

# Introduction to Model Checking

Lecture # 1: Motivation, Background, and Course Organization

Prof. Joost-Pieter Katoen

Chair Software Modeling and Verification



October 14, 2013

# The Importance of Software Correctness

Rapidly increasing **integration of ICT** in different applications

- embedded systems
- communication protocols
- transportation systems

⇒ reliability increasingly depends on software!

Defects can be **fatal** and extremely **costly**

- products subject to mass-production
- safety-critical systems

# What is System Verification?

## Folklore “definition”

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”

## 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

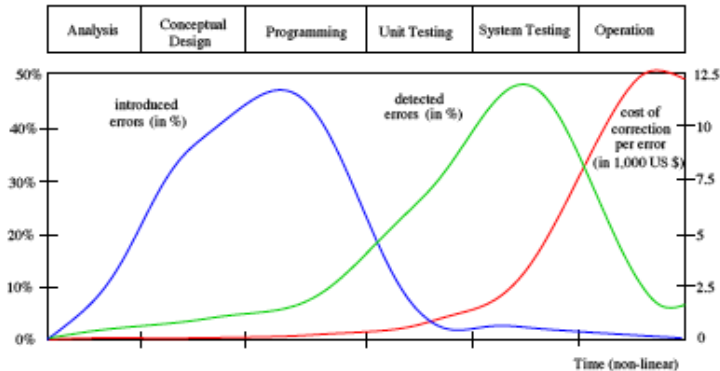
## Testing

- 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 defects per 1,000 code lines

# Bug Hunting: the Sooner, the Better



## Intuitive description

Formal methods are the

“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 by IEC, FAA, and NASA for safety-critical software

# Formal Verification Techniques for Property $P$

## Deductive methods

- method: provide a formal **proof** that  $P$  holds
- tool: theorem prover/proof assistant or proof checker
- applicable if: system has form of a mathematical theory

## Model checking

- method: **systematic check** on  $P$  in all states
- tool: model checker (SPIN, NUSMV, UPPAAL, ...)
- applicable if: system generates (finite) behavioural model

## Model-based simulation or testing

- method: test for  $P$  by **exploring possible behaviours**
- applicable if: system defines an executable model

# 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

## About testing ...

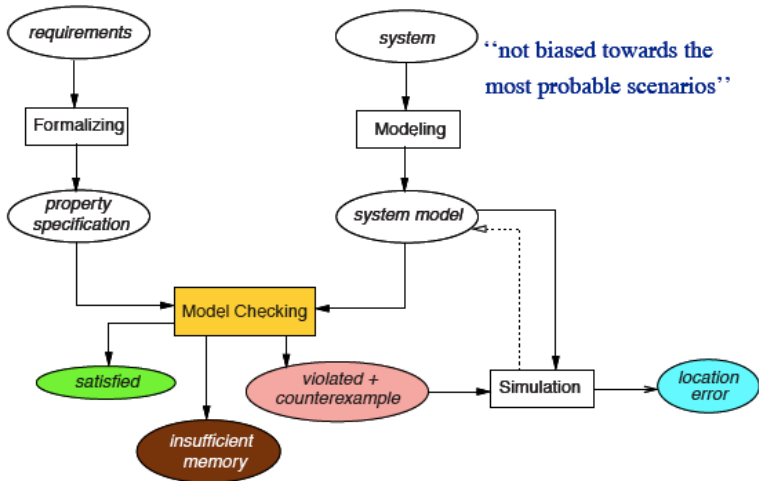
testing/simulation can show the presence of errors, **not** their absence



# Milestones in Formal Verification

- **Mathematical program correctness** (Turing, 1949)
- **Syntax-based technique for sequential programs** (Hoare, 1969)
  - for a given input, does a computer 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**
  - model-based instead of proof-rule based approach
  - does the concurrent program satisfy a given (logical) property?

# 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, ...



Moshe Vardi



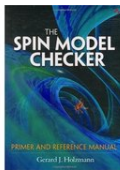
Pierre Wolper

“For work on model checking with finite automata.”

