Lecture 1
Introduction

Prof. Alessandro Abate

Department of Computer Science
University of Oxford

Course information

- Prerequisites/background
  - basic computer science/maths, no probability knowledge assumed, logical underpinnings recalled
- Lectures/Exercises
  - 20 lectures, all in person
  - Usually Mon AM-Tue PM-Wed midday
  - but also Thu midday available
- Assessment
  - short take-home assignment in June, to be submitted electronically to me
- My contacts: aabate@cs.ox.ac.uk

Course outline

- Discrete-time Markov chains (DTMCs) and their properties
- Temporal logics: LTL, CTL, etc.
- Probabilistic temporal logics: PCTL, (p)LTL, etc.
- PCTL model checking for DTMCs
- PRISM model checker
- Costs & rewards

- Continuous-time Markov chains (CTMCs)

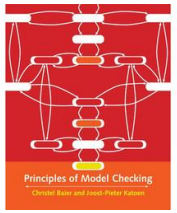
- Markov decision processes (MDPs)
- Probabilistic Verification and Strategy synthesis

- Statistical MC, Frontiers methods and Current research

Further information

- Course lecture notes are self-contained

- For further reading material…
  - two online tutorial papers also cover a lot of the material
    - Stochastic Model Checking
      Marta Kwiatkowska, Gethin Norman and David Parker
    - Automated Verification Techniques for Probabilistic Systems
      Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, David Parker
  - material is based on (mostly, Chapter 10 of):



Principles of Model Checking
Christel Baier and Joost-Pieter Katoen
MIT Press, 2008

  - Various material and examples also appear courtesy of BK08
  - PRISM web site: http://www.prismmodelchecker.org/

Probabilistic model checking

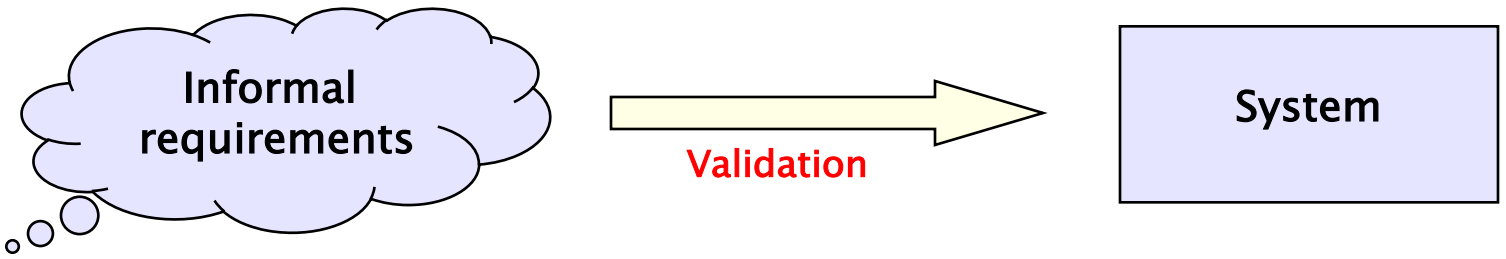
- Probabilistic model checking…
  - is a formal verification technique for modelling and analysing systems that exhibit probabilistic behaviour
- Formal verification…
  - is the application of rigorous, mathematics-based techniques to establish the correctness of computerised systems

Outline

- Introducing probabilistic model checking…
- Topics for this lecture
  - the role of automatic verification
  - what is probabilistic model checking?
  - why is it important?
  - where is it applicable?
  - what does it involve?
- About this course
  - aims and organisation
  - information and links

