COMP30112: Concurrency
Introduction to Course & Introduction to FSP

Howard Barringer

Room KB2.20: email: Howard.Barringer@manchester.ac.uk

February 2009
Acknowledgement

The course material is largely based on that of:

Concurrency: State Models and Java Programs
Profs Jeff Magee and Jeff Kramer

of Imperial College, Dept of Computing, London.
Acknowledgement

The course material is largely based on that of:

Concurrency: State Models and Java Programs
Profs Jeff Magee and Jeff Kramer

of Imperial College, Dept of Computing, London.

Many thanks to all concerned.
Contents

**Topic 1: Introduction**
- General Background
- On Concurrency
- Examples
- Implementation

**Topic 2: Modelling Processes with FSP - I**
- Labelled Transition Systems
- FSP: Basic Elements
- Summary
Outline

Topic 1: Introduction
General Background
On Concurrency
Examples
Implementation

Topic 2: Modelling Processes with FSP - I
Labelled Transition Systems
FSP: Basic Elements
Summary
## Course Structure and Topics

<table>
<thead>
<tr>
<th>Lectures</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction and Overview</td>
</tr>
<tr>
<td>5</td>
<td>FSP: Modelling Processes</td>
</tr>
<tr>
<td>5</td>
<td>Properties: Safety, liveness</td>
</tr>
<tr>
<td>2</td>
<td>Process Equality</td>
</tr>
<tr>
<td>2</td>
<td>Java Threads</td>
</tr>
<tr>
<td>6</td>
<td><strong>Concurrency Patterns:</strong></td>
</tr>
<tr>
<td></td>
<td>Mutual Exclusion</td>
</tr>
<tr>
<td></td>
<td>Monitors + Semaphores</td>
</tr>
<tr>
<td></td>
<td>Producers/Consumers</td>
</tr>
<tr>
<td></td>
<td>Readers/Writers</td>
</tr>
<tr>
<td></td>
<td>GUIs</td>
</tr>
<tr>
<td></td>
<td>Termination</td>
</tr>
<tr>
<td>1</td>
<td>Revision</td>
</tr>
</tbody>
</table>
General Comments

- Some concepts familiar from COMP20051 and COMP20081
- We follow much of Magee and Kramer, but more on modelling
- Java $\sim$ COMP20051
- Java used to illustrate — BUT this is NOT a programming course
Supporting and Background Material

Books


LTSA: Magee and Kramer’s modelling and analysis tool (associated with book)
Exercises: offline and in lectures
Lecture Slides: hardcopy and PDF
Notes on FSP
Assessment

Two-hour examination
Outline

**Topic 1: Introduction**
- General Background
- On Concurrency
- Examples
- Implementation

**Topic 2: Modelling Processes with FSP - I**
- Labelled Transition Systems
- FSP: Basic Elements
- Summary
What is Concurrency?

A set of sequential programs executed in abstract parallelism
What is Concurrency?

A set of sequential programs executed in abstract parallelism

- thread [of control]
- multi-threading
- light-weight threads
What is Concurrency?

A set of sequential programs executed in **abstract** parallelism

- thread [of control]
- multi-threading
- light-weight threads
- parallel processing
What is Concurrency?

A set of sequential programs executed in **abstract** parallelism

- thread [of control]
- multi-threading
- light-weight threads
- parallel processing
- multi-processing
- multi-tasking
What is Concurrency?

A set of sequential programs executed in abstract parallelism

- thread [of control]
- multi-threading
- light-weight threads
- parallel processing
- multi-processing
- multi-tasking

- shared memory
- protected work-space
What is Concurrency?

A set of sequential programs executed in abstract parallelism

- thread [of control]
- multi-threading
- light-weight threads
- parallel processing
- multi-processing
- multi-tasking

- shared memory
- protected work-space
- message-passing (synchronous or asynchronous)
Why use Concurrency?
Why use Concurrency?

- many processes often closely models application
Why use Concurrency?

- many processes often closely models application
- sometimes closely fits intuition, a good abstraction
Why use Concurrency?

- many processes often closely models application
- sometimes closely fits intuition, a good abstraction
- performance issues
Why use Concurrency?

