Concurrent Execution II

Lecture 5

Alma Rahat

CS-210: Concurrency

9 February, 2021

Questions and comments: shorturl.at/knJM5

What did we do in the last session?

- We can model a thread's lifecycle using FSP.
- We can use indices in FSP: `store[i:0..3]`.
- We can use guarded actions: `(when B x-> | y -> Q)`.
- We model processes as logically parallel, but in reality they may be interleaved.

In this lecture...

Learning outcomes.

1. To explain how processes are interleaved during execution.
2. To construct models of parallel processes and shared actions between them.

Outline.

1. Revisit multi-process execution and interleaving.
2. Modelling Concurrency.
    - Parallel composition.
    - Labelling.
    - Sharing between processes
        - Actions.

Process Scheduling (Interleaving) Simulation

If we have one process running on one processor, we have no issues: it runs as you would expect. When we have multiple processes – running in parallel, and potentially each have idle times in-built (e.g. input/output – we need to decide how best to utilise the processors. This is where process scheduling comes to play. There are various algorithms, e.g. first-come-first-serve and round-robin.

Scheduling Criteria

- CPU utilisation: Is the CPU busy at least most of the time?
- Throughput: What is the number of processes completed per unit time?
- Turnaround time: How much does a particular process take to complete?
- Waiting time: How long does a process queue waiting for its turn?
- Response time: How long does it take to handle interactions?

Simulator: `ess.cs.tu-dortmund.de/Software/AnimOS/CPU-Scheduling/`

As discussed before, we will not worry about the interleaving: we will write models and codes to scenarios with multiple processes that work in a concurrent manner and correctly irrespective to scheduling.

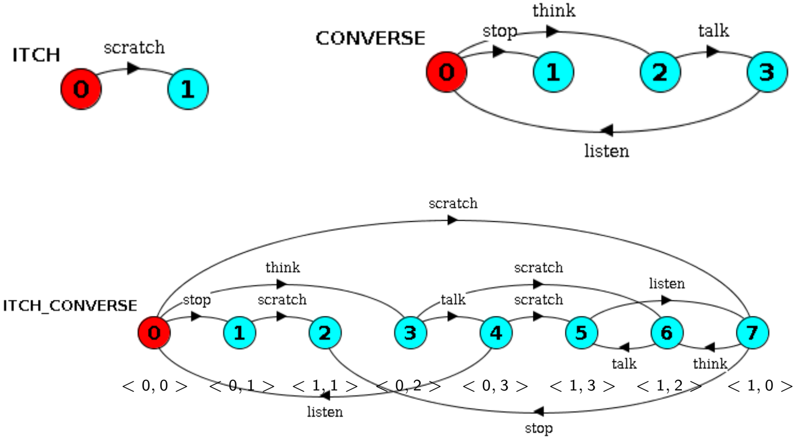
Parallel Composition

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

Consider itching while having a conversation!

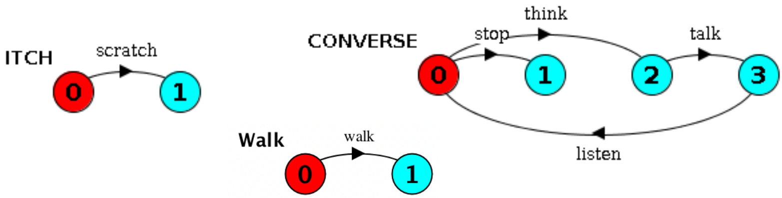
```
ITCH = (scratch -> STOP).\\process for itching.
CONVERSE = (think -> talk -> listen -> CONVERSE |
stop -> STOP).\\process for conversing.
||ITCH_CONVERSE = (ITCH || CONVERSE).\\composition of
two processes.
```

How many combined states?



Cartesian product of two sets of states: $\{< i, j > | \forall i \in P \wedge \forall j \in Q \wedge P||Q\}$.
Therefore, total number of states: $|P| \times |Q|$.
`ITCH` has two states, and `CONVERSE` has four states: so `ITCH||CONVERSE` should $2 \times 4 = 8$ states (all possible combinations of actions.)

How many combined states?



Exercise: How many states do you think the parallel composition of *ITCH*, *CONVERSE* and *Walk* would have?

Please go to `www.menti.com` and use the code 13 58 67 0.

Algebraic Laws

Commutative (P || Q) = (Q || P).

