LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

Introduction to a Calculus of Communicating Systems

by

David Walker

Expository Report

Introduction to a Calculus of Communicating Systems

ECS-LFCS-87-22

(also published as CSR-227-87)

March 1987
Revised June 1988

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Introduction to a Calculus of Communicating Systems

David Walker

Laboratory for Foundations of Computer Science,
Department of Computer Science,
University of Edinburgh

February 1987

Preface

In December 1986 the Laboratory for Foundations of Computer Science in Edinburgh ran a one-week course for industrial and academic computer scientists entitled "Foundations of Concurrent Programming". This course was designed to provide an introduction to Robin Milner's Calculus of Communicating Systems (CCS). The bulk of the material presented was drawn from a set of lecture notes written by Milner in the autumn of 1986, these being a development and refinement of their author's monograph "A Calculus of Communicating Systems" published by Springer-Verlag in 1980. Milner's notes provide both a description of the calculus and a full account of its mathematical foundations. Their comprehensive nature, however, does not help to make them an ideal introduction to the calculus for a reader not primarily interested in the underlying mathematics. These notes are intended to provide such an introduction.

After a short introductory section setting out the principles on which the calculus is based, there follows a description of the language of CCS together with an explanation of how the language may be used to describe systems of concurrent or communicating processes. The operational semantics, or meaning, of the language is then described. Next, the question of when two expressions of the language of CCS are to be considered as describing the same behaviour is addressed. To this important question there exists at present no universally accepted answer. In these notes, as seems appropriate given their nature, we do not consider all of the notions of "equivalence" which have been proposed, but rather concentrate on one of the most prominent, Milner's *observational equivalence*. However, it should be reiterated that there do exist other notions of "equivalence" applicable to the language of CCS. Laws for analysing the behaviour of systems described by expressions of the language are presented, together with examples illustrating their use. Then a manageable technique for tackling the problem of determining whether or not two expressions of the language are observationally equivalent is described and illustrated. Finally there is a short discussion of the topic of divergence, or "livelock", in communicating systems.

At the end of each of the six sections there are some exercises on the material covered. Solutions to these appear at the end of the notes. I should be grateful to receive any criticism of either the text or the exercises.

The idea of writing a set of notes such as these developed in a conversation during the December course between Kevin Mitchell, David Park and myself. I am indebted to Joachim Parrow for introducing me to *Macdraw* and to Katharine Grevling for assistance in preparing these notes. Finally I should like to express my gratitude to Robin Milner, who not only permitted me to plagiarise the opening paragraphs of his notes (upon which I could not improve), with the effect that the Introduction here is little more than a precis of them, but also offered encouragement and valuable suggestions during the writing of these notes. I have been supported by a grant from the Venture Research Unit of the British Petroleum Company.

Contents

Introduction

We begin by analysing the transmission of information from one agent to another. Our aim is to isolate certain fundamental ideas concerning the nature of communication which will form a starting point in formulating a theory of communicating systems.

We may start with the natural idea that three entities are involved in any transmission of information: a *Sender*, a *Receiver* and a *Medium* of transmission through which items of information pass from *Sender* to *Receiver*. Furthermore, it is natural to consider each such message, containing one item of information, as corresponding to a single act of sending: each message is sent exactly once and may be received at most once.

Media may be classified according to the discipline under which sending and receiving take place. For example an (idealized) unbounded *Ether* may be characterised as a medium whose discipline of transmission is as follows:
(a) at any time *Sender* may send a message to *Ether*;
(b) at any time at which *Ether* is not empty, *Receiver* may receive a message from *Ether*;
(c) the order in which messages are received by *Receiver* is not related to that in which they are sent by *Sender*.
As a second example we may describe the discipline of transmission of a bounded *Buffer* as follows:
(a) at any time at which *Buffer* is not full, *Sender* may send a message to *Buffer*;
(b) at any time at which *Buffer* is not empty, *Receiver* may receive a message from *Buffer*;
(c) the order in which messages are received by *Receiver* is the same as that in which they are sent by *Sender*.

We could proceed to consider further possible disciplines of transmission. However it is clear from the above examples that in modelling communication it will be necessary to take into account properties of the media involved. To reinforce this view consider the medium of a *Shared Memory* consisting of a collection of registers. In this case the appropriate discipline may be described as follows:
(a) at any time *Sender* may write an item to a register;
(b) at any time *Receiver* may read an item from a register;
(c) the reading and writing of items may occur in any order.
Notice that we have used the term "item" rather than "message" in the above description, for since the contents of a register may be read on more than one occasion, we can not accurately speak of a message being transmitted from *Sender* to *Receiver* via *Shared Memory*.

One medium of transmission may sometimes be implemented using another: a bounded buffer is often implemented in terms of a shared memory, for example. Consideration of this fact, in conjunction with the above discussion, leads us to deny any latent distinction which may have existed between "active" agents, such as *Sender* and *Receiver*, and "passive" media, such as *Buffer* and *Shared Memory*, at least to the extent that we regard the passing of an item of information from *Sender* to *Buffer* or from *Shared Memory* to *Receiver* as being an event in which each entity concerned is an active participant. Furthermore, we regard such events as atomic actions, indivisible in time.

Thus we are led to take as basic just one kind of entity, the *agent*, and to think of pairs of agents as engaging in indivisible acts of communication, experienced by both participants. These are the fundamental ideas from which we develop the calculus of communicating systems.

1. The language of CCS—agents

In this section we introduce the language of the calculus and illustrate how the expressions of the language, which we call *agents*, may be used to model systems of communicating processes. We think of a process as being endowed with *ports* through which it may communicate with other processes. With each port is associated a *label*. To facilitate the modelling of communication between processes, we assume that ports occur in complementary pairs. It is often convenient to think of one port in a complementary pair as an *input port*, that is as a port at which a signal or a value may be input, and the other as an *output port*, from which a signal or a value may be output. Indeed we shall often speak of *input labels* and *output labels* rather than labels of input ports and output ports, and we shall distinguish output labels from input labels by marking the former with an overbar. Thus *in* might be an input label and $\overline{in}$ the complementary ouptut label. We shall adopt the convention that labels always begin with a lower case letter. Thus we assume that we have a set $\mathcal{L}$ of labels composed of a set $\mathcal{I}$ of input labels and a set $\mathcal{O}$ of ouput labels, and that there is a one-to-one correspondence between the sets $\mathcal{I}$ and $\mathcal{O}$. We use l, k to denote arbitrary labels, and if l denotes a label then $\overline{l}$ denotes the complementary label.

We now introduce, via examples, the *operators*, or *constructors*, of the language and explain their intended meanings. Each of the operators describes a means of forming a new agent from existing agents and corresponds to some means of combining processes to form a more complex process. To get the constructions started we introduce a simple agent *Nil* which is intended to represent a process which can not perform any actions. Thus *Nil* describes a process which can do nothing. From *Nil* we can construct agents describing processes more interesting than that described by *Nil* itself. For example we may describe the behaviour of a match which may be struck once and thereafter engage in no further activity. To achieve this we introduce a label *strike* and form the agent

$$strike.Nil.$$

The intended meaning here is that the process described by the agent *strike.Nil* is capable initially of performing the action *strike* and thereafter behaving as the process described by *Nil*. Of course we have described only a small part of the behaviour of a real match, but we have done so accurately and succinctly. This example has served merely to introduce the first constructor of the calculus, which we call *action-prefixing*: given any label l and agent P we may form the agent

$$l.P$$

which represents a process which may initially perform the event described by l to become the process described by P. Notice that in such a composite agent we have always a label on the left of the '.' and an agent on the right. We will adopt the convention that agents always begin with an upper case letter, and we use P, Q and R to denote arbitrary agents.

We may introduce agents by means of definitions. For example we might define an agent *Match* as follows:

$$Match \stackrel{def}{=} strike.Nil.$$

Here we use $\stackrel{def}{=}$ to mean that the agent, *Match*, on the left hand side is defined to be the expression (which is itself an agent) on the right hand side. We may use recursion in such definitions. For example to describe the behaviour of a clock that is capable of ticking any number of times we could introduce a label *tick* and define

$$Clock \stackrel{def}{=} tick.Clock.$$

By repeatedly substituting the defining expression *tick.Clock* of *Clock* in the right hand side, we can see that *Clock* does indeed describe the behaviour of the clock described above.

As a second example of a recursive definition, consider the agent *Addone* with input label *in* and output label $\overline{out}$ defined as follows:

$$Addone \stackrel{def}{=} in(x).\overline{out}(x+1).Addone.$$

Notice that here we have parameterised the label *in* by a variable, x, and the label $\overline{out}$ by an expression, $x+1$. The behaviour of *Addone* is repeatedly to input a value (an integer) at the port *in* and then to ouput at the port $\overline{out}$ that value increased by 1.

In giving recursive definitions of agents we need not be restricted to a single equation: we may define agents by mutually recursive definitions. For example we can reformulate the definition of *Addone* by giving the following mutually recursive definitions of the agents *Addone* and *Holding*(x):

$$\begin{aligned} Addone &\stackrel{def}{=} in(x).Holding(x) \\ Holding(x) &\stackrel{def}{=} \overline{out}(x+1).Addone. \end{aligned}$$

Notice that the agent *Holding(x)* is parameterised by the variable x. *Holding(x)* represents the state of the process described by *Addone* when an value has been input and bound to the variable x: the next action will be the output of that value increased by 1. We will see later mutually recursive definitions in which the number of equations is greater than two.

So far we can only describe processes whose pattern of behaviour is rigidly fixed. For example we can describe the behaviour of a change-giving machine with one input slot and one output slot which is capable of accepting a 50 pence coin and then delivering in turn a 20 pence coin, a second 20 pence coin, and finally a 10 pence coin, repeatedly. To do so we introduce labels *fifty* and $\overline{out}$ and define:

$$Change \stackrel{def}{=} fifty.\overline{out}(20).\overline{out}(20).\overline{out}(10).Change.$$

Thus *fifty* is the label of a port at which 50 pence coins may be accepted, and $\overline{out}$ of a port at which 20 pence and 10 pence coins may be delivered. Suppose, however, that we wish to describe a change-giving machine with two input slots and one output slot which, in addition to the above capability, is able to give two fifty pence coins in change for a one pound coin. Thus the machine is prepared initially to accept either a 50 pence coin

or a one pound coin and having received one of these coins will dispense the appropriate change before returning to its initial state. To describe this behaviour we introduce a new constructor + and, introducing a new label *hundred*, define the agent *Newchange* as follows:

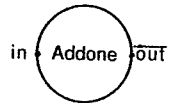
$$\begin{aligned} \mathit{Newchange} \stackrel{def}{=} \; & \mathit{fifty}.\overline{\mathit{out}}(20).\overline{\mathit{out}}(20).\overline{\mathit{out}}(10).\mathit{Newchange} \\ & + \mathit{hundred}.\overline{\mathit{out}}(50).\overline{\mathit{out}}(50).\mathit{Newchange} \end{aligned}$$

We intend that + conjoins the capabilities of the agents which appear either side of it. Thus *Newchange* is prepared initially to engage in either of the actions *fifty* and *hundred* and depending on which actually occurs will dispense the appropriate change and return to its original state. In general, given two agents P and Q we may form the composite agent $P+Q$ which represents the process whose capabilities are the conjunction of those of P and Q. We call such an agent a *sum*, and the operation of forming a sum, *summation*. Thus + is the *summation operator*.

In saying that the agent $P+Q$ represents a process whose capabilities are the conjunction of those of the processes described by P and Q, we have made a statement which may be interpreted in more than one way. In later sections the precise nature of the operator + will be discussed further. In particular, we shall give a formal operational semantics, or meaning, to agents which will enable us to speak with definiteness about the "behaviour" of an agent.

We now consider how we may model communication between processes within the calculus. At the beginning of this section we stated that we regard processes as communicating through ports with complementary labels. It is sometimes convenient to picture an agent in such a way as to exhibit the labels through which it may communicate. We may do so by drawing a *flow graph* of the agent. This consists of a node marked with the name of the agent on the border of which are points each marked with the name of a label through which it may communicate.