- many processes often closely models application
- sometimes closely fits intuition, a good abstraction
- performance issues
- increased responsiveness and throughput (esp. GUIs)
Why is concurrency hard?
Why is concurrency hard?

- algorithm development
Why is concurrency hard?

- algorithm development
- efficiency and performance
Why is concurrency hard?

- algorithm development
- efficiency and performance
- simulation and testing: NON_DETERMINISM
Why is concurrency hard?

- algorithm development
- efficiency and performance
- simulation and testing: NON_DETERMINISM
- analysis of properties: deadlock, livelock, fairness, liveness, etc.
Why is concurrency hard?

- algorithm development
- efficiency and performance
- simulation and testing: NON_DETERMINISM
- analysis of properties: deadlock, livelock, fairness, liveness, etc.

We will consider: Modelling, Analysis and Implementation (in Java)
Outline

Topic 1: Introduction
  General Background
  On Concurrency
  Examples
  Implementation

Topic 2: Modelling Processes with FSP - I
  Labelled Transition Systems
  FSP: Basic Elements
  Summary
Example: Mutual Exclusion

Design a basic control protocol to ensure two processes never execute some “critical” region of program together.

Is it OK?
Example: The Firing Squad Synchronisation Problem

- On the command “FIRE”, the chain of control units must mutually synchronise to fire each gun simultaneously.
- The control units must be identical and work for any size chain of artillery.
Example: The Firing Squad Synchronisation Problem

- On the command “FIRE”, the chain of control units must mutually synchronise to fire each gun simultaneously.
- The control units must be identical and work for any size chain of artillery.
Example: The Firing Squad Synchronisation Problem

- On the command “FIRE”, the chain of control units must mutually synchronise to fire each gun simultaneously.
- The control units must be identical and work for any size chain of artillery.
Modelling Concurrency
Modelling Concurrency

- a ‘simplified’ representation of the real world?
Modelling Concurrency

- a ‘simplified’ representation of the real world?
- modelling before implementing
Modelling Concurrency

- a ‘simplified’ representation of the real world?
- modelling before implementing
- model captures interesting aspects: concurrency
- animation
- analysis
Modelling Concurrency

- a ‘simplified’ representation of the real world?
- modelling before implementing
- model captures interesting aspects: concurrency
- animation
- analysis

- Model Description Language: **FSP** (*Finite State Processes*)
- Models: **LTS** (*Labelled Transition Systems*)
Example: Cruise Control System

- Does it do what we expect? Is it safe?
FSP: Animation

\[
\text{INPUTSPEED} = ( \text{engineOn} \rightarrow \text{CHECKSPEED} ), \\
\text{CHECKSPEED} = ( \text{speed} \rightarrow \text{CHECKSPEED} \\
\mid \text{engineOff} \rightarrow \text{INPUTSPEED} ).
\]
set Sensors = {engineOn, engineOff, on, off, 
               resume, brake, accelerator}

set Engine = {engineOn, engineOff}

set Prompts = {clearSpeed, recordSpeed, 
               enableControl, disableControl}
Outline

**Topic 1: Introduction**
- General Background
- On Concurrency
- Examples
- Implementation

**Topic 2: Modelling Processes with FSP - I**
- Labelled Transition Systems
- FSP: Basic Elements
- Summary
Implementation in Java

- **Thread** class; **Runnable** interface
- starting, stopping, suspending threads
- mutual exclusion: **synchronized** methods and code blocks
- monitors, condition synchronization
- **wait**, **notify**, **notifyAll**
- **sleep**, **interrupt**
- suspend, resume, stop
- properties: safety, liveness
Outline

Topic 1: Introduction
General Background
On Concurrency
Examples
Implementation

Topic 2: Modelling Processes with FSP - I
Labelled Transition Systems
FSP: Basic Elements
Summary
Labelled Transition Systems

What is an LTS?:

\[ \text{LTS} = (S, A, \sigma, s_0) \]

- \( S \) : set of states
- \( A \) : alphabet \( A \subseteq \text{Act} \)
- \( \sigma \) : transition relation \( \sigma \subseteq (S \times A \times S) \)
- \( s_0 \) : initial state \( s_0 \in S \)

