

Concurrent Processes

CS 3SD3

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton,
Ontario, Canada

Concurrent Processes

- Concurrent process is a **composition** of sequential processes.
- *Hidden assumption*: Concurrent systems can be decomposed into sequential systems.

① Process (sequential): A sequence of action



② Model of a process: Finite state machine



③ A possible implementation of processes: Threads in Java.

The approach 1,2,3 is not the only one, but we will concentrate on it in this course.

Concepts: Processes - units of sequential execution

Models: Finite State Processes (FSP)

To model processes as sequences of actions

Labelled Transition Systems (LTS)

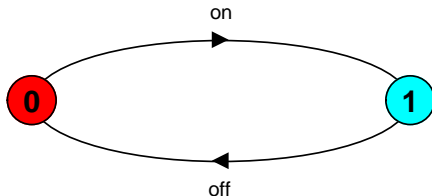
To analyze, display and animate behaviour

Practice: Java threads

-
- LTS - graphical form
 - FSP - algebraic form
-

- Tool LTSA takes FSP and analyses them.
- Different names for the same concepts:
 - LTS - automata, state machines
 - FSP - CSP (Communicating Sequential Processes), Processes in Process Algebras

- A process is the execution of a sequential program. It is modeled as a finite state machine which transits from state to state by executing a sequence of atomic actions.



a light switch
LTS

on→off→on→off→on→off→

a sequence of
actions or *trace*

- How can it be modelled by an algebraic expression?

- If x is an action and P is a process then

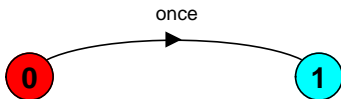
$$(x \rightarrow P)$$

describes a process that initially engages in the action x and then behaves exactly as described by P .

ONESHOT = (**once** \rightarrow **STOP**).

ONESHOT state
machine

(terminating process)

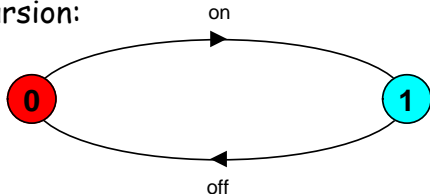


- Convention: actions begin with lowercase letters while PROCESSES begin with uppercase letters

FSP -action prefix and recursion

Repetitive behaviour uses recursion:

```
SWITCH = OFF,  
OFF    = (on -> ON),  
ON     = (off-> OFF).
```



Substituting to get a more succinct definition:

```
SWITCH = OFF,  
OFF    = (on ->(off->OFF)).
```

And again:

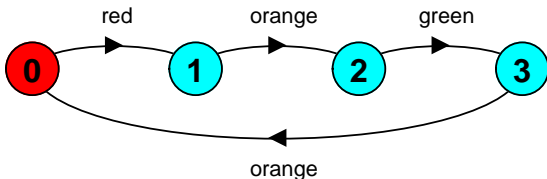
```
SWITCH = (on->off->SWITCH).
```

FSP model of a traffic light (in Europe)

FSP model of a traffic light :

```
TRAFFICLIGHT = (red->orange->green->orange  
-> TRAFFICLIGHT).
```

LTS generated using *L TSA*:



Trace:

red→orange→green→orange→red→orange→green ...

FSP - choice

- If x and y are actions then

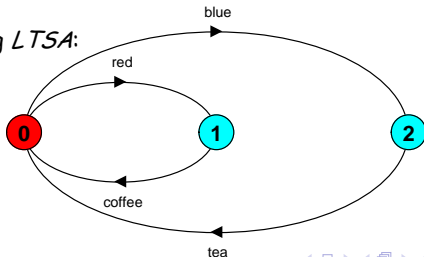
$$(x \rightarrow P \mid y \rightarrow Q)$$

describes a process which initially engages in either of the actions x or y . After the first action has occurred, the subsequent behavior is described by P if the first action was x and Q if the first action was y .

FSP model of a drinks machine :

```
DRINKS = (red->coffee->DRINKS  
          | blue->tea->DRINKS  
          ).
```

LTS generated using *LTSA*:



Possible traces?