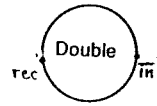
Addone, for example, could be pictured



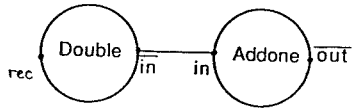
while the agent *Double* defined by

$$Double \stackrel{def}{=} rec(x).\overline{in}(2x).Double$$

might be represented as



To describe the parallel composition of the agents *Double* and *Addone*, which describes a process with the possibility of communication through the ports labelled *in* and $\overline{in}$, we introduce a new constuctor |, the *composition operator*. Thus if P and Q are agents then $P|Q$ is an agent which represents the parallel composition of P and Q in such a way that each of P and Q may proceed independently of the other and there is in addition the possibility of communication between them. We represent this possibility pictorially by joining with an arc each pair of complementary labels. Thus, for example, the agent $DA \stackrel{def}{=} Double \mid Addone$ may be pictured



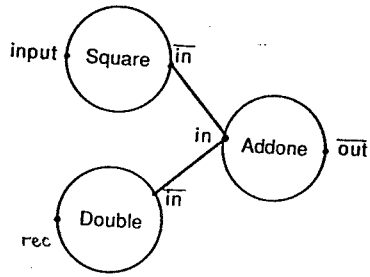
DA may accept a value v at label *rec* and after internal communication between the component agents *Double* and *Addone*, ouput the value $2v+1$ at label $\overline{out}$. In combining two agents with the composition operator we do not require that they *must* communicate with each other, merely that such behaviour is possible. Each agent retains the capacity to communicate with other agents. Thus if we compose DA with the agent *Square* defined by

$$Square \stackrel{def}{=} input(y).\overline{in}(y^2).Square$$

to obtain the agent SDA defined by

$$SDA \stackrel{def}{=} Square \mid DA,$$

then we represent *SDA* as follows:



Thus in *SDA* communication is possible between the agents *Square* and *Addone* and the agents *Double* and *Addone*.

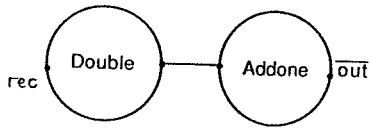
The behaviour of composite agents of the form $P|Q$ will be discussed in greater depth in the next section, but for the moment we proceed to consider the possibility that having composed two agents, so that pairs of complementary ports become "connected", we may wish to "isolate" the connected ports so that no further connection is possible. For example, having formed *DA* by composing *Double* and *Addone*, we may wish to prevent the $\overline{in}$ port of *Square* being connected to the *in* port of *Addone* in forming the composition of *DA* with *Square*. We achieve this by introducing a new constructor, the *restriction operator*, $\backslash$. Given an agent P and a label l,

$$P \setminus l,$$

P restricted on l, is an agent which behaves like P, except that $P \setminus l$ is unable to communicate through either of the ports labelled l and $\overline{l}$, whereas P may have such a capability. Notice that by restricting on the label l, communication through both l and $\overline{l}$ is prohibited. Thus restricting P on l has the same effect as restricting P on $\overline{l}$, for any agent P and label l. We may represent restriction pictorially by deleting the restricted labels. For example, *DAR* defined by

$$DAR \stackrel{def}{=} DA \setminus in$$

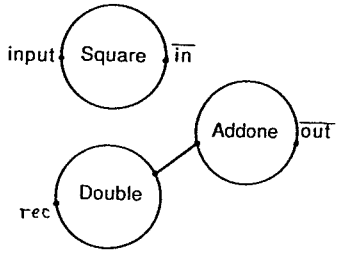
may be pictured



If we compose *Square* with *DAR* to get

$$SDAR \stackrel{def}{=} Square \mid DAR,$$

then in contrast to the picture of *SDA*, *SDAR* may be represented as



It is sometimes convenient to restrict an agent not by a single label but by a set of labels. Thus if L is a set of labels and P an agent then

$$P \setminus L,$$

P restricted on L, is an agent whose behaviour is like that of P except that any capability which P may have of communicating through any label l such that either l or $\overline{l}$ lies in the restricted set L is lost. The introduction of restriction by sets of labels, rather than individual labels, is no more than a convenience. We should expect that the behaviour of an agent $P \setminus L$ would be the same as that of an agent obtained by restricting P on all labels in the (finite) set L, one after the other. To substantiate such an expectation, however, we should have to be precise about what we mean by two agents "having the same behaviour", a matter to which we will devote much consideration in subsequent sections.

As an illustration of the use of restriction by a set of labels, and a means of introducing the last but one constructor of the calculus, consider the following agents.

$$User1 \stackrel{def}{=} in1(x).(\overline{send1}(x).rec1(y).\overline{out1}(y).User1 + \overline{send2}(x).rec2(y).\overline{out1}(y).User1)$$
$$User2 \stackrel{def}{=} in2(x).\overline{send2}(x).rec2(y).\overline{out2}(y).User2$$
$$Test1 \stackrel{def}{=} send1(z).\mathit{if}\ z = 0\ \mathit{then}\ \overline{rec1}(\mathtt{true}).Test1\ \mathit{else}\ \overline{rec1}(\mathtt{false}).Test1$$
$$Test2 \stackrel{def}{=} send2(z).\mathit{if}\ z = 0\ \mathit{then}\ \overline{rec2}(\mathtt{true}).Test2\ \mathit{else}\ \overline{rec2}(\mathtt{false}).Test2.$$

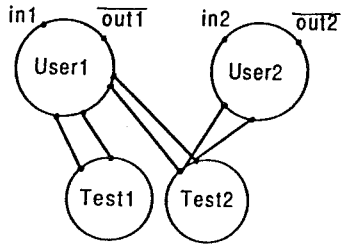
Here *Test1* and *Test2* are agents which repeatedly accept a value and, depending on whether or not that value is 0, output the string "`true`" or the string "`false`". In defining these agents we have introduced a further constructor of the calculus, the *conditional operator*. If P and Q are agents and e is a Boolean expression, then

$$\text{if } e \text{ then } P \text{ else } Q$$

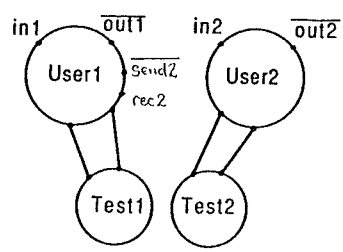
is an agent which behaves either like P or like Q depending on whether e evaluates to `true` or to `false`. To continue with the illustration of restriction by a set of labels, define

$$SystemA \stackrel{def}{=} ((User2 \mid Test2) \mid (User1 \mid Test1)) \setminus \{send1, send2, rec1, rec2\}$$
$$SystemB \stackrel{def}{=} ((User2 \mid Test2) \setminus \{send2, rec2\}) \mid (User1 \mid Test1) \setminus \{send1, rec1\}.$$

We may picture the agents *SystemA* and *SystemB*:



SystemA



SystemB

Notice that the order in which the compositions and restrictions are performed determines the structure of the composite system. In forming *SystemB*, by restricting *User2* | *Test2* by the set $\{send2, rec2\}$, the labels in this set are rendered inaccessible; thus *Test2* may communicate only with *User2*. By contrast, in *SystemA* both *User1* and *User2* may communicate with *Test2*.

If we look at the definitions of *Test1* and *Test2* , we see that, apart from the difference in the names of the labels through which they may communicate, they have very similar behaviour. This observation leads us to introduce the final constructor of the calculus, the *relabelling operator* which enables us to rename the ports of an agent. Consider the agent *Test* defined by

$$Test \stackrel{def}{=} \text{if } z = 0 \text{ then } \overline{rec}(\texttt{true}).Test \text{ else } \overline{rec}(\texttt{false}).Test.$$

Now let f be the function defined on the set $\mathcal{L}$ of labels by $f(send) = send1$, $f(\overline{send}) = \overline{send1}$), $f(rec) = rec1$, $f(\overline{rec}) = (\overline{rec1})$, and $f(l) = l$ for all other labels l. Then we may form the agent $Test[f]$, *Test relabelled by f*, whose behaviour is exactly that of *Test1*. In general, given an agent P and a *relabelling function* g we may form the agent $P[g]$, *P relabelled by g*, whose behaviour is like that of P except that whenever P may communicate through a port labelled l, $P[g]$ may communicate through the port labelled $g(l)$. We require of a relabelling function only that it respect complements, that is that for any label l, $g(\overline{l}) = \overline{g(l)}$. As a final example recall the agent *Addone* defined by

$$Addone \stackrel{def}{=} in(x).\overline{out}(x+1).Addone.$$

Consider the following relabelling functions, *f1* and *f2*:

$$f1(out) = mid,\ f1(\overline{out}) = \overline{mid},\ f1(l) = l \text{ for all other labels } l$$

$$f2(in) = mid,\ f2(\overline{in}) = \overline{mid},\ f2(l) = l \text{ for all other labels } l.$$

Then consider

$$Addone1 \stackrel{def}{=} Addone[f1]$$

$$Addone2 \stackrel{def}{=} Addone[f2].$$

It is often convenient to write such definitions making explicit those labels on which the relabelling functions are not the identity. Thus we may write

$$Addone1 \stackrel{def}{=} Addone[\overline{mid}/\overline{out}]$$

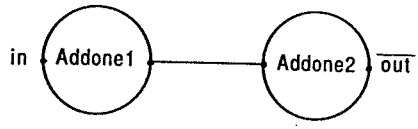
$$Addone2 \stackrel{def}{=} Addone[mid/in].$$

So *Addone1* is *Addone* with $\overline{out}$ relabelled $\overline{mid}$ and *Addone2*, *Addone* with *in* relabelled by *mid*. Notice that we need not record both $\overline{mid}$ / $\overline{out}$ and *mid* / *out*, since the second is a consequence of the first and the fact that *f1* is a relabelling function. Similar remarks apply in the case of *f2*. Finally if we compose *Addone1* amd *Addone2* and restrict on *mid*, we obtain the following agent:

$$Addtwo \stackrel{def}{=} (Addone1) \mid Addone2) \setminus mid$$

which may repeatedly accept a value v at port *in* and after some internal communication

output $v+2$ at port $\overline{out}$:



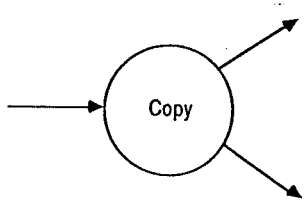
We have now introduced all the constructors of the calculus. We conclude this section by summarising the means we have of constructing agents.
Basic agent. *Nil*
Action prefixing. If l is a label and P an agent then $l.P$ is an agent. If l is an input label it may be parameterised by a variable, while if it is an output label it may be parameterised by an expression.
Summation. If P and Q are agents then $P+Q$ is an agent.
Composition. If P and Q are agents then $P|Q$ is an agent.
Restriction. If P is an agent and L a set of labels (or l is a label) then $P\backslash L$ ($P\backslash l$) is an agent.
Relabelling. If P is an agent and f a relabelling function then $P[f]$ is an agent.
Conditional agent. If P and Q are agents and e is a Boolean expression then *if* e *then* P *else* Q is an agent.
Recursive definition. We may define agents by recursive definitions of the form

$$agent \stackrel{def}{=} expression$$

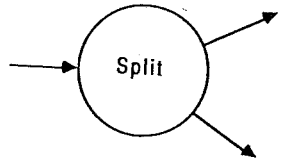
where *expression* may contain *agent*. More generally we may have several mutually recursive defintions.

Exercises 1

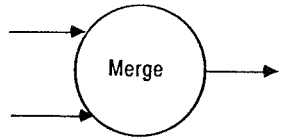
Exercise 1. (a) Define an agent *Copy* which, repeatedly, inputs a value and then outputs that value at two ports:



(b) Define an agent *Split* which repeatedly inputs a value and then outputs that value at one of its two output ports, successive outputs occurring at different ports:

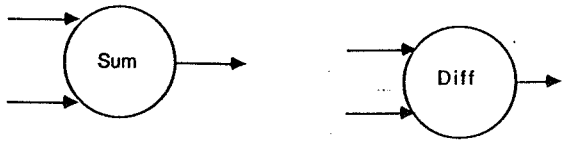


(c) Define an agent *Merge* which repeatedly inputs a value at one of its two input ports before outputting that value, successive inputs occurring at different ports, and the first input occurring at a designated port:



(d) Define agents *Sum* and *Diff* which repeatedly input a number at each of their two input

ports and then output the sum, respectively, the difference, of the numbers input:



(e) Draw a flow diagram representing a system, constructed from the above agents, which repeatedly inputs two numbers at its input port and outputs at its output port their sum followed by their difference. Using the relabelling operator as necessary, define an agent whose behaviour is that of the system represented. Can your system deadlock?

(f) Modify your system if necessary, perhaps by adding new components, so that a new number can not be input until the results from the previous pair have been output.

Exercise 2. (a) Define an agent *Change* which describes the behaviour of a change-giving machine with one input slot and one output slot which is capable initially of accepting either a 20 pence coin or a 10 pence coin, and which after having accepted one coin is capable of dispensing any sequence of 1 pence, 2 pence, 5 pence and 10 pence coins the sum of whose values is equal to that of the coin accepted, before returning to its initial state.

(b) Define an agent *Source* which is capable of repeatedly producing 20 pence coins and an agent *Sink* which is capable of repeatedly accepting coins of any value.

(c) Define an agent *Man* which describes the behaviour of a man who may initially accept a 20 pence coin from *Source*, then insert that coin into the change-giving machine and receive the dispensed change, then send the coins received in decreasing order of value to *Sink*, before returning to the initial state.

