CO3007

COMMUNICATION AND CONCURRENCY

Irek Ulidowski

E-mail: I.Ulidowski@mcs.le.ac.uk
(or simply iu3@mcs.le.ac.uk)

Extension Number: 3801

Office: G12, Mathematics and Computer Science Building

2006/2007, Semester 1

Research interests

1. Process Algebras, for example CCS and CSP, with additional features such as
   - new process operators
   - time, random delays, probabilities, value passing
2. Structural Operational Semantics (SOS)
3. Reversible Process Algebras
4. Models of Reversible Computation
5. Proof Systems and Term Rewriting

Introduction

- Organisation
- Reading List
- Assessment
- Concurrent Systems
- Aims and Objectives

Organisation

- 30 lectures in Weeks 2 - 11:
  - Monday 13:30 BEN LT8
  - Monday 17:30 GP LTA
  - Tuesday 16:30 BEN LT3
- 10 surgeries (tutorials) in Weeks 2 - 11, Thursday 14:30, FJ L67
- a couple of labs in Weeks 5 - 6, Thursday 16:30, CW3 Cityside

Note:

- Students should attend the lectures.
- Surgeries are devoted to solving problems from the worksheets and the lecture notes, and to feedback.
- Surgeries are compulsory: attendance lists.
- Students should complete all lab session exercises.

Reading List

- **Main Textbooks**:
  - Lecture Notes by Irek Ulidowski
  - R. Milner, Communication and Concurrency, Prentice Hall, 1989.
- Other Books:
  - C.A.R. Hoare, Communicating Sequential processes, Prentice Hall, 1985.
  - C. Fencott, Formal Methods for Concurrency, Thomson Computer Press, 1996.
  - A.W. Roscoe, The Theory and Practice of Concurrency, Prentice Hall, 1997.

Course Assessment

Continuous Assessment: 30% of the total credit

- There will be several assessed worksheets including two tests.
- there is no late submissions of work.
- There is no provision for re-siting the coursework.
- Avoid plagiarism.

Written Examination: 70% of the total credit

- 3 hour examination in January

More details can be found in the Study Guide.

Past exam papers are included in the lecture notes. Also available on the Web pages for CO3007.

Questions from the earlier years are also included in the notes after the relevant chapters. Some will also appear as coursework questions.

Aims and Objectives

We will study a formalism called A Calculus of Communicating Systems, or CCS for short, in order to

- understand the essential aspects of concurrent systems,
- learn how to formally **specify** such systems,
- learn how to express such systems' possible **designs or implementations**,
- learn how to **verify** the correctness of a design or an implementation with respect to a specification,
- apply CCS to realistic (small) concurrent systems,
- learn to use CWB-NC.

Concurrent Systems

A concurrent system is a dynamic system that is composed of several parts:

- each part acts concurrently with, and independently of, other parts—**concurrency**,
- parts interact (or communicate, react) with each other to synchronize their behaviour and exchange information—**communication**

Communication and concurrency are complementary notions, which

- are essential in understanding concurrent systems
- distinguish concurrent systems from sequential ones

Concurrent systems = communicating systems = concurrent communicating systems = reactive systems = interacting systems

Concurrent systems are complex in nature. Safety-critical systems, telecommunication systems and industrial plant systems are examples of concurrent systems. Thus their correctness, reliability and safety are crucial.

Examples

1. A university as a system of departments; and a department is as a system of people.
2. A chocolate vending machine system consisting of the machine and its customers.
3. A banking system consists of the computers in the central office and in the branch offices.
4. An UNIX operating system with processes managing devices and execution of user tasks.
5. A multiprocessor system with processors executing processes.
6. A concurrent program with n processes,

   Cobegin $P_1; \ldots; P_n$ **Coend**

7. Communicating networks, digital control systems

And many others ....

What is CCS?

- A process language for representing specifications and possible designs or implementations of concurrent systems at relatively high level of abstraction.
- There are several theories that support verification and testing
  - Structured Operational Semantics with the notions of bisimulations,
  - Axiomatic semantics with sets of laws (axioms) and rules of equational reasoning.
- The software tool CWB-NC to support the analysis and reasoning about concurrent systems.

Chapter 1
Modelling Concurrent Systems

Goals for Chapter 1

- The notion of agents.
- The handshake communication between agents.
- The five basic ways of constructing expressions for agents (five operators).
- Equations between agent expressions.
- Examples of simple systems.
- Simple examples of specification, implementation and verification process.

