2. Modelling Sequential and Parallel Systems

Computer-Aided Verification

Dave Parker

University of Birmingham

2017/18

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

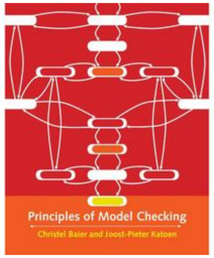
- Tutorials (feedback on exercises) (not all weeks):
  - Thur 4-5 (surnames A-L, by default):
    - UG06, Murray Learning Centre
  - Fri 10-11 (surnames M-Z, by default):
    - Lecture Theatre 1, Sports and Exercise Sciences

Assessment

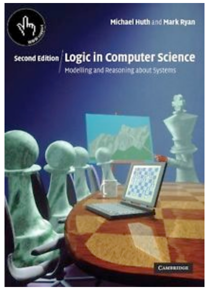
- Split:
  - 80% exam (1.5hr, in the summer)
  - 20% continuous assessment
- Continuous assessment
  - 4 exercises (1st is formative; 2–4 are assessed: 6%/8%/6%)
  - 1 week for each: due Thurs of weeks 3, 5, 8, 11
  - 1 extra assessed exercise for "extended" version (due week 10)
  - submitted through Canvas
  - solutions worked through in tutorial sessions

Reference material

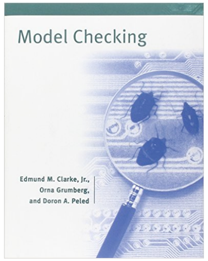
- Useful books



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)