(d) Define an agent *Woman* whose behaviour is the same as that of *Man* except that when all the change has been received from *Change*, no coin can be sent to *Sink* if at least one 10 pence coin has been received. In that case *Woman* should attempt to change all 10 pence coins received for coins of value at most 5 pence. If and when this has been achieved *Woman* should send all the coins held, in increasing order of value, to *Sink*, before return-

ing to the original state.

(e) Examine the behaviour of the agent *System* defined by

$$System \stackrel{def}{=} (Source \mid Sink \mid Change \mid Man \mid Woman) \setminus L$$

where L is the set of labels through which *Man* and *Change*, and *Woman* and *Change* communicate. In particular, consider the following question: is it possible for *System* to go through an infinite number of "state transitions" in none of which *Sink* is an active participant? If so, how could one component of *System* be modified to remove this possibility?

Exercise 3. Consider the following agents:

$$\begin{aligned}
Input &\stackrel{def}{=} in(x).\overline{send}(x).Input \\
Output(\langle\rangle) &\stackrel{def}{=} receive(x).Output(\langle x\rangle) \\
Output(\langle x_1 \ldots x_n\rangle) &\stackrel{def}{=} receive(x).Output(\langle x, x_1 \ldots x_n\rangle) \;+\; \overline{out}(x_n).Output(\langle x_1 \ldots x_{n-1}\rangle) \\
Output_i(s) &\stackrel{def}{=} Output(s)[receive_i \,/\, receive,\ out_i \,/\, out] \quad i = 0 \text{ or } 1 \\
List &\stackrel{def}{=} getlist.putlist.List \\
Stamp &\stackrel{def}{=} getstamp.putstamp.Stamp \\
Worker &\stackrel{def}{=} send(x).Worker'(x) \\
Worker'(x) &\stackrel{def}{=} \overline{getlist}.\overline{getstamp}.\overline{receive}_{d(x)}(x).Worker'' \\
&\quad + \overline{getstamp}.\overline{getlist}.\overline{receive}_{d(x)}(x).Worker'' \\
Worker'' &\stackrel{def}{=} \overline{putlist}.\overline{putstamp}.Worker \\
&\quad + \overline{putstamp}.\overline{putlist}.Worker \\
Processor &\stackrel{def}{=} (Input \mid List \mid Stamp \mid Worker \mid Worker \mid \\
&\quad \mid Output_0 \mid Output_1) \setminus L
\end{aligned}$$

where $L = \{$ *getlist, putlist, getstamp, putstamp, send,* $receive_0$, $receive_1 \}$.
Processor is intended to model a system for processing items of some sort. Each item is input at port *in* and passed to one of the two identical *Workers* who checks the *List* to determine where the item should be output: this information is modelled by the function *d*. The *Worker* then stamps the item (using the agent *Stamp*) to indicate that it has been processed before sending it to the appropriate $Output_i$, i=0 or 1. $Output_i$ may eventually send it out at port $\overline{out_i}$, but is capable of buffering any number of items. Notice that to process any particular item a *Worker* must acquire both *List* and *Stamp*.

(a) Draw a flow-graph of *Processor* and investigate the behaviour of the agent.

(b) Does *Processor* function reliably? In particular, is it possible for *Processor* to deadlock?

(c) If your answer to second question in (b) was positive, suggest possible alterations to *Processor* which would make it free of deadlock.

(d) Suppose that *Processor* has been modified in such a way that it is free of deadlock. Is the behaviour of the modified *Processor* the same as that of the agent P defined as follows:

$$\begin{aligned} P &\stackrel{def}{=} P(\langle\rangle) \\ P(\langle\rangle) &\stackrel{def}{=} in(x).P(\langle x\rangle) \\ P(\langle x, x_1 \ldots x_{n-1}\rangle) &\stackrel{def}{=} in(y).P(\langle x, x_1 \ldots x_{n-1}, y\rangle) + \overline{out}_{d(x)}(x).P(\langle x_1 \ldots x_{n-1}\rangle). \end{aligned}$$

2. The operational semantics of agents

In this section we consider more carefully what we mean by the "behaviour" of an agent. It is often convenient to consider a process as passing through various "states" during its execution, and to think of changes of state, *state transitions*, as taking place on occurrences of certain "events". For example, in Section 1 we introduced an agent *Change* to model certain aspects of the behaviour of a change-giving machine:

$$Change \stackrel{def}{=} fifty.\overline{out}(20).\overline{out}(20).\overline{out}(10).Change.$$

For convenience we reformulate this definition as follows:

$$\begin{aligned} Change &\stackrel{def}{=} fifty.Change' \\ Change' &\stackrel{def}{=} \overline{out}(20).Change'' \\ Change'' &\stackrel{def}{=} \overline{out}(20).Change''' \\ Change''' &\stackrel{def}{=} \overline{out}(10).Change. \end{aligned}$$

We may think of each of the agents *Change*, *Change'*, *Change''* and *Change'''* as representing a particular state of the change-giving machine. *Change*, for example, represents the machine in the state in which it will accept a 50 pence coin, while *Change''* represents the state in which one 20 pence coin has been dispensed and a balance of 30 pence remains. Furthermore, the actions *fifty*, $\overline{out}(20)$ and $\overline{out}(10)$ represent the events whose occurrences attend a change of state. Thus rather than speak of the "states of an agent" we shall use the terms "state" and "agent" synonomously.

It is convenient to introduce some notation to represent state transitions. We write

$$Change \xrightarrow{fifty} Change'$$

to indicate that the agent *Change* may perform the action *fifty* to become the agent *Change'*. In general, given agents P and Q and an action a, we write

$$P \xrightarrow{a} Q$$

to indicate that the agent P may perform the action a to become the agent Q.

How are we to determine when an assertion of the form $P \xrightarrow{a} Q$ holds of given agents P and Q and given action a? We do so by means of rules from which we may infer such transitions. The rules are intended to encapsulate the informal operational understanding of the behaviour of agents presented in Section 1. Before describing the rules, we must consider state transitions of composite agents which occur as the result of communication between some of the components.

We may describe the behaviour of a person who wishes, by interacting with the machine described by the agent *Change*, to obtain change for a 50 pence coin by defining an agent *Person* as follows:

$$Person \stackrel{def}{=} \overline{fifty}.Person'$$
$$Person' \stackrel{def}{=} \ldots$$

where the definition of *Person'* is for the moment unimportant. Then given our understanding of the action-prefixing operator, we should expect (and the rules to be introduced will confirm our expectation) that the following transition be possible:

$$Person \xrightarrow{\overline{fifty}} Person'.$$

Also we will have, as suggested above, that

$$Change \xrightarrow{fifty} Change'$$

describes a possible state transition of the agent *Change*. Thus *Person* and *Change* may perfom the complementary actions $\overline{fifty}$ and *fifty*, and so given our understanding of the composition operator we should expect that the composite agent *Person*|*Change* can undergo a state transition to become the agent *Person'*|*Change'* as the result of a "communication event" through the complementary ports. Thus our rules should guarantee that a state transition of the form

$$Person|Change \xrightarrow{?} Person'|Change'$$

is possible. But what action should we write instead of the "?" here? In other words, how should we model the occurrence of a communication event between agents?

In CCS we regard an occurrence of such an event as being unobservable by the environment of the communicating agents. We are thus led to introduce a single new action to represent the occurrence of a communication event between agents. We use τ (the Greek letter tau) for this purpose. We introduce only a single new action, rather than many such actions. This is, as we shall see, consistent with the intention to regard communications as special events, invisible to the environment of the agents involved.

We are now ready to introduce the rules from which we may infer the possibility of transitions between agents. Recalling that an agent may either be given by a definition or be constructed from existing agents using one of the constructors of the calculus, we are led to give a set of rules covering all the possible forms which an agent may take. In each case, except for that of action-prefixing which is particularly simple, the rules enable us to infer a transition of an agent from transitions of "simpler" agents, or in the case of recursively defined agents from those of the defining expressions. In stating the rules we use a to denote an arbitrary member of the set $\mathcal{L}$ of labels or the special action τ.

Nil For *Nil* there is no rule. This corresponds to our understanding of *Nil* as representing a process which can do nothing.

Action-prefixing If a is an action and P an agent then $a.P$ is an agent which initially is capable of performing exactly one action, namely a, and thereafter behaving as the agent P. This is expressed by the rule `Act`:

$$\texttt{Infer}\ a.P \xrightarrow{a} P.$$

If a=$l(x)$ is an input label parameterised by a variable, then for each value v of the appropriate type we introduce a new label l_v and modify the rule to read

$$\texttt{Infer } l(x).P \xrightarrow{l_v} P\{v/x\}$$

where $P\{v/x\}$ is the agent obtained from P by substituting v for all occurrences of x. If a=$\overline{l}(e)$ is an ouput label parameterised by an expression, and v is the value of the expression e, then we modify the rule to read

$$\overline{l}(e).P \xrightarrow{\overline{l_v}} P.$$

Summation $P+Q$ is an agent with the capabilities of both P and Q. This is expressed by the rules Sum1 and Sum2:

$$\begin{array}{l}\texttt{From } P \xrightarrow{a} P' \texttt{ infer } P+Q \xrightarrow{a} P' \\ \texttt{From } Q \xrightarrow{a} Q' \texttt{ infer } P+Q \xrightarrow{a} Q'.\end{array}$$

Composition $P|Q$ is an agent whose behaviour is such that each of P and Q may act independently of the other, while P and Q may together engage in a communication whenever thay are able to perform complementary actions. There are three rules to express these possibilities, Com1, Com2 and Com3:

$$\begin{array}{lll}\texttt{From} & P \xrightarrow{a} P' & \texttt{infer } P|Q \xrightarrow{a} P'|Q \\ \texttt{From} & Q \xrightarrow{a} Q' & \texttt{infer } P|Q \xrightarrow{a} P|Q' \\ \texttt{From} & P \xrightarrow{l} P' \text{ and } Q \xrightarrow{\overline{l}} Q' & \texttt{infer } P|Q \xrightarrow{\tau} P'|Q'.\end{array}$$

Notice that we model a state transition occurring on a communication between two agents as a τ-action.

Restriction $P\backslash L$ is an agent which behaves like P except that it may not engage in any action l such that l or $\overline{l}$ lies in L. This is expressed by the rule Res:

$$\texttt{If } a, \overline{a} \texttt{ are not in } L \texttt{ then from } P \xrightarrow{a} P' \texttt{ infer } P\backslash L \xrightarrow{a} P'\backslash L.$$

Notice that P' is restricted on L. Notice also that L does not contain τ, since L is a set of labels: the silent action can not be restricted.

Relabelling $P[f]$ is an agent whose behaviour is like that of P except that the labels are relabelled as specified by the function f. This is expressed by the following rule Rel:

$$\texttt{From } P \xrightarrow{a} P' \texttt{ infer } P[f] \xrightarrow{f(a)} P'[f].$$

Note that P' is relabelled by f. Of a relabelling function we require that $f(\tau)$=τ, so that the silent action can not be relabelled.

Conditional `if` e `then` P `else` Q is an agent which behaves like P or like Q depending on whether the Boolean expression e evaluates to `true` or to `false`. This is expressed by the rules Cond1 and Cond2:

$$\text{If e evaluates to } \texttt{true} \text{ then } \texttt{from } P \xrightarrow{a} P' \texttt{ infer if } e \texttt{ then } P \texttt{ else } Q \xrightarrow{a} P'$$

If e evaluates to `false` then `from` $Q \xrightarrow{a} Q'$ `infer if` e `then` P `else` $Q \xrightarrow{a} Q'$.

Recursive definition If $P \stackrel{def}{=} A$ then the behaviour of the defined agent P is that of the defining agent A as expressed by the rule Rec:

$$\texttt{From } A \xrightarrow{a} P' \texttt{ infer } P \xrightarrow{a} P'.$$

From these rules we may infer the possibility of certain transitions for agents. We stipulate that a transition of the form $P \xrightarrow{a} Q$ is possible precisely when it may be inferred by some finite number of applications of the rules. As an example consider the agents *Person* and *Change* introduced earlier:

$$\begin{aligned} Person &\stackrel{def}{=} \overline{fifty}.Person' \\ Change &\stackrel{def}{=} fifty.Change' \end{aligned}$$

where the definitions of *Person'* and *Change'* are not for the moment significant. According to our understanding of the operators of the calculus we should expect that *Person* and *Change* should be capable of communicating through the complementary actions $\overline{fifty}$ and *fifty*. Thus we should be able to infer

$$Person|Change \xrightarrow{\tau} Person'|Change'.$$

Since $Person|Change$ is a composition we look at the rules (Com1), (Com2) and (Com3) to see if any is applicable. (Com1) would enable us to infer the transition (A) if we could show that $Person \xrightarrow{\tau} Person'$ and $Change = Change'$. Now in fact $Change \neq Change'$, but since we are presently interested in understanding the transition rules, let us continue to see whether we could infer $Person \xrightarrow{\tau} Person'$. Since *Person* is given by a definition we consider the rule (Rec), and since the defining expression of *Person* is an action-prefix we consider the rule (Act). But since $\tau \neq \overline{fifty}$ we see that this rule does not enable us to infer the transition. Thus (Com1) is useless in trying to establish the transition (A). Similarly (Com2) is of no help. (Com3) however, would enable us to infer the transition (A), provided that we could find some pair of complementary labels l and $\overline{l}$ and show that $Person \xrightarrow{l} Person'$ and $Change \xrightarrow{\overline{l}} Change'$. The only rule by means of which we are able to infer transitions of *Person* and *Change* is (Act), and applying this rule we see that $Person \xrightarrow{\overline{fifty}} Person'$ and $Change \xrightarrow{fifty} Change'$. Thus we have satisfied the conditions required for an application of the rule (Com3) and thus we may infer the transition (A).

