Model Checking

Introduction, Background, Course Organisation

Joost-Pieter Katoen and Tim Quatmann

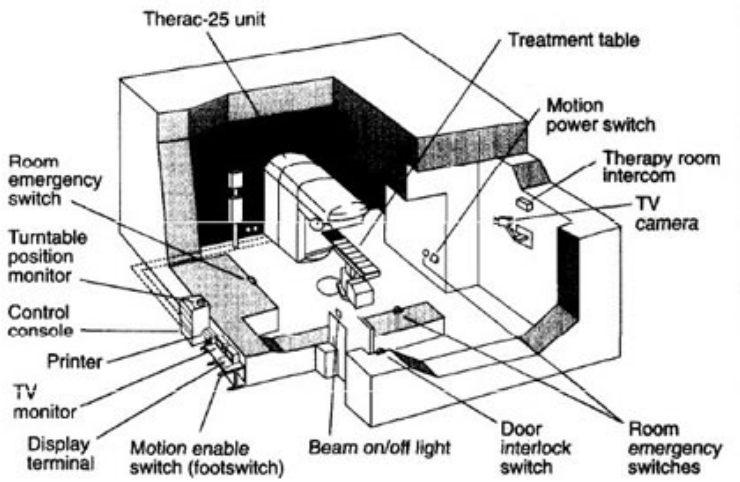
Software Modeling and Verification Group

RWTH Aachen, SoSe 2022

Overview

Overview

Software Reliability: Therac-25



- Radiation machine for cancer patients
- At least 6 cases of overdosis (≈ factor 100) in 1985–1987
- Three cancer patients died
- Source: Design error in the control software: race condition
- Software written in assembly language

Software Reliability: Ariane 5 Flight 501

- Crash of European Ariane 5-missile in 1996
- Source: conversion from a 64-bit floating point to 16-bit signed integer
- Efficiency considerations had led to disabling of the software handler (in `Ada`)
- Overflow conditions crashed both primary and backup computers
- Costs: more than 500 million US$, 8 billion US$ development costs

The Quest for Software Correctness

"It is fair to state, that in this digital era correct systems for information processing are more valuable than gold."

Speech@50-years CWI Amsterdam

Henk Barendregt

The Importance of Software Correctness

- Rapid increase of software in different applications
  - embedded systems
  - communication protocols
  - transportation systems

$\Rightarrow$ reliability increasingly depends on software!

- Defects can be fatal and extremely costly[1]
  - products subject to mass-production
  - safety-critical systems

Software reliability is one of the grand challenges of the German Society of Computer Science.

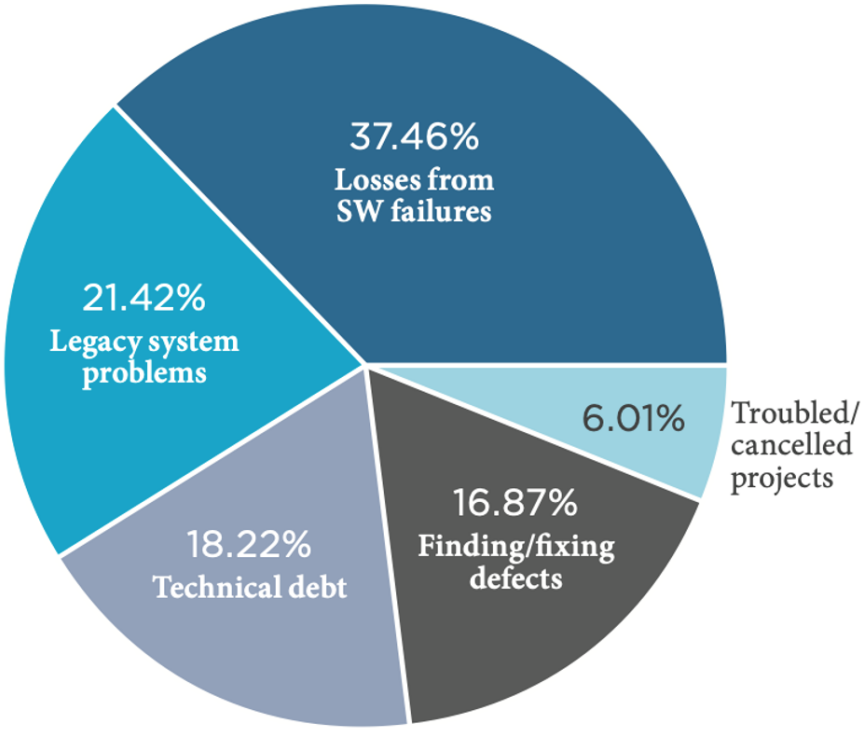
[1] See `https://raygun.com/blog/costly-software-errors-history/`

The Cost of Poor Quality Software

2,840,000,000,000 US dollar per year

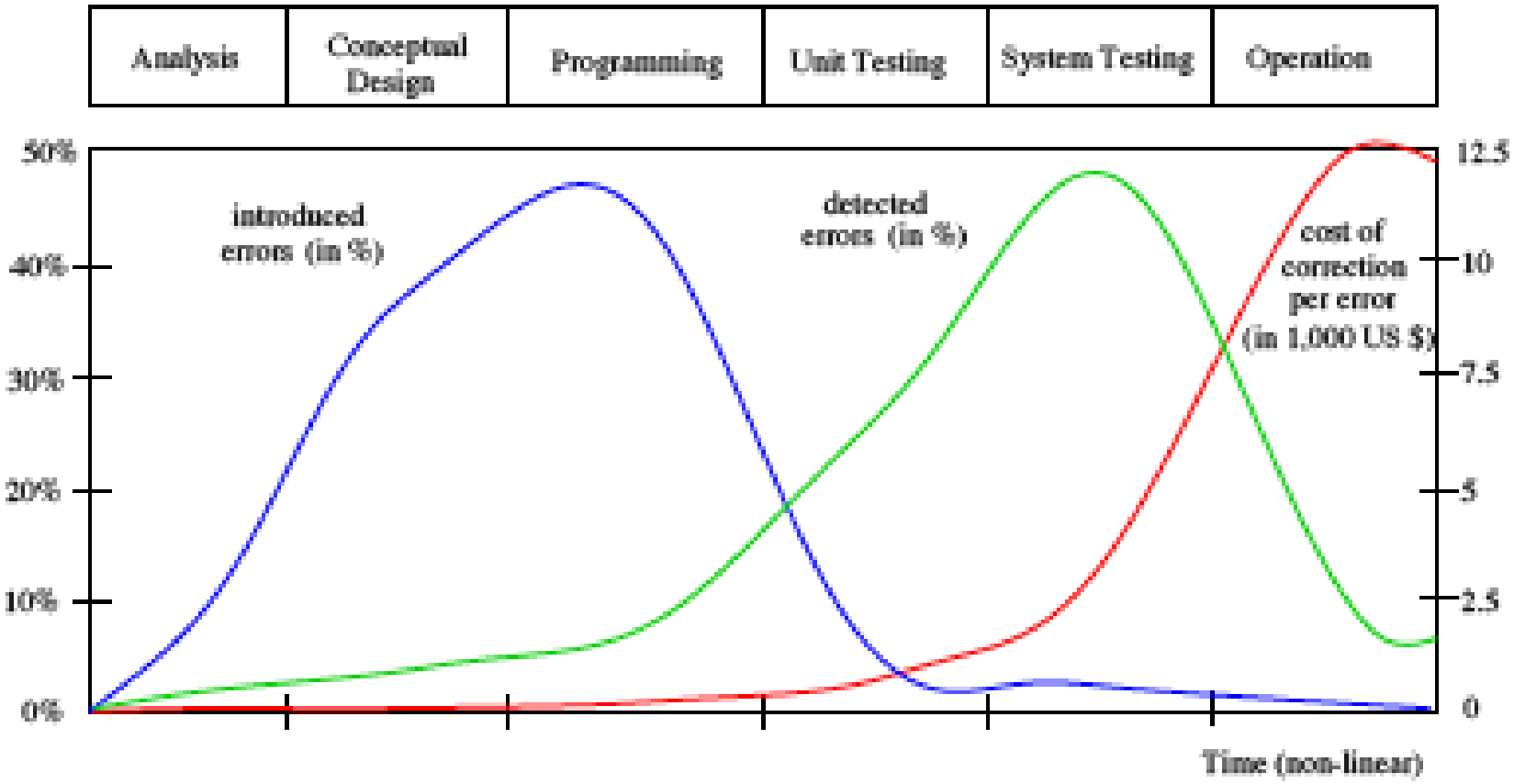
CISQ
Consortium for IT Software Quality

[The cost of poor quality software in the US: a 2018 report.]



