

# Initiation à la vérification

## Basics of Verification

<https://wikimpri.dptinfo.ens-cachan.fr/doku.php?id=cours:c-1-22>

Paul Gastin

Paul.Gastin@lsv.ens-cachan.fr  
<http://www.lsv.ens-cachan.fr/~gastin/>

MPRI – M1  
2013 – 2014

## Outline

### 1 Introduction

### Models

### Temporal Specifications

### Satisfiability and Model Checking

### More on Temporal Specifications

## Need for formal verification methods

### Critical systems

- ▶ Transport
- ▶ Energy
- ▶ Medicine
- ▶ Communication
- ▶ Finance
- ▶ Embedded systems
- ▶ ...

## Disastrous software bugs

### Mariner 1 probe, 1962

See [http://en.wikipedia.org/wiki/Mariner\\_1](http://en.wikipedia.org/wiki/Mariner_1)

- ▶ Destroyed 293 seconds after launch
- ▶ Missing hyphen in the data or program? **No!**
- ▶ Overbar missing in the mathematical specification:

$\bar{R}_n$ :  $n$ th smoothed value of the time derivative of a radius.

Without the smoothing function indicated by the bar, the program treated normal minor variations of velocity as if they were serious, causing spurious corrections that sent the rocket off course.



## Disastrous software bugs

### Ariane 5 flight 501, 1996

See [http://en.wikipedia.org/wiki/Ariane\\_5\\_Flight\\_501](http://en.wikipedia.org/wiki/Ariane_5_Flight_501)

- ▶ Destroyed 37 seconds after launch (cost: 370 millions dollars).
- ▶ data conversion from a 64-bit floating point to 16-bit signed integer value caused a hardware exception (arithmetic overflow).
- ▶ Efficiency considerations had led to the disabling of the software handler (in Ada code) for this error trap.
- ▶ The fault occurred in the inertial reference system of Ariane 5. The software from Ariane 4 was re-used for Ariane 5 without re-testing.
- ▶ On the basis of those calculations the main computer commanded the booster nozzles, and somewhat later the main engine nozzle also, to make a large correction for an attitude deviation that had not occurred.
- ▶ The error occurred in a realignment function which was not useful for Ariane 5.

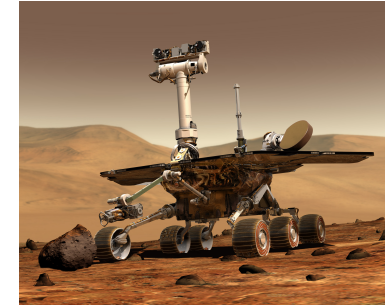


## Disastrous software bugs

### Spirit Rover (Mars Exploration), 2004

See [http://en.wikipedia.org/wiki/Spirit\\_rover](http://en.wikipedia.org/wiki/Spirit_rover)

- ▶ Landed on January 4, 2004.
- ▶ Ceased communicating on January 21.
- ▶ Flash memory management anomaly: too many files on the file system
- ▶ Resumed to working condition on February 6.



## Disastrous software bugs

### Other well-known bugs

- ▶ Therac-25, at least 3 death by massive overdoses of radiation.  
Race condition in accessing shared resources.  
See <http://en.wikipedia.org/wiki/Therac-25>
- ▶ Electricity blackout, USA and Canada, 2003, 55 millions people.  
Race condition in accessing shared resources.  
See [http://en.wikipedia.org/wiki/Northeast\\_Blackout\\_of\\_2003](http://en.wikipedia.org/wiki/Northeast_Blackout_of_2003)
- ▶ Pentium FDIV bug, 1994.  
Flaw in the division algorithm, discovered by Thomas Nicely.  
See [http://en.wikipedia.org/wiki/Pentium\\_FDIV\\_bug](http://en.wikipedia.org/wiki/Pentium_FDIV_bug)
- ▶ Needham-Schroeder, authentication protocol based on symmetric encryption.  
Published in 1978 by Needham and Schroeder  
Proved correct by Burrows, Abadi and Needham in 1989  
Flaw found by Lowe in 1995 (man in the middle)  
Automatically proved incorrect in 1996.  
See [http://en.wikipedia.org/wiki/Needham-Schroeder\\_protocol](http://en.wikipedia.org/wiki/Needham-Schroeder_protocol)

## Formal verifications methods

### Complementary approaches

- ▶ Theorem prover
- ▶ Model checking
- ▶ Static analysis
- ▶ Test

# Model Checking

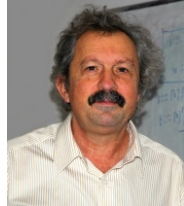
- ▶ Purpose 1: **automatically** finding software or hardware bugs.
- ▶ Purpose 2: **prove correctness** of abstract models.
- ▶ Should be applied during design.
- ▶ Real systems can be analysed with abstractions.