1. Agents.

We treat a concurrent system as a network of agents.

- Each agent has its own identity which persists throughout time.
- An agent is used to identify where an event occurs.
- An agent may be decomposed into a network of several sub-agents acting concurrently and communicating with each other.
- A system itself is an agent.

Example

Leicester University is a network of departments, and a department is a network of people:

- A department is an agent with a name , e.g. Maths & Computer Science.
- Departments act concurrently with and independently of each other; departments also communicate or interacts with each other.
- Activities of departments: teaching, research, development, publication
- A department is a network of people.
- Each person is an atomic agent.
- Leicester University itself is an agent.

Note

- An agent may mean any system whose behaviour consists of discrete actions;
- An agent which for one purpose we take to be atomic may, for other purposes, be decomposed into sub-agents acting concurrently and interacting.

2. Communication

Each action of an agent is either

- an interaction (communication) with other agents,
- or it is an independent event that may occur concurrently with actions of other agents.

Independent actions of an agent are attempts of communication with the external environment, or among the components of that agent.

The behaviour of a system is its entire capability to communicate (perform actions).

The behaviour of a system is exactly what is observable.

To observe a system is exactly to communicate with it.

Communication Media

How does the transmission of information from one agent to another take place?

Agents: *Sender*, *Receiver*

MEDIUM: a place where items reside while in transit.

Items are transmitted in the form of messages:

- each message corresponds to a single act of sending; a message is therefore sent exactly once;
- each act of receiving involves a distinct message; so each message is received at most once.

There are several different types of communication media:

1. ETHER

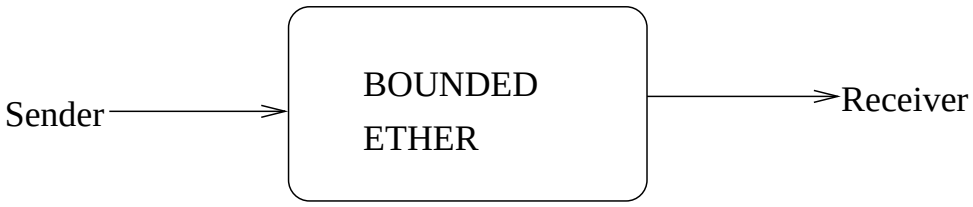
   The ETHER protocol is:

   (a) The *Sender* may always send a message.

   (b) The *Receiver* may always receive a message, provided the medium is not empty.

   (c) The orders in which messages are sent and received may be different.

2. BOUNDED ETHER



The BOUNDED ETHER protocol:

(a) The *Sender* may always send a message, provided the medium is not full.

(b) As for ETHER.

(c) As for ETHER.

3. BUFFER

The BUFFER protocol:

(a) As for ETHER

(b) As for ETHER

(c) Receiving oder = Sending order

4. BOUNDED BUFFER

The BUFFER protocol:

(a) As for BOUNDED ETHER

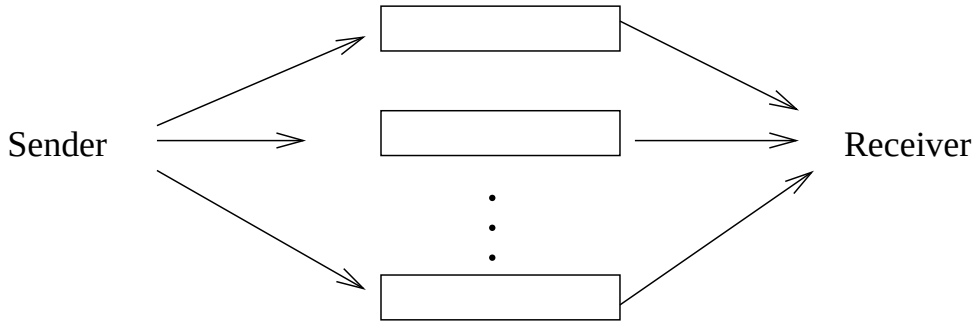
(b) As for ETHER

(c) As for BUFFER

Shared Memory

Is it always possible for the items of information in all media to be considered as messages?