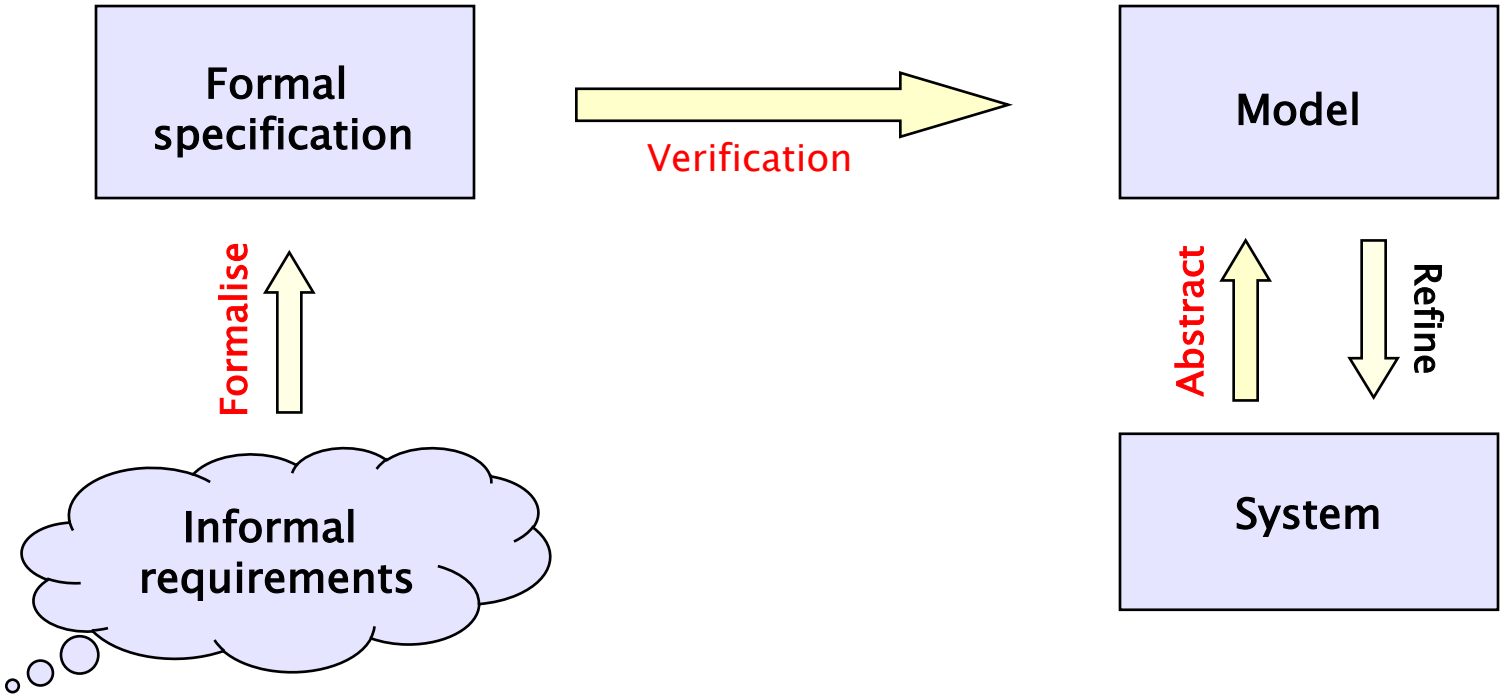
Conventional software engineering

- From requirements to software system
  - apply design methodologies
  - code directly in programming language
  - validation via testing, code walkthroughs
  - finding errors does not prove their absence



Rigorous software engineering

- From requirements to formal specification
  - formalise specification
  - derive model of system
  - formally verify correctness



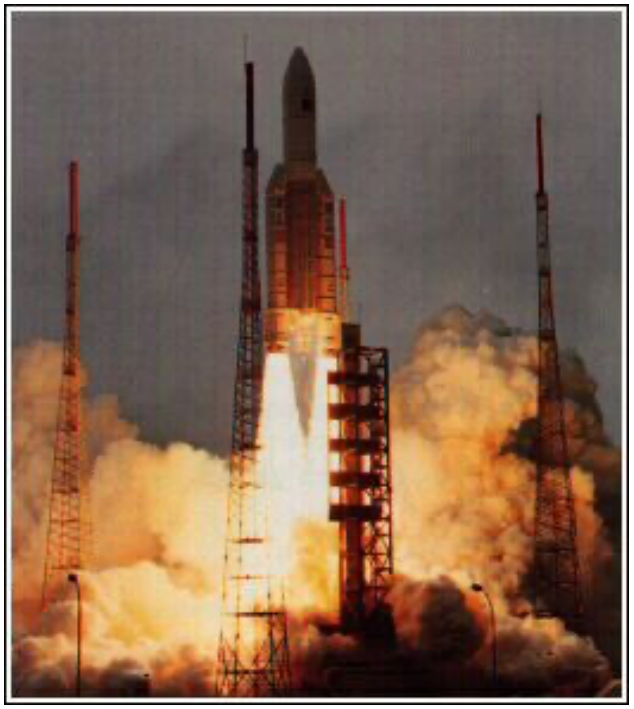
  - paradigm applies beyond SW, to V&V Engineering problems

But my program works!

- True, there are many successful large-scale complex computer systems…
  - online banking, electronic commerce
  - information services, online libraries, business processes
  - supply chain management
  - mobile phone networks
- Yet many new potential application domains with far greater complexity and higher expectations
  - autonomous driving, robots
  - medical sensors: heart rate & blood pressure monitors
  - intelligent buildings and spaces, environmental sensors
- Learning from mistakes can be costly…

Ariane 5

- ESA (European Space Agency) Ariane 5 launcher
  - shown here in maiden flight on 4th June 1996
- 37secs later, it self-destructs
  - uncaught exception: numerical overflow in a conversion routine results in incorrect altitude sent by the on-board computer
- Expensive, embarrassing…



Infusion pumps

F.D.A. Steps Up Oversight of Infusion Pumps

The New York Times

Published: April 23, 2010

Pump producers now typically conduct 'simulated' testing of devices by users

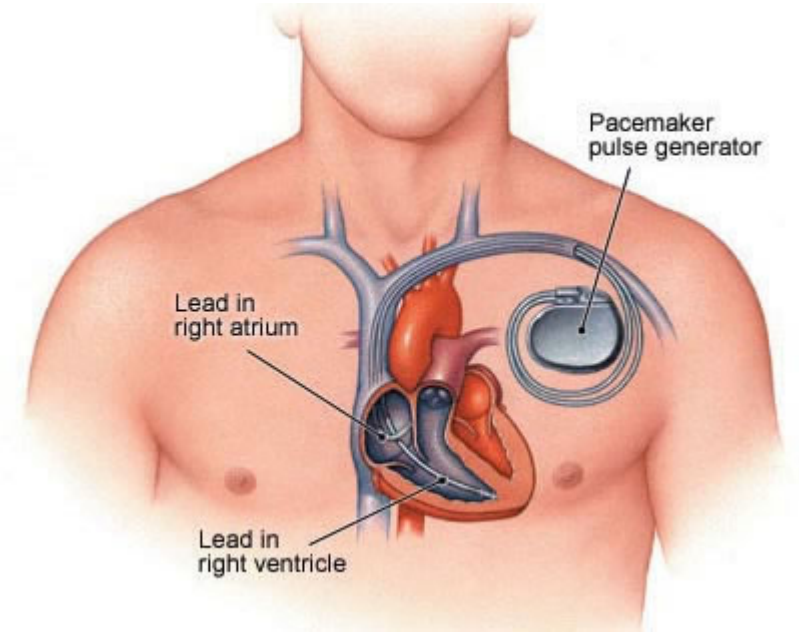
Over the last five years, […] 710 patient deaths linked to problems with the devices.

Some of those deaths involved patients who suffered drug overdoses accidentally, either because of incorrect dosage entered or because the device's software malfunctioned.

Manufacturers […] issued 86 recalls, among the highest for any medical device.