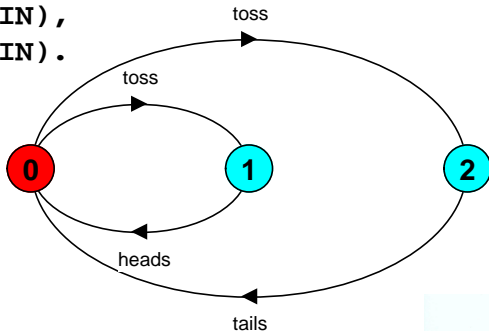
Non-deterministic choice

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

```
COIN = (toss->HEADS | toss->TAILS),  
HEADS = (heads->COIN),  
TAILS = (tails->COIN).
```

**Tossing a
coin.**

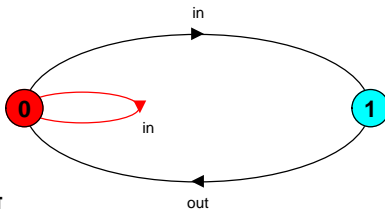
Possible traces?



Modeling failure

- How do we model an unreliable communication channel which accepts **in** actions and if a failure occurs produces no output, otherwise performs an **out** action?

Use non-determinism...



```
CHAN = (in->CHAN  
        | in->out->CHAN  
        ).
```

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

```
BUFF = (in[i:0..3]->out[i]-> BUFF).
```

equivalent to

```
BUFF = (in[0]->out[0]->BUFF
| in[1]->out[1]->BUFF
| in[2]->out[2]->BUFF
| in[3]->out[3]->BUFF
).
```

indexed actions
generate labels of
the form
action.index

or using a **process parameter** with default value:

```
BUFF(N=3) = (in[i:0..N]->out[i]-> BUFF).
```

More helpful syntax

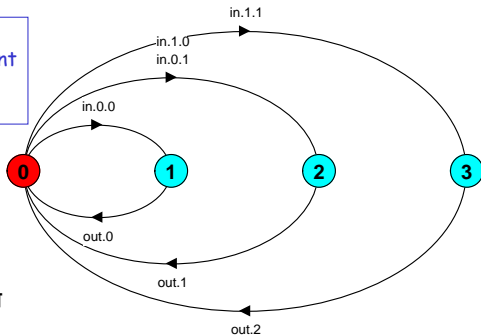
Local indexed process definitions are equivalent to process definitions for each index value

index expressions to model calculation:

```
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).
```

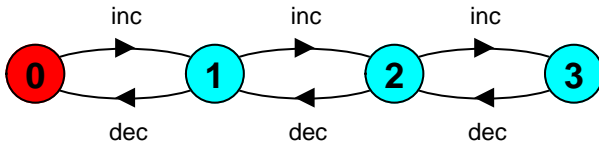
- Notation: $in[0][1] = in.0.1$, $out[2] = out.2$, etc.



FSP - guarded actions

- The choice ($\text{when } B \ x \rightarrow P \mid y \rightarrow Q$) means that when the guard B is true then the actions x and y are both eligible to be chosen, otherwise if B is false then the action x cannot be chosen.

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

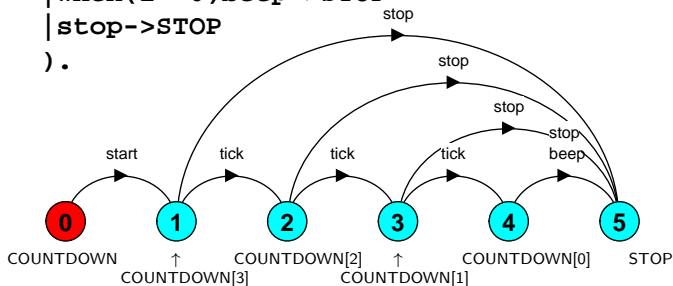


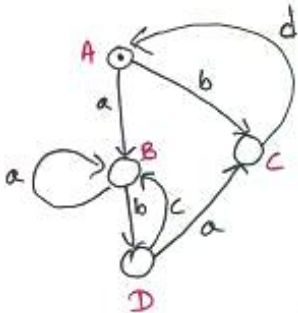
- It usually occurs in the form ($\text{when } B \ x \rightarrow P \mid \text{when } \neg B \ y \rightarrow Q$), so the choice between $x \rightarrow P$ and $y \rightarrow Q$ is exclusive.