E.M. Clarke



E.A. Emerson



J. Sifakis

Prix Turing 2007.

# Model Checking

## 3 steps

- ▶ Constructing the model  $M$  (transition systems)
- ▶ Formalizing the specification  $\varphi$  (temporal logics)
- ▶ Checking whether  $M \models \varphi$  (algorithmics)

## Main difficulties

- ▶ Size of models (combinatorial explosion)
- ▶ Expressivity of models or logics
- ▶ Decidability and complexity of the model-checking problem
- ▶ Efficiency of tools

## Challenges

- ▶ Extend models and algorithms to cope with more systems.  
Infinite systems, parameterized systems, probabilistic systems, concurrent systems, timed systems, hybrid systems, ... See Modules 2.8 & 2.9
- ▶ Scale current tools to cope with real-size systems.  
Needs for modularity, abstractions, symmetries, ...

# References

- [1] Christel Baier and Joost-Pieter Katoen.  
*Principles of Model Checking*.  
MIT Press, 2008.
- [2] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci,  
Ph. Schnoebelen.  
*Systems and Software Verification. Model-Checking Techniques and Tools*.  
Springer, 2001.
- [3] E.M. Clarke, O. Grumberg, D.A. Peled.  
*Model Checking*.  
MIT Press, 1999.
- [4] Z. Manna and A. Pnueli.  
*The Temporal Logic of Reactive and Concurrent Systems: Specification*.  
Springer, 1991.
- [5] Z. Manna and A. Pnueli.  
*Temporal Verification of Reactive Systems: Safety*.  
Springer, 1995.

# Outline

## Introduction

- 2 Models
  - Transition Systems
  - ... with Variables
  - Concurrent Systems
  - Synchronization and Communication

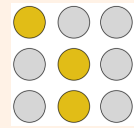
## Temporal Specifications

## Satisfiability and Model Checking

## More on Temporal Specifications

## Model and abstractions

Example: Golden face



Each coin has a golden face and a silver face.  
At each step, we may flip simultaneously the 3 coins of a line, column or diagonal.  
Is it possible to have all coins showing its golden face ?  
If yes, what is the smallest number of steps.

## Model and Specification

Example: Men, Wolf, Goat, Cabbage



Model = Transition system

- State = who is on which side of the river
- Transition = crossing the river
- Specification
  - Safety: Never leave WG or GC alone
  - Liveness: Take everyone to the other side of the river.

## Transition system or Kripke structure

Definition: TS  $M = (S, \Sigma, T, I, AP, \ell)$

- $S$ : set of states (finite or infinite)
- $\Sigma$ : set of actions
- $T \subseteq S \times \Sigma \times S$ : set of transitions
- $I \subseteq S$ : set of initial states
- AP: set of atomic propositions
- $\ell : S \rightarrow 2^{AP}$ : labelling function.

Every discrete system may be described with a TS.

Example: Digicode ABA

## Description Languages

Pb: How can we easily describe big systems?

Description Languages (high level)

- Programming languages
- Boolean circuits
- Modular description, e.g., parallel compositions  
problems: concurrency, synchronization, communication, atomicity, fairness, ...
- Petri nets (intermediate level)
- Transition systems (intermediate level)  
with variables, stacks, channels, ...  
synchronized products
- Logical formulae (low level)

Operational semantics

High level descriptions are translated (compiled) to low level (infinite) TS.