Cardiac pacemakers

- The Food and Drug Administration (FDA)
  - issued 23 recalls of defective pacemaker devices during the first half of 2010
  - classified as "Class I," meaning there is "reasonable probability that use of these products will cause serious adverse health consequences or death"
  - six of those due to software defects



- "Killed by code" report
  - many similar medical devices
  - wireless, implantable, e.g. glucose monitors

Toyota

- February 2010
  - unintended acceleration
  - resulted in accidents
- Engine Control Module
  - source code found defective
  - no mirroring: stack overflow, recursion was used

- "Killed by firmware"
  - millions of cars recalled, at huge costs
  - handling of the incident prompted much criticism, bad publicity
  - fined $1.2 billion for concealing safety defects

What do these stories have in common?

- Programmable computing devices
  - conventional computers and networks
  - software embedded in devices
    - airbag controllers, mobile phones, medical devices, etc.
- Programming error direct cause of failure

- Software critical
  - for safety
  - for business
  - for performance
- High costs incurred: not just financial

- Failures avoidable…

Why must we verify?

"Testing can only show the presence of errors, not their absence."

To rule out errors need to consider all possible executions, often not feasible mechanically!

- need formal verification…

"In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, computers are without precedent in the cultural history of mankind."

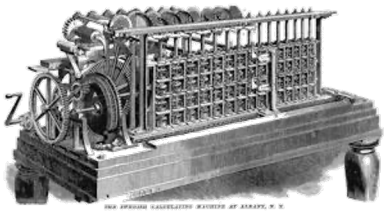
Edsger Dijkstra

1930–2002

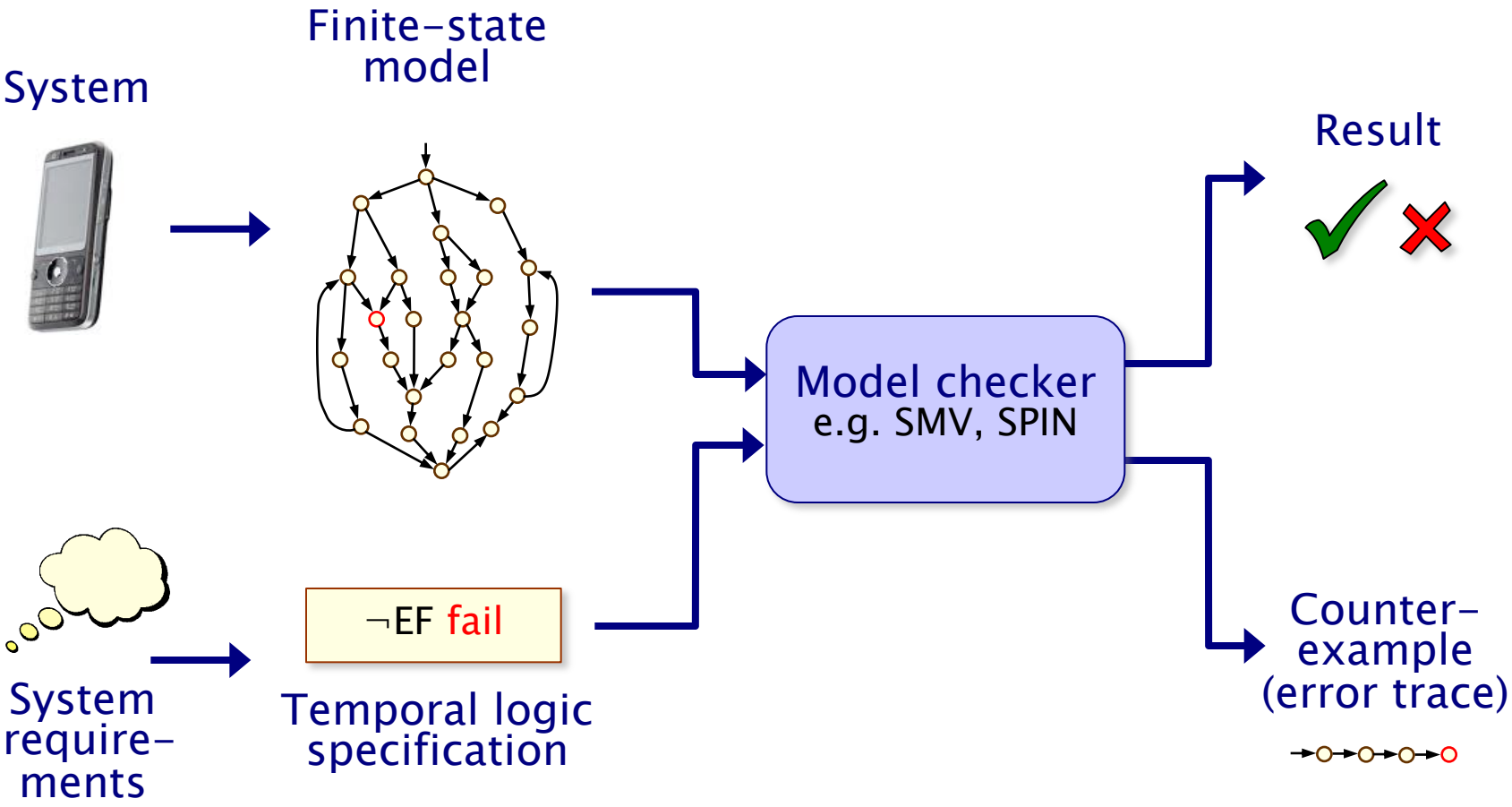
Automated formal verification

- Formal verification…
  - the application of rigorous, mathematics-based techniques to establish the correctness of computerised systems
  - essentially: proving that a program satisfies its specification
  - many techniques: manual proof, assisted theorem proving, static analysis, model checking, …

- Automated verification…
  - mechanical, push-button technology
  - performed without human intervention



Verification via model checking



Model checking in practice

- Model checking now routinely applied to real-life systems
  - not just “verification”…
  - model checkers used as a debugging tool
  - e.g., security bugs detected at Microsoft that could not be found with simulations
- Now widely accepted in industrial practice
  - Intel, Cadence, Microsoft, Amazon, Facebook, Google, ...
- Many software tools, both commercial and academic
  - CBMC, SPIN, NuSMV, FDR2, Infer, ...
  - software (memory safety, security), hardware, protocols, …
- Extremely active research area
  - 2008 Turing Award for model checking
  - see YouTube keynotes from Byron Cook (Amazon WS/UCL) and Peter O’Hearn (Facebook/UCL) at FLoC 2018 in Oxford