2. Modelling Sequential and Parallel Systems

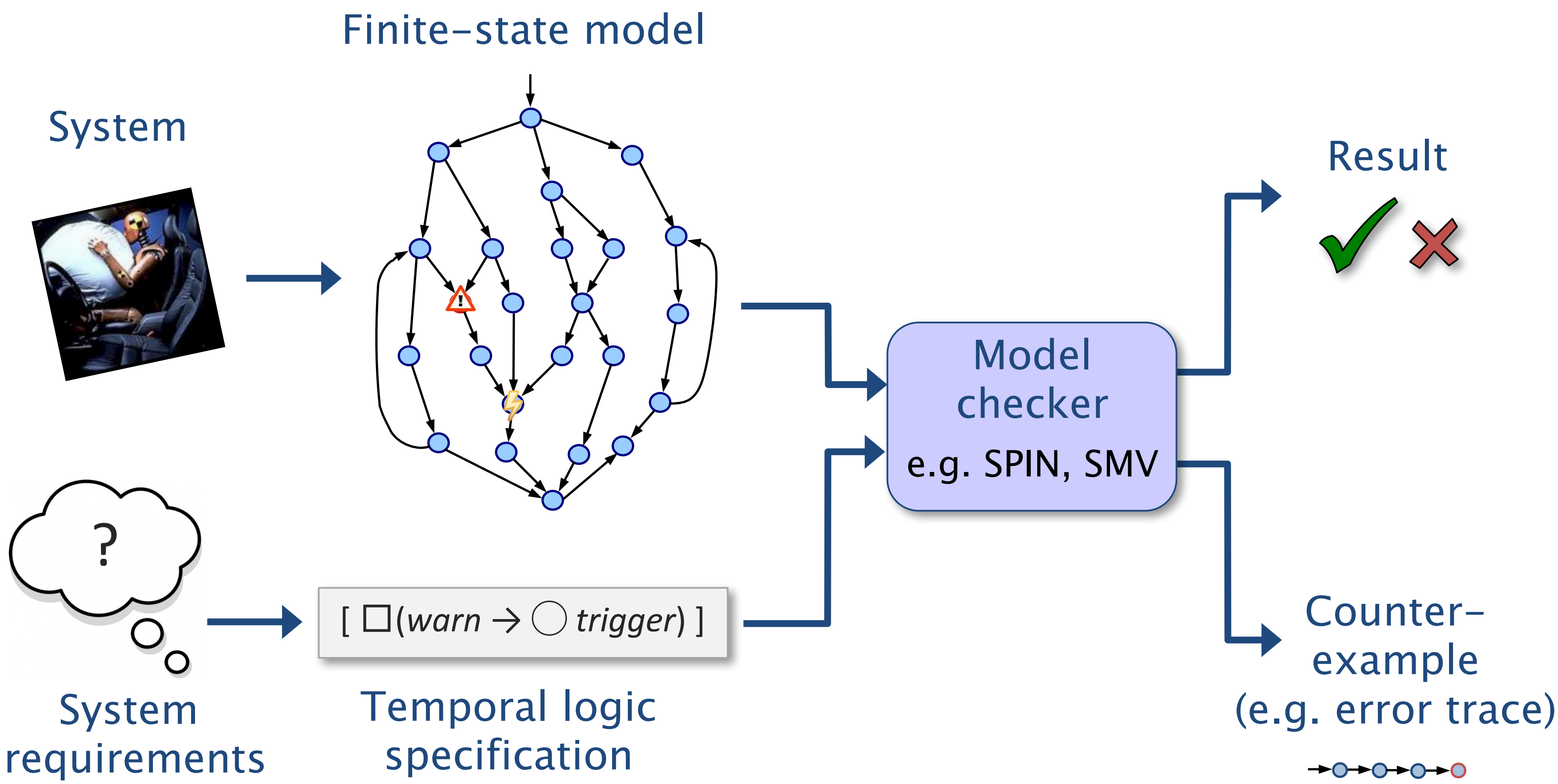
Computer-Aided Verification

Dave Parker

University of Birmingham

2016/17

Model checking

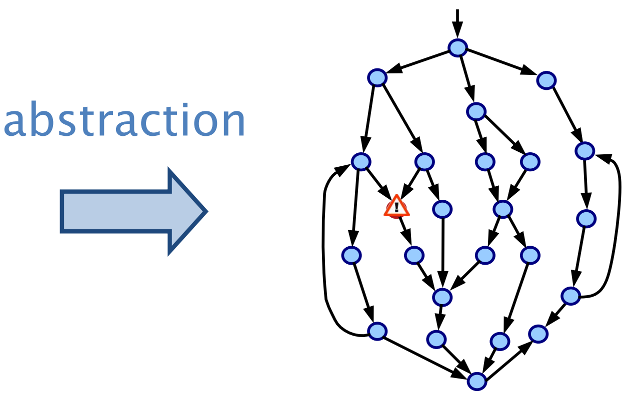


Models

- To verify computerised systems
  - we need precise mathematical models of their behaviour
- "All models are wrong, but some are useful" [George Box]
  - results of verification are only as good as the model
- It's all about abstraction

```
do {
    if (is_request) {
        size = makerequest(WRQ, name, dp, mode) - 4;
    } else {
        size = readit(file, &dp, 0);
        if (size < 0) {
            nak(errno + 100, NULL);
            break;
        }
        dp->th_opcode = _htons((u_short)DATA);
        dp->th_block  = _htons((u_short)block);
    }

    timeout = 0;
timeout:
    if (trace)
        tpacket("sent", dp, size + 4);
```



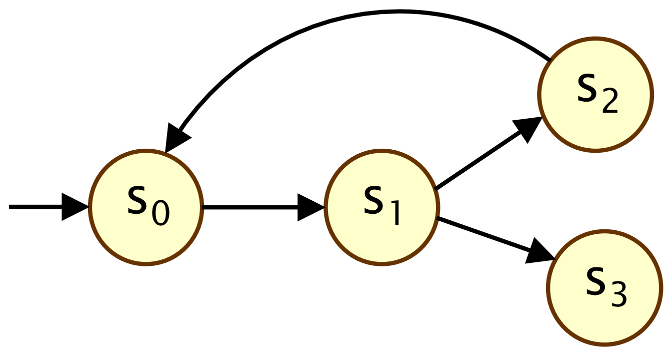
Overview

- Labelled transition systems
  - definitions, notation, …
- Modelling sequential systems
  - e.g. simple programs
- Nondeterminism
- Parallelism and concurrency
  - interleaving, shared variables, handshaking
  - SOS-style semantics