## Transition systems with variables

- Definition: TSV**  $M = (S, \Sigma, \mathcal{V}, (D_v)_{v \in \mathcal{V}}, T, I, AP, \ell)$
- $\mathcal{V}$ : set of (typed) variables, e.g., boolean,  $[0..4]$ ,  $\mathbb{N}$ , ...
  - Each variable  $v \in \mathcal{V}$  has a domain  $D_v$  (finite or infinite). Let  $D = \prod_{v \in \mathcal{V}} D_v$ .
  - Guard or Condition  $g$  with semantics  $\llbracket g \rrbracket \subseteq D$  (unary predicate)  
Symbolic descriptions:  $x < 5$ ,  $x + y = 10$ , ...
  - Instruction or Update  $f$  with semantics  $\llbracket f \rrbracket : D \rightarrow D$   
Symbolic descriptions:  $x := 0$ ,  $x := (y + 1)^2$ , ...
  - $T \subseteq S \times (\text{Guard} \times \Sigma \times \text{Update}) \times S$   
Symbolic descriptions:  $s \xrightarrow{x < 50, ?\text{coin}, x := x + \text{coin}} s'$
  - $I \subseteq S \times \text{Guard}$   
Symbolic descriptions:  $(s_0, x = 0)$

### Example: Vending machine

- coffee: 50 cents, orange juice: 1 euro, ...
- possible coins: 10, 20, 50 cents
- we may shuffle coin insertions and drink selection

## Transition systems with variables

### Semantics: low level TS

- $S' = S \times D$
- $I' = \{(s, \nu) \mid \exists (s, g) \in I \text{ with } \nu \models g\}$
- Transitions:  $T' \subseteq (S \times D) \times \Sigma \times (S \times D)$

$$\frac{s \xrightarrow{g, a, f} s' \wedge \nu \models g}{(s, \nu) \xrightarrow{a} (s', f(\nu))}$$

### SOS: Structural Operational Semantics

- AP': we may use atomic propositions in AP or guards such as  $x > 0$ .

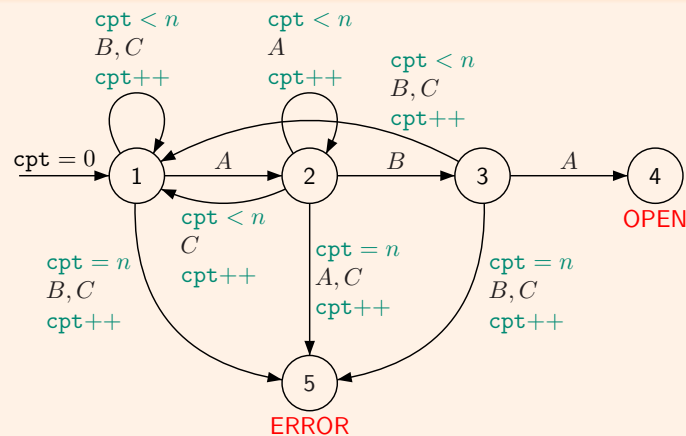
### Programs = Kripke structures with variables

- Program counter = states
- Instructions = transitions
- Variables = variables

### Example: GCD

## TS with variables ...

### Example: Digicode



## Only variables

The state is nothing but a special variable:  $s \in \mathcal{V}$  with domain  $D_s = S$ .

### Definition: TSV $M = (\mathcal{V}, (D_v)_{v \in \mathcal{V}}, T, I, AP, \ell)$

- $D = \prod_{v \in \mathcal{V}} D_v$ ,
- $I \subseteq D$ ,  $T \subseteq D \times D$

### Symbolic representations with logic formulae

- $I$  given by a formula  $\psi(\nu)$
- $T$  given by a formula  $\varphi(\nu, \nu')$   
 $\nu$ : values **before** the transition  
 $\nu'$ : values **after** the transition
- Often we use boolean variables only:  $D_v = \{0, 1\}$
- Concise descriptions of boolean formulae with Binary Decision Diagrams.

### Example: Boolean circuit: modulo 8 counter

$$\begin{aligned} b'_0 &= \neg b_0 \\ b'_1 &= b_0 \oplus b_1 \\ b'_2 &= (b_0 \wedge b_1) \oplus b_2 \end{aligned}$$

## Modular description of concurrent systems

$$M = M_1 \parallel M_2 \parallel \dots \parallel M_n$$

### Semantics

- ▶ Various semantics for the parallel composition  $\parallel$
- ▶ Various communication mechanisms between components: Shared variables, FIFO channels, Rendez-vous, ...
- ▶ Various restrictions

Atomic propositions are inherited from the local systems.