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



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



Edmund Clarke



E. Allen Emerson

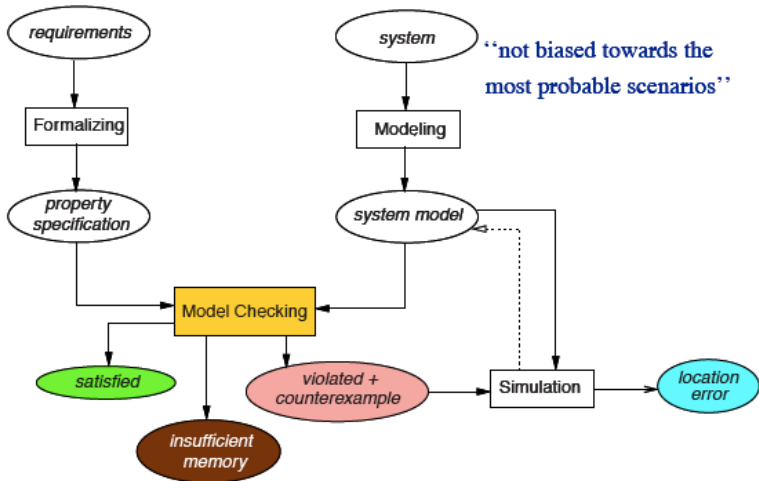


Joseph Sifakis

“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



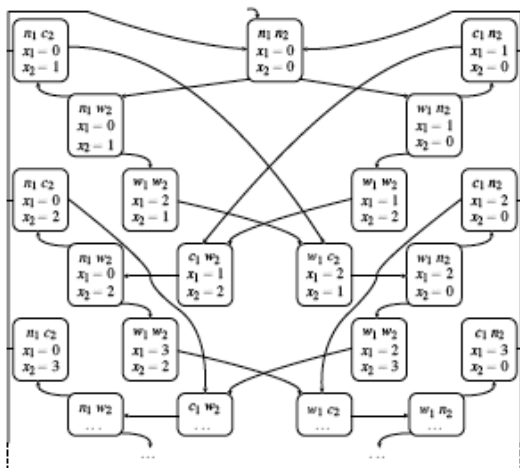
# What is Model Checking?

## Informal description

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

## Expressivity

- Programs are transition systems
- Multi-threading programs are transition systems
- Communicating processes are transition systems
- Hardware circuits are transition systems
- What else?

# 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?

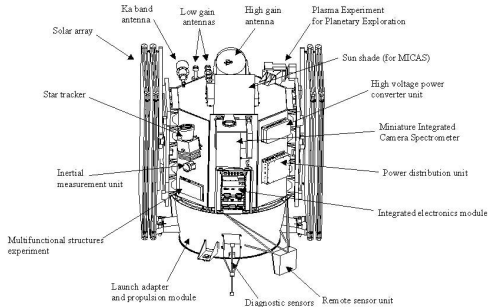
## Temporal logic

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

# NASA's Deep Space-1 Spacecraft

## Model checking

has been applied to several modules of this spacecraft



launched in October 1998

# A Small Program Fragment

```
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
```

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

# 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

# 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 counterexample is provided
- sound and interesting mathematical foundations
- not biased to the most possible scenarios (such as testing)

# The Cons of Model Checking

- main focus on **control-intensive** applications (less data-oriented)
- model checking is only as “good” as the system model
- no guarantee about **completeness** of results
- impossible to check **generalisations** (in general)

Nevertheless:

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



# 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 current/next generation of space missiles**
  - NASA's Mars Pathfinder, Deep Space-1, JPL LARS group

## What are appropriate **models**?

- transition systems
- from programs to transition systems
- from circuits to transition systems
- multi-threading, communication, ...
- nanoPromela: an example modeling language

## What are **properties**?

- safety: “something bad never happen”
- liveness: “something good eventually happens”
- fairness: “if something may happen frequently, it will happen”

## How to check **regular** properties?

- finite-state automata and regular safety properties
- Büchi automata and  $\omega$ -regular properties
- model checking: nested depth-first search

## How to express properties **succinctly**?

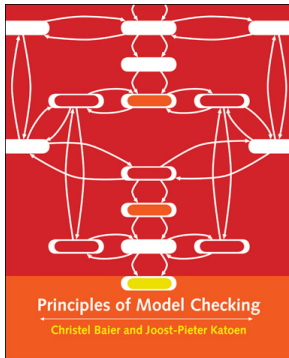
- Linear-time Temporal Logic (LTL): syntax and semantics
- What can be expressed in LTL?
- LTL model checking: algorithms, complexity
- How to treat fairness in LTL

## How to express properties **succinctly**?

- Computation Tree Logic (CTL): syntax and semantics
- What can be expressed in CTL?
- CTL model checking: algorithms, complexity
- How to treat fairness in CTL

## How to make models smaller?

- Equivalences and pre-orders on transition systems
- Which properties are preserved?
- Minimization algorithms



## Principles of Model Checking

CHRISTEL BAIER

TU Dresden, Germany

JOOST-PIETER KATOEN

RWTH Aachen University, Germany

Gerard J. Holzmann, NASA JPL, Pasadena:

*"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."*

## Aim of the course

It's about the **theoretical foundations** of model checking, not its usage!

## Prerequisites

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