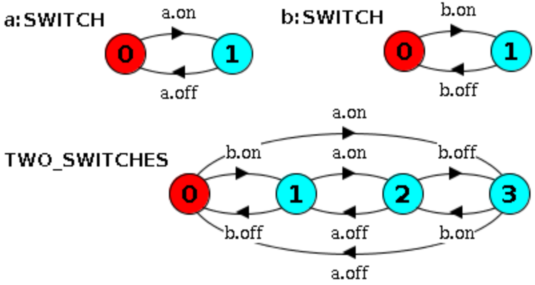
Associative ((P || Q) || R)
= (P || (Q || R))
= (P || Q || R).

Process Instances and Labelling

a:P creates an instance of process P and prefixes each action label in the alphabet of P with a.

Two instances of SWITCH process.

```
SWITCH = (on -> off -> SWITCH).
||TWO_SWITCH = (a:SWITCH || b:SWITCH).
```



What happens if there are no labels?

An Array of Switches

```
SWITCH = (on -> off -> SWITCH).
||SWITCHES(N = 3) = (forall[i:1..N] s[i]:SWITCH).
```

```
SWITCH = (on -> off -> SWITCH).
||SWITCHES(N = 3) = (s[i:1..N]:SWITCH).
```

Any questions?

Sharing Between Processes

Simple rules to understand the nature of sharing:

Action Common behaviour (what the process does; method) across processes, e.g. two users can play chess with their computers.

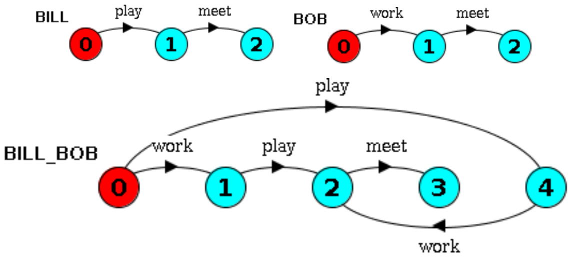
> **Remember** You cannot perform a common action until all pre-requisite actions by all sharer have been performed.

Resource Common property (has-a, uses-a or controls-a type relationship; process or object) across processes, e.g. multiple users at a household have a toaster.

Shared Actions

So far, we have seen composition of processes with disjoint alphabets: no actions are common between processes, and we can arbitrarily interleave them. In case of shared (or common) actions, we must complete all pre-requisite actions (in any order) before performing a common action.

Scenario: Bill plays tennis and then meets Bob before stopping, while Bob works and then meets Bill before stopping.



Exercise

Do you think `S1` and `S2` describe the same behaviour?

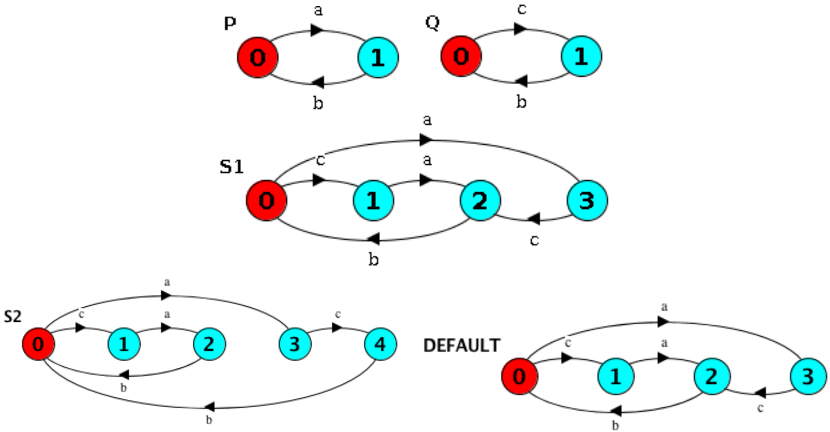
```
P = (a -> b -> P).
Q = (c -> b -> Q).
||S1 = (P || Q).
```

```
S2 = (a -> c -> b -> S2 | c -> a -> b -> S2).
```

Hint: Draw the LTS of `S1`, and see if it is equivalent to `S2`.

Please go to `www.menti.com` and use the code 10 83 21 7.

Solution



$$S2 \equiv DEFAULT$$

Use `Build > Minimise` to produce `DEFAULT` from `S2` on the LTSA tool.

Summary

- We use `(P||Q)` for parallel composition of processes `P` and `Q`.
- We can model shared actions.