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

```

COUNTDOWN (N=3)  = (start->COUNTDOWN[N]),
COUNTDOWN[i:0..N] =
    (when(i>0) tick->COUNTDOWN[i-1]
    | when(i==0) beep->STOP
    | stop->STOP
    ).
  
```





$$A = (a \rightarrow B \mid b \rightarrow C)$$

$$B = (a \rightarrow B \mid b \rightarrow D)$$

$$C = (d \rightarrow A)$$

$$D = (a \rightarrow C \mid c \rightarrow B)$$

FSP \rightarrow LTS

$$A = (a \rightarrow b \rightarrow B \mid b \rightarrow (a \rightarrow c \rightarrow A \mid b \rightarrow B))$$

$$B = (a \rightarrow c \rightarrow (a \rightarrow A \mid b \rightarrow b))$$

often some parentheses can be omitted for readability, i.e., we may write:

$$A = a \rightarrow b \rightarrow B \mid b \rightarrow (a \rightarrow c \rightarrow A \mid b \rightarrow B)$$

$$B = a \rightarrow c \rightarrow (a \rightarrow A \mid b \rightarrow B)$$

$$B_1 = b \rightarrow B$$

$$A_1 = c \rightarrow A$$

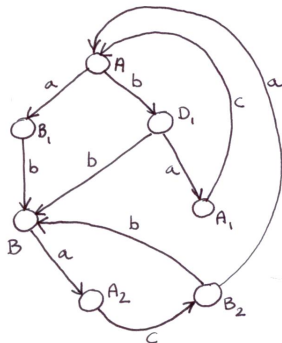
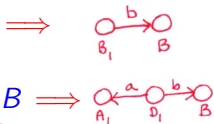
$$D_1 = a \rightarrow A_1 \mid b \rightarrow B \Rightarrow$$

$$A = a \rightarrow B_1 \mid b \rightarrow D_1$$

$$B_2 = a \rightarrow A \mid b \rightarrow B$$

$$A_2 = c \rightarrow B_2$$

$$B = a \rightarrow A_2$$



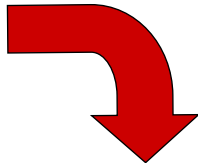
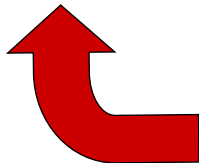
- The alphabet of a process is the set of actions in which it can engage.
- Process alphabets are **implicitly** defined by the actions in the process definition.
- Alphabet extension can be used to extend the implicit alphabet of a process:

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

Alphabet of *WRITER* is the set

$$\{write[0..3]\} = \{write[0], write[1], write[2], write[3]\}.$$

Modeling **processes** as
finite state machines
using FSP/LTS.



Implementing **threads**
in Java.

- Process \implies models as FSP or LTS
- Thread \implies implementation in Java

Concurrency, Parallelism: definitions

- **Concurrency:** Logically simultaneous processing. Does not imply multiple processing elements (PEs). Requires interleaved execution on a single PE.
- **Parallelism:** Physically simultaneous processing. Involves multiple PEs and/or independent device operations.

The textbook uses the terms parallel and concurrent interchangeably and generally do not distinguish between real and pseudo-concurrent execution.

- **These are the authors definitions!**
- **They are NOT universally accepted!**
- **WHAT ABOUT SIMULTANEITY AND SIMULTANEOUS EXECUTIONS?! They may make a substantial difference!**

- How should we model process execution speed?
Arbitrary speed (we abstract away time)
- How do we model concurrency?
Arbitrary relative order of actions from different processes
(**interleaving** but preservation of each process order)

!!? MANY CONSIDER THIS APPROACH AS AN
OVERSIMPLIFICATION!

- What is the result?
It provides a general model independent of scheduling
(**asynchronous** model of execution)

!!? MANY CONSIDER THE LAST STATEMENT AS AN
UNJUSTIFIED OVERSTATEMENT!