Consider a shared memory as a medium



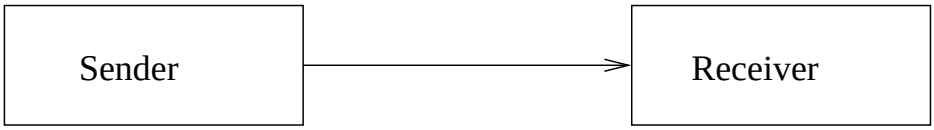
The SHARED MEMORY protocol:

1. The *Sender* may always write an item in to a register.
2. The *Receiver* may always read an item from a register.
3. Writing and reading may occur in any order.

The assumption that information should be transmitted in the form of messages is not essential.

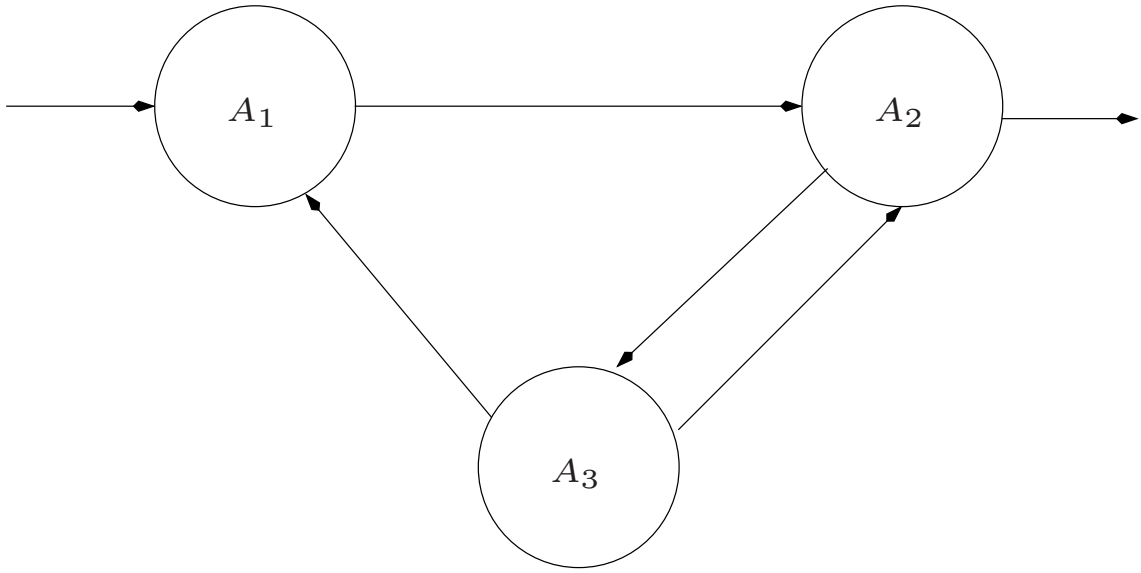
Handshake Communication

These are the basic points common to the above described forms of transmission:



- Each **arrow** is a vehicle for a single action, indivisible in time—atomic action.
- Each single action consists of the passage of 'an item' between two entities, a sender and a receiver.
- "Performers" and "passive entities", or "Senders" and "Receivers" **all** participate in these single acts of communication.
- Such acts of communication are experienced simultaneously by both participants.
- These acts of communication are called handshakes.

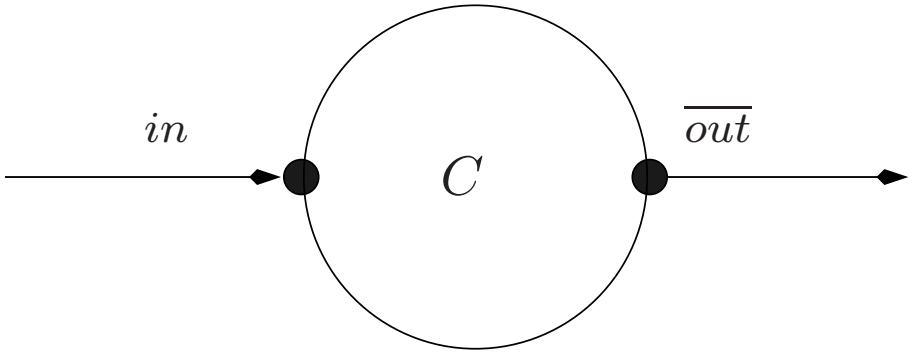
Graphical Representation



- Linked arrows indicate handshakes between agents.
- Non-linked arrows indicate handshakes with the environment.
- We shall treat non-linked and linked arrows quite differently when we consider the behaviour of a system.

3. Agent Expressions

A Cell



C is the agent's name; in and $\overline{out}$ are port labels

Behaviour

- When empty, C may accept an item at port in;
- When holding a value, C may deliver it at port $\overline{out}$.

