# Communication and Concurrency: Introduction

Colin Stirling (cps)

School of Informatics

16th September 2013

# Goals

- Modelling: a notation for describing concurrent systems (CCS)

# Goals

- Modelling: a notation for describing concurrent systems (CCS)
- Equivalence: when two descriptions are the same system

# Goals

- Modelling: a notation for describing concurrent systems (CCS)
- Equivalence: when two descriptions are the same system
- Properties: modal and temporal properties of systems.

# Goals

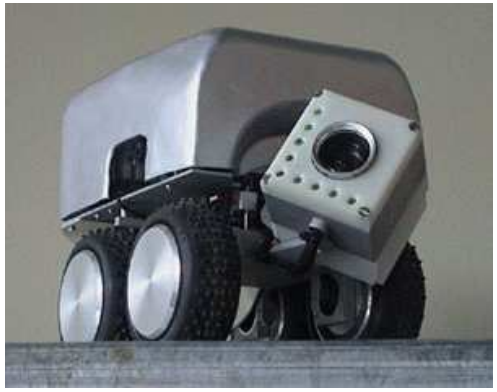
- Modelling: a notation for describing concurrent systems (CCS)
- Equivalence: when two descriptions are the same system
- Properties: modal and temporal properties of systems.
- Model checking: algorithmic techniques for checking equivalence and properties.

# Goals

- Modelling: a notation for describing concurrent systems (CCS)
- Equivalence: when two descriptions are the same system
- Properties: modal and temporal properties of systems.
- Model checking: algorithmic techniques for checking equivalence and properties.
- Software tools: automatically checks properties and equivalence

# An Example: Mutual Exclusion

Mutual Exclusion
Protocol



# Specification: Temporal Properties

- Mutual exclusion

# Specification: Temporal Properties

- Mutual exclusion
- Absence of deadlock

# Specification: Temporal Properties

- Mutual exclusion
- Absence of deadlock
- Absence of starvation

# CCS model of Peterson's solution

$$
\begin{array}{rcl}
\mathsf{B1f} & = & \overline{\mathrm{b1rf}}.\mathsf{B1f} + \mathsf{b1wf}.\mathsf{B1f} + \mathsf{b1wt}.\mathsf{B1t} \\
\mathsf{B1t} & = & \overline{\mathrm{b1rt}}.\mathsf{B1t} + \mathsf{b1wt}.\mathsf{B1t} + \mathsf{b1wf}.\mathsf{B1f} \\
 & & \\
\mathsf{B2f} & = & \overline{\mathrm{b2rf}}.\mathsf{B2f} + \mathsf{b2wf}.\mathsf{B2f} + \mathsf{b2wt}.\mathsf{B2t} \\
\mathsf{B2t} & = & \overline{\mathrm{b2rt}}.\mathsf{B2t} + \mathsf{b2wt}.\mathsf{B2t} + \mathsf{b2wf}.\mathsf{B2f} \\
 & & \\
\mathsf{K1} & = & \overline{\mathrm{kr1}}.\mathsf{K1} + \mathsf{kw1}.\mathsf{K1} + \mathsf{kw2}.\mathsf{K2} \\
\mathsf{K2} & = & \overline{\mathrm{kr2}}.\mathsf{K2} + \mathsf{kw2}.\mathsf{K2} + \mathsf{kw1}.\mathsf{K1} \\
 & & \\
\mathsf{P1} & = & \overline{\mathrm{b1wt}}.\mathsf{req1}.\overline{\mathrm{kw2}}.\mathsf{P11} \\
\mathsf{P11} & = & \mathsf{b2rt}.\mathsf{P11} + \mathsf{b2rf}.\mathsf{P12} + \mathsf{kr2}.\mathsf{P11} + \\
 & & \mathsf{kr1}.\mathsf{P12} \\
\mathsf{P12} & = & \mathsf{enter1}.\mathsf{exit1}.\overline{\mathrm{b1wf}}.\mathsf{P1} \\
 & & \\
\mathsf{P2} & = & \overline{\mathrm{b2wt}}.\mathsf{req2}.\overline{\mathrm{kw1}}.\mathsf{P21} \\
\mathsf{P21} & = & \mathsf{b1rf}.\mathsf{P22} + \mathsf{b1rt}.\mathsf{P21} + \mathsf{kr1}.\mathsf{P21} + \\
 & & \mathsf{kr2}.\mathsf{P22} \\
\mathsf{P22} & = & \mathsf{enter2}.\mathsf{exit2}.\overline{\mathrm{b2wf}}.\mathsf{P2} \\
 & & \\
\mathsf{Peterson} & = & (\mathsf{P1} \mid \mathsf{P2} \mid \mathsf{K1} \mid \mathsf{B1f} \mid \mathsf{B2f}) \setminus L
\end{array}
$$