To aid in recording such inferences as that of the transition (A), it is convenient to introduce the following notation. We write

$$\frac{P_1 \xrightarrow{a_1} P_1' \ \cdots \ P_n \xrightarrow{a_n} P_n'}{P \xrightarrow{a} P'} \ (\texttt{X})$$

to indicate that using rule (X) we may infer from the transitions above the line the transition below the line. Thus, for example, to indicate the application of the rule (Com3) in deriving the transition (A) above we write

$$\frac{Person \xrightarrow{\overline{fifty}} Person' \quad Change \xrightarrow{fifty} Change'}{Person|Change \xrightarrow{\tau} Person'|Change'} \text{ (Com3).}$$

By placing representations such as this for all rules used in a particular derivation in an appropriate arrangement, we have a concise record of that derivation. For example we may record the derivation of the transition (A) above as follows:

$$\frac{\dfrac{}{Person \xrightarrow{\overline{fifty}} Person'} \text{(Act)} \quad \dfrac{}{Change \xrightarrow{fifty} Change'} \text{(Act)}}{Person \mid Change \xrightarrow{\tau} Person' \mid Change'} \text{ (Com3).}$$

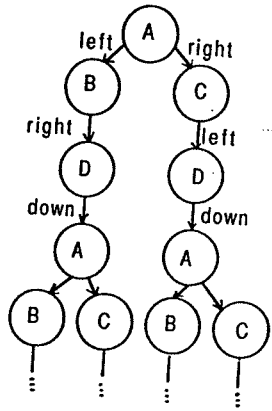
Note that above the lines marked (Act) we have no transitions: this reflects the fact that no condition other than that the agent be an action-prefix is required to be satisfied in order that this rule be applicable.

We can regard the transition rules as providing an operational semantics, or meaning, for the agents of the calculus. The behaviour of an agent may be represented as a sort of labelled tree. To construct that tree we start with a node labelled by the agent P in question, the root of the tree. Then for each agent Q for which there is some transition of the form $P \xrightarrow{a} Q$—each such agent is said to be a *successor* of P— we introduce a new node labelled Q and draw an arc from the node labelled P to the node labelled Q and label that arc with the action a. Then we repeat the same process, regarding each of the new nodes in turn as we regarded the root node before; and so on. Now this process may not terminate if the agent involves a recursive definition. However in that case the tree constructed, which we refer to as the *derivation tree* of the agent P, *may* exhibit a repetitive structure. For example, the agent A defined by

$$A \stackrel{def}{=} left.right.\overline{down}.A + right.left.\overline{down}.A$$

which may repeatedly input signals at ports labelled *left* and *right*, in either order, and

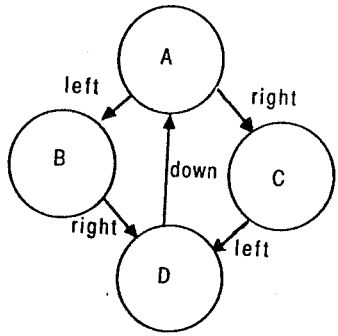
then output a signal at a port labelled $\overline{down}$, has the following derivation tree:



where $B \stackrel{def}{=} right.D$, $C \stackrel{def}{=} left.D$ and $D \stackrel{def}{=} \overline{down}.A$, and we have indicated that the structure of the tree is repeated indefinitely. Each agent labelling a node in the derivation tree of the agent P is called a *derivative* of P.

It is useful to introduce a name for the set of all actions which an agent may perform at any time during its evolution. Thus we say that the *sort* of an agent P, $sort(P)$, is the set of labels in $\mathcal{L}$ which label an arc in the derivation tree of P. We may think of $sort(P)$ as the set of labels through which P may communicate at least once during its lifetime.

Notice that on account of the repetitive structure of the derivation tree of the agent A, all essential information about the behaviour of A is contained in the first four lines of the tree. Thus in this case we may "fold up" the derivation tree to obtain the following *transition diagram* of the agent A:



In such diagrams there is an arc labelled by an action a from a node labelled by an agent P to a node labelled by an agent Q precisely when $P \xrightarrow{a} Q$ holds. In closing this section we remark that it is not always possible to obtain a finite transition diagram representing the behaviour of an agent, as the derivation trees of some agents have an infinite number of nodes labelled by distinct agents.

Exercises 2

Exercise 1. Using the transition rules find all pairs (d, B') such that $B \xrightarrow{d} B'$ where B is defined by

$$B \stackrel{def}{=} ((\overline{a}.Nil + b.B) \mid (\overline{b}.Nil + a.B))\,[f] \setminus b)$$

where $f(b) = c$, $f(a) = b$ and $f(l) = l$ for all l other than b, $\overline{b}$, a and $\overline{a}$.

Exercise 2. Draw part of the derivation trees of the agents A and B defined by

$$\begin{aligned} A &\stackrel{def}{=} a.(\tau.A + b.Nil) + \tau.a.A \\ B &\stackrel{def}{=} ((a.Nil \mid \overline{a}.Nil + b.Nil) \setminus a) + c.B' \\ B' &\stackrel{def}{=} d.B' \end{aligned}$$

and from this construct transition diagrams of A and B.

Exercise 3. Let

$$\begin{aligned} C_1 &\stackrel{def}{=} a.\overline{b}.c.C_1 + \overline{b}.c.a.C_1 + c.a.\overline{b}.C_1 \\ C_2 &\stackrel{def}{=} d.\overline{c}.b.C_2 + \overline{c}.b.d.C_2 + b.d.\overline{c}.C_2 \\ C &\stackrel{def}{=} (C_1 \mid C_2) \setminus \{b, c\}. \end{aligned}$$

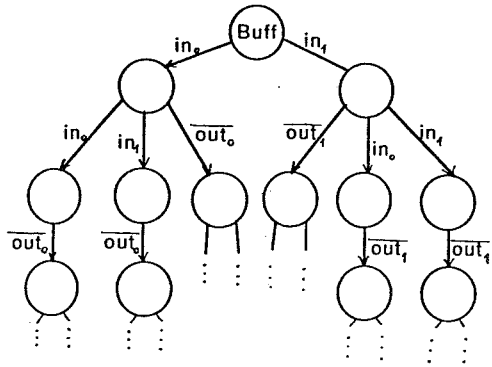
Construct a transition graph of C. Define an agent D not involving the composition, restriction and relabelling operators with the same transition diagram. Is it possible to delete any of the τ-transitions from the diagram in such a way that the modified diagram represents an agent with the same observable behaviour as C?

3. Observational equivalence and congruence

Having analysed the behaviours of agents, and seen how we may represent such behaviours as labelled trees, in this section we consider the problem of what we mean when we say that two agents "have the same behaviour". We begin by considering the following agents:

$$\begin{aligned} \mathit{Buff} &\stackrel{def}{=} in(x).\mathit{Buff}'(x) \\ \mathit{Buff}'(x) &\stackrel{def}{=} in(y).\mathit{Buff}''(x,y) + \overline{out}(x).\mathit{Buff} \\ \mathit{Buff}''(x,y) &\stackrel{def}{=} \overline{out}(x).\mathit{Buff}'(y). \end{aligned}$$

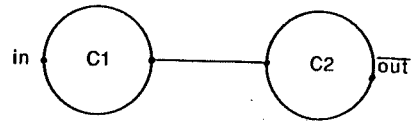
We may regard these agents as providing a specification of the behaviour of a two-place buffer, *Buff*, *Buff'* and *Buff''* representing, respectively, the states in which zero, one or two values are held in the buffer. Depending on the nature of the values which the buffer may hold, the derivation tree of *Buff* may contain an infinite number of states. Let us assume, for simplicity, that the only values which may be held in the buffer are 0 and 1. Then introducing pairs of actions in_0, in_1 and $\overline{out}_0$, $\overline{out}_1$ to represent the communication of the particular values, we may represent the derivation tree of *Buff* as follows:



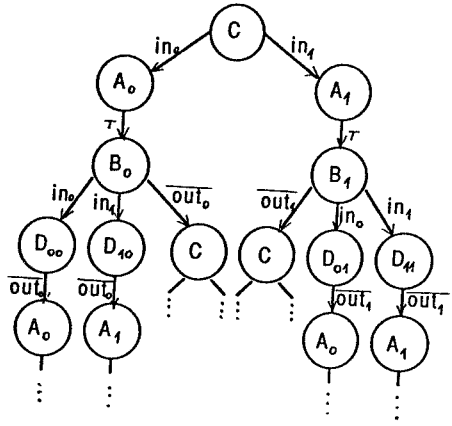
Now consider the following agents:

$$\begin{aligned} \mathit{Cell} &\stackrel{def}{=} in(x).\mathit{Cell}'(x) \\ \mathit{Cell}'(x) &\stackrel{def}{=} \overline{out}(x).\mathit{Cell} \\ \mathit{C1} &\stackrel{def}{=} \mathit{Cell}[\overline{mid}/\overline{out}] \\ \mathit{C2} &\stackrel{def}{=} \mathit{Cell}[mid/in] \\ \mathit{C} &\stackrel{def}{=} (\mathit{C1} \mid \mathit{C2}) \setminus mid. \end{aligned}$$

We may picture the agent C, constructed from two copies of the agent *Cell* by relabelling, composition and restriction, as follows:



Assuming that the only values which these agents may communicate are 0 and 1, the derivation tree of C is as follows:



where $A_i \stackrel{def}{=} (C1'(i) \mid C2) \setminus mid$, $B_i \stackrel{def}{=} (C1 \mid C2'(i)) \setminus mid$ and $D_{ij} \stackrel{def}{=} (C1'(j) \mid C2'(i)) \setminus mid$ for $i,j = 0$ or 1.

If we compare these two trees and recall our intention to regard τ-actions as representing occurrences of events invisible to the environment, then we are led to the view that the agents *Buff* and C exhibit the same observable behaviour. Thus we should like to be able to regard *Buff* and C as "equivalent" agents, in the sense that whenever one were a component of some complex system, the other could be substituted for it without affecting the behaviour of the complex system. Notice that here we speak of the complex system, and a modification of that system, as exhibiting "the same behaviour". Thus we are led to consider the following questions: (1) Can we define a notion of "equivalence" between agents which encapsulates some intuitive notion of "having the same behaviour"? (2)

Assuming a positive answer to (1), can we develop techniques by means of which we may determine whether or not two given agents are "equivalent"? Question (1) does indeed have a positive answer. In fact, several different notions of equivalence have been proposed, each capturing *some* idea of what it is for two agents "to have the same behaviour". We concentrate on one particular equivalence, *observational equivalence*. This equivalence provides a very reasonable formalization of the notion of "having the same behaviour". Furthermore for this equivalence we may answer question (2) affirmatively. This assertion will be fully discussed later, particularly in section 5.

