Computer-Aided Verification

Dave Parker

University of Birmingham

2017/18

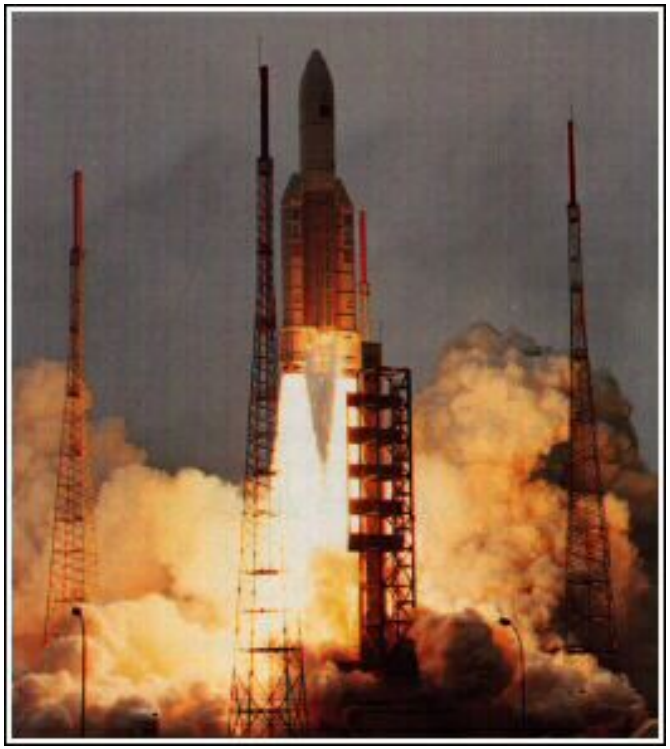
- Formal verification
  - is the application of rigorous, mathematical methods to establish the correctness of computerised systems
  - "system" = software, hardware, protocol, …
- Computer-aided verification
  - automated verification: tools, algorithms, …

Overview

- Motivation
  - why verify?
- Computer-aided verification
  - model checking
  - example
- This module
  - syllabus
  - aims, delivery, resources, …

Ariane 5

- ESA (European Space Agency) Ariane 5 launcher
  - shown here in maiden flight on 4th June 1996



- 37secs later self-destructs
  - numerical overflow in a conversion routine (64-bit float to 16-bit int)
  - results in incorrect altitude sent by the on-board computer
  - exception handling disabled
- Expensive, embarrassing…
  - more than $500 million

Toyota Prius

- Toyota Prius
  - first mass-produced hybrid vehicle

- February 2010
  - software “glitch” in anti-lock braking system
  - in response to numerous complaints/accidents
  - eventually fixed via software update
  - 185,000 cars recalled, at huge cost
  - much criticism of handling, significant bad publicity

- Further recalls in 2015
  - 625,000 Toyota hybrids recalled due to a software "glitch"
  - can cause loss of power while driving

Correctness is crucial

- What do the previous examples have in common?
  - failures/errors in programmable computerised devices
  - high costs incurred (and not just financial)
  - failures were avoidable

- Trends in computing
  - computerised devices are everywhere…
  - including business-, safety-critical domains
    - avionic/automotive embedded software, driverless cars, medical sensors/devices
  - software is increasingly complex
  - and increasingly interconnected

