CS238: Concurrent Processes
Seminar Session 2

Seminar Tutor: Nick Papanikolaou
Office 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. Warm-up for Session 2

Today's session aims to clarify certain issues that arose last week and to give you some practice with agent expressions and state transition diagrams. The concept of a **buffer** will be introduced and then applied in the context of the Producers & Consumers Problem. The last part of the seminar will be devoted to this problem, which will serve as your first case study in CCS. The topics will be introduced in the following order:

1. Leftovers from Session 1
   - Why `a.(P+Q)` is different from `a.P+a.Q`
   - Why `a.(b.0 | c.0)` is different from `a.b.c.0+a.c.b.0`
2. How to evaluate/expand agent expressions
   - This is a fundamental exam skill.
3. Simplifications and State Transition Diagrams
   - Including the answer to the last question of Session 1
4. The Concept of a Buffer
5. The Producers and Consumers Problem [Case Study]

2. Leftovers from Session 1

Students often express concern about the difference between factored and unfactored agent expressions. In other words, it seems unclear whether

```
a.( P + Q )
```

is the same as

```
a.P + a.Q
```

The same question arises for concurrent composition, i.e. as to whether

```
a.( b.0 | c.0 )
```

is the same as

```
a.b.c.0 + a.c.b.0
```

Now we will study these issues in more detail.

In CCS, the **+** sign denoting nondeterministic choice **denotes a choice of agents (<u>behaviours</u>), but not a choice of actions**. Thus,

$$\texttt{a.(P + Q)} \neq \texttt{a.P + a.Q}$$

in general (it depends on `P` and `Q`, see below). Observe that,

$$\texttt{a.(b.0 + 0)} \neq \texttt{a.b.0 + a.0}$$

because `b.0 + 0 = b.0`. The right-hand side can execute an `a` action not followed by a `b`, but the left-hand side cannot.

"But **why** is `a.(P + Q)` different from `a.P + a.Q`?", someone insisted.

The difference lies in **when** a choice is made. Look at the transition diagrams for the two agents:

$$\begin{tikzcd}
\texttt{a.(P + Q)} \arrow[d, "\texttt{a}"] \\
\texttt{P + Q}
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
 & \texttt{a.P + a.Q} \arrow[dl, "\texttt{a}"'] \arrow[dr, "\texttt{a}"] & \\
\texttt{P} & & \texttt{Q}
\end{tikzcd}$$

Here, the choice between behaving as `P` *or* `Q` *is made* ***after*** *the action* `a` *is executed. Choice is delayed.*

Here, action `a` *is performed at the* ***same time*** *that the choice is being made. Choice happens immediately.*

Based on previous course material by S.G. Matthews

In `a.(P + Q)` we make a choice after the action `a`, while in `a.P + a.Q` we make our choice (and so commit ourselves) when we execute the action `a`. Once a choice is made there is no going back to undo it. To make a choice in an agent of the form `P + Q` is to choose one of `P` or `Q` and execute its first action. We cannot choose an agent and then execute its first action later.

What about concurrent composition?

Concurrent composition is related to nondeterministic choice in the following way:

$$a.P \mid b.Q = a.(P \mid b.Q) + b.(a.P \mid Q)$$

as long as a and b are unrestricted. A first action in a concurrent composition is the first action of either component agent.

Examples:

$$0 \mid 0 = 0$$

$$a.0 \mid b.0 = a.b.0 + b.a.0$$

$$a.0 \mid a.0 = a.a.0$$

$$a.(b.0 \mid c.0) \neq a.b.c.0 + a.c.b.0$$

$$\begin{aligned} a.0 \mid (b.0 + c.0) &= (a.0 \mid b.0) + (a.0 \mid c.0) \\ &= a.b.0 + b.a.0 + a.c.0 + c.a.0 \end{aligned}$$

$$(a.0 \mid b.0) + c.0 = a.b.0 + b.a.0 + c.0$$

---

3. How to evaluate an agent expression, step by step

When you are given an agent definition, it is not always obvious exactly how that agent will behave when executed. For example, to clarify the difference between $a.(P + Q)$ and $a.P + a.Q$ we used a state transition diagram. Such diagrams are a visual representation of agent behaviour. But since CCS is an algebra, there are rules that we can apply to evaluate an agent step by step. An important skill is being able to do precisely that.