To motivate the definition of observational equivalence, let us try to isolate some properties which we would expect of a reasonable formalization of the notion of "having the same behaviour". Firstly, and quite apart from any specific considerations of behaviour, there are certain properties which a relation must possess to be an acceptable notion of equivalence between agents. Using $=$ to denote the relation under consideration, we may express these as follows:
(1) Every agent should be equivalent to itself, i.e.

$$P = P$$

should hold for all agents P.
(2) The notion of equivalence should be *symmetric* in the sense that for all agents P and Q

$$\textit{if } P = Q \textit{ then } Q = P.$$

(3) The relation should be *transitive* in the sense that for all agents P, Q and R

$$\textit{if } P = Q \textit{ and } Q = R \textit{ then } P = R.$$

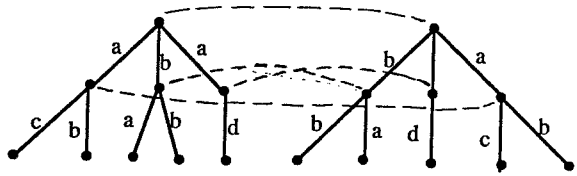
(4) We should expect the relation to have the "substitutivity" property described above: that if $P{=}Q$ and S is some complex system of which P is a component, then the agents S' obtained by replacing P by Q should be equivalent to S. We may express this as follows: for all actions a, restriction sets L, relabelling functions f, Boolean expressions e and agents R, A and B

$$\begin{aligned}
\text{if } P = Q \text{ then } \quad & a.P = a.Q \\
& P + R = Q + R \\
& P \mid R = Q \mid R \\
& P \setminus L = Q \setminus L \\
& P[f] = Q[f] \\
& \texttt{if } e \texttt{ then } P \texttt{ else } R = \texttt{if } e \texttt{ then } Q \texttt{ else } R \\
& \text{if } A \stackrel{def}{=} P \text{ and } B \stackrel{def}{=} Q \text{ then } A = B.
\end{aligned}$$

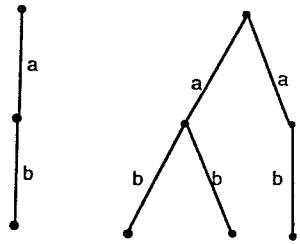
Any relation satisfying (1), (2) and (3) is said to be an *equivalence relation* of agents, while a relation satisfying in addition (4) is said to be a *congruence relation* of agents (with respect to the operators of the calculus). Thus in order that a relation be an acceptable

notion of equivalence, it must be a congruence relation of agents. Of course not every congruence relation will provide a suitable formalization of the notion of "having the same behaviour": the relation must also encapsulate our operational understanding. Thus, for example, syntactic identity, although a congruence, is much too restrictive, since we should certainly wish, given two agents P and Q to regard, for example, the agents $P+Q$ and $Q+P$ as equivalent, since the behaviour of each is that of an agent with the capabilities of both P and Q.

Now the derivation trees of $P+Q$ and $Q+P$ might differ only in the order in which the successors of a given node were drawn, and thus we could regard them as "the same tree". So perhaps we might say that two agents are equivalent if the nodes of their derivation trees may be placed in a certain type of one-to-one correspondence, as illustrated in the example below:



However this also is too restrictive as the nodes of the derivation trees



can not be placed in such a correspondence, yet we should wish to regard the agents which they represent, *a.b.Nil* and *a.b.Nil+a.(b.Nil+b.Nil)*, as equivalent.

As another idea, we might regard two agents as being equivalent precisely when according to the operational semantics they may perform the same sequences of actions. Thus, if given agents P and Q and a sequence $s = \langle a_1 \ldots a_n \rangle$ of actions we write

$$P \xrightarrow{s} Q$$

whenever for some agents $P_1, \ldots, P_{n-1}$ we have

$$P \xrightarrow{a_1} P_1,\ P_1 \xrightarrow{a_2} P_2,\ \ldots\ P_{n-1} \xrightarrow{a_n} Q,$$

we might regard agents P and Q as equivalent precisely when for all sequences s of actions, for some P', $P \xrightarrow{s} P'$ holds if and only if for some Q', $P' \xrightarrow{s} Q'$ holds.

However this equivalence is insufficiently discriminating as we may see by considering the agents $a.(b.Nil + c.Nil)$ and $a.b.nil + a.c.Nil$. Both of these agents may perform the sequences $\langle\rangle$ (the empty sequence), $\langle a\rangle$, $\langle a, b\rangle$ and $\langle a, c\rangle$. The latter however has the capability of performing the action a in such a way that the action c is not possible, whereas the former does not have this capability. On account of this difference we should not wish to regard these agents as equivalent and we must therefore reject the above suggestion.

From the above examples we see that for two agents to be equivalent not only must they be capable of performing the same actions, but also there must be a close relationship between the states which may be reached from each by the performance of a given action. In considering the *observable* behaviour of agents we shall have to take into account our intention to regard τ-actions as special events whose occurrences are invisible to the environment. If, however, we neglect this consideration for the moment, and consider what we mean by "a close relationship between the states", then bearing in mind that in saying that two agents "have the same behaviour" we are thinking of the behaviour of the agents throughout their lifetimes, then we are led to consider seeking a notion of equivalence with the following property (*):

P and Q are "equivalent" if and only if for each action a,

(a) if P' is such that $P \xrightarrow{a} P'$, then there is Q' such that $Q \xrightarrow{a} Q'$ and P' and Q' are "equivalent", and conversely,

(b) if Q' is such that $Q \xrightarrow{a} Q'$, then there is P' such that $P \xrightarrow{a} P'$ and P' and Q' are "equivalent".

The property (*) can not itself be taken as a *definition* of "equivalent" because the term "equivalent" appears on both sides of the "if and only if", and thus as a definition (*) is circular. Indeed it is not clear whether there is *any* relation satisfying (*); far less is it apparent that there is exactly one such relation. It turns out, however, that there is indeed a relation satisfying (*) and, moreover, that amongst all such relations there is a *largest* one, in the sense that whenever two agents are related by *some* relation satisfying (*), they are related by this largest relation. It is in fact the case that this relation is a congruence relation of agents. Thus it might be considered as a candidate for our notion of "equivalence" between agents. However, recall that (*) explicitly ignores our intention to regard τ-actions as representing occurrences of events invisible to the environment: for two agents to be equivalent under the largest relation satisfying (*), every τ-action of one would have to be "matched" by a τ-action of the other. Thus, for example, the agent Nil and $\tau.Nil$, and hence also the agents $a.Nil$ and $a.\tau.Nil$, would not be equivalent under the relation. But we should certainly wish to regard them as equivalent, so we are led to consider some modification of (*) which takes into account the special status of τ-actions.

First we introduce some notation. Given agents P and Q and an action a, we write

$$P \stackrel{a}{\Longrightarrow} Q$$

if $P \xrightarrow{\langle\tau\rangle^m\langle a\rangle\langle\tau\rangle^n} Q$ for some m,n$\geq$0, where $\langle\tau\rangle^k$ is a sequence of k τ's and juxtaposition of sequences denotes concatenation. Thus $P \stackrel{a}{\Longrightarrow} Q$ if P may be transformed to Q by performing the action a preceded and succeeded by any finite number (possibly zero) of

τ-actions. For example if $P \stackrel{def}{=} \tau.a.\tau.Nil$ then

$$\begin{aligned} P &\stackrel{\tau}{\Longrightarrow} a.\tau.Nil \\ P &\stackrel{a}{\Longrightarrow} \tau.Nil \\ P &\stackrel{a}{\Longrightarrow} Nil. \end{aligned}$$

Note in particular that $P \stackrel{\tau}{\Longrightarrow} Q$ if and only if P may become Q by performing any sequence of length *at least one* of τ-actions. We may now modify (*) to obtain the following property (**) which takes account of the special status of τ-actions:
P and Q are "equivalent" if and only if for each action a,
(a) if P' is such that $P \stackrel{a}{\longrightarrow} P'$, then there is Q' such that $Q \stackrel{a}{\Longrightarrow} Q'$ and P' and Q' are "equivalent", and conversely,
(b) if Q' is such that $Q \stackrel{a}{\longrightarrow} Q'$, then there is P' such that $P \stackrel{a}{\Longrightarrow} P'$ and P' and Q' are "equivalent".

Now this is not quite what we want, since assuming that there is a relation satisfying (**), two agents would be "equivalent" under it only if whenever either could perform a τ-action, the other could perform a sequence of τ-actions of length *at least one* to reach an "equivalent" state, and so the agents $\tau.Nil$ and Nil, and hence also the agents $a.\tau.Nil$ and $a.Nil$, would not be "equivalent" under this relation. Thus we modify (**) to obtain instead the property (***):
P and Q are "equivalent" if and only if for each action a
(a) if P' is such that $P \stackrel{a}{\longrightarrow} P'$, then *either* (1) there is Q' such that $Q \stackrel{a}{\Longrightarrow} Q'$ and P' and Q' are "equivalent, *or* (2) $a{=}\tau$ and P' is "equivalent" to Q, and conversely,
(b) if Q' is such that $Q \stackrel{a}{\longrightarrow} Q'$ then *either* (1) there is P' such that $P \stackrel{a}{\Longrightarrow} P'$ and P' and Q' are "equivalent", *or* (2) $a{=}\tau$ and P is "equivalent" to Q'.

Thus we do not require that every τ-action of P be "matched" by at least one τ-action of Q, *provided* that Q is itself "equivalent" to the agent reached from P via a τ-action; and similarly with P and Q interchanged. Now assuming that there exists a relation satisfying (***), $\tau.Nil$ and Nil, and hence also $a.\tau.Nil$ and $a.Nil$, will be "equivalent" under it. More generally, (***) seems to capture in a very reasonable manner an idea of what it means for two agents to have "the same behaviour", taking into account the special nature of τ-actions.

As with (*), (***) can not itself be taken as a *definition* of "equivalent". However it turns out that there is in fact a relation satisfying (***) and moreover that among all such relations there is a largest. We call this relation *observational equivalence* and write $P \approx Q$ to express that P and Q are related by it, that is that P and Q are *observationally equivalent*. Thus $\approx$ is such that: $P \approx Q$ if and only if for each action a,
(a) if P' is such that $P \stackrel{a}{\longrightarrow} P'$, then *either* (1) there is Q' such that $Q \stackrel{a}{\Longrightarrow} Q'$ and $P' \approx Q'$, *or* (2) $a{=}\tau$ and $P' \approx Q$, and conversely,
(b) if Q' is such that $Q \stackrel{a}{\longrightarrow} Q'$, then *either* (1) there is P' such that $P \stackrel{a}{\Longrightarrow} P'$ and $P' \approx Q'$, *or* (2) $a{=}\tau$ and $P \approx Q'$.
Moreover $\approx$ is the largest such relation, in the sense that if $\mathcal{R}$ is any relation satisfying (***) and $P \mathcal{R} Q$, then $P \approx Q$.

It is possible to show that observational equivalence is *almost* a congruence relation of agents, that is that it is very often possible to substitute in a complex system an agent

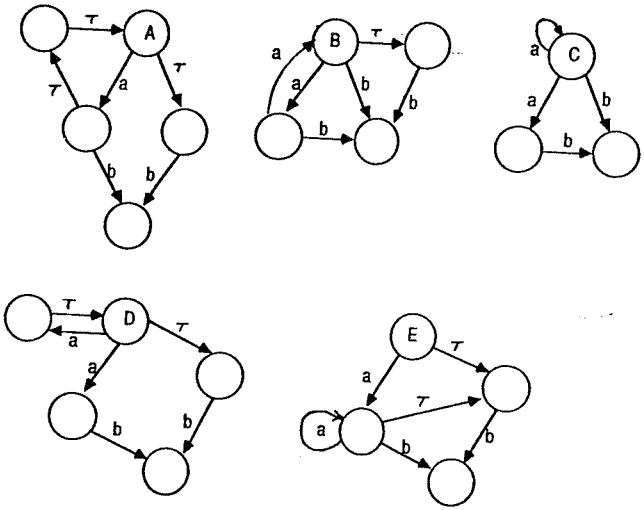
for an observationally equivalent agent and thereby obtain an observationally equivalent system. However it is *not* in general the case that if $P \approx Q$ then $P+R \approx Q+R$. For we have already seen that $\tau.a.Nil \approx a.Nil$, and so if the above implication were true then we should have that $\tau.a.Nil + l.Nil \approx a.Nil + l.Nil$ where l is any label. But this is not so for the agent on the left may perform a τ-action to become $a.Nil$, but the agent on the right may not perform any sequence of τ-actions of length at least one to become an agent observationally equivalent to $a.Nil$, nor is it itself observationally equivalent to $a.Nil$ (since it may perform the action l). This example illustrates the pre-emptive power of τ-actions: an occurrence of an internal communication, an event invisible to the environment, may significantly affect the behaviour of an agent. It is this possibility, which must certainly be taken into account if we are satisfactorily to model communicating systems, which leads us to refine observational equivalence slightly to obtain our notion of equivalence of agents.

This relation, which we call *observational congruence*, is the largest relation included in observational equivalence which is a congruence relation of agents. We write $P=Q$ if the agents P and Q are in this relation, i.e. are *observationally congruent*. Thus observational congruence is a fully substitutive equivalence relation.

It is possible to give an explicit characterisation of the relation $=$, but for the moment we shall merely reiterate that we have now arrived at a notion of equivalence which meets the criteria, set out at the beginning of this section, which must be satisfied by a relation if it is to provide a satisfactory formalization of the intuitive idea of two agents "having the same behaviour". It is somewhat unfortunate that observational equivalence ($\approx$) is not itself an congruence relation of agents. However it is "almost" so, and, for a reason to be explained in section 5, in order to establish that two agents are observationally congruent, it is often sufficient to prove them observationally equivalent. This fact, in conjunction with the existence of a manageable technique for determining whether or not two agents are observationally equivalent, a technique which will be described in section 5, is a significant factor in support of observational congruence in any attempt to balance the usefulness of the various possible notions of equivalence.