Overview

Bug Hunting: the Sooner, the Better



Formal Methods

Formal methods are:

- "applied mathematics for modelling and analysing ICT systems"

[2] Federal Aviation Authority

[3] Automotive Safety Integrity Level

Formal Methods

Formal methods are:

- "applied mathematics for modelling and analysing ICT systems"

Formal methods 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

[2] Federal Aviation Authority

[3] Automotive Safety Integrity Level

Formal Methods

Formal methods are:

- "applied mathematics for modelling and analysing ICT systems"

Formal methods 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

Usage of formal methods:

- Highly recommended for safety-critical software by FAA[2], NASA, ...
- Required by ISO for autonomous vehicles at ASIL[3] Level D

[2] Federal Aviation Authority

[3] Automotive Safety Integrity Level

Formal Methods for Verifying Property φ

Deductive methods: provide a formal proof that φ holds

- tool: theorem prover (Isabelle/HOL, Coq, ...)
- applicable if: system has form of a mathematical theory
- pros: general applicable, high user involvement, hard guarantees

Model checking: systematic check on φ in all states

- tool: model checker (Spin, NuSMV, UppAal, ...)
- applicable if: system generates (finite) behavioural model
- pros: highly (fully) automated, hard guarantees

Model-based testing: test for φ by program execution

- applicable if: system defines an executable model
- tool: text generation and execution (JTorX, RT-Tester, ...)
- pros: useful for finding bugs, not their absence

Milestones in Formal Verification

- Mathematical program correctness (Turing, 1949)

Milestones in Formal Verification

- Mathematical program correctness (Turing, 1949)
- Syntax-based technique for sequential programs (Hoare, 1968)
  - for a given input, does a program generate the correct output?
  - based on compositional proof rules expressed in predicate logic

Milestones in Formal Verification

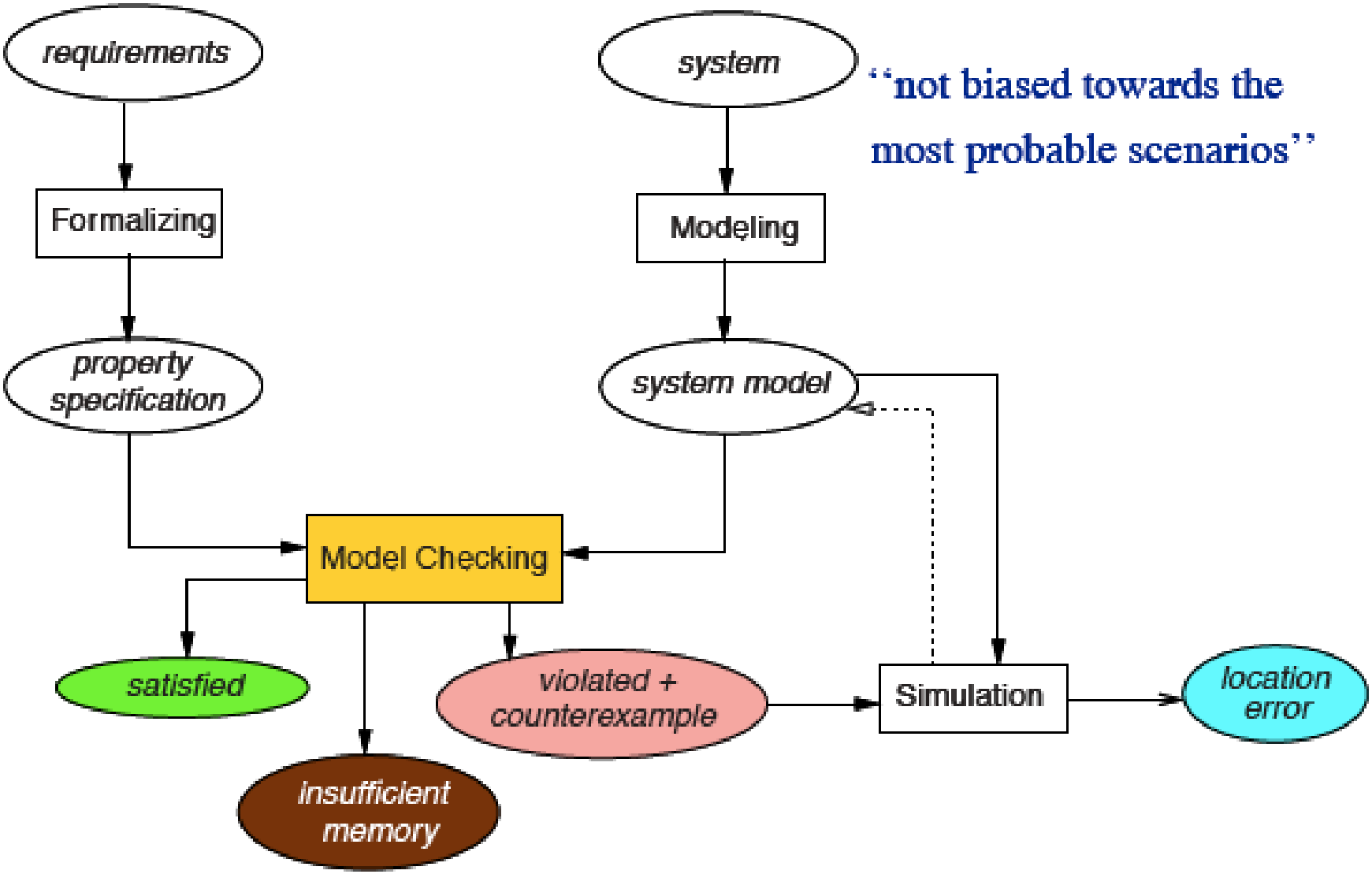
- Mathematical program correctness (Turing, 1949)
- Syntax-based technique for sequential programs (Hoare, 1968)
  - for a given input, does a program generate the correct output?
  - based on compositional proof rules expressed in predicate logic
- Syntax-based technique for concurrent programs (Pnueli, 1977)
  - handles properties referring to states during the computation
  - based on proof rules expressed in temporal logic

Milestones in Formal Verification

- Mathematical program correctness (Turing, 1949)
- Syntax-based technique for sequential programs (Hoare, 1968)
  - for a given input, does a program generate the correct output?
  - based on compositional proof rules expressed in predicate logic
- Syntax-based technique for concurrent programs (Pnueli, 1977)
  - handles properties referring to states during the computation
  - based on proof rules expressed in temporal logic
- Automated verification of concurrent programs (Clarke & Emerson 1981 Queille & Sifakis 1982)
  - model-based instead of proof-rule based approach
  - does the concurrent program satisfy a given (logical) property?