Agent Expressions

$$C \stackrel{def}{=} in(x).C'(x)$$

$$C'(x) \stackrel{def}{=} \overline{out}(x).C$$

Remarks

- Agent names, e.g. C and $C'(x)$, can take parameters.
- The Prefix '.' is used for sequencing of events.
- The Prefix $in(x)$ stands for a handshake at port in.
- The agent $in(x).C'(x)$ performs the handshake $in(x)$ and then proceeds according to the definition of $C'(x)$.
- $\overline{out}(x).C$ outputs the value of x at port $\overline{out}$ and then proceeds according to the definition of C.

Scope of a Variable

- A variable is either bound by an input Prefix: $in(x).C'(x)$
- or by the left-hand side of a defining equation: $C'(x) \stackrel{def}{=} \overline{out}(x).C$.

Remark.

1. $C'(x) \stackrel{def}{=} \overline{out}(x).C$ can also be written as
$$C'(y) \stackrel{def}{=} \overline{out}(y).C$$
2. Output Prefix like $\overline{out}(x)$ does not make the variable x bound in $\overline{out}(x).C$.
3. Agent expression $C'(x)$ is auxiliary:
$$C \stackrel{def}{=} in(x).\overline{out}(x).C$$

Exercise

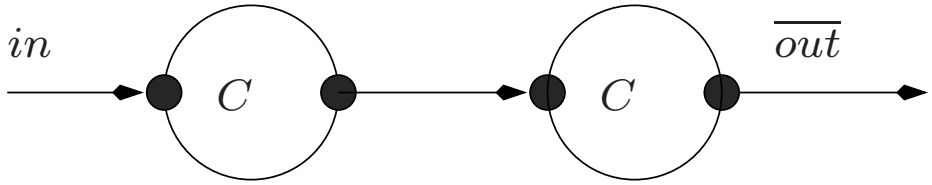
Consider the equation

$$A \stackrel{def}{=} in(x).in(y).\overline{out}(x).\overline{out}(y).A$$

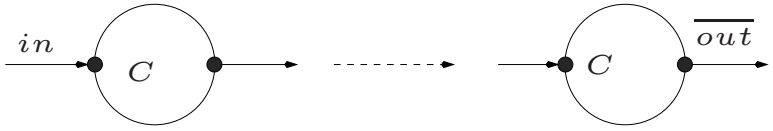
1. How does the behaviour of A differ from that of C?
2. Write it as a pair, or as four equations.

Bounded Buffer

Consider linking two or more copies of C



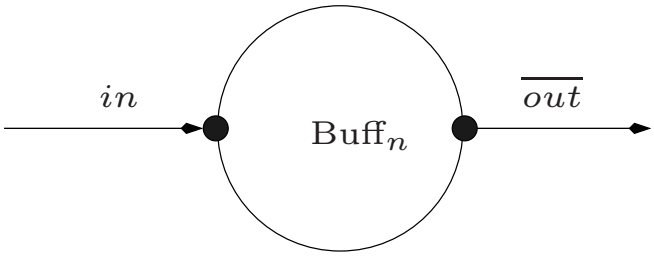
$C^{(2)} \stackrel{def}{=} C \frown C$



$C^{(n)} \stackrel{def}{=} C \frown C \frown \ldots \frown C$

$C^{(n)}$ behaves as a buffer of capacity n.

Buffer of capacity n



$Buff_n <> \stackrel{def}{=} in(x).Buff_n < x >$

$Buff_n < v_1, \ldots, v_n > \stackrel{def}{=} \overline{out}(v_n).Buff_n < v_1, \ldots, v_{n-1} >$

$Buff_n < v_1, \ldots, v_k > \stackrel{def}{=}$

$in(x).Buff_n < x, v_1, ..., v_k > + \overline{out}(v_k).Buff_n < v_1, \ldots, v_{k-1} >$