Exercises 3

Exercise 1. Find agents A, B, C, D and E with the following transition diagrams:



For each pair (P, Q) of these agents try to determine whether or not $P \approx Q$ holds. Given that $\approx$ is the largest relation satisfying (***), try to prove rigorously that one of $P \approx Q$ and $P \not\approx Q$ which holds of some particular pair.

Exercise 2. (a) Consider the following agents:

$$\begin{aligned} P &\stackrel{def}{=} in(x).\overline{mid}(x).P \\ Q &\stackrel{def}{=} mid(y).\overline{out}(y).Q \\ R &\stackrel{def}{=} (P \mid Q) \setminus mid \\ S &\stackrel{def}{=} in(x).\overline{out}(x).S. \end{aligned}$$

Try to determine whether or not $R \approx S$.

(b) Consider the following agents:

$$
\begin{aligned}
T &\stackrel{def}{=} in1(x).T_1(x) + in2(y).T_2(y) \\
T_1(x) &\stackrel{def}{=} \overline{out1}(x).T + in2(y).T_{12}(x,y) \\
T_2(y) &\stackrel{def}{=} \overline{out2}(y).T + in1(x).T_{12}(x,y) \\
T_{12}(x,y) &\stackrel{def}{=} \overline{out1}(x).T_2(y) + \overline{out2}(y).T_1(x)
\end{aligned}
$$

$$
\begin{aligned}
U &\stackrel{def}{=} in1(x).U_1(x) + send2(y).U_2(y) \\
U_1(x) &\stackrel{def}{=} \overline{send1}(x).ack1.U + send2(y).U_3(x,y) \\
U_2(y) &\stackrel{def}{=} \overline{out2}(y).\overline{ack2}.U \\
U_3(x,y) &\stackrel{def}{=} \overline{out2}(y).\overline{ack2}.U_1(x)
\end{aligned}
$$

$$
\begin{aligned}
V &\stackrel{def}{=} in2(y).V_1(y) + send1(x).V_2(x) \\
V_1(y) &\stackrel{def}{=} \overline{send2}(y).ack2.V + send1(x).V_3(x,y) \\
V_2(x) &\stackrel{def}{=} \overline{out1}(x).\overline{ack1}.V \\
V_3(x,y) &\stackrel{def}{=} \overline{out1}(x).\overline{ack1}.V_1(y)
\end{aligned}
$$

$$
W \stackrel{def}{=} (U \mid V) \setminus \{send1,\ send2,\ ack1,\ ack2\}.
$$

Examine the behaviour of these agents and try to determine whether or not $T \approx W$ holds.

4. Properties of observational congruence

Having introduced observational congruence, we devote this section to describing some of its properties. First we recall that observational congruence is a congruence relation of agents, that is that it is a fully substitutive equivalence relation. The remainder of the properties, which we express as laws relating agents of the calculus, when combined with the proof technique for observational equivalence to be described in section 5, provide a means of tackling the problem of establishing the observational congruence of agents.

The laws may conveniently be divided into four groups: the Dynamic Laws, expressing properties of the dynamic constructors, action-prefixing and summation; the Static Laws, describing properties of the static constructors, composition, restriction and relabelling; the Expansion Principle, which relates the static and dynamic constructors; and the Recursion Laws, useful in reasoning about recursively defined agents.

Dynamic Laws

Laws 1

$$\begin{array}{lrcl} (a) & P+Q & = & Q+P \\ (b) & P+(Q+R) & = & (P+Q)+R \\ (c) & P+P & = & P \\ (d) & P+Nil & = & P. \end{array}$$

These laws express various properties of the summation operator. In each of (a)–(d), and in all subsequent laws, the congruence holds for all agents P, Q and R. Thus (a) expresses that the order of agents in a summation is not significant; (b) that the same is true of the grouping of agents; (c) that in a summation duplicates of agents may be neglected; and (d) that the inclusion of *Nil* in a summation is of no significance.

The remainder of the Dynamic Laws concern the silent action τ. Each law enables us to manipulate agents containing τ. Their significance will become more fully apparent when we come to apply them.

Laws 2

$$\begin{array}{lrcl} (a) & a.\tau.P & = & a.P \\ (b) & P+\tau.P & = & \tau.P \\ (c) & a.(P+\tau.Q)+a.Q & = & a.(P+\tau.Q) \\ (d) & P+\tau.(P+Q) & = & \tau.(P+Q). \end{array}$$

These laws may appear somewhat strange at first. With familiarity, however, this should become less so. There is a precise sense in which they capture our understanding of τ as a silent action. Unfortunately to explain this remark fully would require a fairly long detour into the mathematics, so we must leave the assertion unsubstantiated.

At this point it is worth noting explicitly a couple of "non-laws", that is equations which express properties which agents do not generally enjoy. The first arises from our understanding of the silent action τ which may lead us to think that

$$P \stackrel{?}{=} \tau.P$$

always holds. However as we saw in section 3, an occurrence of a τ-action may pre-empt certain capabilities of an agent, and thus in general $P = \tau.P$ does not hold.

The second "non-law" which we consider is what we might call the "distributive property":

$$a.(P + Q) \stackrel{?}{=} a.P + a.Q.$$

This equivalence does not hold in general as we may see by considering the agents $P = b.Nil$ and $Q = c.Nil$. From the fact that $\approx$ satisfies the property (***) of section 3 we can see that $a.(P + Q) \not\approx a.P + a.Q$, since $a.(P + Q) \xrightarrow{a} P + Q$ whereas $a.P + a.Q$ can not progress (even with silent moves) to reach an equivalent state as $P + Q$ may perform both a b action and a c action while neither P nor Q has this capability.

Static Laws

The first set of laws express properties of the composition operator:

Laws 3

$$\begin{array}{lrcl} (a) & P|Q & = & Q|P \\ (b) & P|(Q|R) & = & (P|Q)|R \\ (c) & P|Nil & = & P. \end{array}$$

(a) and (b) express that the order and grouping of agents in a composition is insignificant, while (c) expresses that the presence of Nil in a composition may be disregarded.

Next we have laws expressing properties of the restriction operator and its relationship to the other static constructors. In subsequent laws L and K are arbitrary sets of labels and f and g arbitrary relabelling functions.

Laws 4

$$\begin{array}{lrcl} (a) & P\backslash L & = & P, \text{ provided } sort(P) \cap (L \cup \overline{L}) = \emptyset \\ (b) & (P\backslash L)\backslash K & = & P\backslash(L \cup K) \\ (c) & (P[f])\backslash L & = & (P\backslash f^{-1}(L))[f] \\ (d) & (P|Q)\backslash L & = & (P\backslash L)|(Q\backslash L), \text{ provided } sort(P) \cap \overline{sort(Q)} \cap (L \cup \overline{L}) = \emptyset. \end{array}$$

Recall that *sort(P)* is the set of all labels through which P may communicate during its lifetime. In (a) and (d) we use sorts to describe the conditions under which the laws hold. (a) expresses that restricting an agent by a set of labels none of which is in the sort of the agent has no effect; (b) expresses that the effect of a repeated restriction may be achieved

by restricting on all relevant labels at once; (c) expresses that a relabelling followed by a restriction is congruent to a modified restriction followed by a relabelling: $f^{-1}(L)$ is the set of all labels mapped by f to labels in L; finally (d) expresses that under a certain condition, the restriction of a composition of agents is congruent to the composition of the restricted agents. In general such a congruence does not hold and it is easy to construct an example to illustrate this. However the congruence does hold under the stated condition.

The remainder of the Static Laws describe properties of the relabelling operator.

Laws 5

$$\begin{array}{lrcl} (a) & P[Id_L] & = & P \\ (b) & P[f] & = & P[g], \text{ provided } f \text{ and } g \text{ agree on } sort(P) \\ (c) & (P[f])[g] & = & P[g \circ f] \\ (d) & (P|Q)[f] & = & (P[f])|(Q[f]), \text{ provided } f \text{ is one} - \text{to} - \text{one on } sort(P|Q) \cup \overline{sort(P|Q)}. \end{array}$$

Here Id_L is the identity function on $\mathcal{L}$, so (a) expresses that changing no labels has no effect; (b) expresses that in relabelling P, only the labels in *sort(P)* are of significance: (c) expresses that a double relabelling is congruent to a single relabelling by the composite relabelling function; and (d) expresses that provided f has a certain property, a composition followed by a relabelling is congruent to relabelling each agent and then composing. As with (d) of Laws 4, the side condition of f is essential here.

Often we may deal with simple relabelling functions and it is worthwhile to state here some further laws which are actually special cases of earlier laws.

Laws 6

$$\begin{array}{lrcl} (a) & P[k/l] & = & P, \text{ provided } l, \overline{l} \notin sort(P) \\ (b) & P\backslash l & = & (P[k/l])\backslash k. \text{ provided } k, \overline{k} \notin sort(P) \\ (c) & (P\backslash L)[k/l] & = & (P[k/l])\backslash L, \text{ provided } k, l \notin L \cup \overline{L}. \end{array}$$

(a) expresses that relabelling an agent at a label not in its sort has no effect; (b) expresses that a new label may be introduced to effect a restriction; and (c) expresses that if a restriction involves neither the argument nor the value of a relabelling, then the order of the restriction and the relabelling is not significant.

The Expansion Principle

We now consider an important principle relating the dynamic and static constructors. It is often convenient to model a system as a composition of agents, that is as an agent of the form $P_1|P_2|\ldots|P_n$, or as a restricted composition, an agent of the form $(P_1|P_2|\ldots|P_n)\backslash L$. Indeed given an agent it is often possible, using the Static Laws described above, to find an observationally congruent agent of the form

$$(P_1[f_1] \mid P_2[f_2] \mid \ldots\ P_n[f_n]) \setminus L$$

where $f_1, \ldots f_n$ are relabelling functions. An agent of one of the above forms can proceed in one of two ways: either one of the component agents may proceed independently of the others by performing some action (which in the latter two cases is not restricted), or two component agents may communicate resulting in a τ-action of the composite agent. The Expansion Principle expresses the behaviour of such an agent in terms of the behaviours of its components and the dynamic operators, action-prefixing and summation. We first state the Principle in its simplest form.

Law 7A

$$\begin{aligned} P_1|\ldots|P_n \;=\; & \Sigma\{a.(P_1|\ldots|P_i'|\ldots|P_n) \mid 1 \le i \le n,\ P_i \xrightarrow{a} P_i'\} \\ & + \Sigma\{\tau.(P_1|\ldots|P_i'|\ldots|P_j'|\ldots|P_n) \\ & \mid 1 \le i < j \le n, \text{ for some } l\ P_i \xrightarrow{l} P_i',\ P_j \xrightarrow{\bar{l}} P_j'\} \end{aligned}$$

This expresses that the agent $P_1|\ldots|P_n$ is congruent to the agent on the right hand side which represents the behaviour described above, the first set of summands resulting from independent progress of one component and the second set from communications between pairs of components.

In its most general form the Expansion Principle is slightly more complicated. Here we have an agent of the form $(P_1[f_1]|\ldots|P_n[f_n])\backslash L$ and we must be careful that the relabelled actions correspond in any communication and that restricted events do not occur. These factors are taken into account in the general principle which is as follows:

Law 7B (The Expansion Principle)

$$\begin{aligned} (P_1[f_1]|\ldots|P_n[f_n])\backslash L \;=\; & \Sigma\{f_i(a).(P_1[f_1]|\ldots|P_i'[f_i]|\ldots|P_n[f_n])\backslash L \\ & \mid 1 \le i \le n,\ P_i \xrightarrow{a} P_i',\ f_i(a) \notin L \cup \bar{L}\} \\ & + \Sigma\{\tau.(P_1[f_1]|\ldots|P_i'[f_i]|\ldots|P_j'[f_j]|\ldots|P_n[f_n])\backslash L \\ & \mid 1 \le i < j \le n, \text{ for some } l,\ k\ P_i \xrightarrow{l} P_i',\ P_j \xrightarrow{k} P_j',\ \overline{f_i(l)} = f_j(k)\} \end{aligned}$$

This completes our discussion of the Expansion Principle. Finally we consider laws useful in reasoning about recursively defined agents.

Recursion Laws

The first law assures us that in making a, possibly recursive, definition we obtain an agent congruent to its defining expression.