New challenges for verification

- Devices, ever smaller
  - laptops, phones, sensors…
- Networking, wireless and wired
  - 5G wireless, pervasive internet of things
- New design and engineering challenges
  - adaptive computing, ubiquitous/pervasive computing, context-aware systems
  - DNA computing and biosensing
  - trade-offs between e.g. performance, security, power usage, battery life, …
  - cyber-physical systems
  - control engineering, machine learning

New challenges for verification

- Many properties other than correctness are important
- Need to guarantee…
  - safety, reliability, performance, dependability
  - resource usage, e.g. battery life
  - security, privacy, trust, anonymity, fairness
  - and much more…

- Quantitative, as well as qualitative requirements:
  - "how reliable is my car's Bluetooth network?"
  - "how efficient is my phone's power management policy?"
  - "how secure is my bank's web-service?"

- This course: probabilistic verification

Why probability?

- Some systems are inherently probabilistic…
- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding
- Examples: real-world protocols featuring randomisation
  - Randomised back-off schemes
    - IEEE 802.3 CSMA/CD, IEEE 802.11 Wireless LAN
  - Random choice of waiting time
    - IEEE 1394 Firewire (root contention), Bluetooth (device discovery)
  - Random choice over a set of possible addresses
    - IPv4 Zeroconf dynamic configuration (link-local addressing)
  - Randomised algorithms for anonymity, contract signing, …
  - Quantum systems

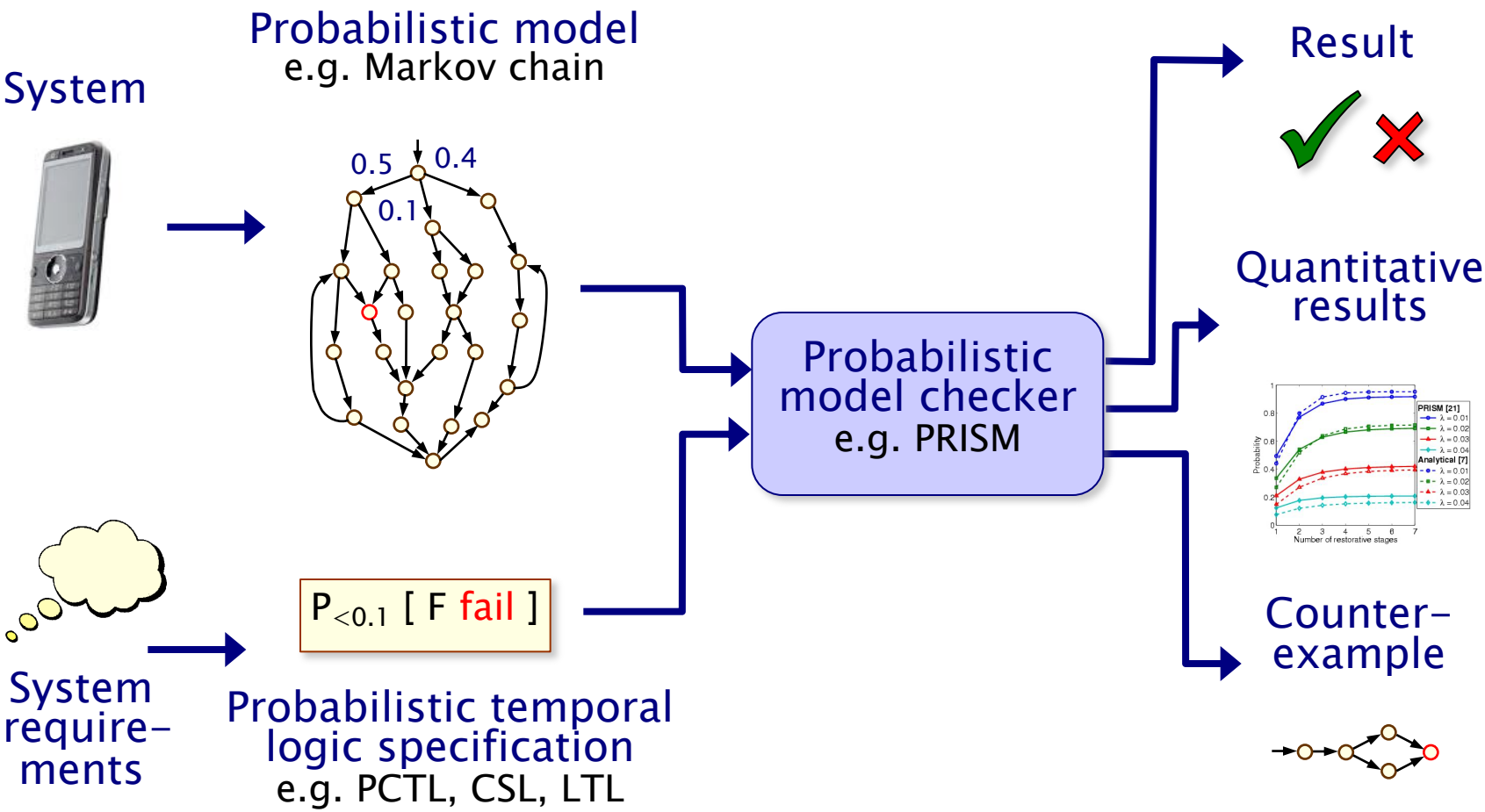
Why probability?

- Some systems are inherently probabilistic…
- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding
- Modelling uncertainty and performance
  - to quantify rate of failures, to express Quality of Service
- Examples:
  - computer networks, embedded systems
  - power management policies
  - nano-scale circuitry: reliability through defect-tolerance

Why probability?

