Computational Logic

Introduction to Model Checking

René Thiemann

Institute of Computer Science
University of Innsbruck

WS 2010/2011

Outline

- Organization
- On the role of system verification
- Formal verification techniques
- Model Checking
- Course Objectives

Outline

- Organization
- On the role of system verification
- Formal verification techniques
- Model Checking
- Course Objectives

Organization

LVA 703503 VO 1

cl-informatik.uibk.ac.at/teaching/ws10/imc

VO Wednesday 12:15 – 13:45 HS F
(no lecture on October 20, lecture ends in December)

online registration – required until 23:59 on October 30

		consultation hours
René Thiemann	3N01	Monday 13:00 – 15:00

Literature

the course is mainly based on the following books

- Christel Baier and Joost-Pieter Katoen
  Principles of Model Checking
  MIT Press, 2008
- Edmund M. Clarke, Orna Grumberg, and Doron A. Peled
  Model Checking
  MIT Press, 1999

Organization

- there will be exercises that help to understand the material of the lecture
- some lectures will be replaced by a discussion of the exercises; students can present their solutions and will be awarded with extra points for the exam (depending on the quality of the solution)
- some examples and proofs are developed on the blackboard

$\Rightarrow$ these parts are not in the slides

$\Rightarrow$ if you cannot attend a lecture see to it that some friend of yours makes notes

- Bachelor students cannot attend first exam (15 Dec)
  (but perhaps second or third exam in March/April or June)

Outline

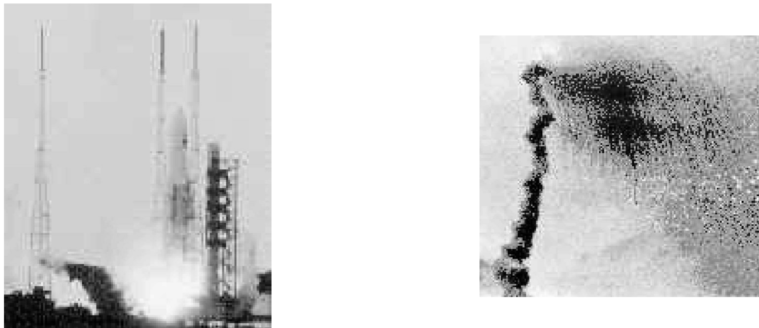
- Organization
- On the role of system verification
- Formal verification techniques
- Model Checking
- Course Objectives

The importance of software correctness

- rapidly increasing integration of ICT in different applications:
  - embedded systems
  - communication protocols
  - transportation systems
- reliability increasingly depends on hard- and software integrity
- defects can be fatal and extremely costly
  - products subject to mass-production
  - safety-critical systems

(ICT = information and computation technology)

A famous example: Ariane-5



the Ariane-5 launch on June 4, 1996; it crashed 36 seconds after the launch due to a conversion of a 64-bit floating point into a 16-bit integer value

What is system verification?

system verification amounts to check whether a system fulfills the qualitative requirements that have been identified

verification $\neq$ validation

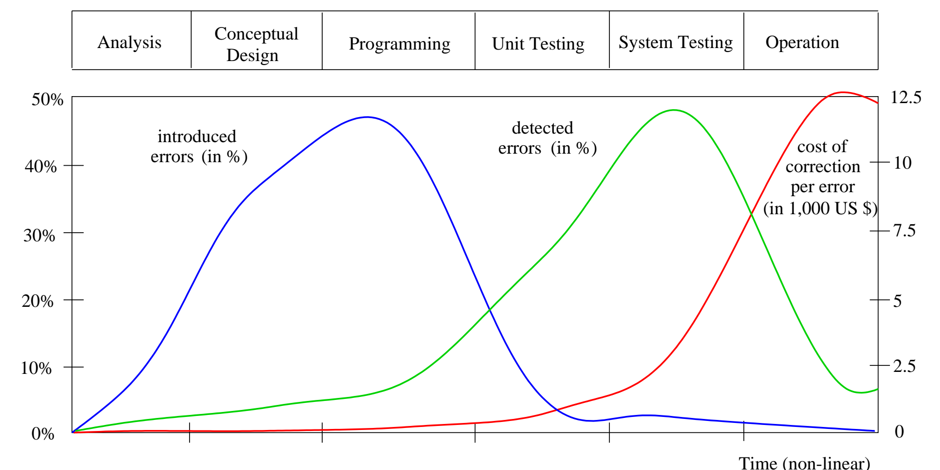
verification = "check that we are building the thing right"

