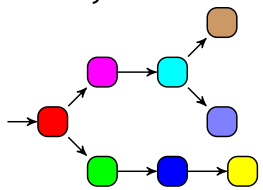
Forward reachability: Algorithm

Algorithm $isReachable(\mathcal{TS}, S_0, S_F)$

input : Set S_0 of initial states, set S_F of final states
output : `true` if S_F is reachable from S_0, `false` otherwise

$S \leftarrow S_0$;
repeat
  if $S \cap S_F \neq \emptyset$ **then**
    ;
  $S \leftarrow$;
until ;
return ;

Exercise: Forward reachability



Problem: is  reachable?

$S_0 = \{ \quad \}$

$S_F = \{ \quad \}$

Answer:

Backward Reachability

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

Definition (Pre)

By extension, we write $Pre^*(S')$ for the set of all states from which one can reach states of S'.

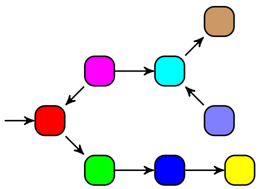
Backward Reachability: Algorithm

$isReachableBack(\mathcal{TS}, S_0, S_F)$

input : Set S_0 of initial states, set S_F of final states
output : `true` if S_F is reachable from S_0, `false` otherwise

```
S ← S_F ;
repeat
    if S ∩ S_0 ≠ ∅ then
        ;
    S ←            ;
until          ;
return     ;
```


Exercise: Backward reachability



Problem: is  reachable?

$S_0 = \{ \quad \}$

$S_F = \{ \quad \}$

Answer:

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 (1/3)

Design an observer for the coffee machine and the drinker verifying that whenever the coffee comes, no cup was put to the washing machine before.

(…and check the validity of the property)

Observers for the coffee machine (2/3)

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 (3/3)

Design an observer for the coffee machine and the drinker verifying that it is possible to order a coffee with exactly one dose of sugar.

(…and check the validity of the property)

Sources and references

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

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.

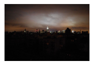
Baier, C. and Katoen, J.-P. (2008).
Principles of Model Checking.
MIT Press.

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.

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

Additional explanation

Explanation for the 4 pictures in the beginning



Allusion to the Northeast blackout (USA, 2003)
Computer bug
Consequences: 11 fatalities, huge cost
(Picture actually from the Sandy Hurricane, 2012)

Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
No fatalities
Computer bug: inaccurate finite element analysis modeling
(Picture actually from the Deepwater Horizon Offshore Drilling Platform)

Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
28 fatalities, hundreds of injured
Computer bug: software error (clock drift)
(Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)

Error screen on the earliest versions of Macintosh

License

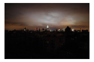
Source of the graphics (1)

Title: Clock 256
Author: Everaldo Coelho
Source: `https://commons.wikimedia.org/wiki/File:Clock_256.png`
License: 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

License of this document

Author: Étienne André

(LaTeX source available on demand)