# Formalising Temporal Properties

$$
\begin{aligned}
\text{Mutex} &= \text{AG}\ ([\text{exit1}]\text{ff} \vee [\text{exit2}]\ \text{ff}) \\
\text{NoDeadlock} &= \text{AG}\ \langle - \rangle\ \text{tt} \\
\text{NoStarvation} &= \text{AG}([\text{req1}]\ \text{AF}\ \langle \text{exit1} \rangle\ \text{tt}) \wedge \\
&\quad\ \ \text{AG}([\text{req2}]\ \text{AF}\ \langle \text{exit2} \rangle\ \text{tt})
\end{aligned}
$$

# Model checking

- The Edinburgh Concurrency Workbench
  - A tool for simulating and verifying CCS agents
  - http://homepages.inf.ed.ac.uk/perdita/cwb/

# Model checking

- The Edinburgh Concurrency Workbench
  - A tool for simulating and verifying CCS agents
  - http://homepages.inf.ed.ac.uk/perdita/cwb/
- Proving Peterson's solution correct

# Model checking

- The Edinburgh Concurrency Workbench
  - A tool for simulating and verifying CCS agents
  - http://homepages.inf.ed.ac.uk/perdita/cwb/
- Proving Peterson's solution correct
  - `Command:` checkprop(Peterson,Mutex);
  - `Command:` checkprop(Peterson,NoDeadlock);
  - `Command:` checkprop(Peterson,NoStarvation);

# Model checking

- The Edinburgh Concurrency Workbench
  - A tool for simulating and verifying CCS agents
  - http://homepages.inf.ed.ac.uk/perdita/cwb/
- Proving Peterson's solution correct
  - `Command: checkprop(Peterson,Mutex);`
  - `true`
  - `Command: checkprop(Peterson,NoDeadlock);`
  - `true`
  - `Command: checkprop(Peterson,NoStarvation);`
  - `true`

# In Reality . . .

Modelling and model checking large (and infinite state) systems

- Circuits: since Pentium-bug Intel uses model checking

# In Reality . . .

Modelling and model checking large (and infinite state) systems

- Circuits: since Pentium-bug Intel uses model checking
- Software: Microsoft prototype software model checking

# In Reality . . .

Modelling and model checking large (and infinite state) systems

- Circuits: since Pentium-bug Intel uses model checking
- Software: Microsoft prototype software model checking
- ⋮

# In Reality . . .

Modelling and model checking large (and infinite state) systems

- Circuits: since Pentium-bug Intel uses model checking
- Software: Microsoft prototype software model checking
- ⋮
- Life: cells and pathways (Systems biology: huge new area)

# In Reality . . .

Modelling and model checking large (and infinite state) systems

- Circuits: since Pentium-bug Intel uses model checking
- Software: Microsoft prototype software model checking
- ⋮
- Life: cells and pathways (Systems biology: huge new area)

Paper on hardware verification and one on BLAST tool for software verification on course web page

# In Reality . . .

Modelling and model checking large (and infinite state) systems

- Circuits: since Pentium-bug Intel uses model checking
- Software: Microsoft prototype software model checking
- ⋮
- Life: cells and pathways (Systems biology: huge new area)

Paper on hardware verification and one on BLAST tool for software verification on course web page
Look up "model checking" in Wikipedia, Google, . . .