Overview

Model Checking Overview



Paris Kanellakis Theory and Practice Award 1998

Randal Bryant

Edmund Clarke

E. Allen Emerson

Ken McMillan

For their invention of "symbolic model checking,"
a method of formally checking system designs,
which is widely used in the computer hardware industry
and starts to show significant promise also in
software verification and other areas.

Some other winners: Rivest et al., Paige and Tarjan, Buchberger, ...

Gödel Prize 2000

Moshe Vardi

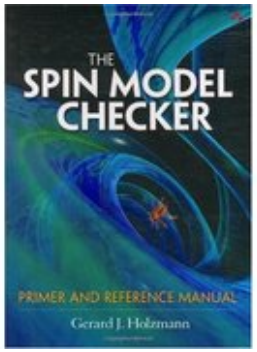
Pierre Wolper

"For work on model checking with finite automata."

Some other winners: Shor, Sénizergues, Agrawal et al., ...

ACM System Software Award 2001

Gerard J. Holzmann



SPIN book

SPIN is a popular open-source software tool, used by thousands of people worldwide, that can be used for the formal verification of distributed software systems.

Some other winners: TeX, Postscript, UNIX, TCP/IP, Java, Smalltalk

ACM Turing Award 2007

Edmund Clarke

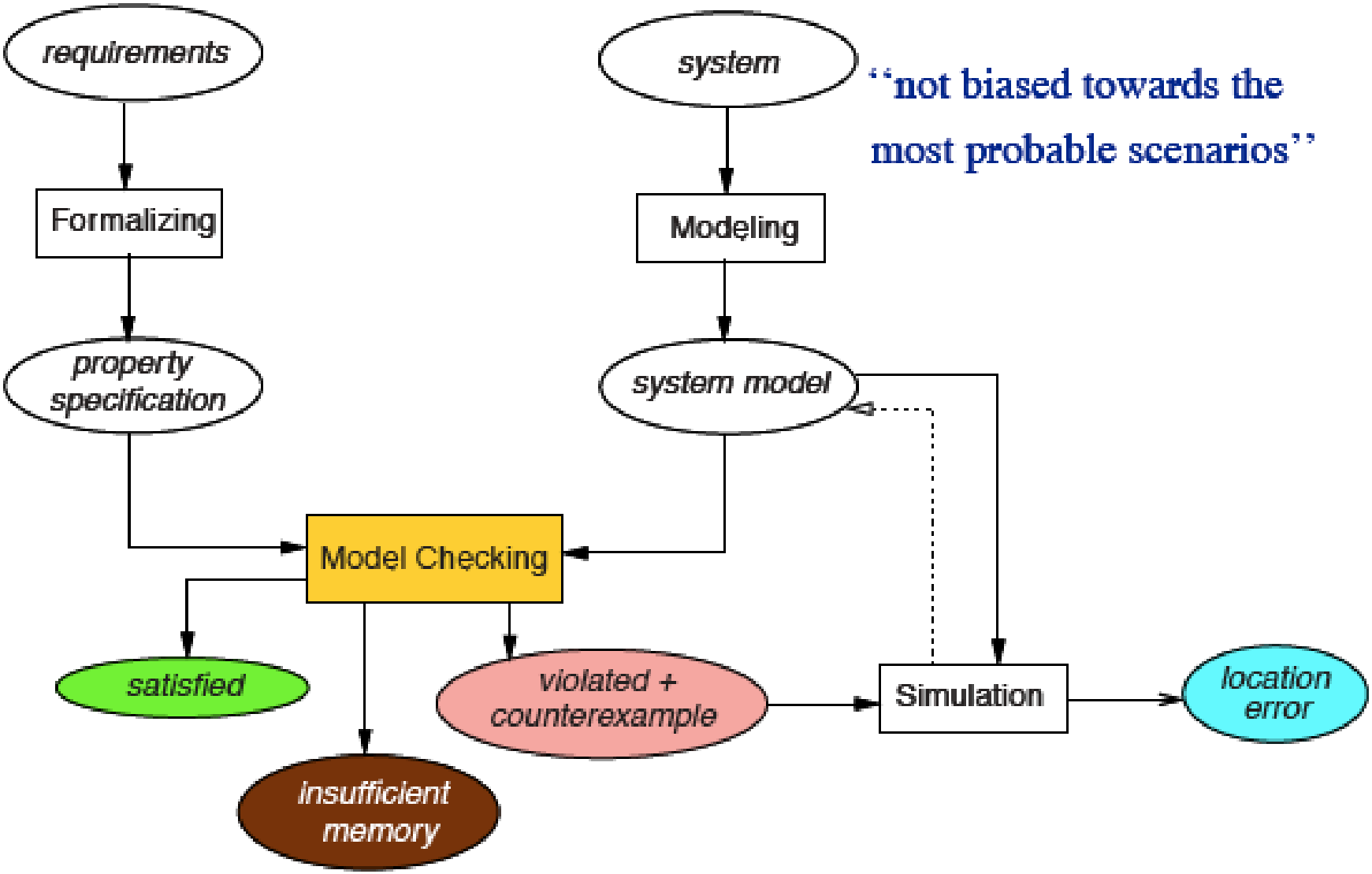
E. Allen Emerson

Joseph Sifakis

"For their role in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries."