- Some systems are inherently probabilistic…
- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding
- Modelling uncertainty and performance
  - to quantify rate of failures, to express Quality of Service
- For quantitative analysis of software and systems
  - to quantify resource usage given a policy
    "the minimum expected battery capacity for a scenario…"
- And many others, e.g. bio-chemical processes

Probabilistic model checking

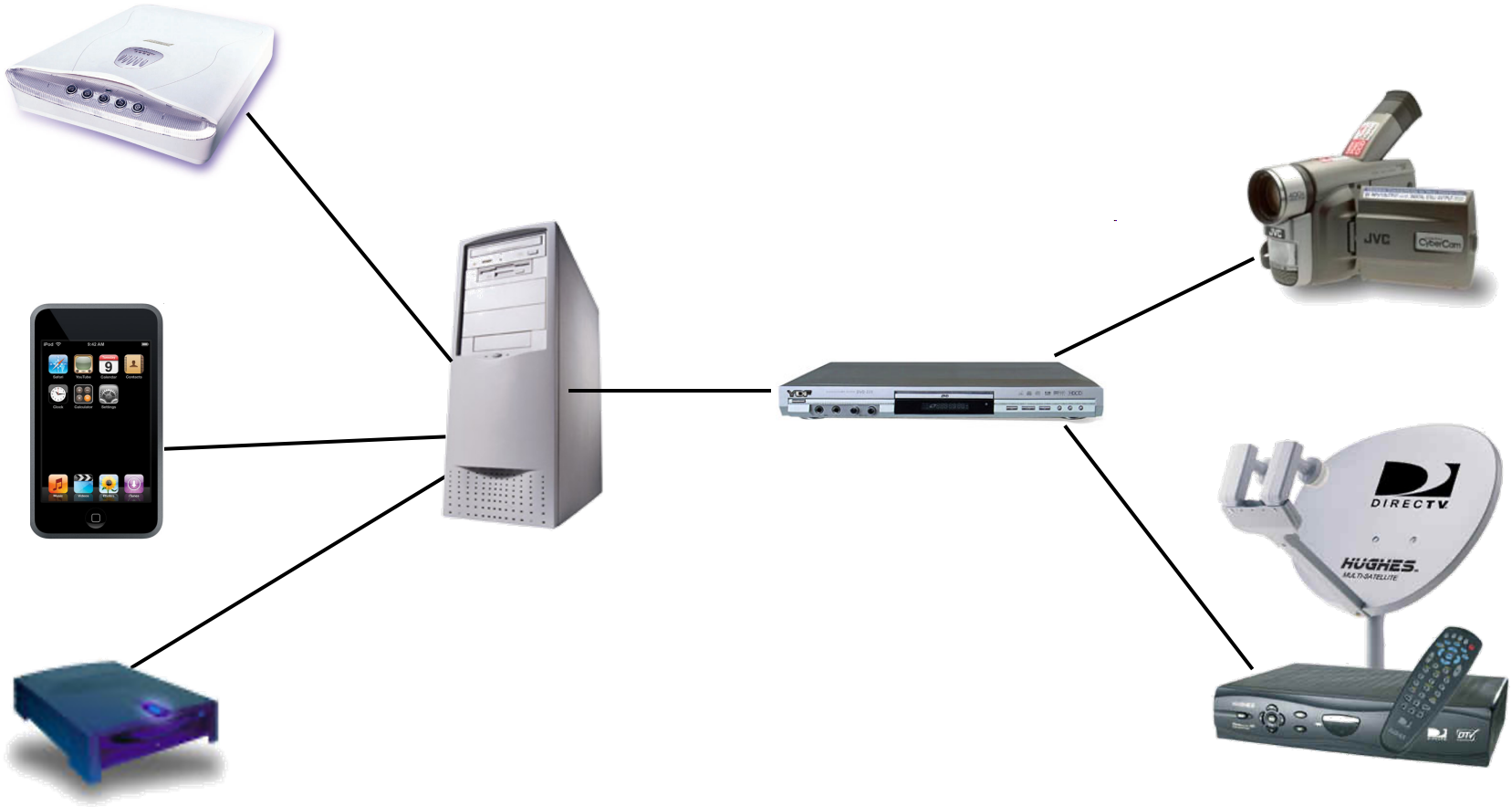


Case study: FireWire protocol

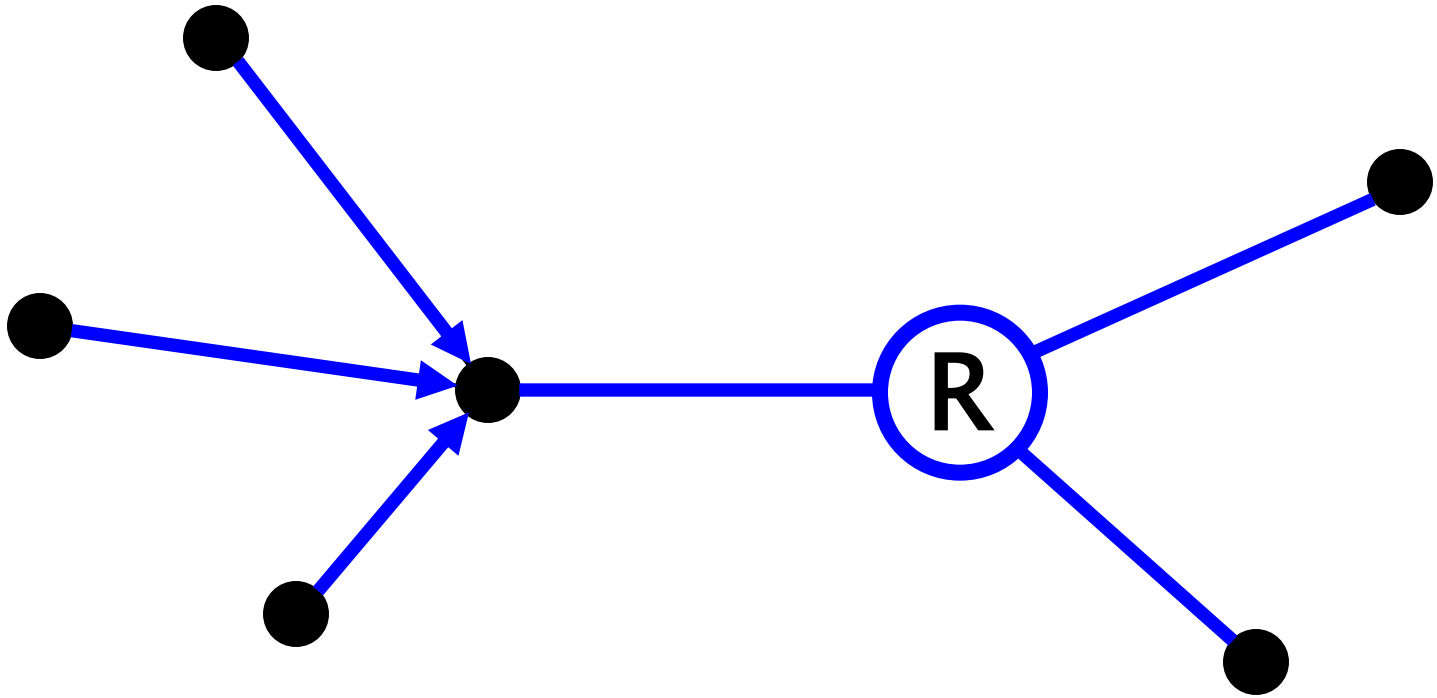
- FireWire (IEEE 1394)
  - high-performance serial bus for networking multimedia devices; originally by Apple
  - "hot-pluggable" - add/remove devices at any time
  - no requirement for a single PC (need acyclic topology)

