CS238: Concurrent Processes
Term 1 Seminar Sessions

Seminar Tutor: Nick Papanikolaou
Lab 327, Department of Computer Science
http://www.dcs.warwick.ac.uk/~nikos/
Email: nikos@dcs.warwick.ac.uk
(Send queries here with 'CS238' as the message subject)

1. Administrative Matters

The CS238 module is a companion to CS237 'Concurrent Programming'. There will be a total of about 10 lectures, accompanied by 4 seminar sessions. Dates and times for these follow:

Session	Date	Time	Room
Theory 1	9 November 2004	2:00pm	ACCR
Theory 2	16 November 2004	2:00pm	ACCR
Theory 3	23 November 2004	2:00pm	ACCR
Theory 4	30 November 2004	2:00pm	ACCR

2. Goals of these Sessions

These sessions are intended to:

- Help you with lecture material
- Allow you to ask questions
- Give you examples and practice
- Show you how to use the CWB tool to experiment with CCS on a PC

The structure of a seminar session is as follows:

- To motivate the session, we kick off with a review of topics and a general Warm-Up (10-15 min.).
- Then, examples and sample problems are presented.

3. Warm-up for Session 1

- Concurrency is a general phenomenon, not only applicable to computers. Concurrent *programming* views the subject from a computing perspective only. The goal of CS238 is to regard concurrency in a machine-independent way, to allow us to reason about concurrent systems in general.
- Why is formal reasoning about concurrency necessary? Looking at classical sequential programs in a formal way is recognised as a good way of debugging them in them in the first place; this is even more important for concurrent programs, which are much more complicated.
- Concurrent programs are becoming increasingly important due to applications such as:
    - Multiprocessor systems
    - Real-time systems
    - Safety-critical systems
- Formal methods such as the CCS algebra allow us to prove that such systems (and programs running on them) will operate according to specification.
- We cannot use formal methods that apply for classical systems, because they are not powerful enough.

4. Key Topics for Session 1

1) Defining simple agents;
2) Getting agents to execute at the same time (*concurrent composition*)
3) Getting agents to *synchronize*, i.e. do the same thing at the same time.
4) Semaphores as examples of synchronization

Topic 1: Defining simple agents

An *agent* is anything or anyone capable of performing an *action*. But in general it is thought of as representing a *state*, to which changes occur.
Examples of agents:

$$\mathtt{Student} = \mathtt{wakeup.breakfast.study.sleep.Student}$$

$$\mathtt{Badstudent} = \mathtt{sleep.0}$$

$$\mathtt{Lecturer} = \mathtt{teach.Ordinaryman}$$

$$\mathtt{Ordinaryman} = \mathtt{livenow.paylater.Lecturer}$$

This last example is one of mutual recursion: a lecturer behaves like an ordinary man for a while, and then behaves like a lecturer…

Topic 2: Getting agents to execute at the same time (*concurrent composition*)

If we are given the agents

$$\mathtt{Startcar} = \mathtt{keystop1.keystop2.keystop3.go.0}$$

$$\mathtt{Talktowife} = \mathtt{listen.listen.say.listen.Talktowife}$$

then the concurrent composition of **Startcar** and **Talktowife** represents one of many possible interleavings of their actions, for example:

$$(\mathtt{Startcar}|\mathtt{Talktowife}) \rightarrow$$
$$\rightarrow \mathtt{listen.keystop1.listen.keystop2.}$$
$$\mathtt{keystop3.say.listen.go.Talktowife}$$

Topic 3: Getting agents to *synchronize*, i.e. do the same thing at the same time

Your wife would expect you to listen carefully to what she is saying, so the definitions of agents **Startcar** and **Talktowife** have to be amended:

$$\mathtt{Startcar} = \mathtt{keystop1.wait.keystop2.keystop3.go.0}$$
$$\mathtt{Talktowife} = \mathtt{listen.listen.}\underline{\mathtt{wait}}\mathtt{.say.listen.Talktowife}$$

$$(\mathtt{Startcar}|\mathtt{Talktowife})\backslash\{\mathtt{wait}\} \rightarrow$$
$$\rightarrow \mathtt{listen.keystop1.listen.}\tau\mathtt{.keystop2.}$$
$$\mathtt{keystop3.say.go.listen.Talktowife}$$

Note that the conventional notation for interactions, i.e. actions to synchronize upon, is to place a bar *over* the action name, not to underline it as we have here!

