Basics of Verification

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

Thomas Chatain
Marie Fortin
Philippe Schnoebelen
Stefan Schwoon

MPRI – M1
2016 – 2017

Disastrous software bugs

Ariane 5 flight 501, 1996

http://en.wikipedia.org/wiki/Ariane_5_Flight_501

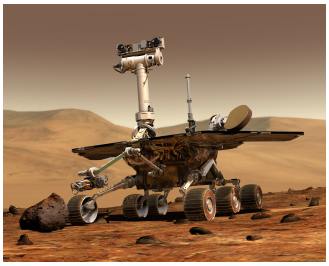
- Destroyed 37 seconds after launch (cost: 370 millions dollars).
- data conversion from a 64-bit floating point to 16-bit signed integer value caused a hardware exception (arithmetic overflow).
- Efficiency considerations had led to the disabling of the software handler (in Ada code) for this error trap.
- The fault occured in the inertial reference system of Ariane 5. The software from Ariane 4 was re-used for Ariane 5 without re-testing.

Disastrous software bugs

Spirit Rover (Mars Exploration), 2004

See 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
- Pentium FDIV bug, 1994.
  Flaw in the division algorithm, discovered by Thomas Nicely.
  See 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

Need for formal verification methods

Critical systems

- Transportation
- Energy
- Medicine
- Communication
- Embedded systems
- Security
- ...

A few things one can check automatically

- Result of an arithmetic operation
  - $36443 \times 28376 + 29385 = 1034135953$?
- Check that a finite graph is (strongly) connected
- Correctness of a proof
  - Proof checker
- Static analysis of programs
  - Typing
  - Undeclared, unused variables

Exercise: complexity of these problems?

Many things one can *not* check automatically a.k.a. Undecidable problems

Halting problem for Turing machines [Turing, 1936]

- Assume $P : (Q, x) \mapsto \{\text{true}, \text{false}\}$
  such that $P(Q, x)$ iff $Q(x)$ halts
- Build R such that $R(Q)$ halts iff $Q(Q)$ does not halt (exercise)
- Then $R(R)$ halts iff $R(R)$ does not halt!

Rice, 1953

All non trivial problems about Turing machines are undecidable

...and even most problems about most (even very simple) extensions of automata!

Formal verification methods

Based on

- a formal model of the system
  - written in a not too expressive formalism
- a formal semantics
- a formal specification

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

Turing Award 2007.

Model Checking

3 steps

- Constructing the model M (transition systems)
- Formalizing the specification φ (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

References

[1] Ph. Schnoebelen.
The Complexity of Temporal Logic Model Checking.
In AiML'02, pages 393-436. King's College Publication, 2003. Invited paper.

[2] Christel Baier and Joost-Pieter Katoen.
Principles of Model Checking.
MIT Press, 2008.

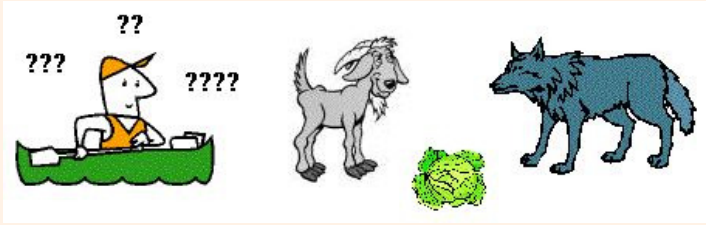
[3] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen.
Systems and Software Verification. Model-Checking Techniques and Tools.
Springer, 2001.

[4] E.M. Clarke, O. Grumberg, D.A. Peled.
Model Checking.
MIT Press, 1999.

[5] Z. Manna and A. Pnueli.
The Temporal Logic of Reactive and Concurrent Systems: Specification.
Springer, 1991.

Model and Specification

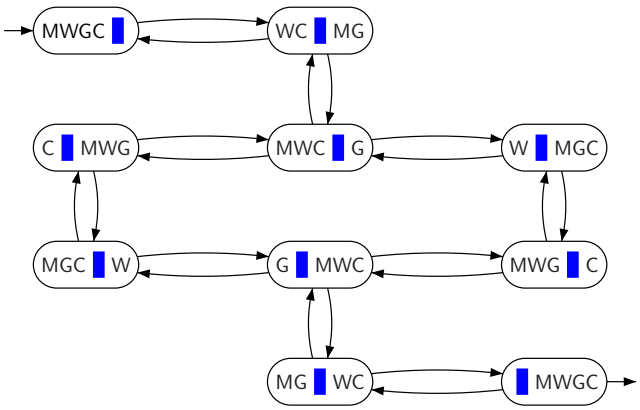
Example: Man, Wolf, Goat, Cabbage



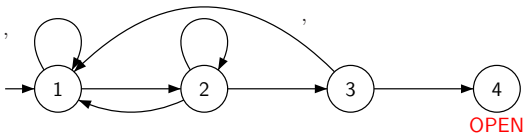
Model = Transition system

- 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



Example: Timed Digicode with UPPAAL



Construct a timed version and check it with UPPAAL.

Exercise

Describe a thechnique to automatically detect a bug in the following function:

```
function GCD(x:  int, y:  int) {
 while x ≠ 0 do
   if x > y then {tmp := y; y := x; x := tmp}
   x := y - x
 return y
}
```

Exercise

```
function f91(x:  int) {
 if (x > 100)
   then return x - 10;
   else return f91(f91(x + 11));
}
```

Describe a transition system and a property that expresses correctness.