\( \text{Act} \) is our set of atomic transition labels, or actions.
First Exercise

**DAY1:** A Day In the Life Of:

- Get up — action: up,
- then have a cup of tea — action: tea
- then work — action: work

★ LTS for DAY1? ★
First Exercise

**DAY1:** A Day In the Life Of:

- Get up — action: up,
- then have a cup of tea — action: tea
- then work — action: work

★ LTS for DAY1? ★

**DAY2:** Now repeat the Day

★ LTS for DAY2? ★
First Exercise

**DAY1**: A Day In the Life Of:

- Get up — action: \textit{up},
- then have a cup of tea — action: \textit{tea}
- then work — action: \textit{work}

★ LTS for DAY1? ★

**DAY2**: Now repeat the Day

★ LTS for DAY2? ★

**DAY3**: Now be able to choose coffee — action: \textit{coffee} — instead of tea

★ LTS for DAY3? ★
FSP: A Textual Representation for LTS

FSP - Finite State Processes
What FSP Constructs Are Required??

- sequence
FSP: A Textual Representation for LTS

FSP - Finite State Processes
What FSP Constructs Are Required??

- sequence
- STOP
FSP: A Textual Representation for LTS

FSP - Finite State Processes
What FSP Constructs Are Required??

- sequence
- STOP
- process definition, with recursion
FSP: A Textual Representation for LTS

FSP - Finite State Processes
What FSP Constructs Are Required??

- sequence
- STOP
- process definition, with recursion
- choice
Outline

Topic 1: Introduction
- General Background
- On Concurrency
- Examples
- Implementation

Topic 2: Modelling Processes with FSP - I
- Labelled Transition Systems
- FSP: Basic Elements
- Summary
Action Prefix

If $x$ is an action and $P$ is a process then $(x\rightarrow P)$ describes a process that initially engages in the action $x$ and then behaves exactly as described by $P$.

$(\text{once}\rightarrow \text{STOP})$.

Convention:

- actions begin with a lower case letter
- PROCESS NAMES begin with an upper case letter
- $STOP$ is a specially pre-defined FSP process name.
Action Prefix

If $x$ is an action and $P$ is a process then $(x \rightarrow P)$ describes a process that initially engages in the action $x$ and then behaves exactly as described by $P$.

$(\text{once} \rightarrow \text{STOP})$.

Convention:

- actions begin with a lower case letter
- PROCESS NAMES begin with an upper case letter
- $STOP$ is a specially pre-defined FSP process name.

★ FSP for DAY1? ★
Process Definition

Basic form:

\[ \text{ProcId} = \text{process\_expression} \]

The meaning of \textit{ProcId} will be given by the meaning of \textit{process\_expression}.

\textit{ProcId} should start with an upper-case letter.

(more complex forms possible — see later...)
Process Definition

Basic form:

\[ \text{ProcId} = \text{process\_expression} \]

The meaning of \( \text{ProcId} \) will be given by the meaning of \( \text{process\_expression} \).

\( \text{ProcId} \) should start with an upper-case letter.

(more complex forms possible — see later...)  
★ FSP for DAY2? ★
Choice

- If $x$ and $y$ are actions then $(x->P \ | \ y->Q)$ describes a process which initially engages in either of the actions $x$ or $y$.
- After that the subsequent behaviour is described by
  - $P$ if the first action was $x$,
  - $Q$ if the first action was $y$. 
Choice

• If $x$ and $y$ are actions then $(x\rightarrow P \mid y\rightarrow Q)$ describes a process which initially engages in either of the actions $x$ or $y$.
• After that the subsequent behaviour is described by
  • $P$ if the first action was $x$,
  • $Q$ if the first action was $y$.

