C340 Concurrency:
Modelling Processes

Wolfgang Emmerich

1

Processes and Threads

■ *Execution of a program is a <u>process</u>*

■ *<u>Concurrent programs</u> consist of multiple processes*

■ *<u>Threads</u> are lightweight processes*

■ *Both threads and processes can be modelled in the same way*

■ *We use finite state machines for that*

2

Labelled Transition Systems

- **Special form of finite state machines**
- **Used to model states of concurrent programs and transitions between them**
- **LTS:=(S,T,A,δ,c) where**
  - **S (a finite set of states)**
  - **T $\subseteq$ S$\times$S (a finite set of transitions)**
  - **A (an alphabet of atomic actions)**
  - **δ: T$\rightarrow$ A (a transition labelling)**
  - **c$\in$ S (the current state)**

© Wolfgang Emmerich, 1998/99

3

Graphic LTS Notation
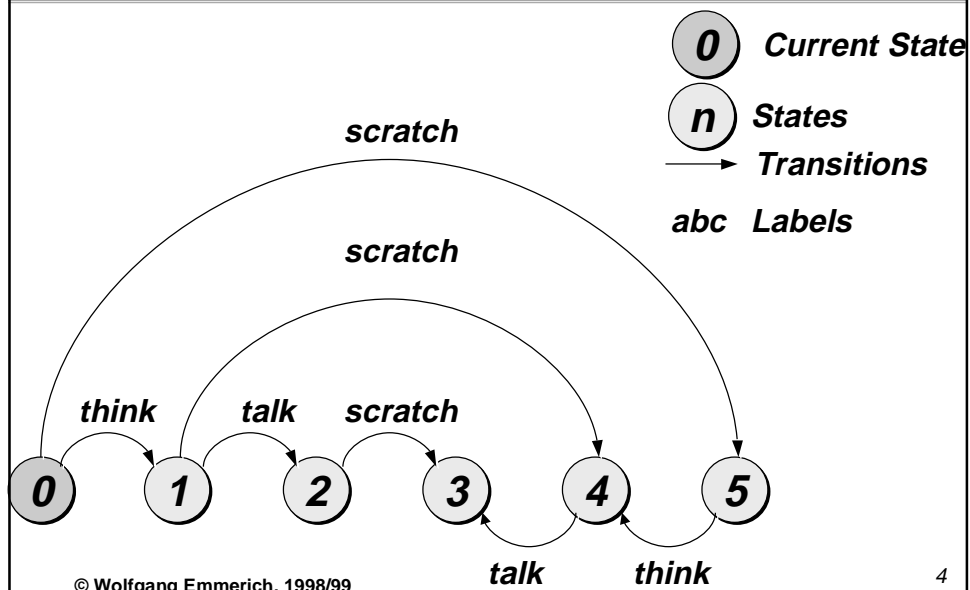
© Wolfgang Emmerich, 1998/99

4

LTS Semantics

- ***All actions that are annotations of transitions starting from the current state are enabled***

- ***If process engages in enabled action target of transition becomes current state***

Demo

- ***In this way LTS determines all possible traces of process***

5

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***

6

FSP Intro: Action Prefix

- *Let* **x** *be an action and* **P** *a process. The* *action prefix* **(x->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:* **ONESHOT=(once->STOP).**
- *Equivalent LTS:*
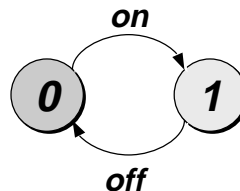
once

0 → **1**
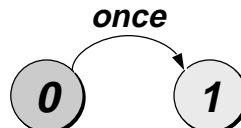
7

FSP Intro: Recursion

- *Let* **P** *be a process. Then* **P** *may be used in action prefixes in a recursive way.*
- *Used to model repetitive behaviour*
- *Example:*
  ```
  SWITCH=OFF.
  OFF    =(on->ON).
  ON     =(off->OFF).
  ```
- *Equivalent LTS:*

on

0 ⇄ **1**

off

- *Note: Processes are equivalent to states*

8

FSP Intro: Local Processes

- *It is not necessary for all states/processes to be globally visible.*
- *Restricting states/processes by use of ','*
- *Example:*
  ```
  SWITCH=OFF,
  OFF=(on->ON),
  ON=(off->OFF).
  ```
- OFF *and* ON *are not visible outside* SWITCH
- *Equivalent to:*
  ```
  SWITCH=(on->off->SWITCH).
  ```
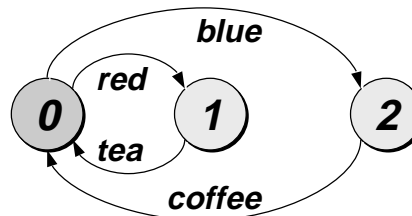
© Wolfgang Emmerich, 1998/99

FSP Intro: Choice

- (x->P|y->Q) *describes a <u>choice</u> that engages either in* x *or* y*. After* x *it continues with* P*, after* y *it continues with* Q
- *Example:* DRINKS=(
  ```
        red->tea->DRINKS
      | blue->coffee->DRINKS
      ).
  ```
- *Equivalent LTS:*

© Wolfgang Emmerich, 1998/99

FSP Intro: 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] is declares an indexed process.**
- **A _process index variable_ is valid within the process, an indexed action is valid within the scope of the choice.**

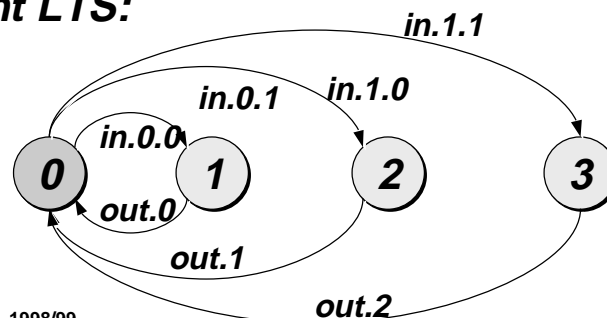## FSP Intro: Index Example

```
const N =1
range T =0..N
range R =0..2*N
SUM      =(in[a:T][b:T]->OUT[a+b],
OUT[s:R]=(out[s]->SUM).
```

- **Equivalent LTS:**
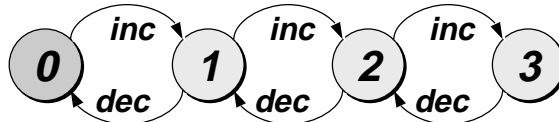

in.1.1

in.0.1   in.1.0

in.0.0

(0) (1) (2) (3)

out.0

out.1

out.2

FSP Intro: Guarded Actions

- *The <u>guarded action</u> `when B x->P` means that when the guard `B` is true action `x` is enabled and the process proceeds as `P`.*

- *Example:*
  ```
  COUNT(N=3) =COUNT[0],
  COUNT[i:0..N]=(when(i<N) inc->COUNT[i+1]
                |when(i>0) dec->COUNT[i-1]
                ).
  ```

- *Equivalent LTS:*

13

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*

14