Processes and Threads

- **Execution of a sequential program is a process**
- **Concurrent programs consist of multiple processes.**

*What about OS processes?*
- A traditional Unix process had only one thread of control.
  - Modern operating systems allow a heavyweight process to consist of multiple lightweight processes, each with its own thread of control.
- **Threads are lightweight processes.**
Processes and Threads

Single-threaded processes and threads can be modelled in the same way.
• (ignoring performance, scheduling and timing issues)

We use finite state machines for this modelling.

Labelled Transition Systems

Special form of finite state machines.
Used to model states of concurrent programs and transitions between them.

$LTS := (S, T, A, [], c)$ where
• $S$ (a finite set of states)
• $T \subseteq S \times S$ (a finite set of transitions)
• $A$ (an alphabet of atomic actions)
• $[]: T \rightarrow A$ (a transition labelling)
• $c: S$ (the current state)
Graphic LTS Notation

LTS Semantics

- All actions that are annotations of transitions starting from the current state are enabled.
- If a process engages in the enabled action, the target of the transition becomes the current state.
- In this way LTS determines all possible traces of the process.
Finite State Processes (FSP)

- LTS become unmanageable for large number of states and transitions.
- Process algebras determine LTSs in a more concise way.

Finite State Processes (FSP):
- machine readable notation for a process algebra.
- For each FSP model an equivalent LTS can be constructed automatically.

FSP Introduction: Action Prefix

Let \( x \) be an action and \( P \) a process.

*The action prefix \( (x \rightarrow P) \) is process that initially engages in action \( x \) and then behaves in the same way as process \( P \).*

- Used to model atomic actions
- Actions have lower case identifiers, states have upper case identifiers

Example:

\[
\text{ONESHOT} = (\text{once} \rightarrow \text{STOP}).
\]

Equivalent LTS:

\[
\begin{array}{c}
0 \\
\downarrow \text{once} \\
1
\end{array}
\]
FSP Introduction: Recursion

Let \( P \) be a process. Then \( P \) may be used in action prefixes in a recursive way.

- Used to model repetitive behaviour

Example:

LTS:

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

- Note: Processes (OFF, ON, etc) are equivalent to states in the LTS.

FSP Introduction: Local Processes

It is not necessary for all processes (states) to be globally visible.

- Restrict the processes using ‘,’

Example:

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

- OFF and ON are not visible outside SWITCH

- Equivalent to:

\[
\text{SWITCH} = (\text{on} \rightarrow \text{off} \rightarrow \text{SWITCH}).
\]
FSP Introduction: Choice

\((x \rightarrow P \mid y \rightarrow Q)\) describes a choice that engages either in \(x\) or \(y\).

After \(x\) it continues with \(P\)
After \(y\) it continues with \(Q\)

Example:

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

Equivalent LTS:

```
FSP Introduction: Indexes

A range type is a finite and scalar type:

- Example: range \( T = 0..3 \)

If \( T \) is a range type then \( x[i:T] \) is the declaration of an action index and \( P[i:T] \) declares an indexed process.

- A process index variable is valid within the process, an indexed action is valid within the scope of the choice.
```
FSP Introduction: Index Example

\[
\begin{align*}
\text{const } N &= 1 \\
\text{range } T &= 0..N \\
\text{range } R &= 0..2N \\
\text{SUM} &= (\text{in}[a:T][b:T]->\text{OUT}[a+b]), \\
\text{OUT}[s:R] &= (\text{out}[s]->\text{SUM}).
\end{align*}
\]

**Equivalent LTS:**

```
\begin{array}{c}
0 \\
\text{in.0.0} \\
\text{out.0} \\
1 \\
\text{in.0.1} \\
\text{out.1} \\
2 \\
\text{in.1.0} \\
\text{out.2} \\
3
\end{array}
```

FSP Intro: Guarded Actions

The *guarded action* when \( B x->P \) means that when the guard \( B \) is true action \( x \) is enabled and the process proceeds as \( P \).

**Example:**

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

**Equivalent LTS:**

```
\begin{array}{c}
0 \\
\text{inc} \\
\text{dec} \\
1 \\
\text{inc} \\
\text{dec} \\
2 \\
\text{inc} \\
\text{dec} \\
3
\end{array}
```
Summary

- Formal Definition of LTS
- Algebraic notation in FSP
- Equivalence between LTS and FSP
- FSP and LTS concepts introduced so far are sufficient for sequential programs

- Next session: FSP constructs for modelling concurrent programs
- Solve Exercises 1 and 2 of tutorial sheet