- Root contention protocol
  - leader election algorithm, when nodes join/leave
  - symmetric, distributed protocol
  - uses electronic coin tossing and timing delays
  - nodes send messages: "be my parent"
  - root contention: when nodes contend leadership
  - random choice: "fast"/"slow" delay before retry

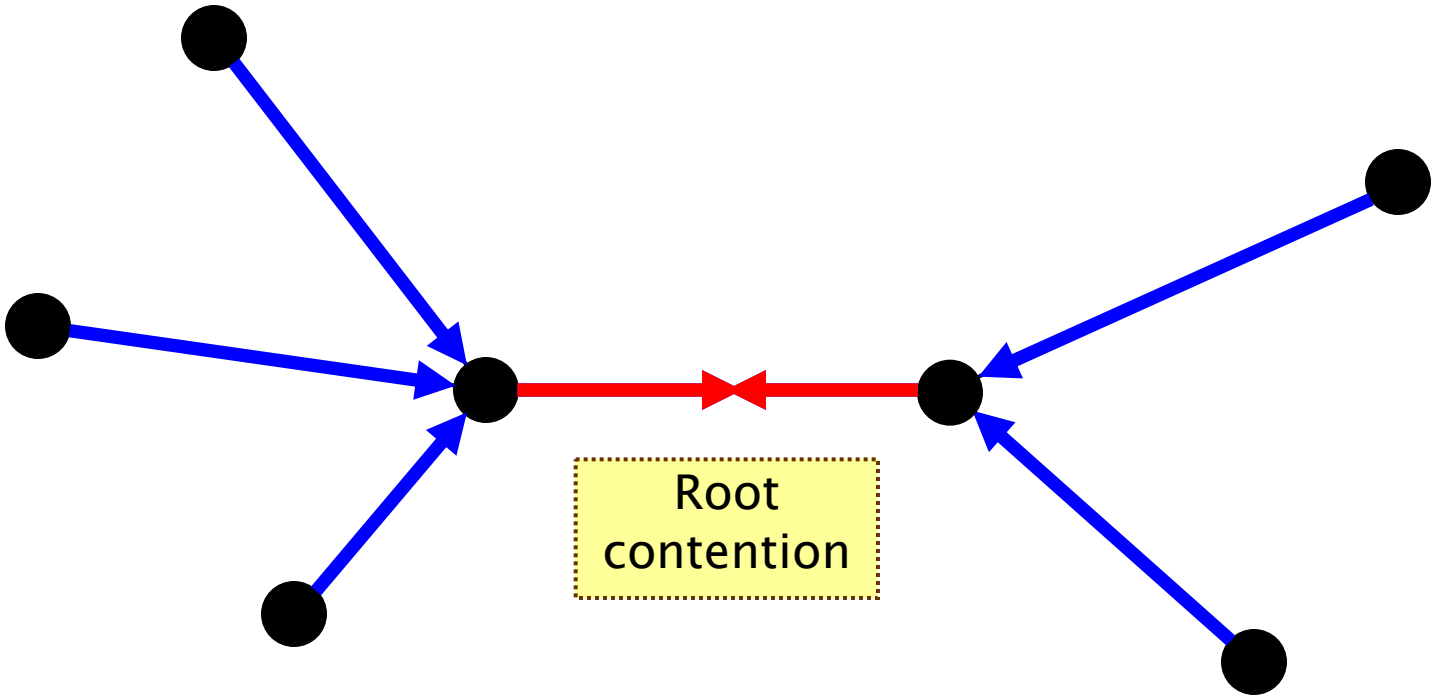
FireWire example



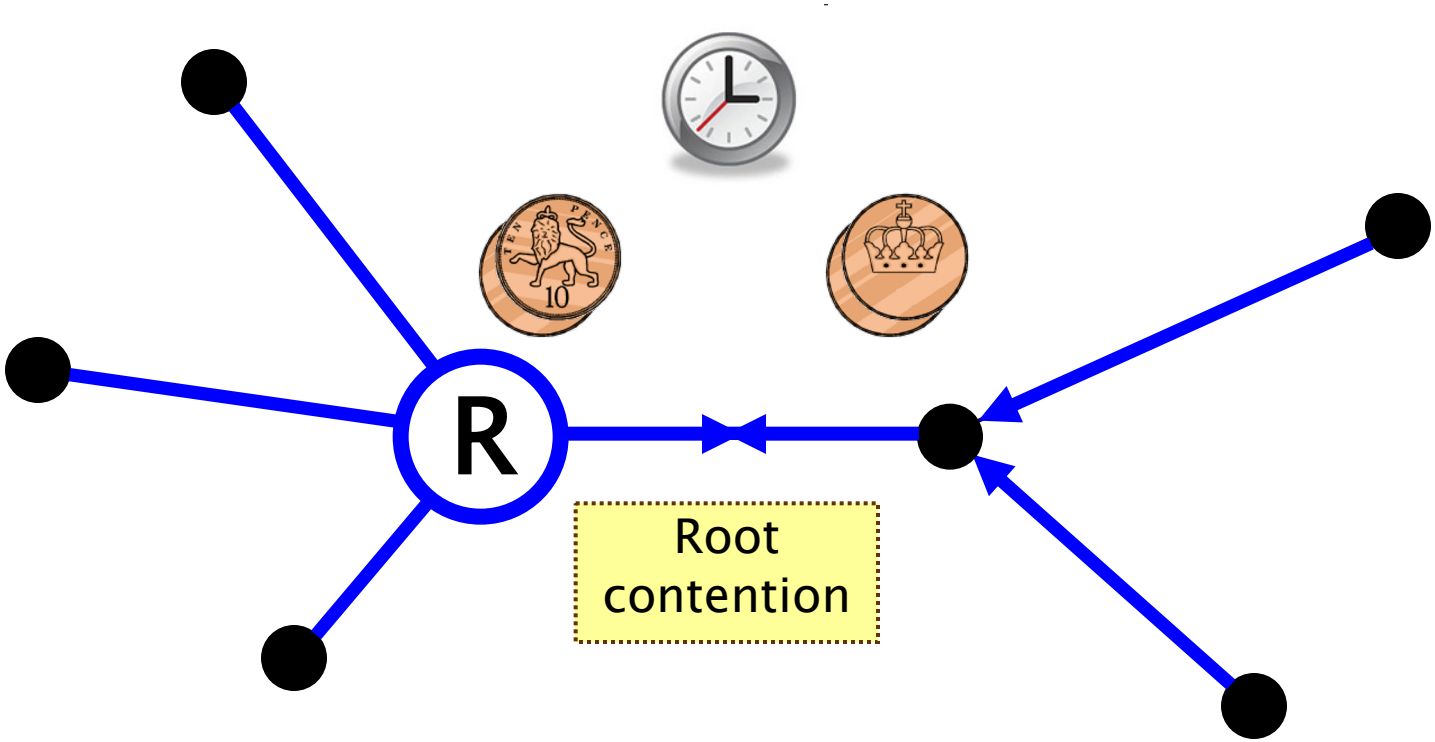
FireWire leader election



FireWire root contention

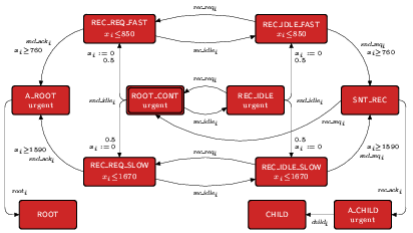


FireWire root contention

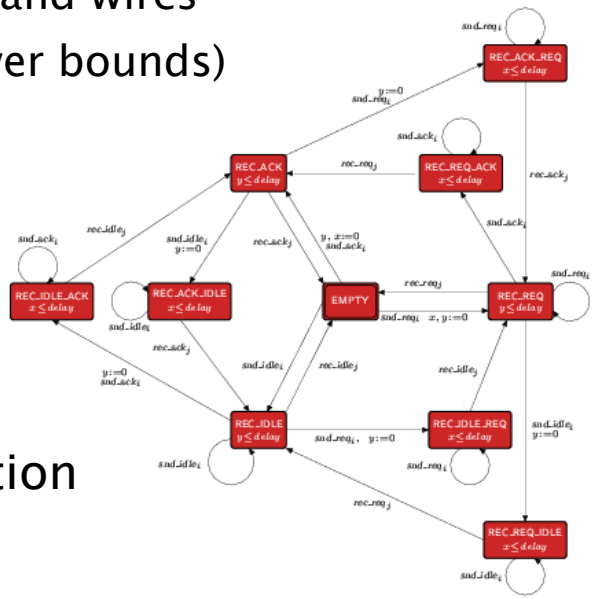


FireWire analysis

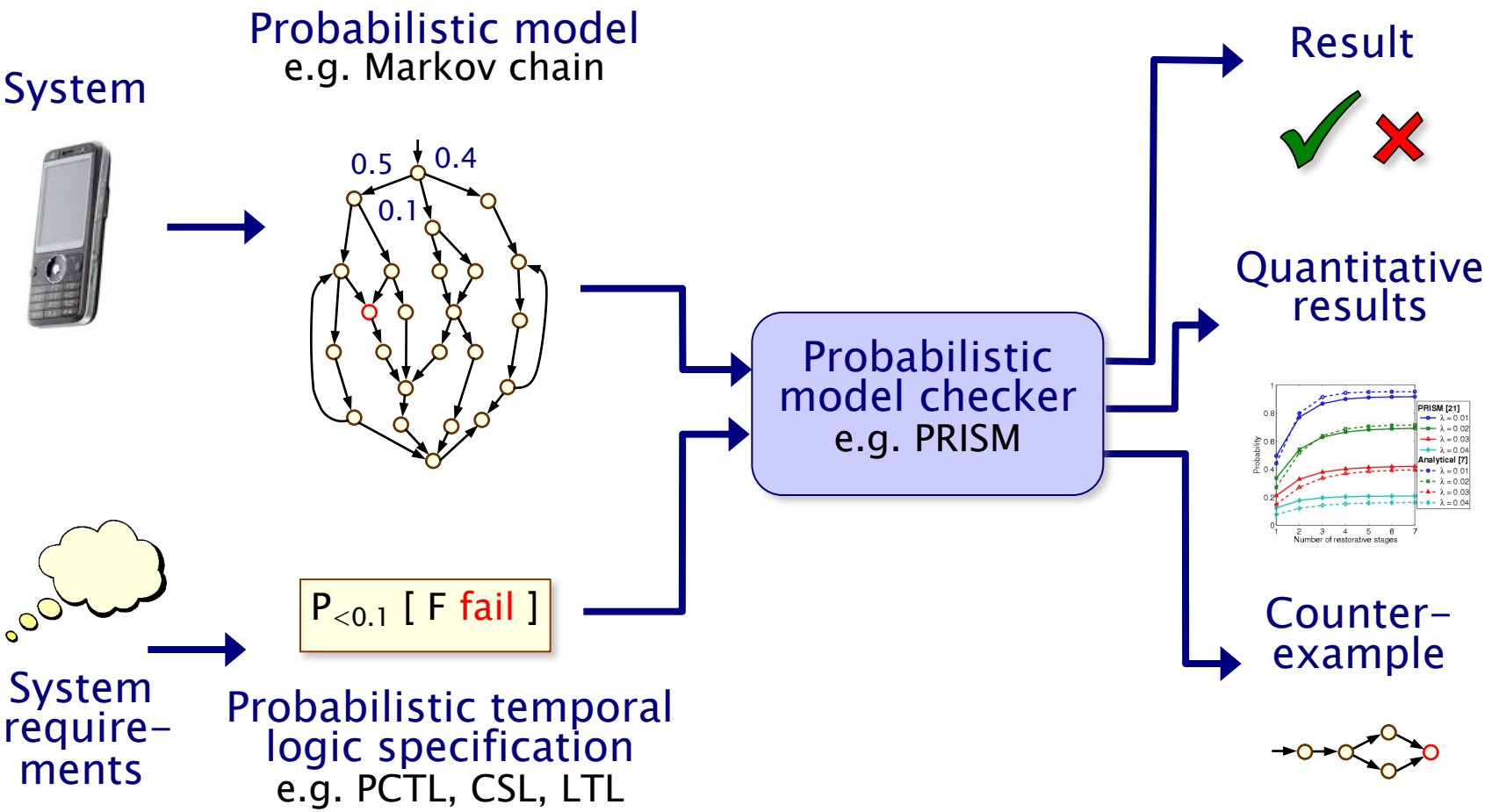
- Probabilistic model checking
  - model constructed and analysed using PRISM
  - timing delays taken from IEEE standard
  - model includes:
    - concurrency: messages between nodes and wires
    - underspecification of delays (upper/lower bounds)
  - max. model size: 170 million states



- Analysis:
  - verified that root contention always resolved with probability 1
  - investigated time taken for leader election
  - and the effect of using biased coin



Probabilistic model checking



Probabilistic model checking – Inputs

- Models: variants of Markov chains