Parallel composition - action interleaving

- If P and Q are processes then $(P||Q)$ represents the concurrent execution of P and Q . The operator $||$ is the parallel composition operator.

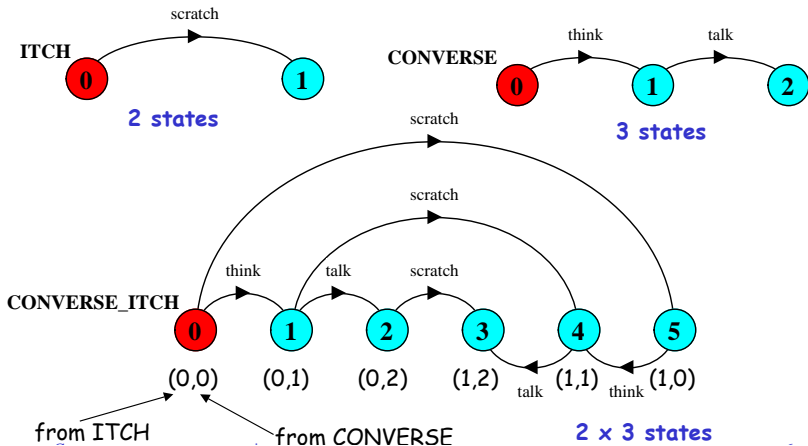
```
ITCH  = (scratch->STOP).  
CONVERSE = (think->talk->STOP).  
  
||CONVERSE_ITCH = (ITCH || CONVERSE).
```

Disjoint
alphabets

```
think->talk->scratch  
think->scratch->talk  
scratch->think->talk
```

Possible traces as
a result of action
interleaving.

Parallel composition - action interleaving



- Transformation into LTS is NOT the best solution, transformation into *Petri nets* is better!

Parallel composition: Algebraic Laws

- **Commutativity:** $P \parallel Q = Q \parallel P$
 - **Associativity:** $P \parallel (Q \parallel R) = (P \parallel Q) \parallel R = P \parallel Q \parallel R$
-

Problem: *What these equalities mean?*

- The set of traces that is generated is the same for the left and the right side, but is this sufficient?
- Semantics is not defined! In a decent scientific paper such “laws” would not survive!
- *Semantics should be defined before!*
- LTS are also the same for the left and right side of equations? Do they define semantics?

Example (Clock Radio)

$CLOCK = tick \rightarrow CLOCK$

$RADIO = on \rightarrow off \rightarrow RADIO$

$\parallel CLOCK_RADIO = CLOCK \parallel RADIO$

Modelling interaction - shared actions

- If processes in a composition have actions in common, these actions are said to be *shared*. Shared actions are the way that process interaction is modeled. While unshared actions may be arbitrarily interleaved, a *shared action must be executed at the same time* by all processes that participate in the shared action.

Example (Maker-user)

$MAKER = make \rightarrow ready \rightarrow MAKER$

$USER = ready \rightarrow use \rightarrow USER$

$\parallel MAKER_USER = Maker \parallel USER$

Traces:

$make \rightarrow ready \rightarrow use \rightarrow make \rightarrow ready \rightarrow make \rightarrow use \rightarrow \dots$

Example (Maker-user)

MAKER = *make* \rightarrow *ready* \rightarrow MAKER

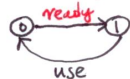
USER = *ready* \rightarrow *use* \rightarrow USER

\parallel MAKER_USER = Maker \parallel USER

MAKER

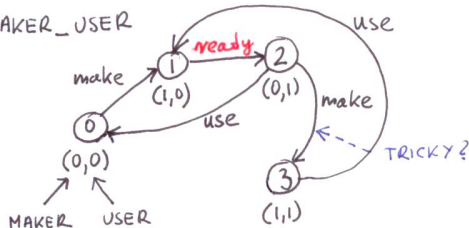


USER



LTS:

\parallel MAKER_USER



- IT IS MUCH EASIER AND MORE INTUITIVE TO REPRESENT SYSTEMS LIKE MAKER-USER WITH *PETRI NETS*!