validation = "check that we are building the right thing"

Software verification techniques

- peer reviewing
  - static technique: manual code inspection, no software execution
  - detects between 31 and 93 % of defects with median of about 60 %
  - subtle errors (concurrency and algorithm defects) hard to catch
- testing
  - dynamic technique in which software is executed
- some figures
  - 30 to 50 % of software project costs devoted to testing
  - more time and effort is spent on validation than on construction
  - accepted defect density: about 1 defect per 1,000 code lines

Catching software bugs: the sooner, the better



Outline

- Organization
- On the role of system verification
- Formal verification techniques
- Model Checking
- Course Objectives

Formal methods

formal methods are the
"applied mathematics for modelling and analysing ICT systems"

they offer a large potential for

- obtaining an early integration of verification in the design process
- providing more effective verification techniques (higher coverage)
- reducing the verification time

highly recommended by several large institutions (NASA, ...) for safety-critical software

Model-based formal verification

- starting-point is a model of the system under consideration
- modelling—a piece of art—already reveals several inconsistencies and ambiguities
- accompanied with efficient algorithms for realistic systems
  - improvements in data structures and algorithms + better computers

any verification using model-based techniques is only
as good as the model of the system

Formal verification techniques for property φ

- model checking
  - method: systematic check on φ in all states of model
  - tool: model checker (SPIN, NUSMV, UPPAAL, ...)
  - applicable if: system generates (finite) behavioural model
- deductive methods, model-based simulation or testing, ...

On the relevance of model checking

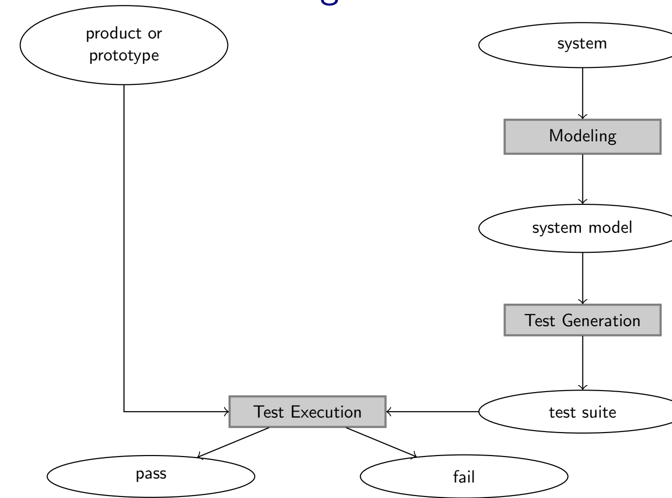
Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis are the winners of the 2007 A.M. Turing Award for their original and continuing research in a quality assurance process known as Model Checking.