Some other winners: Dijkstra, Cook, Hoare, Rabin and Scott

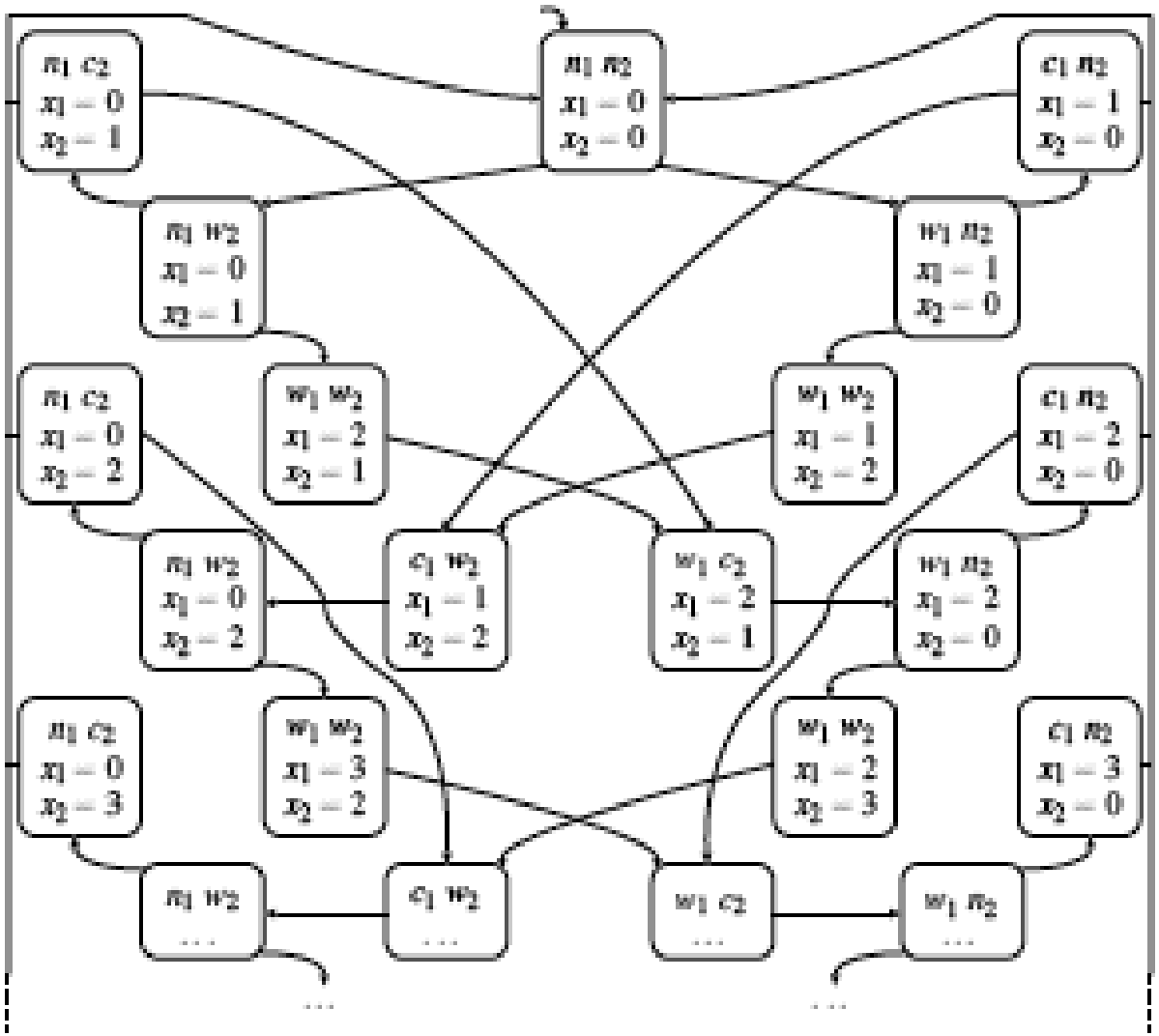
Model Checking Overview



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.

What are Models?



What are Models?

Transition systems

- States labelled with basic propositions
- Transition relation between states
- Action-labelled transitions to facilitate composition

What are Models?

Transition systems

- States labelled with basic propositions
- Transition relation between states
- Action-labelled transitions to facilitate composition

Expressivity

- Programs are transition systems
- Multi-threaded programs are transition systems
- Hardware circuits are transition systems
- Petri nets are transition systems
- .....

What are Properties?

Example properties:

- Can the system reach a deadlock situation?
- Can two processes ever be simultaneously in a critical section?
- On termination, does a program provide the correct output?
- Can the system be reset in every possible system state?

What are Properties?

Example properties:

- Can the system reach a deadlock situation?
- Can two processes ever be simultaneously in a critical section?
- On termination, does a program provide the correct output?
- Can the system be reset in every possible system state?

Temporal logic:

- Propositional logic
- Modal operators such as $\Box$ “always” and $\Diamond$ “eventually”
- Interpreted over infinite state sequences (linear)
- Or over infinite trees of states (branching)

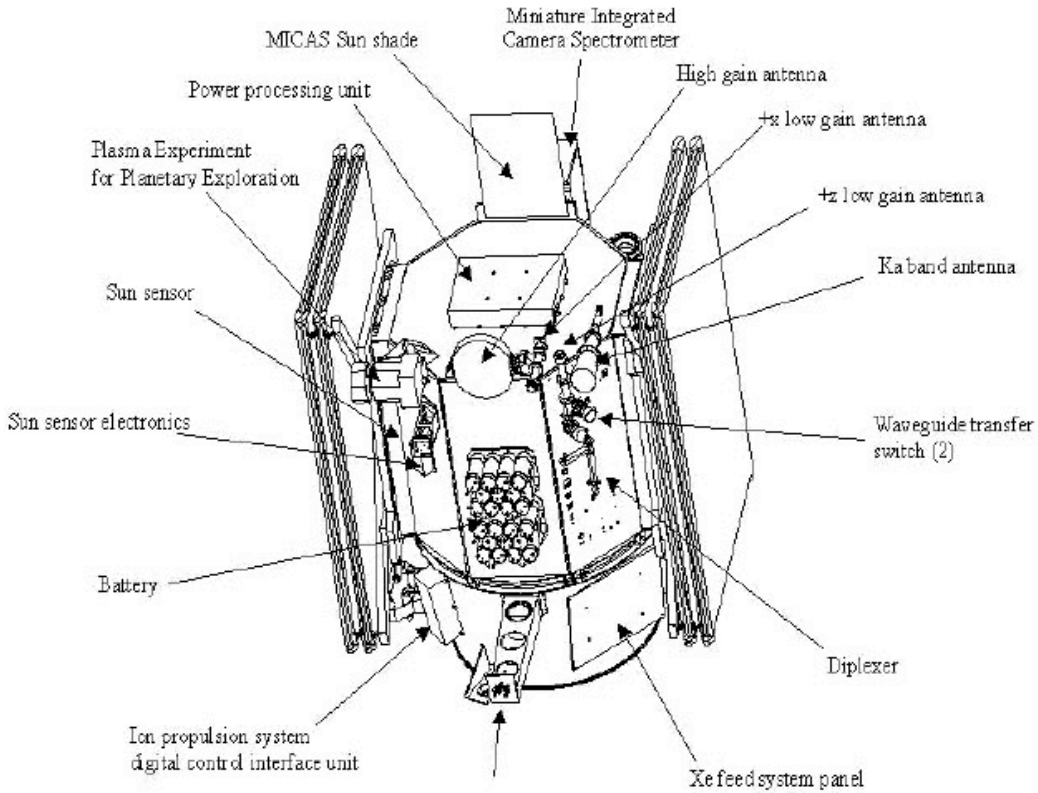
The Model Checking Problem

Let M be a model, i.e., a finite transition system.
Let φ be a formula in temporal logic, i.e., the specification.

Aim: Find all initial states s in M such that $M, s \models \varphi$.

Example: NASA's Deep Space-1 Spacecraft

Model checking has been applied to several modules of this spacecraft



launched in October 1998

A Program Snippet

process Inc = **while** *true* **do if** $x < 200$ **then** $x := x + 1$ **od**

process Dec = **while** *true* **do if** $x > 0$ **then** $x := x - 1$ **od**

process Reset = **while** *true* **do if** $x = 200$ **then** $x := 0$ **od**

Question: *is x always between (and including) 0 and 200?*

Modelling in Promela

```
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() }
```

How to Check?

Extend the model with a "monitor" process that checks $0 \leq x \leq 200$:

How to Check?

Extend the model with a "monitor" process that checks $0 \leq x \leq 200$:

```
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() }
```

Running a Model Checker

```
> spin -run -bfs program.pml
# [...]
pan:1: assertion violated ((x>=0)&&(x<=200)) (at depth 613)
```

```
> spin -p -replay program.pml
Starting Inc with pid 1
  1:    proc  0 (:init::1) program.pml:19 (state 1) [(run Inc())]
# [...]
 10:    proc  1 (Inc:1) program.pml:4 (state 1) [(1)]
 11:    proc  2 (Dec:1) program.pml:8 (state 2) [((x>0))]
 12:    proc  1 (Inc:1) program.pml:4 (state 2) [((x<200))]
 13:    proc  1 (Inc:1) program.pml:4 (state 3) [x = (x+1)]
# [...]
609:    proc  3 (Reset:1) program.pml:12 (state 2)  [((x==200))]
610:    proc  3 (Reset:1) program.pml:12 (state 3)  [x = 0]
611:    proc  3 (Reset:1) program.pml:12 (state 1)  [(1)]
612:    proc  2 (Dec:1) program.pml:8 (state 3) [x = (x-1)]
613:    proc  2 (Dec:1) program.pml:8 (state 1) [(1)]
spin: program.pml:16, Error: assertion violated
# [...]
```

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 Model Checking Process

- Modeling phase
  - model the system under consideration
  - as a first sanity check, perform some simulations
  - formalise the property to be checked

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
- Execution phase
  - run the model checker to check the validity of the property in the 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
- Execution phase
  - run the model checker to check the validity of the property in the model
- Analysis phase
  - property satisfied? → check next property (if any)

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
- Execution 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

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
- Execution 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