Topic 4: Semaphores as examples of synchronization

Semaphores were presented in CS237 as a trick to block two processes from gaining access to the same resource at the same time. A binary semaphore allows only one agent to acquire a resource at a time. Each semaphore agent for a given problem denotes that resource.

$$\mathtt{Forksem} = \mathtt{get.put.Forksem}$$
$$\mathtt{John} = \underline{\mathtt{get}}\mathtt{.eat.talk.}\underline{\mathtt{put}}\mathtt{.0}$$
$$\mathtt{George} = \mathtt{talk.talk.}\underline{\mathtt{get}}\mathtt{.eat.}\underline{\mathtt{put}}\mathtt{.0}$$
$$\mathtt{Dinner} = (\mathtt{John}|\mathtt{George}|\mathtt{Forksem})\backslash\{\mathtt{get,put}\}$$

5. Questions and Problems

1) Which of the following actions can take place when agent `P` is executed?

```
P = a.b.0 + c.d.0
```

One) `a`
Two) `b`
Three) `c`
Four) `d`

---

2) We have a vending machine V which sells Mars and Cadbury chocolate bars. A Mars bar costs 50 pence and a Cadbury bar costs 20 pence. Given that the vending machine's behaviour is specified by the agent

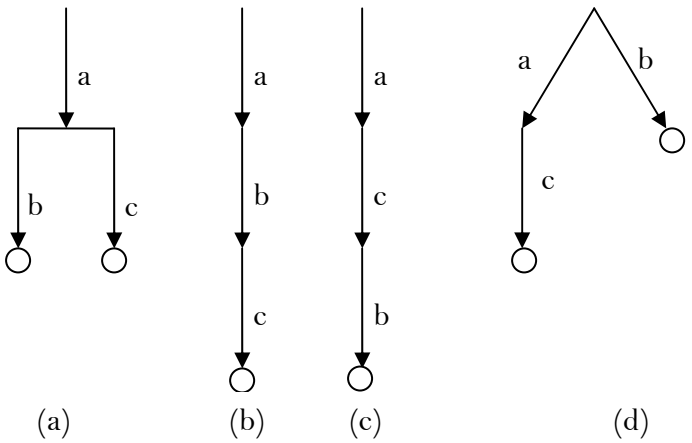
```
V = p50.mars.collect.V +
  + p20.(cadbury.collect.V +
         p20.p10.mars.collect.V) +
  + p10.(p10.cadbury.collect.V)
```

Which of the following choices will be accepted by the vending machine?

One) Cadbury's, 20 pence coin
Two) Mars, 5 × 10 pence coins
Three) Mars, 3 × 10 pence coins + 20 pence coin
Four) Mars, 2 × 20 pence coins + 10 pence coin

---

3) Match the following transition graphs to the corresponding agent definitions.



(a) (b) (c) (d)

One) `a.c.0 + b.0`
Two) `a.(b.0|c.0)`
Three) `a.c.b.0`
Four) `a.b.c.0`

4) In the agent expression

$$a.0[b/a]$$

what operation is being performed?
One) Nondeterministic choice
Two) Relabelling
Three) Restriction

---

5) Given the expressions

$$P = a.b.c.delta.0$$
$$Q = play.\underline{b}.run.Q$$

what do we call the operation $(P|Q)\backslash b$?
One) restriction on the action b
Two) random choice between P and Q

---

6. Problems

- Define an agent `Once` which can either perform the action `a` and then stop, or else perform the action `b` and then stop.
- Define the agent `T` which can either perform the action `a` once and then stop, or else perform the action `a` twice and then stop.
- Define the agent `Choose` which can infinitely often either execute an action `a` or an action `b`.
- Define the agent `X` which can either perform an action `p` and then stop, or else produce an action `p` and behave like `X`.
- Complete the following transition graph, given that
  `A = a.b.c.A` and `B = d.`$\underline{\texttt{c}}$`.B`