Initiation à la vérification
Basics of Verification


Paul Gastin

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

MPRI – M1
2012 – 2013

Outline

- Introduction
- Models
- Specifications
- Satisfiability and Model Checking for LTL
- Branching Time Specifications

Need for formal verifications 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
- Destroyed 293 seconds after launch
- Missing hyphen in the data or program? No!
- Overbar missing in the mathematical specification:
  \( \overline{R}_n \): \( n \)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
- 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.

#### Spirit Rover (Mars Exploration), 2004
- Ceased communicating on January 21.
- Flash memory management anomaly: too many files on the file system.
- Resumed to working condition on February 6.

### Other well-known bugs

### 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.

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

Bibliography


Outline

Introduction
- Models
  - Transition systems
  - . . . with variables
  - Concurrent systems
  - Synchronization and communication

Specifications

Satisfiability and Model Checking for LTL

Branching Time 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**

_{\text{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 $$M = (S, \Sigma, V, (D_v)_{v \in V}, T, I, AP, \ell)$$

- $$V$$: set of (typed) variables, e.g., boolean, [0..4], N, ...
- Each variable $$v \in V$$ has a domain $$D_v$$ (finite or infinite).
- Let $$D = \prod_{v \in V} D_v$$.
- Guard or Condition $$g$$ with semantics $$[g] \subseteq D$$ (unary predicate)
  Symbolic descriptions: $$x < 5$$, $$x + y = 10$$, ...
- Instruction or Update $$f$$ with semantics $$[f] : D \rightarrow D$$
  Symbolic descriptions: $$x := 0$$, $$x := (y + 1)^2$$, ...
- Transitions: $$T \subseteq (S \times D) \times \Sigma \times (S \times D)$$
  Symbolic descriptions: $$s \xrightarrow{g,a,f} s' \wedge v = g$$

I Guard or Condition
I Instruction or Update
I SOS: Structural Operational Semantics

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

Example: GCD

TS with variables ...

Example: Digicode

Only variables

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

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

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

Symbolic representations with logic formulae

- I given by a formula $$\psi(v)$$
- $$T$$ given by a formula $$\varphi(v,v')$$
- $$v'$$: values before the transition
- $$v$$: 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

$$b_0' = \neg b_0$$
$$b_1' = b_0 \oplus b_1$$
$$b_2' = (b_0 \land b_1) \oplus b_2$$
Modular description of concurrent systems

\[ M = M_1 \parallel M_2 \parallel \cdots \parallel M_n \]

**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 \( i \):

**Shared variables**

**Definition:** Asynchronous product + shared variables

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

**Semantics (SOS)**

\[ \nu \models g \land s_i \text{ or } s_j' \iff s_j' = s_j \text{ for } j \neq i. \]

**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 $x = 1 \land y = 2$
Program $P_1$: $x := x + y \parallel y := x + y$
Program $P_2$: $(\text{Load}R_1, x) \parallel (\text{Load}R_2, x)$

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

Communication by Rendez-vous

Definition: Rendez-vous
- $!m$ sending message $m$
- $?m$ receiving message $m$
- SOS: Structural Operational Semantics
  - Local actions
    - $s_1 \xrightarrow{a_{11}} s_1'$
    - $s_2 \xrightarrow{a_{22}} s_2'$
  - Rendez-vous
    - $s_1 \xrightarrow{m_{11}} s_1' \land s_2 \xrightarrow{m_{22}} s_2'$

Example: Elevator with 1 cabin, 3 doors, 3 calling devices
- $?\text{up}$ is uncontrollable for the cabin
- $?\text{leave}_i$ is uncontrollable for door $i$
- $?\text{call}_0$ is uncontrollable for the system

Atomicity

Definition: Atomic statements: $\text{atomic}(ES)$

Elementary statements (no loops, no communications, no synchronizations)
- $ES ::= \text{skip} \mid \text{await} c \mid x := e \mid ES \parallel ES \mid \text{when } c \text{ do } ES \mid \text{if } c \text{ then } ES \text{ else } ES$

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

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 $\text{await}(y = 1)$

Channels

Example: Leader election
We have $n$ processes on a directed ring, each having a unique id $\in \{1, \ldots, n\}$.

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 [∞] of int unbounded
  c : channel [0] of colors Rendez-vous
- Primitives:
  empty(c) = 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
  $D_c = D_m^\infty$ unbounded
  $D_c = \{\epsilon\}$ Rendez-vous
- Politics: FIFO, LIFO, BAG, ...

Semantics: (lossy) FIFO
- Send
  $s_i \xrightarrow{c!e} s_i' \land \nu'(c) = \nu(c) \cdot \nu(e)$
  $\nu''(c) = \nu''(e)$
- Receive
  $s_i \xrightarrow{c?x} s_i' \land \nu'(c) = \nu'(e) \cdot \nu'(x)$
- Lossy send
  $s_i \xrightarrow{c!s} s_i' \land \nu''(c) = \nu''(s)$

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.
- ...

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.
- ...