★ FSP for DAY3? ★
Example: Various Switches

Repetitive behaviour uses **recursion**:

\[
\text{SWITCH} = \text{OFF}, \\
\text{OFF} = (\text{on} \rightarrow \text{ON}), \\
\text{ON} = (\text{off} \rightarrow \text{OFF}).
\]
Example: Various Switches

Repetitive behaviour uses recursion:

\[
\begin{align*}
\text{SWITCH} & = \text{OFF}, \\
\text{OFF} & = (\text{on} \rightarrow \text{ON}), \\
\text{ON} & = (\text{off} \rightarrow \text{OFF}).
\end{align*}
\]

Substituting to get a more concise definition:

\[
\begin{align*}
\text{SWITCH} & = \text{OFF}, \\
\text{OFF} & = (\text{on} \rightarrow \text{off} \rightarrow \text{OFF}).
\end{align*}
\]
Example: Various Switches

Repetitive behaviour uses recursion:

\[
\begin{align*}
\text{SWITCH} &= \text{OFF}, \\
\text{OFF} &= (\text{on}\to\text{ON}), \\
\text{ON} &= (\text{off}\to\text{OFF}).
\end{align*}
\]

Substituting to get a more concise definition:

\[
\begin{align*}
\text{SWITCH} &= \text{OFF}, \\
\text{OFF} &= (\text{on}\to\text{off}\to\text{OFF}).
\end{align*}
\]

And again:

\[
\begin{align*}
\text{SWITCH} &= (\text{on}\to\text{off}\to\text{SWITCH}).
\end{align*}
\]
Example: Various Switches

Repetitive behaviour uses recursion:

\[
\begin{align*}
SWITCH &= OFF, \\
OFF &= (on\rightarrow ON), \\
ON &= (off\rightarrow OFF).
\end{align*}
\]

Substituting to get a more concise definition:

\[
\begin{align*}
SWITCH &= OFF, \\
OFF &= (on\rightarrow off\rightarrow OFF).
\end{align*}
\]

And again:

\[
\begin{align*}
SWITCH &= (on\rightarrow off\rightarrow SWITCH).
\end{align*}
\]

★ Are these FSP SWITCH definitions the same? ★
Example: Traffic Light

FSP model of a traffic light:

\[
\text{TRAFFICLIGHT} = (\text{red} \rightarrow \text{amber} \rightarrow \text{green} \rightarrow \text{amber} \rightarrow \text{TRAFFICLIGHT}).
\]
Example: Traffic Light

FSP model of a traffic light:

\[
\text{TRAFFICLIGHT} = (\text{red} \rightarrow \text{amber} \rightarrow \text{green} \\
\rightarrow \text{amber} \rightarrow \text{TRAFFICLIGHT}).
\]

★ LTS generated using LTSA? ★
Example: Traffic Light

FSP model of a traffic light:

\[
\text{TRAFFICLIGHT} = (\text{red} \rightarrow \text{amber} \rightarrow \text{green} \rightarrow \text{amber} \rightarrow \text{TRAFFICLIGHT}).
\]

★ LTS generated using LTSA? ★

★ Trace? ★
Example: Traffic Light

FSP model of a traffic light:

\[
\text{TRAFFICLIGHT} = (\text{red} \rightarrow \text{amber} \rightarrow \text{green} \rightarrow \text{amber} \rightarrow \text{TRAFFICLIGHT}).
\]

★ LTS generated using LTSA? ★

★ Trace? ★

\[
\text{red} \rightarrow \text{amber} \rightarrow \text{green} \rightarrow \text{amber} \rightarrow \text{red} \rightarrow \text{amber} \rightarrow \text{green} \rightarrow \ldots
\]
Example: Vending Machine

FSP model of a drinks machine:

\[
\text{DRINKS} = (\text{red} \rightarrow \text{coffee} \rightarrow \text{DRINKS} | \text{blue} \rightarrow \text{tea} \rightarrow \text{DRINKS}).
\]
Example: Vending Machine

FSP model of a drinks machine:

\[
\text{DRINKS} = (\text{red} \rightarrow \text{coffee} \rightarrow \text{DRINKS} \mid \text{blue} \rightarrow \text{tea} \rightarrow \text{DRINKS})
\]

★ LTS generated using LTSA? ★
Example: Vending Machine

FSP model of a drinks machine:

\[
\text{DRINKS} = (\text{red} \rightarrow \text{coffee} \rightarrow \text{DRINKS} | \text{blue} \rightarrow \text{tea} \rightarrow \text{DRINKS}).
\]