Outline of the course

September 15 to October 20

- Lectures by Thomas Chatain and Philippe Schnoebelen
- Exercises by Marie Fortin
- Content: CTL and LTL model-checking on finite structures
- Mid-term exam on November 3

November 10 to January 5

- Lectures by Stefan Schwoon
- Exercises by Marie Fortin
- Content: verification of distributed systems, partial order reduction, binary decision diagrams, Petri nets...
- Exam (on the second part) on January 12

Basics of Verification

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

Thomas Chatain
Marie Fortin
Philippe Schnoebelen
Stefan Schwoon

MPRI – M1
2016 – 2017

Disastrous software bugs

Ariane 5 flight 501, 1996

`http://en.wikipedia.org/wiki/Ariane_5_Flight_501`

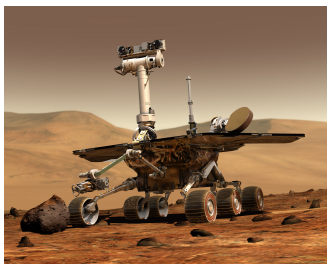
- Destroyed 37 seconds after launch (cost: 370 millions dollars).
- data conversion from a 64-bit floating point to 16-bit signed integer value caused a hardware exception (arithmetic overflow).
- Efficiency considerations had led to the disabling of the software handler (in Ada code) for this error trap.
- The fault occured in the inertial reference system of Ariane 5. The software from Ariane 4 was re-used for Ariane 5 without re-testing.

Disastrous software bugs

Spirit Rover (Mars Exploration), 2004

See `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`
- Pentium FDIV bug, 1994.
  Flaw in the division algorithm, discovered by Thomas Nicely.
  See `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`

Need for formal verification methods

Critical systems

- Transportation
- Energy
- Medicine
- Communication
- Embedded systems
- Security
- ...

A few things one can check automatically

- Result of an arithmetic operation
  - $36443 \times 28376 + 29385 = 1034135953$?
- Check that a finite graph is (strongly) connected
- Correctness of a proof
  - Proof checker
- Static analysis of programs
  - Typing
  - Undeclared, unused variables

Exercise: complexity of these problems?

Many things one can *not* check automatically a.k.a. Undecidable problems

Halting problem for Turing machines [Turing, 1936]

- Assume $P : (Q, x) \mapsto \{\text{true}, \text{false}\}$ such that $P(Q, x)$ iff $Q(x)$ halts
- Build R such that $R(Q)$ halts iff $Q(Q)$ does not halt (exercise)
- Then $R(R)$ halts iff $R(R)$ does not halt!

Rice, 1953

All non trivial problems about Turing machines are undecidable

...and even most problems about most (even very simple) extensions of automata!

Formal verification methods

Based on

- a formal model of the system
  - written in a not too expressive formalism
- a formal semantics
- a formal specification

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

Turing Award 2007.

Model Checking

3 steps

- Constructing the model M (transition systems)
- Formalizing the specification φ (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

References

[1] Ph. Schnoebelen.
The Complexity of Temporal Logic Model Checking.
In AiML'02, pages 393-436. King's College Publication, 2003. Invited paper.

[2] Christel Baier and Joost-Pieter Katoen.
Principles of Model Checking.
MIT Press, 2008.

[3] 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.

[4] E.M. Clarke, O. Grumberg, D.A. Peled.
Model Checking.
MIT Press, 1999.

[5] Z. Manna and A. Pnueli.
The Temporal Logic of Reactive and Concurrent Systems: Specification.
Springer, 1991.

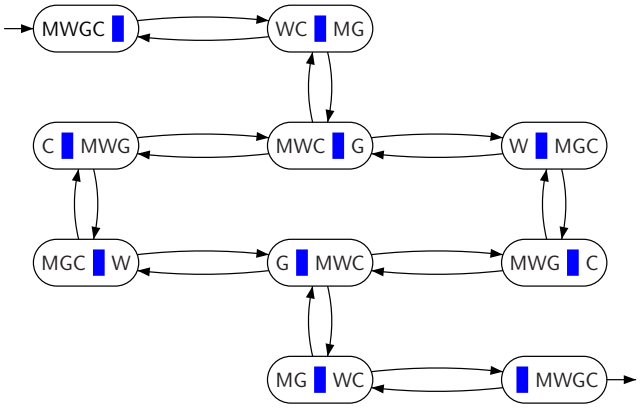
Model and Specification

Example: Man, Wolf, Goat, Cabbage

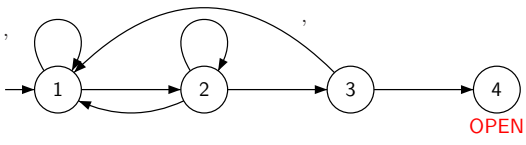
Model = Transition system

- 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



Example: Timed Digicode with UPPAAL



Construct a timed version and check it with UPPAAL.

Exercise

Describe a thechnique to automatically detect a bug in the following function:

```
function GCD(x:  int, y:  int) {
 while x ≠ 0 do
   if x > y then {tmp := y; y := x; x := tmp}
   x := y - x
 return y
}
```

Exercise

```
function f91(x:  int) {
 if (x > 100)
   then return x - 10;
   else return f91(f91(x + 11));
}
```

Describe a transition system and a property that expresses correctness.

Outline of the course

September 15 to October 20

- Lectures by Thomas Chatain and Philippe Schnoebelen
- Exercises by Marie Fortin
- Content: CTL and LTL model-checking on finite structures
- Mid-term exam on November 3

November 10 to January 5

- Lectures by Stefan Schwoon
- Exercises by Marie Fortin
- Content: verification of distributed systems, partial order reduction, binary decision diagrams, Petri nets...
- Exam (on the second part) on January 12