The Pros of Model Checking

- widely applicable (hardware, software, communication protocols, ...)
- potential "push-button" technology (software-tools)
  Uppaal, SPIN, NuSMV, CBMC, Java Pathfinder, Storm, ...
- increased usage in hardware and software industry
  Siemens, Amazon, FaceBook, Intel, Cadence, Ford, ESA, ...
- provides a counterexample if property is refuted
  model checking is an extremely effective bug-hunting technique
- sound mathematical foundations
  logic, automata, data structures and algorithms, complexity
- unlike testing, not biased to the most probable scenarios

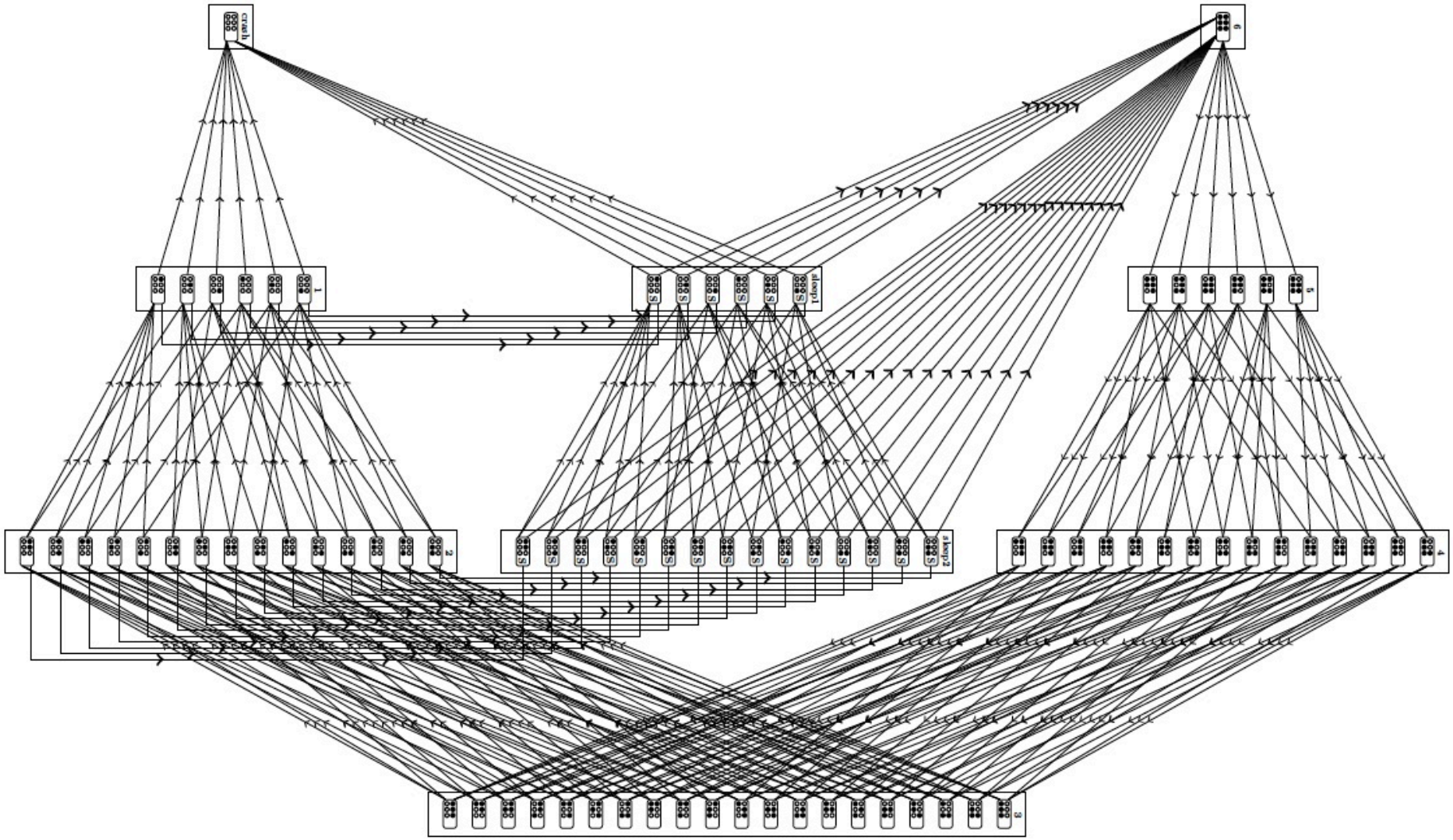
The Cons of Model Checking

- main focus on control-intensive applications (less data-oriented)
- model checking is only as "good" as the system model
- the state-space explosion problem
- mostly not possible to check generalisations

Nevertheless:

> Model checking is a very effective technique
> to expose potential design errors

State Spaces Can Be Gigantic

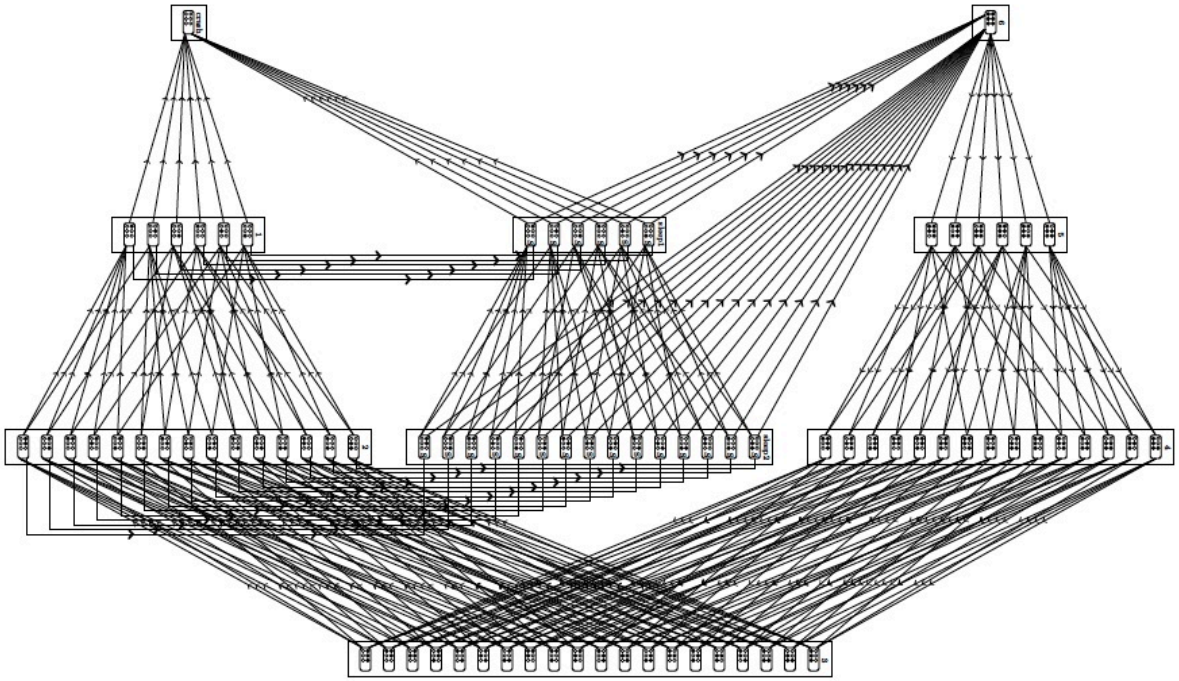


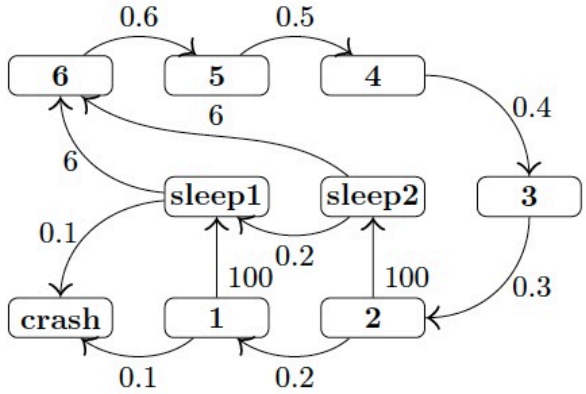
A model of the Hubble telescope

Treating Gigantic Models?