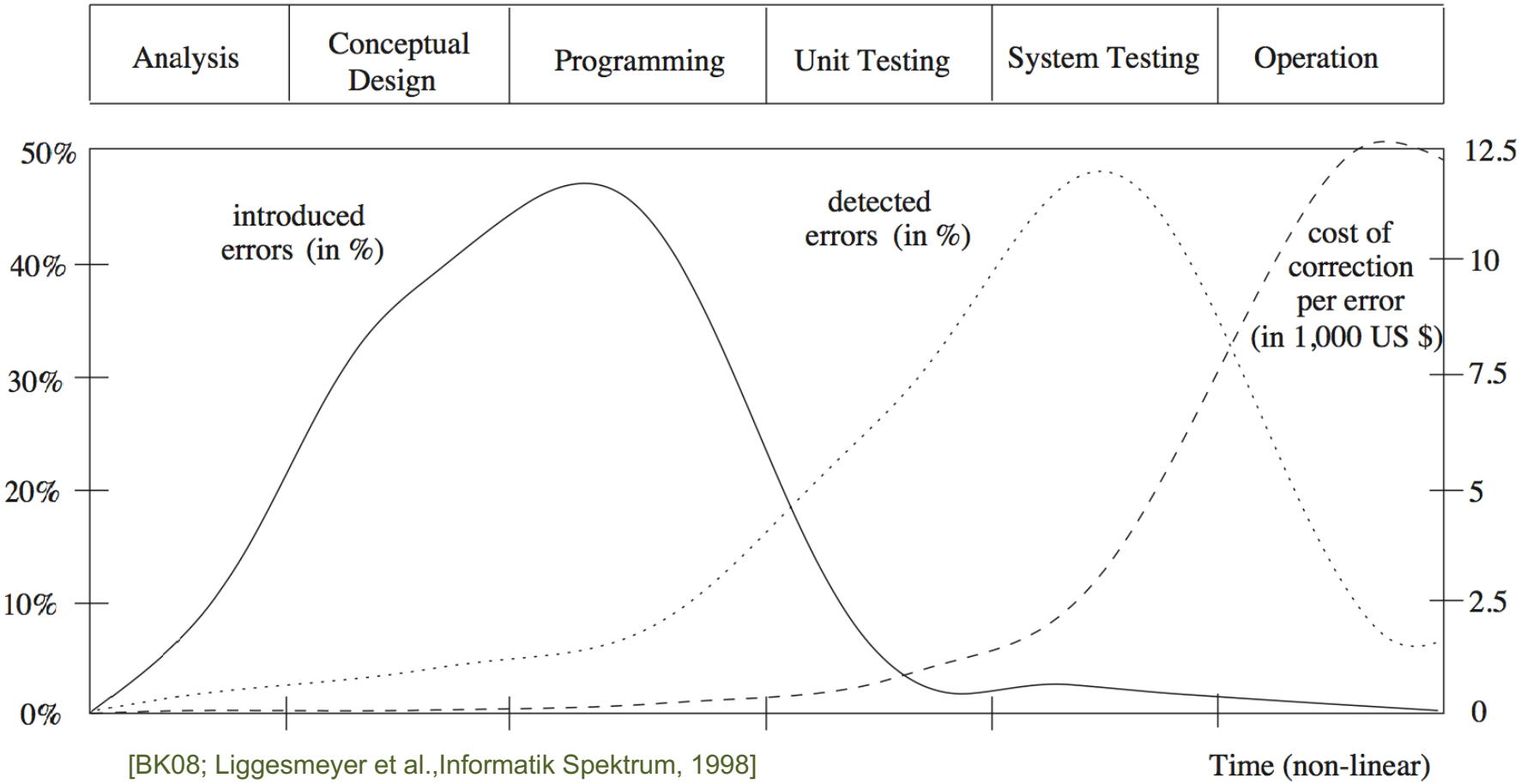
How to ensure correctness?

- Existing methods for checking (software) correctness:

- Code review
  - peer review of code by other programmers
  - static, manual analysis during/after development
- Testing
  - program execution against test suite of known results
  - dynamic analysis performed after development

- Analysis types:
  - dynamic = based on execution of system
  - static = examination of source code/design (no execution)

When to check correctness?



[BK08; Liggesmeyer et al.,Informatik Spektrum, 1998]

Formal verification

- Basic idea:
  - check system correctness using formal, logical reasoning
  - build a mathematical model of a system and then prove that it satisfies a formal specification of correctness

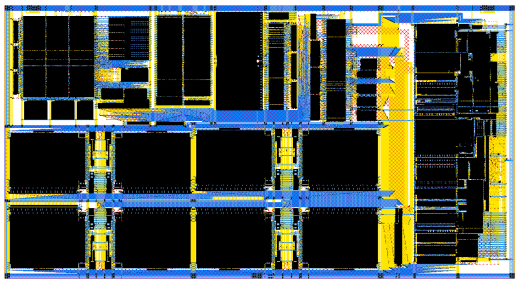
- Key idea: "Testing can only show the presence of errors, not their absence"

Edsger Dijkstra
1930-2002

- Key points:
  - static analysis (based on source code or system design)
  - opportunities for early integration in the design process
  - exhaustive (higher coverage than testing alone)
  - verification, not validation
  - or, help to identify subtle bugs/flaws