Law 8

$$\text{If } A \stackrel{def}{=} P \text{ then } A = P.$$

Now we consider the problem of finding laws to facilitate analysis of recursively defined agents. Consider the two agents

$$A \stackrel{def}{=} l.A$$
$$B \stackrel{def}{=} l.l.B$$

where l is some label. It seems clear that A and B have the same behaviour, namely that of an agent which may perform any number of l actions, and so we should expect that they be observationally congruent, i.e. that A=B. But how can we prove this? To help in this task we introduce a set $\mathcal{V}$ of *agent variables* and allow that we may form complex expressions involving these variables using the operators of the calculus. Thus if X and Y are agent variables then $l.X$ and $X \mid a.Y$ are examples of such expressions. We may think of them as "incomplete agents": by substituting agents for each of the variables in an expression we obtain an agent. Thus, for example, substituting $b.Nil$ and $c.Nil + d.Nil$ for X and Y respectively in the expression $X|a.Y$ above, we obtain the agent $b.Nil|a.(c.Nil + d.Nil)$.

Now suppose that we had an *equation* of the form $X = E$ where X is an agent variable and E an agent expression containing no agent variable other than X, and suppose further that we were able, somehow, to establish that:
(1) whenever P and Q are *solutions* of the equation $X = E$, in the sense that $P = E\{P/X\}$ where $E\{P/X\}$ is the agent obtained by substituting P for X in E (and similarly for Q), then $P = Q$; and,
(2) A and B are both solutions of the equation $X = E$.
Then we could conclude that $A = B$.

Showing that an agent is a solution of an equation of the form $X = E$ is a task to which the preceding algebraic laws are well suited, so we have a means of establishing assertions such as (2). But what of (1)? How can we establish that a given equation has the property that any two of its solutions are observationally congruent? It turns out to be the case, although it is quite difficult to prove, that we can give fairly general conditions under which we can establish this property of an equation. We should note that not all equations have this property: the equation $X = X$, for example, has every agent as a solution. To see what are the general conditions under which we can establish uniqueness, up to observational congruence, of solutions of equations, we return to the agent A defined above: $A \stackrel{def}{=} l.A$. By repeatedly substituting the defining expression $l.A$ of A for A in the right hand side, and using Law 8 to see that $A = l.A$, we see that

$$\begin{aligned} A &= l.A \\ &= l.l.A \\ &= l.l.l.A \\ &= \cdots \end{aligned}$$

The fact that the occurrence of A in the defining agent is "guarded" by the label l allows us to "unwind" the recursion to exhibit the behaviour of A to any finite depth. This "guardedness" will be one of the "general conditions". To be more precise, returning to a general equation $X = E$, we say that X is *guarded* in E if every occurrence of X in E is in some subexpression of E of the form $l.F$ for some label l and expression F. Thus, for example, X is guarded in $l.X + Y$ and in $Nil \mid \tau.l.Nil$, but not in $l.X + X$ or in $\tau.X \mid Y + l.X$.

Unfortunately guardedness is not in itself sufficient to guarantee uniqueness up to $=$ of solutions of an equation. To see this consider the equation

$$X = (l.X \mid \bar{l}.Nil) \setminus l.$$

X is certainly guarded in the expression on the right hand side, but because of the special status of τ-actions, using some of the algebraic laws we can show that if P is any agent whose sort contains neither l nor $\bar{l}$, then $\tau.P$ is a solution of the equation, i.e. that $\tau.P = (l.\tau.P \mid \bar{l}.Nil) \setminus l$. Thus not all solutions of the equation are observationally congruent.

To ensure uniqueness up to $=$ of solutions, we require that in addition to guardedness, an equation have a further property. Thus we say that X is *sequential* in the expression E if no occurrence of X in E lies "within" a static constructor. Thus, for example, X is sequential in $X + a.Nil$ and in $a.X + (Nil \mid b.Y)$, but not in $a.X \mid b.Nil$ or in $a.X + (a.X \mid Y) \setminus b$. It turns out that the conditions of guardedness and sequentiality together are sufficient to guarantee uniqueness up to $=$ of solutions of an equation.

Law 9A

Suppose that $X = E$ is an equation in the agent variable X, that X is guarded in E and that X is sequential in E. Then any two solutions of the equation are observationally congruent, i.e. if $P = E\{P/X\}$ and $Q = E\{Q/X\}$, then $P = Q$.

We may employ Law 9A to establish that the agents A and B introduced earlier are obervationally congruent. Recall that $A \stackrel{def}{=} l.A$ and $B \stackrel{def}{=} l.l.B$. Consider the equation $X = l.l.X$. It is clear that X is both guarded and sequential in $l.l.X$, and thus from Law 9A we know that if P and Q are solutions of the equation then $P = Q$. But from Law 8 we know that $B = l.l.B$, i.e. that B is a solution of the equation. Also from Law 8 we know that $A = l.A$ from which it follows by the substitutivity of $=$ that $A = l.l.A$, i.e. that A is a solution of the equation. Hence $A = B$. Notice that if we had considered instead the equation $X = l.X$ then although we could see that the equation has a unique solution up to $=$ and that A is a solution, it is not clear how we could prove that B is a solution.

In general we may have several mutually recursive definitions of agents, and to enable us to reason about such agents we extend Law 9A a little to the following law.

Law 9B

Suppose that I is an index set, $\{X_i \mid i \in I\}$ a set of agent variables and $\{E_i \mid i \in I\}$ a set of agent expressions, no member of which contains any variable not in $\{X_i \mid i \in I\}$. Suppose that each of the variables X_i is guarded and sequential in each of the expressions E_j. Suppose further that $\{P_i \mid i \in I\}$ and $\{Q_i \mid i \in I\}$ are sets of agents such that for

each $i \in I$ both P_i and Q_i are solutions of the equation $X_i = E_i$. Then for each $i \in I$, $P_i = Q_i$.

We conclude this section with an example which illustrates the use of Law 9B and of some of the other laws. Recall from the beginning of section 3 the specification of a two-place buffer. Below we rewrite this specification on the assumption that the buffer may hold only the values 0 and 1. Notice also that we represent the input of 0 and 1 by distinct actions, in_0 and in_1, and similarly for the other communications.

$$\begin{aligned} Buff &\stackrel{def}{=} in_0.B_0 + in_1.B_1 \\ B_i &\stackrel{def}{=} in_0.B_{i0} + in_1.B_{i1} + \overline{out_i}.Buff \\ B_{ij} &\stackrel{def}{=} \overline{out_i}.B_j. \end{aligned}$$

for $i,j = 0$ or 1.

Recall also the agent C defined at the beginning of section 3. We modify its definition in a similar way to obtain:

$$\begin{aligned} A &\stackrel{def}{=} in_0.A_0 + in_1.A_1 \\ A_i &\stackrel{def}{=} \overline{mid_i}.A \\ A' &\stackrel{def}{=} mid_0.A'_0 + mid_1.A'_1 \\ A'_i &\stackrel{def}{=} \overline{out_i}.A' \\ C &\stackrel{def}{=} (A \mid A') \setminus \{mid_0, mid_1\} \end{aligned}$$

for $i,j = 0$ or 1.

We will prove that $Buff = C$. Consider the equations

$$\begin{aligned} X_1 &= in_0.X_2 + in_1.X_3 \\ X_2 &= in_0.X_4 + in_1.X_5 + \overline{out_0}.X_1 \\ X_3 &= in_0.X_6 + in_1.X_7 + \overline{out_1}.X_1 \\ X_4 &= \overline{out_0}.X_2 \\ X_5 &= \overline{out_0}.X_3 \\ X_6 &= \overline{out_1}.X_2 \\ X_7 &= \overline{out_1}.X_3. \end{aligned}$$

It is clear that each of the variables $X_1 \ldots X_7$ is guarded and sequential in each of the expressions $E_1 \ldots E_7$ on the right hand sides of the equations. Thus from Law 9B we know that the equations have a set of solutions unique up to pointwise observational congruence. From Law 8 it follows that $Buff$, B_0, B_1, B_{00}, B_{01}, B_{10} and B_{11} are solutions of the equations $X_i = E_i$ for i=0, ..., 7 respectively.
Using the Expansion Principle twice and writing L for $\{mid_0, mid_1\}$ we see that

$$\begin{aligned} C &= in_0.(A_0|A')\setminus L + in_1.(A_1|A')\setminus L \\ &= in_0.\tau.(A|A'_0)\setminus L + in_1.\tau.(A|A'_1)\setminus L. \end{aligned}$$

Using the Expansion Principle again we see that for $i = 0$ or 1

$$(A|A'_i)\backslash L \;=\; in_0.(A_0|A'_i)\backslash L \;+\; in_1.(A_1|A'_i)\backslash L \;+\; \overline{out_i}.C$$

and using the principle yet again that for $i, j = 0$ or 1

$$\begin{aligned}(A_i|A'_j)\backslash L &= \overline{out_j}.(A_i|A')\backslash L \\ &= \overline{out_j}.\tau.(A|A'_i)\backslash L.\end{aligned}$$

From the equations using Law 2(a) we see that

$$C \;=\; in_0.(A|A'_0)\backslash L \;+\; in_1.(A|A'_1)\backslash L$$

and for $i, j = 0$ or 1

$$(A_i|A'_j)\backslash L \;=\; \overline{out_j}.(A|A'_i)\backslash L.$$

Hence C, $(A|A'_0)\backslash L$, $(A|A'_1)\backslash L$, $(A_0|A'_0)\backslash L$, $(A_1|A'_0)\backslash L$, $(A_0|A'_1)\backslash L$ and $(A_1|A'_1)\backslash L$ are solutions of the equations $X_i = E_i$ for $i = 1, \ldots, 7$ respectively. Thus from Law 9B we conclude that $Buff = C$ (and that a further six congruences hold).

Exercises 4

Exercise 1. (a) Let

$$\begin{aligned} P &\stackrel{def}{=} a.P + \tau.b.Nil \\ Q &\stackrel{def}{=} a.Q + c.Nil \\ R &\stackrel{def}{=} \overline{c}.b.Nil \\ S &\stackrel{def}{=} (Q \mid R) \setminus c. \end{aligned}$$

Use some of the laws to prove that $P = S$.

(b) Let

$$\begin{aligned} T &\stackrel{def}{=} \overline{a}.d.T \\ U &\stackrel{def}{=} (T \mid S) \setminus a. \end{aligned}$$

Using the Expansion Principle, the Recursion Laws and the fact that if P is an agent such that there are no e and Q with $P \xrightarrow{e} Q$ then $P = Nil$, show that $U = V$ where

$$V \stackrel{def}{=} \tau.(d.V + \tau.(d.b.Nil + b.d.Nil)) + \tau.b.Nil$$

Exercise 2. (a) Let

$$\begin{aligned} A &\stackrel{def}{=} a.\overline{b}.c.A \\ B &\stackrel{def}{=} b.d.\overline{e}.B \\ C &\stackrel{def}{=} e.f.\overline{c}.C \\ D &\stackrel{def}{=} (A \mid B \mid C) \setminus \{b, c, e\}. \end{aligned}$$

Use the Expansion Principle, the Recursion Laws and some of the Dynamic Laws to show that $D = E$ where

$$E \stackrel{def}{=} a.d.f.E.$$

(b) Let

$$\begin{aligned} F &\stackrel{def}{=} D[a'/a, f'/f, \overline{d}/d] \\ G &\stackrel{def}{=} (D \mid F) \setminus d. \end{aligned}$$

Show that $G = H$ where

$$
\begin{aligned}
H &\stackrel{def}{=} a.a'.H1 + a'.a.H1 \\
H1 &\stackrel{def}{=} \tau(f.H2 + f'.H3) \\
H2 &\stackrel{def}{=} f'.H + a.f'.a'.H1 \\
H3 &\stackrel{def}{=} f.H + a'.f.a.H1.
\end{aligned}
$$

Suppose $H1$ is modified by omitting the τ-action. Is G equivalent to the modified H?

5. The bisimulation proof technique

In this section we introduce a technique for determining whether or not two given agents are observationally equivalent. In conjunction with the laws introduced in section 4, this technique is a powerful tool with which to analyse, and establish properties of, agents.

We begin by recalling from section 3 that observational equivalence, $\approx$, is the largest relation such that:

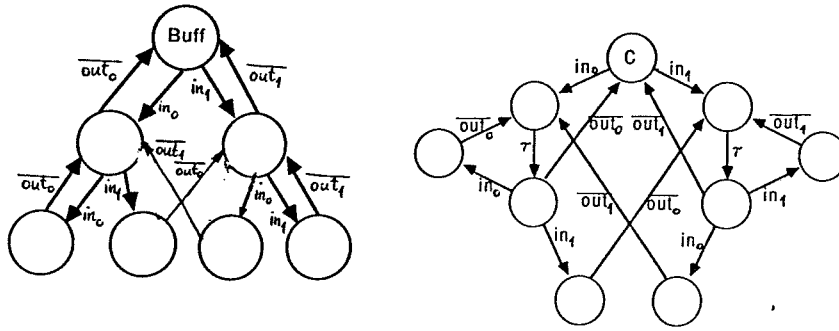
(***) $P \approx Q$ if and only if for each action a,

(a) if P' is such that $P \xrightarrow{a} P'$ then either (1) there is Q' such that $Q \stackrel{a}{\Longrightarrow} Q'$ and $P' \approx Q'$, or (2) $a = \tau$ and $P' \approx Q$, and conversely,

(b) if Q' is such that $Q \xrightarrow{a} Q'$ then either (1) there is P' such that $P \stackrel{a}{\Longrightarrow} P'$ and $P' \approx Q'$, or (2) $a = \tau$ and $P \approx Q'$.