for $0 < k < n$

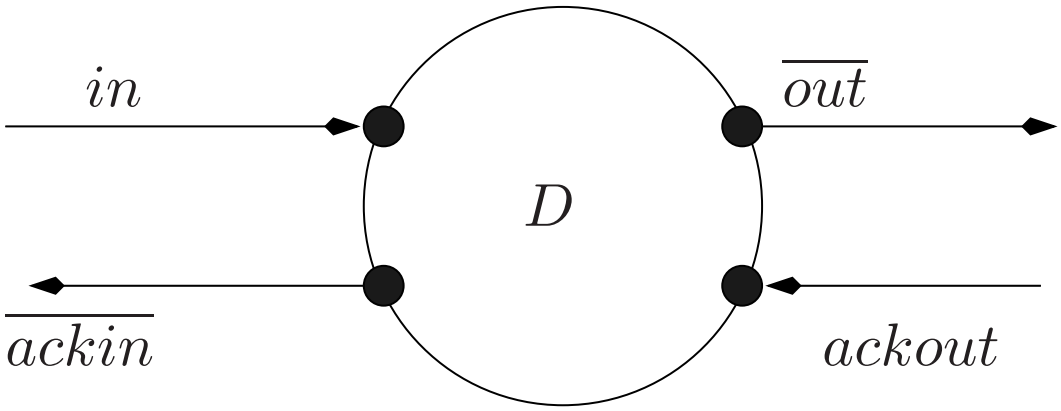
Some Remarks

- We need the agent combinator (or operator) + to build agent expressions.
- $P + Q$ behaves either like agent P or like agent Q; as soon as one of them performs its first action the other is discarded.
- The binary operator + combines two agent expressions as alternatives.
- The Prefix operator '.' is used to produce a sequence of events.
- We can express an agent at different levels of abstraction
  - $Buff_n$ serves as a specification
  - $C^{(n)}$ serves as a design or an implementation.
- We need to have a theory to enable us to prove

$$Buff_n = C^{(n)}$$

More Examples

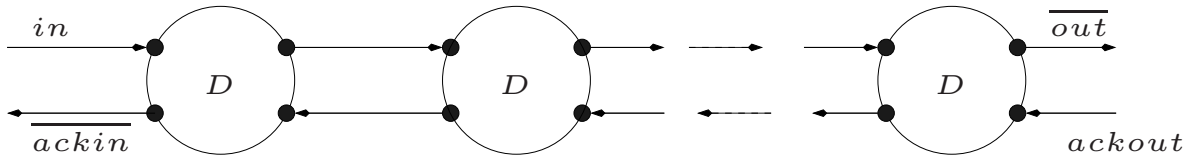
Consider the cell D which is like C, except that it acknowledges the receipt of each message or value.



Assume that D sends acknowledgements after the value arrives at its final destination.

$$D \stackrel{def}{=} in(x).\overline{out}(x).ackout.\overline{ackin}.D$$

Link $n > 1$ copies of D



$D^{(n)} \stackrel{def}{=} D \frown \ldots \frown D$. What is the behaviour of $D^{(n)}$?

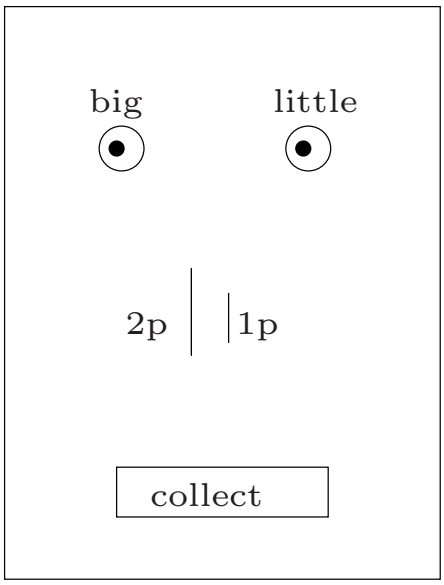
Assume D acknowledges its input as soon as it is received:

$$D' \stackrel{def}{=} in(x).\overline{ackin}.\overline{out}(x).ackout.D'$$

$D'^{(n)}$ equals $Buff'_n <>$, where $Buff'_n$ differs only slightly from $Buff_n$:

it requires the addition of acknowledgement actions.

A Vending Machine



1. To buy a big chocolate, put a 2p coin, press the button *big*, and collect your chocolate.
2. To buy a little chocolate, put a 1p coin, press the button *little*, and collect.

We model collecting big and little chocolates with actions bigchoc and littlechoc respectively. The specification is

$$V \stackrel{def}{=} 2p.big.\overline{bigchoc}.V + 1p.little.\overline{littlechoc}.V$$

Imagine a customer $C \stackrel{def}{=} \overline{1p}.\overline{little}.littlechoc.\overline{happy}.\mathbf{0}$.