Simulation and testing

- basic procedure
  - take a model (simulation) or a realisation (testing)
  - stimulate it with certain inputs, i.e., the tests
  - observe reaction and check whether this is "desired"
- important drawbacks
  - number of possible behaviours is very large (or even infinite)
  - unexplored behaviours may contain the fatal bug

$\Longrightarrow$ testing/simulation can show the presence of errors, not their absence

Model-based testing

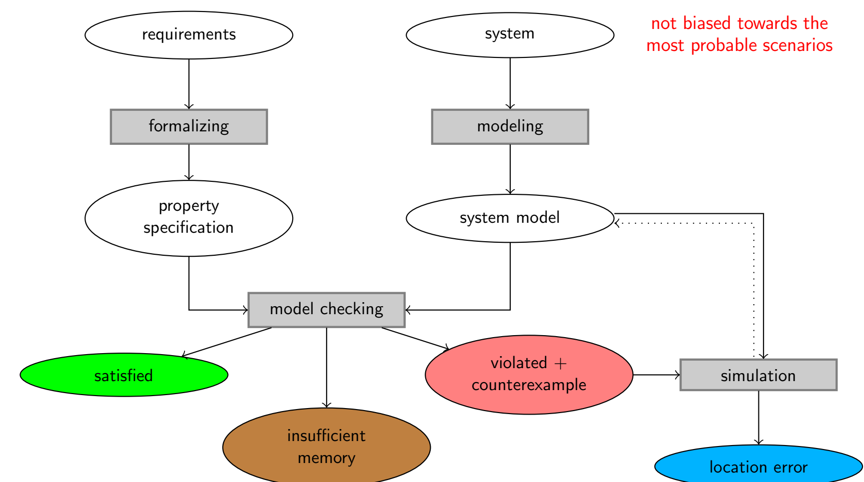


as model checking verifies models and not realisations,
testing is an essential complementary technique

Outline

- Organization
- On the role of system verification
- Formal verification techniques
- Model Checking
- Course Objectives

Model checking overview



What is model checking?

> Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.

The model checking process

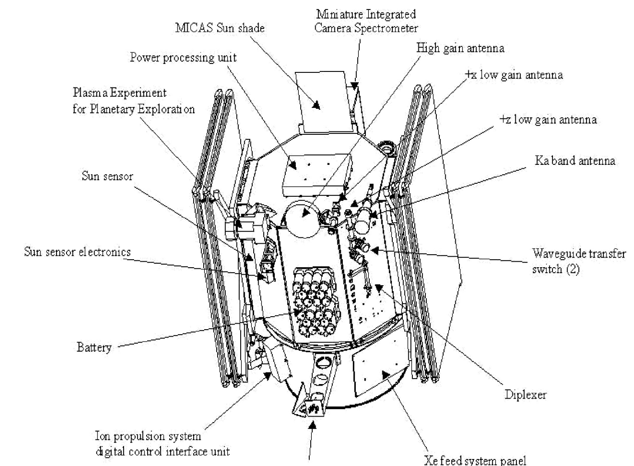
- modeling phase
  - model the system under consideration
  - as a first sanity check, perform some simulations
  - formalise the property to be checked
- running phase
  - run the model checker to check the validity of the property in the model
- analysis phase
  - property satisfied? → check next property (if any)
  - property violated? →
    1. analyse generated counterexample by simulation
    2. refine the model, design, or property . . . and repeat the entire procedure
  - out of memory? → try to reduce the model and try again

Typical model check properties

- is the generated result ok?
- can the system reach a deadlock situation, e.g., when two concurrent programs are mutually waiting for each other and thus halt the entire system?
- can a deadlock occur within 1 hour after a system reset?
- will there be a response to every request?

model checking requires a precise and unambiguous statement of the properties to be examined; this is typically done in temporal logic

Deep Space 1 Spacecraft



modules of NASA's Deep Space 1 spacecraft (launched in October 1998) have been thoroughly examined using model checking

A simple concurrent program

Promela code

```
int x = 0;

proctype Inc() {
  do :: true -> if :: (x < 200) -> x = x + 1 fi od
}
proctype Dec() {
  do :: true -> if :: (x > 0) -> x = x - 1 fi od
}
proctype Reset() {
  do :: true -> if :: (x == 200) -> x = 0 fi od
}
init {
  run Inc() ; run Dec() ; run Reset()
}
```

is x always between (and including) 0 and 200?