$$\begin{aligned} A &= a.b.sync.c.A \\ B &= d.\underline{sync}.0 \end{aligned}$$

$$\begin{aligned} (A \mid B)\backslash\{sync\} &= (a.b.sync.c.A \mid d.\underline{sync}.0)\backslash\{sync\} \\ &\underset{a}{\Rightarrow} (b.sync.c.A \mid d.\underline{sync}.0)\backslash\{sync\} \\ &\underset{b}{\Rightarrow} (sync.c.A \mid d.\underline{sync}.0)\backslash\{sync\} \\ &\underset{d}{\Rightarrow} (sync.c.A \mid \underline{sync}.0)\backslash\{sync\} \end{aligned}$$

$$\underset{\tau}{\Rightarrow} (c.A \mid 0)\backslash\{sync\}$$

$$= (c.A)\backslash\{sync\}$$

$$\underset{c}{\Rightarrow} A\backslash\{sync\}$$

Note that this is only one of several possible ways to evaluate this agent. Instead of starting by performing action a first, we could have performed action d. The important thing is to be consistent.

4. A List of Rules used in CCS

- $P + Q = Q + P$
- $P + (Q + R) = (P + Q) + R$
- $P + P = P$
- $P + 0 = P$
- $\alpha.\tau.P = \alpha.P$
- $P + \tau.P = \tau.P$
- $\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$
- $a.P \mid b.Q = a.(P \mid b.Q) + b.(a.P \mid Q)$

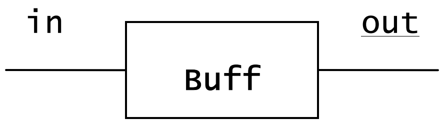
5. State Transition Diagram for Problem in Seminar 1

$$\begin{tikzcd}
 & & \boxed{(A \mid B)\backslash c} \arrow[dl, "a"'] \arrow[dr, "d"] & \\
 & \boxed{(b.c.A \mid B)\backslash c} \arrow[dl, "b"'] \arrow[dr, "d"] & & \boxed{(A \mid \underline{c}.B)\backslash c} \arrow[dl, "a"] \\
\boxed{(c.A \mid B)\backslash c} \arrow[dr, "d"'] & & \boxed{(b.c.A \mid \underline{c}.B)\backslash c} \arrow[dl, "b"] & \\
 & \boxed{(c.A|\underline{c}.B)\backslash c} \arrow[uuur, dash, bend left=60, "\tau"] & &
\end{tikzcd}$$

State Transition Diagram for
$(A|B)\backslash c$ where
$A = a.b.c.A$
$B = d.\underline{c}.B$

6. Buffers

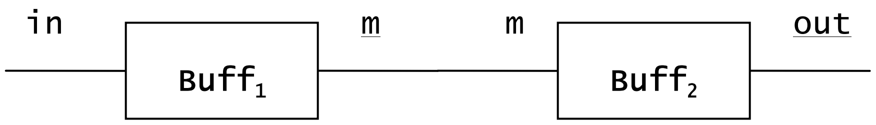
The simplest type of agent in CCS is the **buffer.** A buffer is a place where a datum is temporarily stored and retrieved from. Here is a **flow diagram** for a buffer:



The required behaviour of this process is that it waits until it receives a communication via the action `in` and then makes a communication via the action $\underline{\texttt{out}}$. We are not concerned with the nature of the data itself here. In particular, we want `Buff` to perform the action `in` followed by the action $\underline{\texttt{out}}$ an unspecified number of times:

$$\texttt{Buff} = \texttt{in}.\underline{\texttt{out}}.\texttt{Buff}$$

Consider now the case of two buffers $\texttt{Buff}_1$ and $\texttt{Buff}_2$ communicating with each other so that $\texttt{Buff}_1$ will tell $\texttt{Buff}_2$ that it has received a communication and will wait for the next.