- What will happen when C interacts with V?
- What will happen when the customer pays $1p$ but attempts to presses *big*?
- What will happen when the customer tries to pay $5p$?

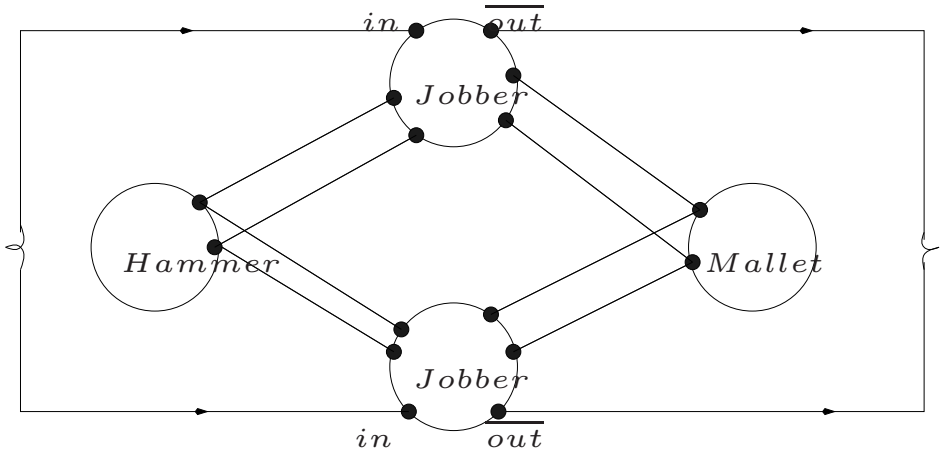
A Jobshop

- Two $Jobbers$ are sharing two tools – $Hammer$ and $Mallet$ – to make objects.
- Each object is made by driving a peg into a block.
- We call a pair consisting of a peg and a block a job.
- Jobs arrive sequentially on a conveyor belt.
- Completed objects depart on a conveyor belt.

What are agents and what are data?

- $Jobbers$, $Hammer$ and $Mallet$ are the agents.
- jobs and objects are the data (values) which enter and leave the system.

A Diagram of the model

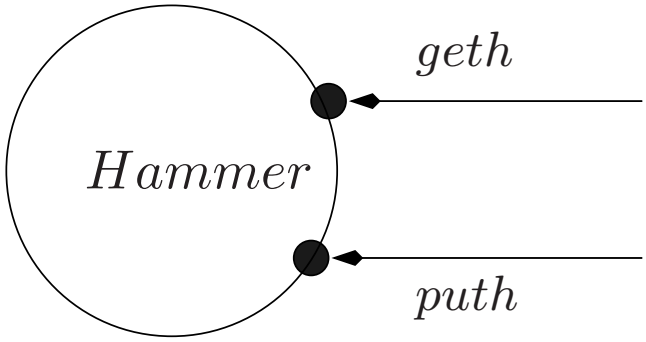


Remarks about the diagram

- A port may be linked to more than one other port.
- This represents the possibility that the two jobbers may compete for the use of a tool.
- This is a potential source of non-determinism.
- in and $\overline{out}$ are the external ports.
- The unlabelled ports (by which the jobbers acquire or release the tools) are internal ones.

Hammer

We regard Hammer as a resource which may be acquired or released by one user only.



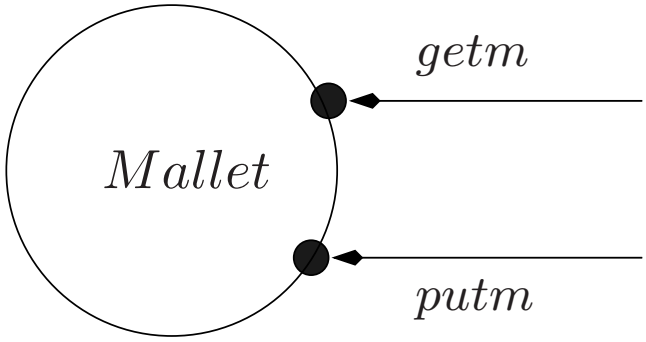
$$Hammer \stackrel{def}{=} geth.Busyhammer$$

$$Busyhammer \stackrel{def}{=} puth.Hammer$$

It is equivalent to

$$Hammer \stackrel{def}{=} geth.puth.Hammer$$

Mallet



$$Mallet \stackrel{def}{=} getm.Busymallet$$

$$Busymallet \stackrel{def}{=} putm.Mallet$$

$$Mallet \stackrel{def}{=} getm.putm.Mallet$$

Notice, that the behaviour of $Mallet$ is exactly the same as that of $Hammer$ if we replace port names $geth$ and $puth$ with $getm$ and $putm$ respectively.