- Use compact data structures
- Make models smaller prior to (or: during) model checking
- Try to make them even smaller
- If possible, try to obtain the smallest possible model
- While preserving the properties of interest
- Do this all algorithmically and possibly fast

Abstraction





	Gigantic	versus	smallest
Is a crash state reachable?	✔		✔
Is a failure repaired on time?	✖		✖

Overview

Striking Model-Checking Examples

- Security: Needham-Schroeder encryption protocol
  - error that remained undiscovered for 17 years unrevealed

Striking Model-Checking Examples

- Security: Needham-Schroeder encryption protocol
  - error that remained undiscovered for 17 years unrevealed
- Transportation systems
  - train model containing 10^{476} states

Striking Model-Checking Examples

- Security: Needham-Schroeder encryption protocol
  - error that remained undiscovered for 17 years unrevealed
- 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

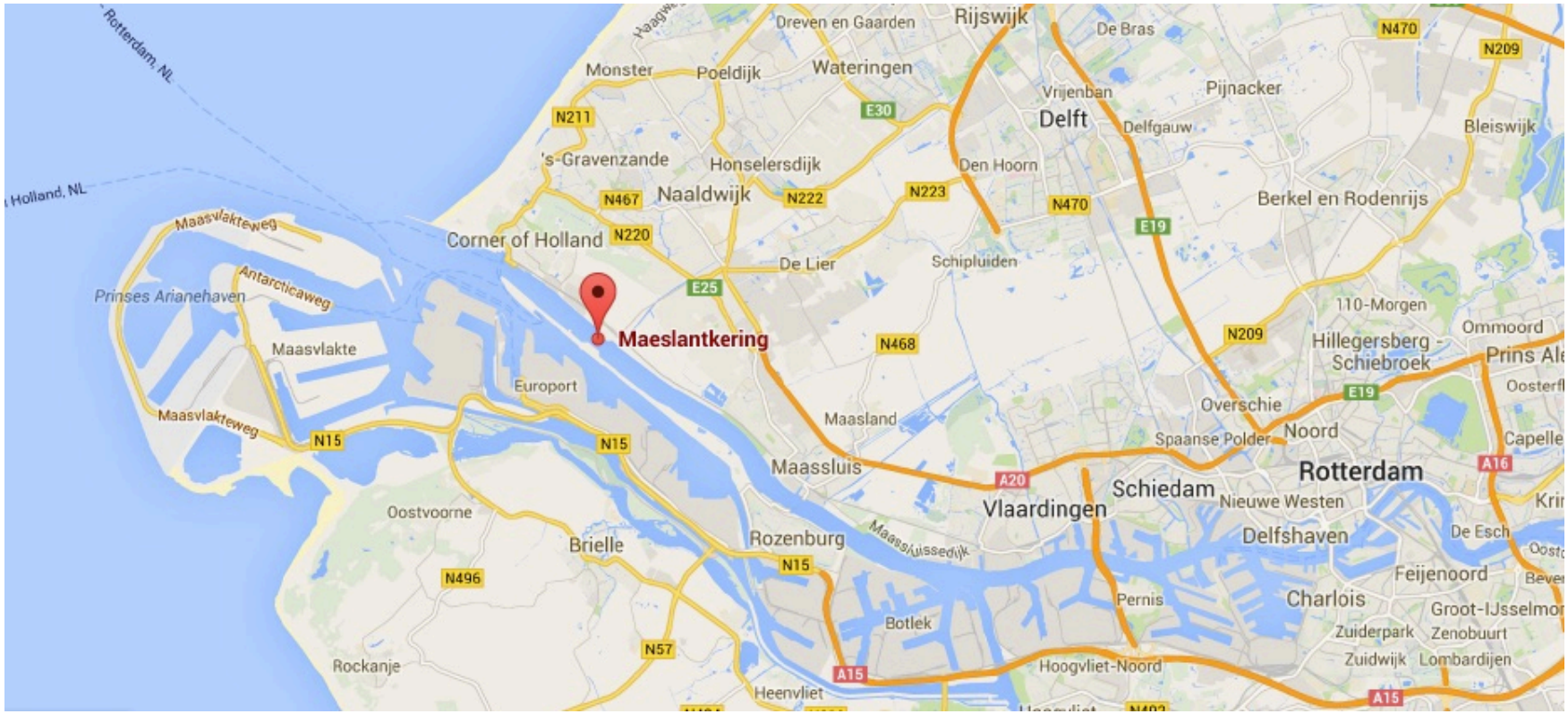
Striking Model-Checking Examples

- Security: Needham-Schroeder encryption protocol
  - error that remained undiscovered for 17 years unrevealed
- 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
- Dutch storm surge barrier in Nieuwe Waterweg

Striking Model-Checking Examples

- Security: Needham-Schroeder encryption protocol
  - error that remained undiscovered for 17 years unrevealed
- 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
- Dutch storm surge barrier in Nieuwe Waterweg
- Software in the space missiles
  - NASA's Mars Curiosity Rover, Deep Space-1, Galileo
  - LARS group@Jet Propulsion Lab

Storm Surge Barrier Maeslantkering



Storm Surge Barrier Maeslantkering

Storm Surge Barrier Maeslantkering

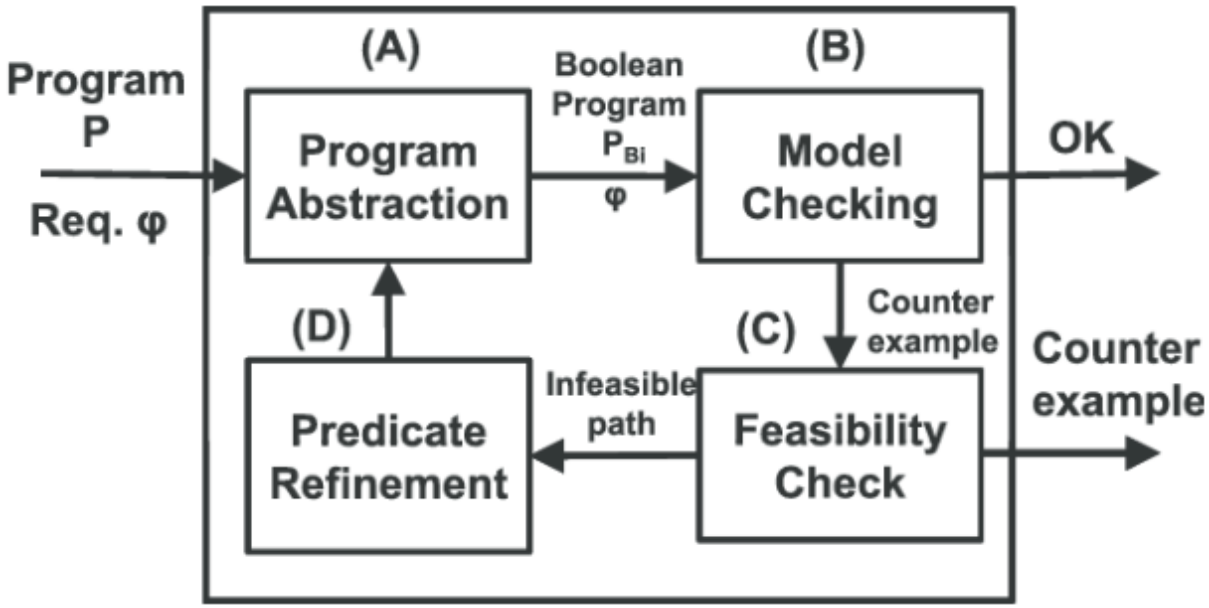
[Kars, Formal Methods in the Design of a Storm Surge Barrier Control System, 1996]

Checking Device Drivers

- 85% of system crashes of Windows XP caused by bugs in third-party kernel-level device drivers (2003)
- Main reason: complexity of the Windows drivers API
- SLAM model checker: automatically checks device drivers for certain correctness properties with respect to the Windows device drivers API
- Nowadays core of Static Driver Verifier (SDV), a tool-set for drivers developers

How to Model Check Device Drivers?