★ LTS generated using LTSA? ★

★ Possible traces? ★
Non-Deterministic Choice

Process \((x \rightarrow P \mid x \rightarrow Q)\) describes a process which engages in \(x\) and then behaves as either \(P\) or \(Q\).

\[
\begin{align*}
\text{COIN} &= (\text{toss} \rightarrow \text{HEADS} \mid \text{toss} \rightarrow \text{TAILS}), \\
\text{HEADS} &= (\text{heads} \rightarrow \text{COIN}), \\
\text{TAILS} &= (\text{tails} \rightarrow \text{COIN}).
\end{align*}
\]

Tossing a coin
Non-Deterministic Choice

Process \((x \rightarrow P \mid x \rightarrow Q)\) describes a process which engages in \(x\) and then behaves as either \(P\) or \(Q\).

\[
\begin{align*}
\text{COIN} & = (\text{toss} \rightarrow \text{HEADS} \mid \text{toss} \rightarrow \text{TAILS}), \\
\text{HEADS} & = (\text{heads} \rightarrow \text{COIN}), \\
\text{TAILS} & = (\text{tails} \rightarrow \text{COIN}).
\end{align*}
\]

Tossing a coin

★ Possible traces? ★
Indexed Processes and Actions

Single slot buffer that inputs a value in the range 0 to 3 and then outputs a value:

\[
\text{BUFF} = (\text{in}[i : 0..3] \rightarrow \text{out}[i] \rightarrow \text{BUFF}).
\]

equivalent to

\[
\text{BUFF} = (\text{in}[0] \rightarrow \text{out}[0] \rightarrow \text{BUFF}) \mid (\text{in}[1] \rightarrow \text{out}[1] \rightarrow \text{BUFF}) \mid (\text{in}[2] \rightarrow \text{out}[2] \rightarrow \text{BUFF}) \mid (\text{in}[3] \rightarrow \text{out}[3] \rightarrow \text{BUFF}).
\]
Indexed Processes and Actions

Single slot buffer that inputs a value in the range 0 to 3 and then outputs a value:

\[ \text{BUFF} = (\text{in}[i : 0..3] \rightarrow \text{out}[i] \rightarrow \text{BUFF}) \].

equivalent to

\[ \text{BUFF} = (\text{in}[0] \rightarrow \text{out}[0] \rightarrow \text{BUFF} \\
| \text{in}[1] \rightarrow \text{out}[1] \rightarrow \text{BUFF} \\
| \text{in}[2] \rightarrow \text{out}[2] \rightarrow \text{BUFF} \\
| \text{in}[3] \rightarrow \text{out}[3] \rightarrow \text{BUFF} \\
) \].
or using a constant and indexed process \texttt{BUFF[i]}:

\begin{align*}
\text{const } & N \quad = \quad 3 \\
\text{BUFF} & \quad = \quad (\text{in}[i : 0..N] \rightarrow \text{BUFF}[i]), \\
\text{BUFF}[i : 0..N] & \quad = \quad (\text{out}[i] \rightarrow \text{BUFF}).
\end{align*}

or using a process parameter with default value:

\[ \text{BUFF}(N = 3) = (\text{in}[i : 0..N] \rightarrow \text{out}[i] \rightarrow \text{BUFF}). \]
Guarded Actions

The choice \( \textbf{when } B \ x \rightarrow P \ | \ y \rightarrow Q \) describes a process that is like \((x \rightarrow P \ | \ y \rightarrow Q)\) except that the action \(x\) can only be chosen \textbf{when} the guard \(B\) is true.
Example: A Counter

\[
\begin{align*}
\text{COUNT}(N = 3) &= \text{COUNT}[0], \\
\text{COUNT}[i : 0..N] &= \begin{cases} 
\text{inc} \to \text{COUNT}[i + 1] & \text{when } (i < N) \\
\text{dec} \to \text{COUNT}[i - 1] & \text{when } (i > 0)
\end{cases}
\end{align*}
\]
Example: A Countdown Timer

A countdown timer which beeps after $N$ ticks, or can be stopped.