- See [BK08] chapter 2 (specifically: 2.1–2.1.1, 2.2–2.2.3)

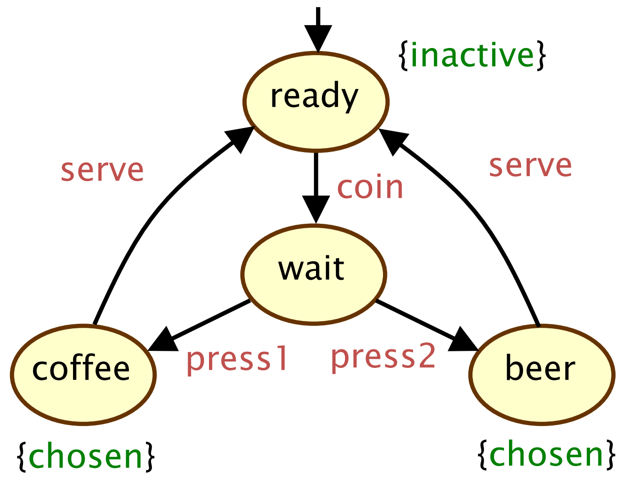
Labelled transition systems

- States = possible configurations of system
  - e.g. valuations of program variables
  - e.g. values of registers in a hardware circuit
- Transitions = possible ways system can evolve
  - e.g. execution of a single program statement
  - e.g. sequential circuit update



Labelled transition systems

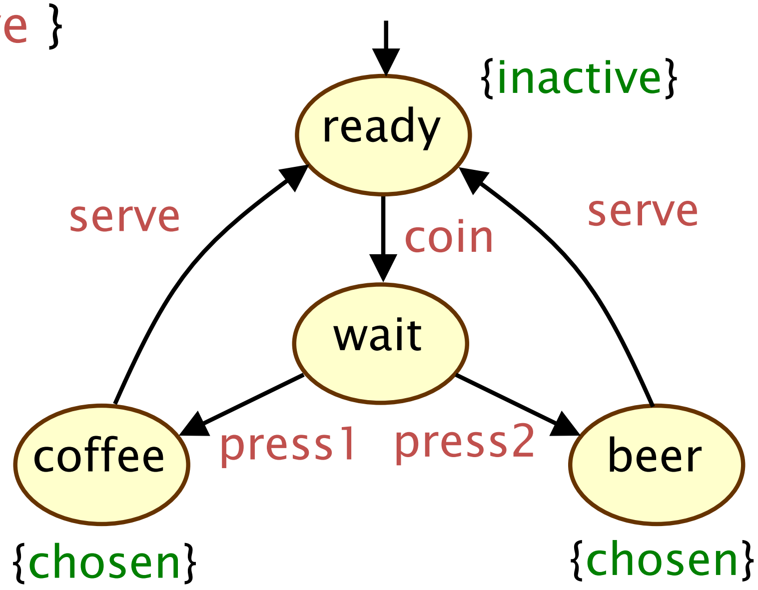
- A labelled transition system (LTS) is:
  - a tuple $(S,Act,\rightarrow,I,AP,L)$
- where:
  - S is a set of states ("state space")
  - Act is a set of actions
  - $\rightarrow \subseteq S \times Act \times S$ is a transition relation
  - $I \subseteq S$ is a set of initial states
  - AP is a set of atomic propositions
  - $L : S \rightarrow 2^{AP}$ is a labelling function



- An LTS is also known as:
  - transition system (TS), state-transition system, Kripke structure
- Essentially: directed graph
  - where nodes/vertices = states, edges = transitions

Example: Drinks machine

- Example LTS $(S,Act,\rightarrow,I,AP,L)$:
  - S = { ready, wait, coffee, beer }
  - Act = { coin, press1, press2, serve }
  - $\rightarrow$ = {
    (ready,coin,wait),
    (wait,press1,coffee),
    (wait,press2,beer),
    (coffee,serve,ready),
    (beer,serve,ready)
    }
  - I = {ready}
  - AP = {inactive,chosen}
  - L(ready) = {inactive},
    L(wait) = $\varnothing$,
    L(coffee) = L(beer) = {chosen}