Sort

- A sort is just a set of port labels.
- An agent P has sort L, if all the (observable) actions of P have the labels in L.
- We write $P : L$ if P has sort L.
- If $P : L$ and $L \subseteq L'$ then $P : L'$.
- We normally find the smallest L such that

$$P : L$$

For example

$$\begin{aligned} Hammer &: \{geth, puth\} \\ Mallet &: \{getm, putm\} \\ Jobshop &: \{in, \overline{out}\} \\ V &: \{\quad\} \\ C &: \{\quad\} \end{aligned}$$

Jobber

Assume that

- There are easy jobs and hard jobs
  - easy jobs will be done just with hands;
  - hard jobs will be done with hammer;
  - other jobs will be done with either hammer or mallet.
- A function *done* from jobs to objects; $done(job)$ is the object made of job.

The *states* of $Jobber$:

$Jobber$	Initial state—waiting to receive a job
$Start(job)$	job received
$Usetool(job)$	tool used
$UseH(job)$	hammer used
$UseM(job)$	mallet used
$Finish(job)$	job completed

$Jobber$ has the sort

$$\{in, \overline{out}, \overline{geth}, \overline{puth}, \overline{getm}, \overline{putm}\}$$

Description of $Jobber$

$$
\begin{aligned}
Jobber &\stackrel{def}{=} in(job).Start(job) \\
Start(job) &\stackrel{def}{=} \textbf{if } easy(job) \textbf{ then } Finish(job) \\
&\quad \textbf{else if } hard(job) \textbf{ then } UseH(job) \\
&\quad\quad \textbf{else } Usetool(job) \\
Usetool(job) &\stackrel{def}{=} UseH(job) + UseM(job) \\
UseH(job) &\stackrel{def}{=} \overline{geth}.\overline{puth}.Finish(job) \\
UseM(job) &\stackrel{def}{=} \overline{getm}.\overline{putm}.Finish(job) \\
Finish(job) &\stackrel{def}{=} \overline{out}(done(job)).Jobber
\end{aligned}
$$

Composing Agents

The subsystem $Jobber$-$Hammer$ is represented by

$$Jobber|Hammer$$

- $|$ is called the parallel composition of CCS.
- $P|Q$ is a system in which P and Q may proceed independently, but may also interact through complementary ports.
- Two ports are complementary if one is labelled by a label l and the other by $\overline{l}$, e.g. $geth$ and $\overline{geth}$, $puth$ and $\overline{puth}$.
- Interaction on complementary ports produces an internal, silent action τ , the action which cannot be communicated upon.
- If $P : L$ and $Q : N$, then $P|Q : L \cup N$.
- Thus, no ports are internalised by composition: e.g. $a.\mathbf{0}|\overline{a}.\mathbf{0}$ can perform a, $\overline{a}$ and τ.

Adding another $Jobber$ to $Jobber|Hammer$ produces

$$(Jobber|Hammer)|Jobber$$

The CCS parallel operator (combinator) has the following properties.

- Composition is commutative

$$Jobber|Hammer = Hammer|Jobber$$

- Composition is associative

$$(Hammer|Jobber)|Jobber = Hammer|(Jobber|Jobber)$$

so we can omit brackets, and write

$$Hammer|Jobber|Jobber$$

- **0** is the identity (unit) for |, e.g.

$$Jobber|\mathbf{0} = Jobber$$

- Unlike $Jobber + Jobber = Jobber$ we **do not** have

$$Jobber|Jobber = Jobber$$

Hiding Ports—Restriction of Actions

Imagine that additional $Jobbers$ want to share $Hammer$.

To prevent this, we have to hide or restrict the ports of $Hammer$.

In general, a port can be hidden so that no further agents can be connected to it. Hiding a port prevents the agent from communicating with its environment through that port, e.g.

$$(Jobber|Hammer)\backslash\{geth, puth\}|Jobber$$

$$(Jobber|Jobber|Hammer)\backslash\{geth, puth\}$$

The Restriction Combinator

To hide a set L of ports of an agent P, we use a combinator $\backslash L$, called restriction, to P:

$$P\backslash L$$

- E.G. $(Jobber|Jobber|Hammer)\backslash\{geth, puth\}$
- $\backslash L$ hides both the ports L and their complements.
- Hiding ports must be done with great care, incorrectly hiding a port may cause deadlock.