- Abstract C programs into Boolean programs
- Apply iterative abstraction-refinement scheme (CEGAR, see below)
- Key: recursive procedure calls (push-down automata)
- Symbolic model checking (binary decision diagrams)
- Points-to analysis + temporal safety properties (monitor)

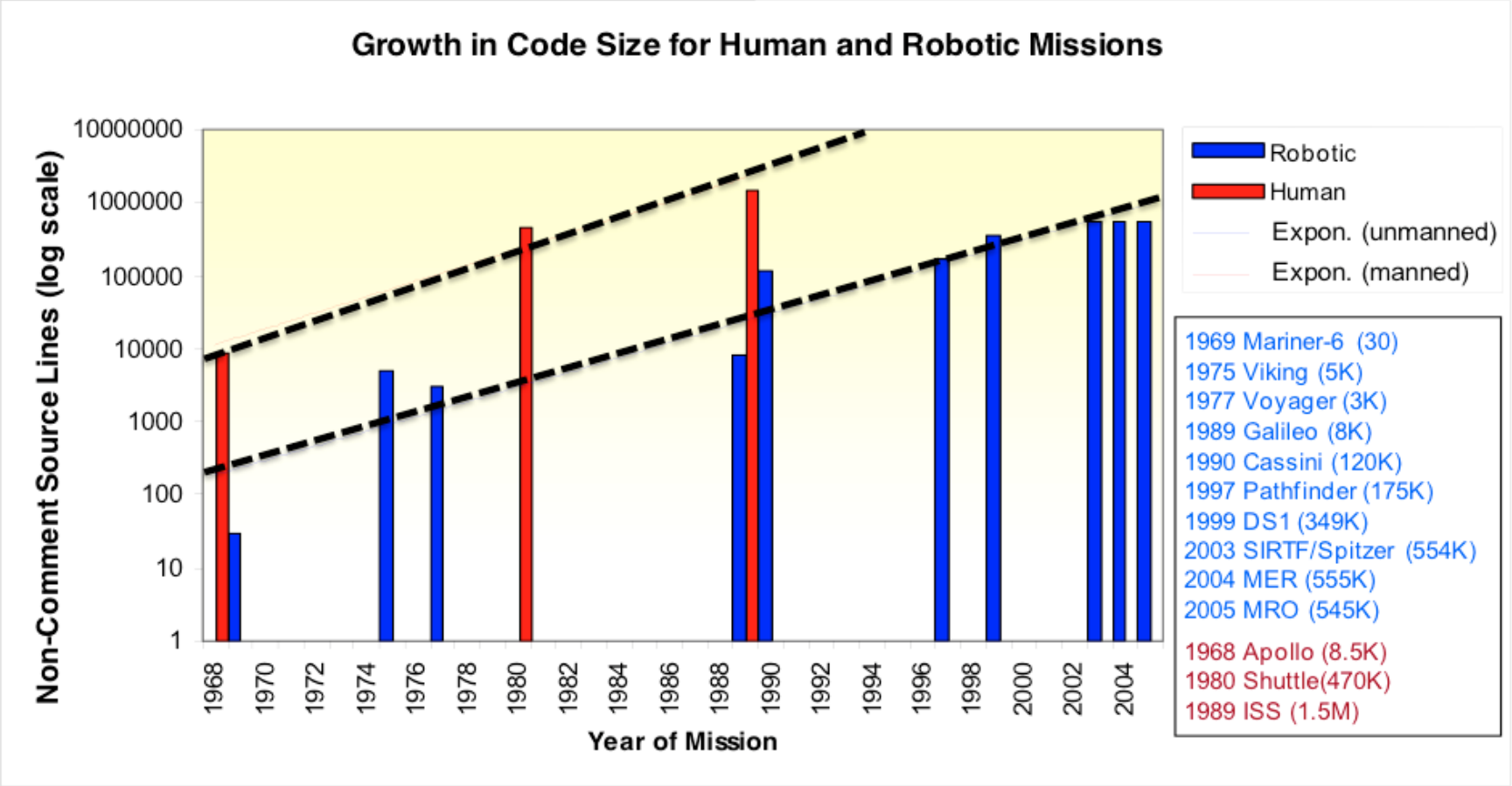


Checking Device Drivers

During development of Windows 7, 270 real bugs found
in 140 device drivers (of ≤ 30,000 lines of code) with SLAM

[Ball et al., A decade of software model checking with SLAM, 2011]

Spacecraft := Flying Software



NASA Study Flight Software Complexity (2009)

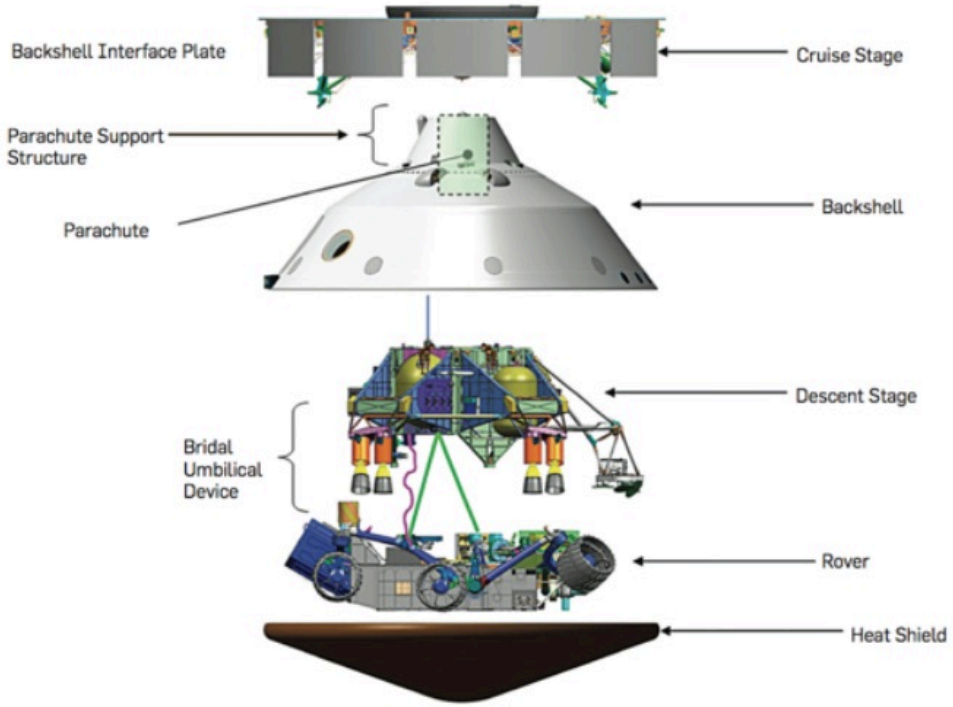
The NASA Curiosity Rover

Software in Space:

- Extremely high reliability requirements are imposed
  - Any small mistake can lead to the loss of a mission
- Extraordinary measures taken in both hardware and software design
  - system debugging and repair from millions of miles away
- Model checking verified intricate software subsystems for absence of races and deadlocks

Mars Rover Landing

The most critical part of the mission



Controlled by one of two computers allocated within the body of the rover.