Automatic verification

- Formal verification
  - essentially: proving that a program satisfies it specification
  - i.e., that all possible system executions behave correctly
  - many techniques: manual proof, automated theorem proving, static analysis, model checking, …

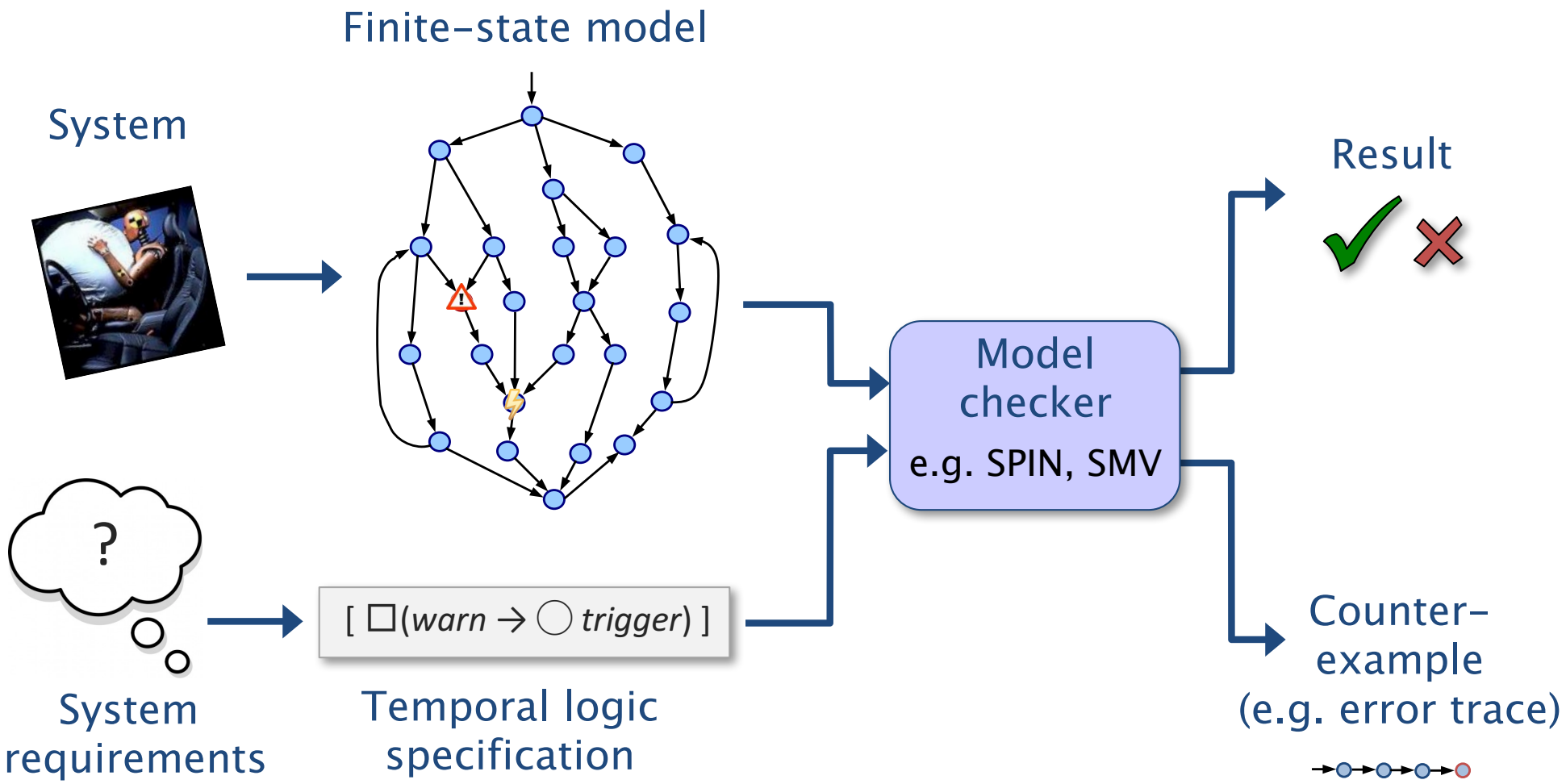


$10^{500,000}$ states

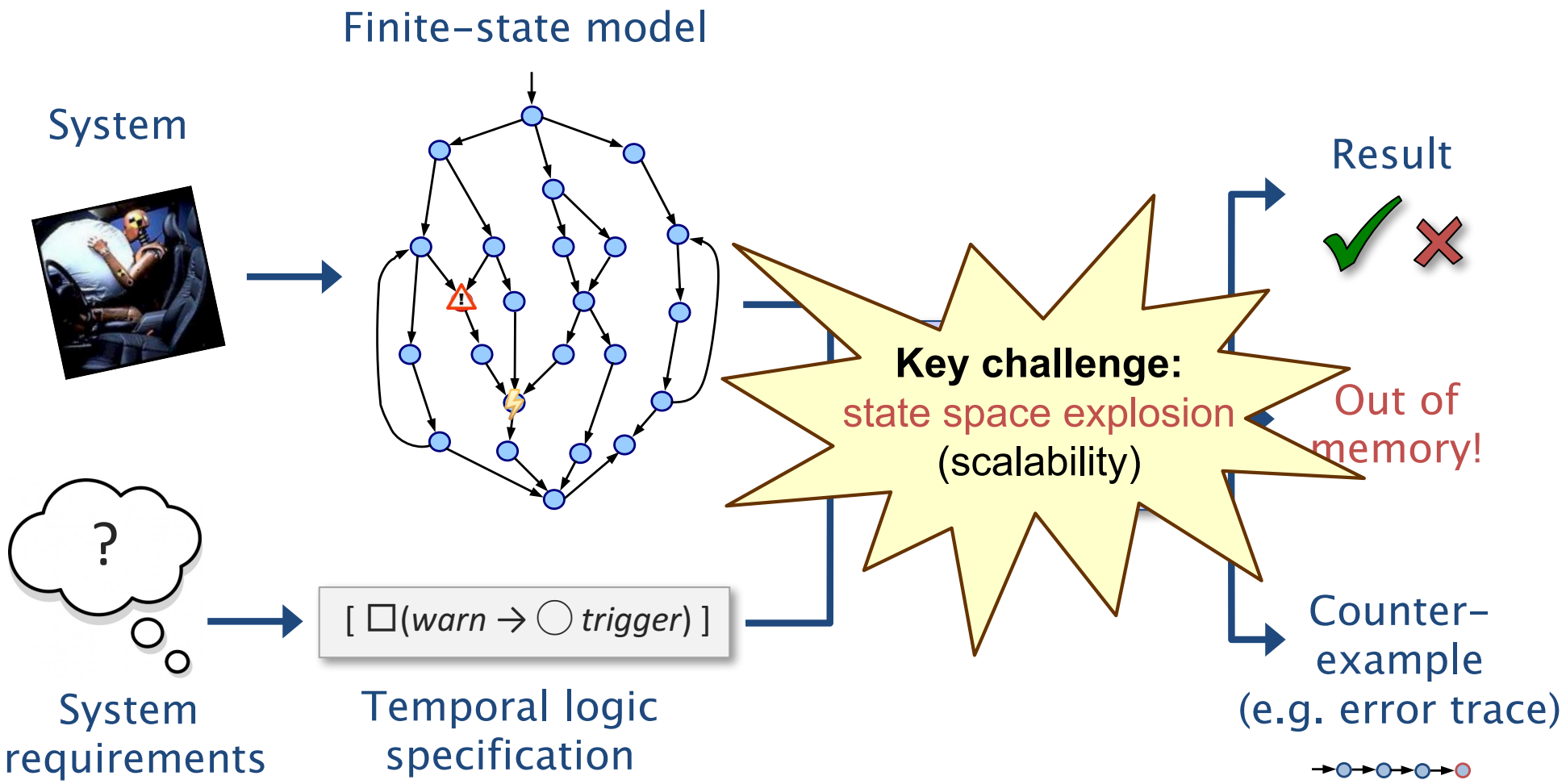
10^{70} atoms

- Computer-aided (automatic) verification
  - push-button technology: algorithms, tools
  - in particular: model checking

Verification via model checking



Verification via model checking



Why study verification?

- Topic of growing importance
  - increasing need for correctness + advances in technology
- Many successful applications in practice
  - NASA: Deep Space 1, Mars Pathfinder
  - Microsoft: SLAM project (device drivers)
  - security: Needham-Schroeder public key protocol (Lowe + FDR)
  - also: hardware industry, avionics, Facebook, …
- Major research area
  - e.g. Turing award for Clarke/Emerson/Sifakis in 2007
- What's involved?
  - automata theory, graph algorithms, logic, data structures, software tools

Example

- A simple concurrent program:

```
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
```

- Specification:
  - variable x always remains in the range {0,1,…,200}

- Is this true?