The System $Jobshop$

We compose $Mallet$ with $Jobber|Jobber|Hammer$, hide appropriate ports and obtain

$$Jobshop \stackrel{def}{=} (Jobber|Jobber|Hammer|Mallet)\backslash L$$

where $L = \{geth, puth, getm, putm\}$

Remarks

- There are many ways of composing a system from given subsystems.
- There are many ways of decomposing a composite systems into subsystems.
- Some of these different ways will be studied as instances of algebraic laws of equivalence.

Renaming Ports

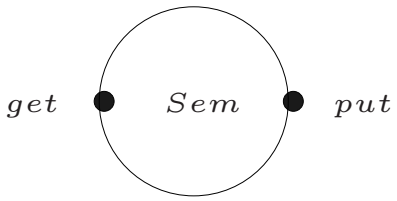
If we correctly relabel the ports of $Hammer$, we will get $Mallet$.

- Extend the "overbar" function to all labels by letting $\overline{\overline{l}} = l$.
- A function f from labels to labels is called a relabelling function if whenever $f(l) = l'$ then $f(\overline{l}) = \overline{l'}$.
- f is written as $l'_1/l_1, \ldots, l'_n/l_n$ if $f(l_i) = l'_i$ and $f(\overline{l_i}) = \overline{l'_i}$ for $i = 1, \ldots, n$, and otherwise $f(l) = l$.
- Given a relabelling function f and an agent P, we use the unary postfix relabelling combinator $\cdot[f]$ in

  $$P[f]$$

  for the agent obtained from P by replacing (renaming) each port l of P with $f(l)$.

Example A semaphore



$$Sem \stackrel{def}{=} get.put.Sem$$

We have the following.

$$Hammer = Sem[geth/get, puth/put]$$

$$Mallet = Sem[getm/get, putm/put]$$

Summary of Combinators

Combinator	**Examples**
Deadlocked Agent	$\mathbf{0}$
Prefix	$in(x).P$, $\overline{get}.Q$
Summation	$P + Q$
Parallel Composition	$P \mid Q$
Restriction	$P \backslash \{l_1, \ldots, l_n\}$
Relabelling	$P[l'_1/l_1, \ldots, l'_n/l_n]$

4. Equality of Agents: a Taster

- $C^{(n)} = Buff_n <>$
- $Hammer = Sem[geth/get, puth/put]$
- Various equivalent expressions of $Jobshop$ in terms of its components.

Specifications of Jobshop

Let S be defined as

$$\begin{array}{lcl} S & \stackrel{def}{=} & in(job).S' \\ S' & \stackrel{def}{=} & in(job).S'' \ + \ \overline{out}(done(job)).S \\ S'' & \stackrel{def}{=} & \overline{out}(done(job)).S' \end{array}$$

We may argue that S is a specification for $Jobshop$:

$$S = Jobshop$$

What other agent you know is somewhat similar to S?

Moreover, let

$$Strongjobber \stackrel{def}{=} in(job).\overline{out}(done(job)).Strongjobber$$

We claim that

$$Jobshop = Strongjobber|Strongjobber$$

Thus, $Strongjobber|Strongjobber$ is another specification of the system which we have described by $Jobshop$.

5. Summary

1. Main ideas:
   - Treat systems as networks of agents.
   - An agent may be treated atomically, or as a composition (a network) of sub-agents, depending on the chosen level of abstraction.
   - An action of an agent is either
     - internal, independent of other agents, so it cannot be communicated upon or controlled, or
     - external and visible, so it can be communicated upon by other agents.
   - A silent action of an agent is nothing more than a communication between sub-agents of that agent.
   - A communication is a handshake between two agents, i.e. an action carried out simultaneously by the two agents.
   - Two agents are equivalent if their external behaviour is the same.

2. Use of Mathematics (algebra, formal language theory):
    - An agent is represented as an expression in terms of symbols and combinators.
    - Symbols for agent constants and agent variables, symbols for actions (port labels), symbols for values (data).
    - Five combinators (operators): Prefix, Summation, Parallel Composition, Restriction, and Relabelling.
3. Two kinds of communication:
    - Synchronisation: Vending machine, $Hammer$, $Mallet$, Sem, etc..
    - Value-passing communication: C, $C^{(n)}$, $Buff_n$, $Jobshop$.