  - discrete-time Markov chains (DTMCs)
    - discrete time, discrete probabilistic behaviours only
  - continuous-time Markov chains (CTMCs)
    - continuous time, continuous probabilistic behaviours
  - Markov decision processes (MDPs)
    - DTMCs plus non-determinism
- Specifications
  - informally:
    - "probability of delivery within time deadline is …"
    - "expected time until message delivery is …"
    - "expected power consumption is …"
  - formally:
    - probabilistic temporal logics (PCTL, CSL, LTL, PCTL*, …)
    - e.g. $P_{<0.05}$ [F err_val>0.1], $P_{=?}$ [$F^{\leq t}$ reply_count=k]

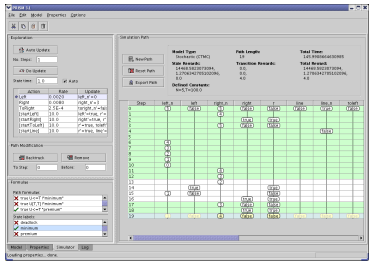
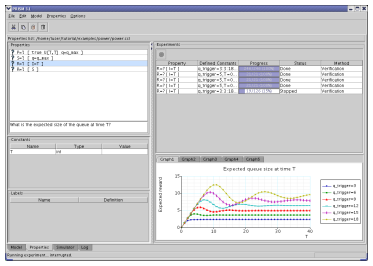
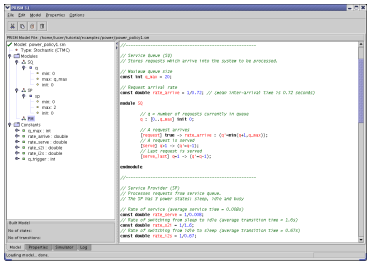
Probabilistic model checking involves…

- Construction of models
  - from a description in a high-level modelling language
- Probabilistic model checking algorithms
  - graph-theoretical algorithms
    - e.g. for reachability, identifying strongly connected components
    - shortest path problems
  - numerical computation
    - linear equation systems, linear optimisation problems
    - iterative methods, direct methods
  - automata for regular languages
  - also sampling-based (statistical) techniques for approximate analysis
    - e.g. statistical hypothesis testing or estimation, based on simulation runs

Probabilistic model checking involves…

- Efficient implementation techniques
  - essential for scalability to real-life systems
  - symbolic data structures based on binary decision diagrams
  - algorithms for bisimulation minimisation, symmetry reduction

- Tool support
  - PRISM: free, open-source probabilistic model checker
  - developed at Oxford (and, earlier, at Birmingham)
  - supports all probabilistic models discussed in this course

  - Other PMC tools exist (e.g., Storm)

In summary, course aims

- Introduce main types of probabilistic models and specification notations
  - theory, syntax, semantics, examples
  - probability, expectation, costs/rewards
- Explain the working of probabilistic model checking
  - algorithms for verification and strategy synthesis
- Introduce software tools
  - probabilistic model checker PRISM
- Examples from wide range of application domains
  - communication & coordination protocols, performance & reliability modelling, biological systems, …
- Mix of theory (probability and logics) and examples (PRISM)