Model Checking of Mars Rover

Despite 145 code reviews, model checking of critical parts (e.g., file system) with SPIN revealed several subtle concurrency flaws.

Model checking was used in the design loop: performed routinely after every change in the code of the file system.

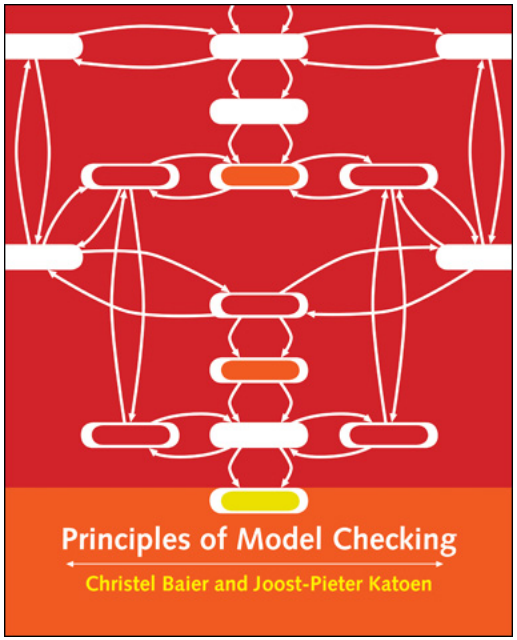
[Holzmann, Mars Code,2014]

Overview

Course Content

- What are transition systems and properties?
- Model checking linear temporal logic
  automata on infinite words, regular properties, complexity
- Model checking branching-time logic
  CTL, expressiveness CTL versus LTL, recursive descent
- How to make models smaller?
  - bisimulation minimisation, simulation, partial-order reduction, CEGAR
- Symbolic model checking
  - binary decision diagrams, bounded model checking

Course Material



Principles of Model Checking:

Christel Baier

TU Dresden, Germany

Joost-Pieter Katoen

RWTH Aachen University, Germany

Gerard J. Holzmann, NASA JPL:

"This book offers one of the most comprehensive introductions to logic model checking techniques available today. The authors have found a way to explain both basic concepts and foundational theory thoroughly and in crystal clear prose."

Lectures

Lecture:

- Mon 10:30 – 12:00, Thu 12:30 – 14:00 both at AH II
- Recordings online

Material:

- Lecture slides are made available on RWTH Moodle
- Many copies of the book are available in the CS library

Website:

`https://moodle.rwth-aachen.de/course/view.php?id=24681`

Contact:

`mc22@i2.informatik.rwth-aachen.de`

Exercises and Examination

Exercise classes:

- Wed 16:30 - 18:00 (AH V, start: April 20)
- Instructors: Alexander Bork and Jip Spel

Weekly exercise series:

- Intended for groups of three to four students
  - Find group-mates in RWTHmoodle
- New series: every Wed in RWTHmoodle (start: April 13)
- Solutions: Wed (before 16:30) one week later via RWTHmoodle
- Participation to exercises strongly encouraged
- Starred exercises are example exam questions

Examination:

- July 28 and September 6 (written exam)
- No particular pre-requisites for exam participation

Course Prerequisites

Aim of the course:

It's about the theoretical foundations of model checking.

Not its practical usage.

Prerequisites:

- Automata and language theory
- Algorithms and data structures
- Computability and complexity theory
- Mathematical logic

Related Courses

- Modelling and verification of probabilistic systems (Katoen)
- Probabilistic programming (Katoen)
- Infinite computations and games (Löding)
- Satisfiability checking (Abráhám)
- Modelling and analysis of hybrid systems (Abráhám)
- Theoretical foundations of the UML (Katoen)
- Semantics and verification of software (Noll)

As well as various master and bachelor theses

Questions?

Questions?

Next Lecture: Monday April 11, 10:30

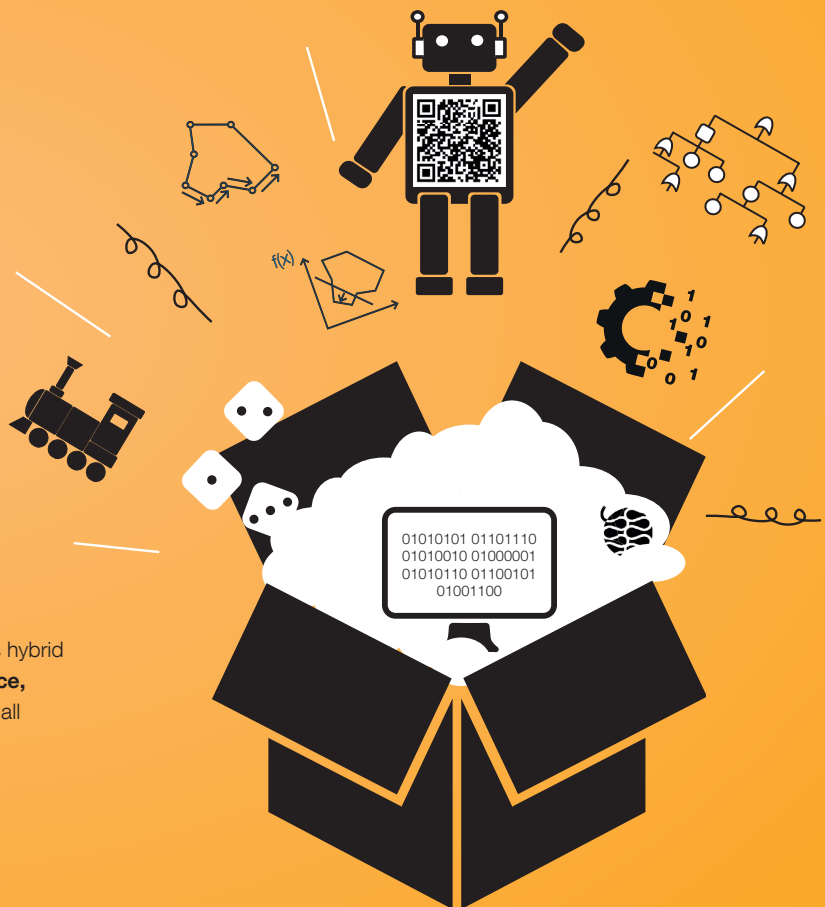
UnRAVeL Research Training Group – Uncertainty and Randomness in Algorithms, Verification, and Logic

RWTH AACHEN UNIVERSITY

UnRAVeL - What's new?

Research in Progress: a Survey Lecture

19.04. Signal Processing on Graphs and Complexes
26.04. Acceptance of Driverless Trains
03.05. Improving Automatic Complexity Analysis of Probabilistic and Non-Probabilistic Integer Programs
10.05. Graph Representations Based on Homomorphisms
17.05. A Branch & Bound Algorithm for Robust Binary Optimization with Budget Uncertainty
24.05. The Challenge of Compositionality for Stochastic Hybrid Systems
14.06. Constructing ω-Automata from Examples
21.06. Uncertainty Bounds for Gaussian Process Regression with Applications to Safe Control and Learning
28.06. Stackelberg Network Pricing Games
05.07. Tractable Reasoning in First-Order Knowledge Bases



In these talks, UnRAVeL professors will present their recently published paper. The survey lecture will take place as hybrid event. Each talk will be given in person, **Tuesdays from 16:30 to 18:00** in the **Department of Computer Science, Ahornstr. 55, building E2 (no. 2356)**, **ground floor, room: B-IT 5053.2.** In addition, you can join via Zoom. For all information, please scan the QR code on the right or visit **https://www.unravel.rwth-aachen.de/cms/~pzix.**

Thinking the Future
Zukunft denken