Example: Elevator with 1 cabin, 3 doors, 3 calling devices

- ▶ Cabin:
- ▶ Door for level  $i$ :
- ▶ Call for level  $i$ :

## Synchronized products

### Definition: General product

- ▶ Components:  $M_i = (S_i, \Sigma_i, T_i, I_i, AP_i, \ell_i)$
- ▶ Product:  $M = (S, \Sigma, T, I, AP, \ell)$  with
 
$$S = \prod_i S_i, \quad \Sigma = \prod_i (\Sigma_i \cup \{\varepsilon\}), \quad \text{and} \quad I = \prod_i I_i$$

$$T = \{(p_1, \dots, p_n) \xrightarrow{(a_1, \dots, a_n)} (q_1, \dots, q_n) \mid \text{for all } i, (p_i, a_i, q_i) \in T_i \text{ or } a_i = \varepsilon \text{ and } p_i = q_i\}$$

$$AP = \biguplus_i AP_i \text{ and } \ell(p_1, \dots, p_n) = \bigcup_i \ell(p_i)$$

### Synchronized products: restrictions of the general product.

Parallel compositions: 2 special cases

- ▶ Synchronous:  $\Sigma_{\text{sync}} = \prod_i \Sigma_i$
- ▶ Asynchronous:  $\Sigma_{\text{async}} = \biguplus_i \Sigma'_i$  with  $\Sigma'_i = \{\varepsilon\}^{i-1} \times \Sigma_i \times \{\varepsilon\}^{n-i}$

Restrictions

- ▶ on states:  $S_{\text{restrict}} \subseteq S$
- ▶ on labels:  $\Sigma_{\text{restrict}} \subseteq \Sigma$
- ▶ on transitions:  $T_{\text{restrict}} \subseteq T$

## Shared variables

### Definition: Asynchronous product + shared variables

$\bar{s} = (s_1, \dots, s_n)$  denotes a tuple of states  
 $\nu \in D = \prod_{v \in \mathcal{V}} D_v$  is a valuation of variables.

Semantics (SOS) 
$$\frac{\nu \models g \wedge s_i \xrightarrow{g, a, f} s'_i \wedge s'_j = s_j \text{ for } j \neq i}{(\bar{s}, \nu) \xrightarrow{a} (\bar{s}', f(\nu))}$$

Example: Mutual exclusion for 2 processes satisfying

- ▶ **Safety:** never simultaneously in critical section (CS).
- ▶ **Liveness:** if a process wants to enter its CS, it eventually does.
- ▶ **Fairness:** if process 1 wants to enter its CS, then process 2 will enter its CS at most once before process 1 does.

using shared variables but without further restrictions: the **atomicity** is

- ▶ testing or reading or writing a **single variable at a time**
- ▶ no test-and-set:  $\{x = 0; x := 1\}$

## Peterson's algorithm (1981)

```

Process i:           // i is not a variable
loop forever
  req[i] := true; turn := 1-i
  wait until (turn = i or req[1-i] = false)
  Critical section
  req[i] := false
    
```

### Exercise:

- ▶ Draw the concrete TS assuming the first two assignments are atomic.
- ▶ Is the algorithm still correct if we swape the first two assignments?

## Atomicity

### Example:

Initially  $x = 1 \wedge y = 2$

Program  $P_1: x := x + y \parallel y := x + y$

Program  $P_2: \left( \begin{array}{l} \text{Load}R_1, x \\ \text{Add}R_1, y \\ \text{Store}R_1, x \end{array} \right) \parallel \left( \begin{array}{l} \text{Load}R_2, x \\ \text{Add}R_2, y \\ \text{Store}R_2, y \end{array} \right)$

Assuming each instruction is atomic, what are the possible results of  $P_1$  and  $P_2$ ?

## Atomicity

Definition: Atomic statements: **atomic(ES)**

Elementary statements (no loops, no communications, no synchronizations)

$$ES ::= \text{skip} \mid \text{await } c \mid x := e \mid ES ; ES \mid ES \square ES \\ \mid \text{when } c \text{ do } ES \mid \text{if } c \text{ then } ES \text{ else } ES$$

Atomic statements: if the ES can be fully executed then it is executed in one step.

$$\frac{(\bar{s}, \nu) \xrightarrow{ES} (\bar{s}', \nu')}{(\bar{s}, \nu) \xrightarrow{\text{atomic}(ES)} (\bar{s}', \nu')}$$

### Example: Atomic statements

- ▶  $\text{atomic}(x = 0; x := 1)$  (Test and set)
- ▶  $\text{atomic}(y := y - 1; \text{await}(y = 0); y := 1)$  is equivalent to  $\text{await}(y = 1)$

## Communication by Rendez-vous

Restriction on transitions is universal but too low-level.