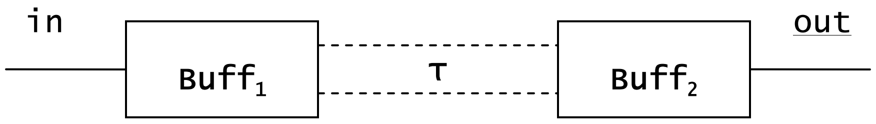


One way of interpreting the system represented by this flow diagram is that it represents a **two-place buffer** that can store a maximum of two communications at any one time. Note that $\texttt{Buff}_1$ cannot communicate with the environment via `m` unless it waits for $\texttt{Buff}_2$ to notify it of a communication received.

$$\texttt{Buff}_1 = \texttt{in}.\underline{\texttt{m}}.\texttt{Buff}_1$$

$$\texttt{Buff}_2 = \texttt{m}.\underline{\texttt{out}}.\texttt{Buff}_2$$

$$\texttt{System} = (\ \texttt{Buff}_1 \mid \texttt{Buff}_2\ )\backslash\{\texttt{m}\}$$

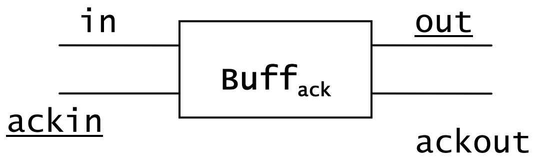


In CCS we refer to visible actions that buffers can perform as **ports.** By convention **output ports are barred** (e.g. $\underline{\text{out}}$).

7. *Acknowledging Communications – A Real-World Scenario*

In real-world communications, noise is likely to occur on any given communications channel. This phenomenon is not taken into account in CCS models, which are rather idealistic. However, one of the typical ways in which system designers deal with this problem is by requiring each communication to be acknowledged. Thus, when a message is sent from A to B, B must respond with an acknowledgment that the message was successfully received.

Consider a buffer which acknowledges the messages it receives and always expects an acknowledgement when it sends a message:

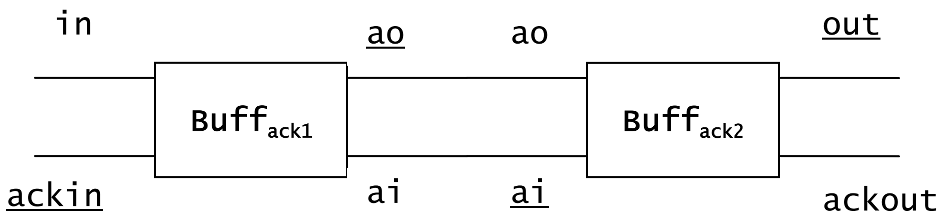


Remember the convention on barred actions: a barred action is a message **sent** by the buffer, whereas an unbarred action is a message **received** by the buffer. In this example, Buff_{ack} sends an acknowledgment $\underline{\text{ackin}}$ of the message received (`in`), sends a copy of the message ($\underline{\text{out}}$) and expects an acknowledgement `ackout`.

A possible definition for this agent would be

$$\text{Buff}_{\text{ack}} = \text{in}.\underline{\text{out}}.\text{ackout}.\underline{\text{ackin}}.\text{Buff}_{\text{ack}}$$

- Is this a safe way of defining this buffer? In other words, can we be sure that messages sent are always received? If not, write a definition which is safer.
- Give a definition of a two-place buffer with acknowledgement. We will call this buffer $\text{Buff}_{\text{ack}}^2$ and it will consist of two copies of Buff_{ack} in parallel:



8. Case Study – The Producers and Consumers Problem

From [C. Fencott, *Formal Methods for Concurrency*, International Thomson Publishing Inc, 1996]:

System Requirements: Producers and Consumers

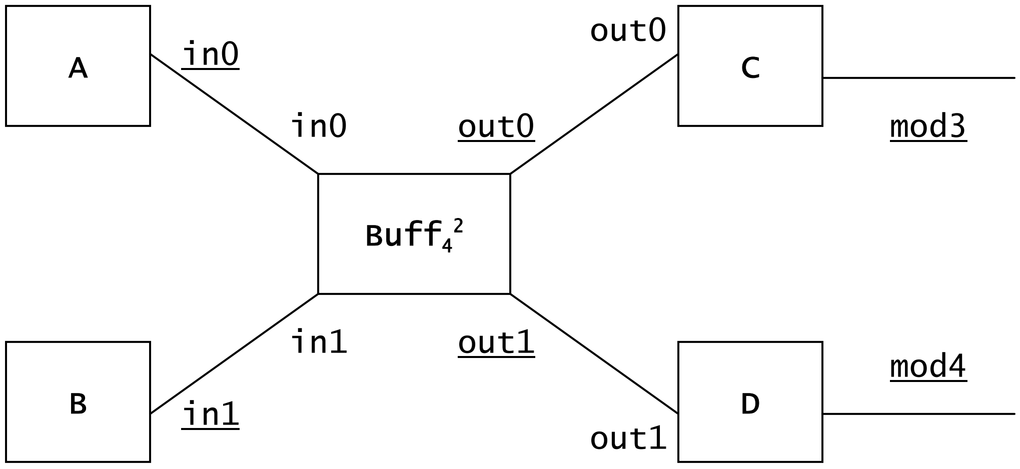
"We are required to specify a system which uses two producers, A and B, to load items into a four-place buffer which can store two types of values. Two further processes, C and D, consume items from the buffer to enable their own particular items. The first of these requires three `0`s from the buffer and the second of these requires four `1`s from the buffer. The names of the two critical actions are `mod3` and `mod4` respectively. There is no control over the order in which items are fed into the buffer or read from the buffer, save in the state of the buffer itself. The only observable actions of the system as a whole should be occurrences of the actions `mod3` and `mod4`.

Our goal is to specify this system in CCS and, in particular, to define the behaviour of the buffer.

There are five agents involved in this problem: $\mathtt{A},\ \mathtt{B},\ \mathtt{Buff}_4^2,\ \mathtt{C}$ and $\mathtt{D}$. The behaviour of the whole system is easily seen to be some composition of these:

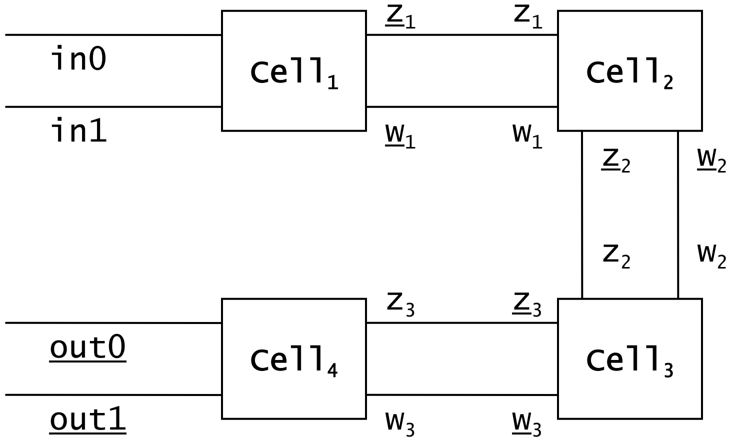
$$\mathtt{System} = (\mathtt{A} \mid \mathtt{B} \mid \mathtt{Buff}_4^2 \mid \mathtt{C} \mid \mathtt{D}) \setminus \mathtt{L}$$

⦿ YOUR TASK is to define `A, B, C` and `D` and thence to obtain the restriction set `L`. You are given the system flow diagram below.


A
$\underline{\mathtt{in0}}$
in0
$\underline{\mathtt{out0}}$
out0
C
mod3
$\mathtt{Buff}_4^2$
in1
$\underline{\mathtt{out1}}$
mod4
B
$\underline{\mathtt{in1}}$
out1
D

The next and final task is to define the buffer. $\mathtt{Buff}_4^2$ consists of four **cells** that are connected in parallel:

$$\mathtt{Buff}_4^2 = (\mathtt{Cell}_1 \mid \mathtt{Cell}_2 \mid \mathtt{Cell}_3 \mid \mathtt{Cell}_4) \setminus \{z_1, z_2, z_3, w_1, w_2, w_3\}$$



⦿ GIVEN THIS DEFINITION and the flow diagram for the buffer, write down the definitions of $\mathtt{Cell}_1$, $\mathtt{Cell}_2$, $\mathtt{Cell}_3$, and $\mathtt{Cell}_4$.

Homework

⦿ AMEND THE SPECIFICATION of `System` so that `C` and `D` are now mutually exclusive on the actions $\underline{\mathtt{mod3}}$ and $\underline{\mathtt{mod4}}$.

Next Week

Next week's seminar will be mostly concerned with solving problems and questions from past papers, and covering the new lecture material (syntax and semantics of CCS and using proof rules).