\[
\begin{align*}
\text{COUNTDOWN}(N = 3) &= (\text{start} \rightarrow \text{COUNTDOWN}[N]), \\
\text{COUNTDOWN}[i: 0..N] &= (\text{when } (i > 0) \text{ tick} \\
&\quad \rightarrow \text{COUNTDOWN}[i - 1] \\
&\quad | \text{when } (i == 0) \text{ beep} \rightarrow \text{STOP} \\
&\quad | \text{stop} \rightarrow \text{STOP})
\end{align*}
\]
Example: A Countdown Timer

A countdown timer which beeps after \( N \) ticks, or can be stopped.

\[
\text{COUNTDOWN}(N = 3) = (\text{start} \rightarrow \text{COUNTDOWN}[N]),
\]
\[
\text{COUNTDOWN}[i : 0..N] = (\text{when } (i > 0) \text{ tick}
\rightarrow \text{COUNTDOWN}[i - 1]
| \text{when } (i == 0) \text{ beep} \rightarrow \text{STOP}
| \text{stop} \rightarrow \text{STOP}
).
\]

★ LTS? ★
Example: What is this?

★ What is the following FSP process equivalent to? ★

\[
\text{const False } = 0 \\
\text{P } = (\text{when (False) do anything } \rightarrow \text{P}).
\]
Constant and Range Declarations

index expressions to model a calculation:

\[
\begin{align*}
\text{const } N &= 1 \\
\text{range } T &= 0..N \\
\text{range } R &= 0..2*N \\
\text{SUM} &= (\text{in}[a:T][b:T] \rightarrow \text{TOTAL}[a+b]), \\
\text{TOTAL}[s:R] &= (\text{out}[s] \rightarrow \text{SUM}).
\end{align*}
\]
**Constant and Range Declarations**

index expressions to model a calculation:

```plaintext
const N = 1
range T = 0..N
range R = 0..2*N

SUM = (in[a:T][b:T] -> TOTAL[a+b]),
TOTAL[s:R] = (out[s] -> SUM).
```

★ Write SUM using basic FSP? ★
Process Alphabets

- The alphabet of a process is the set of actions in which it is allowed to engage.
- This is usually determined implicitly as the actions in which it can engage.
- But the implicit alphabet can be extended:

\[
\text{WRITER} = (\text{write}[1] \rightarrow \text{write}[3] \rightarrow \text{WRITER}) + \{\text{write}[0..3]\}.
\]

The alphabet of \text{WRITER} is the set \{\text{write}[0..3]\}; i.e. the set \{\text{write}[0], \text{write}[1], \text{write}[2], \text{write}[3]\}.
Outline

Topic 1: Introduction
   General Background
   On Concurrency
   Examples
   Implementation

Topic 2: Modelling Processes with FSP - I
   Labelled Transition Systems
   FSP: Basic Elements
   Summary
FSP: Summary

<table>
<thead>
<tr>
<th>Forms of process expression</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix action</td>
<td>(coffee→DRINKS)</td>
</tr>
<tr>
<td>guarded action</td>
<td>(when (i == 0) beep→STOP)</td>
</tr>
<tr>
<td>deterministic choice</td>
<td>(red→COFFEE</td>
</tr>
<tr>
<td>non-deterministic choice</td>
<td>(toss→HEADS</td>
</tr>
<tr>
<td>dependent process</td>
<td>(out[i]→BUFF)</td>
</tr>
<tr>
<td>indexed choice</td>
<td>(in[i : 0..3]→BUFF[i])</td>
</tr>
<tr>
<td>process name</td>
<td>DRINKS,</td>
</tr>
<tr>
<td></td>
<td>BUFF[i]</td>
</tr>
</tbody>
</table>
Process equation: \[
\text{process} \_\text{name} = \text{process} \_\text{expression}
\]
\[
\begin{align*}
\text{declarations} \\
\text{main} \_\text{process} \_\text{equation}, \\
\text{local} \_\text{process} \_\text{equation}, \\
\vdots \\
\text{local} \_\text{process} \_\text{equation}, \\
\text{local} \_\text{process} \_\text{equation}.
\end{align*}
\]