(example from [BK08])

Example – modelling

- Modelled in PROMELA
  - (input language for SPIN model checker)

```
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()
}
```

Example – specification

- Add a "monitor" process that checks whether $0 \leq x \leq 200$

```
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()
}
```

Example – model checking

- Analysis using the SPIN model checker
  - does the assertion always hold?

- No…

```
.....................
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
```

Example – counterexample

- Counterexample trace produced by the model checker:

```
.............
605:   proc 1 (Inc)                                  [((x<200))]
606:   proc 1 (Inc)                                  [x = (x+1)]
607:   proc 2 (Dec)                                  [((x > 0))]
608:   proc 1 (Inc)                                  [(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)))
```

Module syllabus

- Modelling sequential and parallel systems
  - labelled transitions systems, parallel composition
- Temporal logic
  - LTL, CTL and CTL*, etc.
- Model checking
  - CTL model checking algorithms
  - automata-theoretic model checking (LTL)
- Verification tools: SPIN
- Advanced verification techniques
  - bounded model checking via propositional satisfiability
  - (symbolic execution), (symbolic model checking)
- Quantitative verification
  - real-time systems, probabilistic systems

Module aims & focus

- Aims of the module:
  - introduce the basic ideas of automatic verification
  - familiarise you with key techniques & algorithms for verification
  - illustrate uses & applications of automatic verification
  - give practical experience in state of the art verification software
  - provide a foundation for further study in the area of verification

- Mix of: theory + algorithms + tools
  - no programming, but use of modelling languages

- Prerequisites
  - background in: propositional logic, automata, graph algorithms

Module delivery

- Lectures:
  - Tue 11-12:
    - Lecture Room 7, Arts Building
  - Thur 12-1:
    - G29, Mechanical and Civil Engineering

- Tutorials (feedback on exercises) (not all weeks):
  - Thur 4-5 (surnames A-L, by default):
    - UG06, Murray Learning Centre
  - Fri 10-11 (surnames M-Z, by default):
    - Lecture Theatre 1, Sports and Exercise Sciences

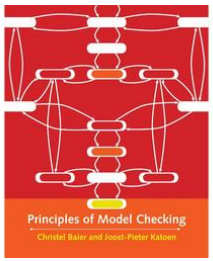
Assessment

- Split:
  - 80% exam (1.5hr, in the summer)
  - 20% continuous assessment

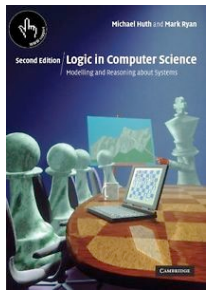
- Continuous assessment
  - 4 exercises (1st is formative; 2-4 are assessed: 6%/8%/6%)
  - 1 week for each: due Thurs of weeks 3, 5, 8, 11
  - 1 extra assessed exercise for "extended" version (due week 10)
  - submitted through Canvas
  - solutions worked through in tutorial sessions

Reference material

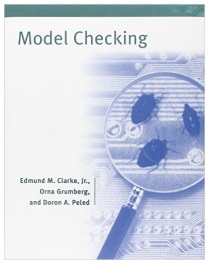
- Useful books



Principles of Model Checking
Christel Baier and Joost-Pieter Katoen [BK08]
The MIT Press, 2008



Logic in Computer Science:
Modelling and reasoning about systems
Michael Huth and Mark Ryan
Cambridge University Press, 2004



Model Checking
Edmund M. Clarke, Orna Grumberg, Doron Peled
2nd edn., MIT Press, 2000

- Links to further papers/tutorials will be added to Canvas

Resources

- Canvas
  - https://canvas.bham.ac.uk/courses/27245
  - lecture slides/videos, links, assessments, announcements
- Facebook group
  - https://www.facebook.com/groups/bham.cav.1718
  - questions, discussion, announcements
- Office hours
  - room 133 (see my door/webpage for times)

Next lecture

- Modelling sequential and parallel systems
  - labelled transition systems
  - parallel composition
  - see Chapter 2 of [BK08]