Labellings and finiteness

- State labelling
  - states are labelled with atomic propositions $a,b,\ldots \in AP$
  - represent facts/observations, e.g. "failed", "size$\leq$max", …
- Transition labelling
  - transitions are labelled with actions $\alpha,\beta,\ldots \in Act$
  - will be used for communication between components
- Finiteness
  - an LTS is finite if S, Act and AP are finite
  - we will usually (but not always) assume finite LTSs

Transitions

- Transitions
  - we write $s \xrightarrow{\alpha} s'$ if $(s,\alpha,s') \in \rightarrow$

- Direct successors/predecessors
  - $Post(s,\alpha) = \{ s' \in S \mid s \xrightarrow{\alpha} s' \}$ and $Post(s) = \cup_{\alpha \in Act} Post(s,\alpha)$
  - $Pre(s,\alpha) = \{ s' \in S \mid s' \xrightarrow{\alpha} s \}$ and $Pre(s) = \cup_{\alpha \in Act} Pre(s,\alpha)$

- Terminal states
  - state s is terminal if $Post(s) = \varnothing$, i.e., has no outgoing transitions
  - might represent termination of a program
  - often represents erroneous/undesired behaviour
  - e.g. deadlock (not all components have terminated)

Paths & reachability

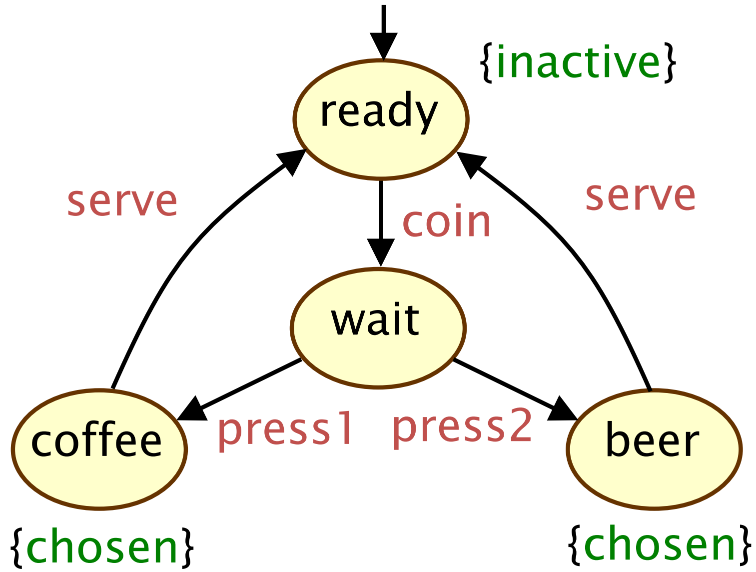
- An path (or run, trajectory, execution) is
  - an alternating sequence $s_0\alpha_0 s_1\alpha_1 s_2\alpha_2\ldots$
  - such that $s_i \xrightarrow{\alpha_i} s_{i+1}$ for all $i \geq 0$ and $s_0 \in I$
- i.e. one possible behaviour/execution of the system modelled

- A finite path is
  - a finite prefix of an (infinite) path, ending in a state
  - e.g. $s_0\alpha_0 s_1\alpha_1\ldots\alpha_{n-1}s_n$

- Reachability
  - state s' is reachable from s if there is a finite path from s to s'
  - s is a reachable state if it is reachable from some $s_0 \in I$
  - reachability (the process of determining reachable states) is a fundamental problem/task in model checking

Example

- Transitions
  - Post(wait,press1) = {coffee}
  - Post(wait) = {coffee,beer}
  - Pre(ready) = {coffee,beer}



- A finite path
  - ready coin wait press1 coffee

- An infinite path/execution
  - ready coin wait press1 coffee serve ready (coin wait press2 beer serve ready)$^{\omega}$

- All states are reachable and non-terminal

Programs as LTSs

- How to model a (sequential) program as an LTS?
  - states are tuples (l_i,x,y) of location & variable values