How to check for the values of x?

extend the model with a "monitor" process that checks $0 \leqslant x \leqslant 200$ ...

```
int x = 0;

proctype Inc() {
  do :: true -> if :: (x < 200) -> x = x + 1 fi od
}
proctype Dec() {
  do :: true -> if :: (x > 0) -> x = x - 1 fi od
}
proctype Reset() {
  do :: true -> if :: (x == 200) -> x = 0 fi od
}
proctype Check() {
  assert (x >= 0 && x <= 200)
}
init {
  run Inc() ; run Dec() ; run Reset() ; run Check()
}
```


How to check for the values of x?

extend the model with a "monitor" process that checks $0 \leqslant x \leqslant 200$ ...

and let the model checker verify whether the assertion holds in every state of the concurrent system ...

```
pan: assertion violated ((x >= 0) && (x <= 200)) (at depth 1802)
pan: wrote pan_in.trail
...................
State-vector 32 byte, depth reached 3598, errors: 1
      12609 states, stored
```


The counter-example

```
..............
605:   proc  1 (Inc)   line   9 "pan_in" (state 2)   [((x<200))]
606:   proc  1 (Inc)   line   9 "pan_in" (state 3)   [x = (x+1)]
607:   proc  2 (Dec)   line   5 "pan_in" (state 2)   [((x > 0))]
608:   proc  1 (Inc)   line   9 "pan_in" (state 1)   [(1)]
609:   proc  3 (Reset) line  13 "pan_in" (state 2)   [((x==200))]
610:   proc  3 (Reset) line  13 "pan_in" (state 3)   [x = 0]
611:   proc  3 (Reset) line  13 "pan_in" (state 1)   [(1)]
612:   proc  2 (Dec)   line   5 "pan_in" (state 3)   [x = (x-1)]
613:   proc  2 (Dec)   line   5 "pan_in" (state 1)   [(1)]
spin: line  17 "pan_in", Error: assertion violated
spin: text of failed assertion: assert(((x>=0)&&(x<=200)))
```

Breaking the error

```
int x = 0;

proctype Inc() {
  do :: true -> atomic { if :: (x < 200) -> x=x+1 fi } od
}
proctype Dec() {
  do :: true -> atomic { if :: (x > 0) -> x=x-1 fi } od
}
proctype Reset() {
  do :: true -> atomic { if :: (x == 200) -> x=0 fi } od
}
proctype Check() {
  assert (x >= 0 && x <= 200)
}
init {
  run Inc() ; run Dec() ; run Reset() ; run Check()
}
```


The pros of model checking

- widely applicable (hardware, software, protocol systems, ...)
- allows for partial verification (only most relevant properties)
- potential "push-button" technology (software-tools)
- rapidly increasing industrial interest
- in case of property violation, a counter-example is provided
- sound and interesting mathematical foundations
- not biased to the most possible scenarios (such as testing)

The cons of model checking

- mainly focused on control-intensive applications (less data-oriented)
- any validation model checking is only as "good" as the system model
- impossible to check generalisations (in general)

nevertheless:

model checking is an effective technique
to expose potential design errors

Striking model checking examples

- security: Needham-Schroeder encryption protocol
  - revealed error that remained undiscovered for 17 years
- transportation systems
  - train model containing 10^{476} states
- model checkers for `C`, `Java` and `C++`
  - used (and developed) by Microsoft, Digital, NASA
  - successful application area: device drivers
- software in the current/next generation of space missiles
  - NASA's Mars Pathfinder, Deep Space 1, JPL LARS group

Outline

- Organization
- On the role of system verification
- Formal verification techniques
- Model Checking
- Course Objectives

Course topics

- basics
  - transition systems
  - Büchi automata
- temporal logics (LTL, CTL*)
  - syntax, semantics
  - formalizations
  - model checking algorithms
- modeling software systems
  - concurrency
  - nanoPromela
  - state-space explosion problem
- ...