Definition: Rendez-vous

- ▶  $!m$  sending message  $m$
- ▶  $?m$  receiving message  $m$
- ▶ SOS: Structural Operational Semantics

Local actions  $\frac{s_1 \xrightarrow{a_1} s'_1}{(s_1, s_2) \xrightarrow{a_1} (s'_1, s_2)} \quad \frac{s_2 \xrightarrow{a_2} s'_2}{(s_1, s_2) \xrightarrow{a_2} (s_1, s'_2)}$

Rendez-vous  $\frac{s_1 \xrightarrow{!m} s'_1 \wedge s_2 \xrightarrow{?m} s'_2}{(s_1, s_2) \xrightarrow{m} (s'_1, s'_2)} \quad \frac{s_1 \xrightarrow{?m} s'_1 \wedge s_2 \xrightarrow{!m} s'_2}{(s_1, s_2) \xrightarrow{m} (s'_1, s'_2)}$

- ▶ It is a restriction on actions.
- ▶ Essential feature of process algebra.

Example: Elevator with 1 cabin, 3 doors, 3 calling devices

- ▶  $?up$  is uncontrollable for the cabin
- ▶  $?leave_i$  is uncontrollable for door  $i$
- ▶  $?call_0$  is uncontrollable for the system

## Channels

Example: Leader election

We have  $n$  processes on a directed ring, each having a unique  $\text{id} \in \{1, \dots, n\}$ .

```
send(id)
loop forever
  receive(x)
  if (x = id) then STOP fi
  if (x > id) then send(x)
```

## Channels

### Definition: Channels

- ▶ Declaration:
  - $c$  : channel [k] of bool    size  $k$
  - $c$  : channel [ $\infty$ ] of int    unbounded
  - $c$  : channel [0] of colors    Rendez-vous
- ▶ Primitives:
  - empty( $c$ )
  - $c!e$     add the value of expression  $e$  to channel  $c$
  - $c?x$     read a value from  $c$  and assign it to variable  $x$
- ▶ Domain: Let  $D_m$  be the domain for a single message.
  - $D_c = D_m^k$     size  $k$
  - $D_c = D_m^*$     unbounded
  - $D_c = \{e\}$     Rendez-vous
- ▶ Politics: FIFO, LIFO, BAG, ...

## Channels

### Semantics: (lossy) FIFO

$$\begin{array}{l} \text{Send} \quad \frac{s_i \xrightarrow{c!e} s'_i \wedge \nu'(c) = \nu(e) \cdot \nu(c)}{(\bar{s}, \nu) \xrightarrow{c!e} (\bar{s}', \nu')} \\ \text{Receive} \quad \frac{s_i \xrightarrow{c?x} s'_i \wedge \nu(c) = \nu'(c) \cdot \nu'(x)}{(\bar{s}, \nu) \xrightarrow{c?e} (\bar{s}', \nu')} \\ \text{Lossy send} \quad \frac{s_i \xrightarrow{c!e} s'_i}{(\bar{s}, \nu) \xrightarrow{c!e} (\bar{s}', \nu)} \end{array}$$

Implicit assumption: all variables that do not occur in the premise are not modified.

### Exercises:

1. Implement a FIFO channel using rendez-vous with an intermediary process.
2. Give the semantics of a LIFO channel.
3. Model the **alternating bit protocol (ABP)** using a lossy FIFO channel.  
Fairness assumption: For each channel, if infinitely many messages are sent, then infinitely many messages are delivered.

## High-level descriptions

### Summary

- ▶ Sequential program = transition system with variables
- ▶ Concurrent program with shared variables
- ▶ Concurrent program with Rendez-vous
- ▶ Concurrent program with FIFO communication
- ▶ Petri net
- ▶ ...

## Models: expressivity versus decidability

### Remark: (Un)decidability

- ▶ Automata with 2 integer variables = Turing powerful  
Restriction to variables taking values in finite sets
- ▶ Asynchronous communication: unbounded fifo channels = Turing powerful  
Restriction to bounded channels or lossy channels

### Remark: Some infinite state models are decidable

- ▶ Petri nets. Several unbounded integer variables but no zero-test.
- ▶ Pushdown automata. Model for recursive procedure calls.
- ▶ Timed automata.
- ▶ ...