Now recall once again the agents *Buff* and C first introduced at the beginning of section 3. At the end of section 4, using some of the laws previously introduced, we proved that $Buff = C$. We will introduce the proof technique for observational equivalence by using it to establish that $Buff \approx C$. We will later see how we may infer from this fact that $Buff = C$.

From the derivation trees of *Buff* and C we may obtain the following transition diagrams:



Here we use the notation introduced in the proof that $Buff = C$ at the end of section 4. Looking at the property (***) of $\approx$, we see that to establish that $Buff \approx C$ it is necessary to set up a suitable correspondence between the states in the respective transition diagrams, with the corresponding states being related by $\approx$. We may build such a correspondence by starting with the pair $(Buff, C)$, seeing which pairs must be in such a correspondence, given that $(Buff, C)$ is such a pair, if it is to be suitable, and repeating this procedure with each of the new pairs so introduced in turn for "as long as necessary". We may fairly easily see that we should expect the agents in each pair in the following correspondence $\mathcal{R}$

to be observationally equivalent:

$$\begin{aligned}\mathcal{R} \;=\; \{&(\mathit{Buff},\ C),\ (B_0,\ (A_0|A')\backslash L),\ (B_0,\ (A|A_0')\backslash L),\\ &(B_1,\ (A_1|A')\backslash L),\ (B_1,\ (A|A_1')\backslash L),\ (B_{00},\ (A_0|A_0')\backslash L),\\ &(B_{01},\ (A_1|A_0')\backslash L),\ (B_{10},\ (A_0|A_1')\backslash L),\ (B_{11},\ (A_1|A_1')\backslash L)\ \}.\end{aligned}$$

This correspondence, consisting of just nine pairs of agents, seems to form a small "self-contained piece" of the total relation $\approx$, in the sense that:
if $(P,\ Q)$ is any pair in $\mathcal{R}$ then for each action a,
(a) if P' is such that $P \xrightarrow{a} P'$ then either (1) there is Q' such that $Q \stackrel{a}{\Longrightarrow} Q'$ and $(P',\ Q')$ is in $\mathcal{R}$, or (2) $a{=}\tau$ and $(P',\ Q)$ is in $\mathcal{R}$, and conversely,
(b) if Q' is such that $Q \xrightarrow{a} Q'$ then either (1) there is P' such that $P \stackrel{a}{\Longrightarrow} P'$ and $(P',\ Q')$ is in $\mathcal{R}$, or (2) $a{=}\tau$ and $(P,\ Q')$ is in $\mathcal{R}$.

This may easily be checked by considering each pair in turn. $\mathcal{R}$ contains the essential information from which, given that $\approx$ is the largest relation satisfying (***), we may infer that $\mathit{Buff} \approx C$. To explain this remark we define a special type of relation between agents as follows.

A relation $\mathcal{B}$ over agents is a *bisimulation* if and only if whenever $(P,\ Q)$ is in $\mathcal{B}$ then for each action a,
(a) if P' is such that $P \xrightarrow{a} P'$ then either (1) there is Q' such that $Q \stackrel{a}{\Longrightarrow} Q'$ and $(P',\ Q')$ is in $\mathcal{B}$, or (2) $a = \tau$ and $(P',\ Q)$ is in $\mathcal{B}$, and conversely,
(b) if Q' is such that $Q \xrightarrow{a} Q'$ then either (1) there is P' such that $P \stackrel{a}{\Longrightarrow} P'$ and $(P',\ Q')$ is in $\mathcal{B}$, or (2) $a = \tau$ and $(P',\ Q)$ is in $\mathcal{B}$.

Notice that a relation $\mathcal{B}$ is a bisimulation precisely when it satisfies one half, the left-to-right implication, of (***). In checking that the relation $\mathcal{R}$ forms a "self-contained piece" of the relation $\approx$, we have in fact established precisely that $\mathcal{R}$ is a bisimulation. Now we may explain the remark above that having constructed $\mathcal{R}$ and shown that it has a certain property, namely that it is a bisimulation, we may conclude that $\mathit{Buff} \approx C$. For it is in fact the case that two agents are observationally equivalent precisely when they are related by some bisimulation, that is $P \approx Q$ if and only if there is a bisimulation $\mathcal{B}$ with $(P,\ Q)$ in $\mathcal{B}$. Thus given that $\mathcal{R}$ is a bisimulation and that $(\mathit{Buff},\ C)$ is in $\mathcal{R}$, it follows that $\mathit{Buff} \approx C$. Notice that since $\approx$ satisfies the left-to-right implication of (***) it is itself a bisimulation; indeed it is the largest bisimulation. However $\approx$ is further distinguished amongst all bisimulations in that it satisfies the right-to-left implication of (***).

Having established that $\mathit{Buff} \approx C$ let us see how we may deduce that $\mathit{Buff} = C$. The fact that $P \approx Q$ holds of agents P and Q does not necessarily imply that $P = Q$, for as we have seen the possibility of initial τ-actions may pre-empt certain behaviours. However neither *Buff* nor C may initially perform a τ-action: their first actions are visible events. It is worthwhile to introduce a term to descibe this quality, so we say that an agent P is *stable* if it can not initially perform a τ-action, that is if there is no agent Q such that $P \xrightarrow{\tau} Q$. Thus *Buff* and C are stable agents.

Stable agents have a very important property which we may state in the following principle:

The Stability Principle

If P and Q are stable agents and $P \approx Q$, then $P = Q$.

Thus in order to establish that two stable agents P and Q are observationally congruent, it is enough to prove that they are observationally equivalent. Using the bisimulation technique we have shown that $Buff \approx C$, and so it follows from the Stability Principle, together with the earlier observation tha $Buff$ and C are stable agents, that $Buff = C$.

Notice that the Stability Principle shows us that observational congruence and observational equivalence differ only when agents capable initially of performing silent actions are considered. Indeed we can give the following explicit characterization of observational congruence:

$P = Q$ if and only if for each action a,

(a) if P' is such that $P \xrightarrow{a} P'$ then there is Q' such that $Q \stackrel{a}{\Longrightarrow} Q'$ and $P' \approx Q'$, and conversely,

(b) if Q' is such that $Q \xrightarrow{a} Q'$ then there is P' such that $P \stackrel{a}{\Longrightarrow} P'$ and $P' \approx Q'$.

In the above example the construction of the bisimulation $\mathcal{R}$ is fairly straightforward as the number of states which must be taken into account is small. Applying this technique to a larger system may become a much more complicated procedure. However the search for a bisimulation containing a given pair of agents can be mechanised. Indeed there is an algorithm which given any two "finite-state" agents, that is agents in whose derivation trees only finitely many distinct agents appear, will determine whether or not the agents are observationally equivalent and, if they are, construct a bisimulation relating them.

We conclude this section by illustrating a refinement of the bisimulation technique which often reduces, sometimes substantially, the amount of work involved in establishing the observational equivalence of two agents. Consider the following agents:

$$\begin{aligned} Sem0 &\stackrel{def}{=} get.Sem1 \\ Sem1 &\stackrel{def}{=} get.Sem2 + put.Sem0 \\ Sem2 &\stackrel{def}{=} get.Sem3 + put.Sem1 \\ Sem3 &\stackrel{def}{=} put.Sem2. \end{aligned}$$

$Sem0$ describes the behaviour of a ternary semaphore which will allow at most three processes to be "active" at one time: a process may become "active" after performing a $\overline{get}$ action and cease to be "active" by performing a $\overline{put}$ action. Now consider the agents:

$$\begin{aligned} S &\stackrel{def}{=} get.S' \\ S' &\stackrel{def}{=} put.S \\ T &\stackrel{def}{=} S|S|S. \end{aligned}$$

We prove that $Sem0 \approx T$. To do so it is sufficient to construct a bisimulation containing the pair $(Sem0,\ T)$. This is fairly straightforward: it is easy to verify that the relation $\mathcal{R}$ below is a bisimulation.

$$\begin{aligned} \mathcal{R} = \{&(Sem0,\ T),\ (Sem1,\ S'|S|S),\ (Sem1,\ S|S'|S), \\ &(Sem1,\ S|S|S'),\ (Sem2,\ S'|S'|S),\ (Sem2,\ S'|S|S'), \\ &(Sem2,\ S|S'|S'),\ (Sem3,\ S'|S'|S')\ \}. \end{aligned}$$

Using some of the laws concerning the composition operator we can show that

$$S'|S|S \approx S|S'|S \approx S|S|S'$$

and

$$S'|S'|S \approx S'|S|S' \approx S|S'|S'.$$

(Recall that for any agents P and Q, $P = Q$ implies $P \approx Q$.) So we may feel that in constructing the bisimulation $\mathcal{R}$, and in the process pairing *Sem1* with $S' \mid S \mid S$, $S \mid S' \mid S$ and $S \mid S \mid S'$, and *Sem2* with $S'|S'|S$, $S'|S|S'$ and $S|S'|S'$, we may be doing more work than necessary to establish that *Sem0* $\approx T$. We cannot remove from $\mathcal{R}$ any of these pairs and retain the property of being a bisimulation. However the implementation of such a procedure produces a relation which is sufficiently close to being a bisimulation to enable us to conclude that *Sem0* $\approx T$. Consider the relation $\mathcal{R}'$ defined by:

$$\mathcal{R}' = \{(Sem0,\ T),\ (Sem1,\ S'|S|S),\ (Sem2,\ S'|S'|S),\ (Sem3,\ S'|S'|S')\ \}$$

$\mathcal{R}'$ has the property that: whenever (P, Q) is in $\mathcal{R}'$ then for each action a,
(a) if P' is such that $P \xrightarrow{a} P'$ then either (1) there are P'', Q' and Q'' such that $Q \stackrel{a}{\Longrightarrow} Q'$, $P' \approx P''$, $Q' \approx Q''$ and (P'',Q'') is in $\mathcal{R}'$, or (2) $a = \tau$ and (P', Q) is in $\mathcal{R}'$, and conversely,
(b) if Q' is such that $Q \xrightarrow{a} Q'$ then either (1) there are Q'', P' and P'' such that $P \stackrel{a}{\Longrightarrow} P'$, $Q' \approx Q''$, $P' \approx P''$ and (P'', Q'') is in $\mathcal{R}'$, or (2) $a{=}\tau$ and (P, Q') is in $\mathcal{R}'$.

We call any relation satisfying these conditions a *bisimulation up to* $\approx$. If $\mathcal{B}$ is such a relation and P and Q are agents with (P, Q) in $\mathcal{B}$, then for each action a, if P' is such that $P \xrightarrow{a} P'$ then, neglecting for the moment the extra possibility which obtains if $a = \tau$, we know *not* that there is Q' such that $Q \stackrel{a}{\Longrightarrow} Q'$ and (P', Q') is in $\mathcal{B}$, but rather that there are P'', Q' and Q'' such that $Q \stackrel{a}{\Longrightarrow} Q'$, $P' \approx P''$, $Q' \approx Q''$ and (P'', Q'') is in $\mathcal{B}$; and similarly with P and Q interchanged. This is the sense in which the relation $\mathcal{R}'$ above is "almost" a bisimulation. It is in fact the case that in order to establish that $P \approx Q$ holds of the agents P and Q, it is enough to construct a relation $\mathcal{B}$ containing the pair (P, Q) which is a bisimulation up to $\approx$. Thus to establish that *Sem0* $\approx T$ it is sufficient to show that the relation $\mathcal{R}'$ is a bisimulation up to $\approx$, a task which we may easily achieve using the laws concerning the composition operator described above. Notice that the relation $\mathcal{R}'$ is much smaller than the bisimulation $\mathcal{R}$. This difference may be of greater significance in dealing with larger systems and then the construction of a bisimulation up to $\approx$ containing a given pair of agents may involve substantially less work than that of a corresponding bisimulation.

Exercises 5

Exercise 1. (a) Let

$$\begin{aligned} Copy &\stackrel{def}{=} in(x).\overline{up}(x).\overline{down}(x).Copy \\ Addone &\stackrel{def}{=} up(x).\overline{out1}(x+1).Addone \\ Subone &\stackrel{def}{=} down(x).\overline{out2}(x-1).Subone \\ Mult &\stackrel{def}{=} out1(y).out2(z).\overline{out}(yz).Mult \\ Square &\stackrel{def}{=} in(x).\overline{down}(x^2).Square \\ Cell &\stackrel{def}{=} out2(x).\overline{out}(x).Cell \end{aligned}$$

Use the bisimulation proof technique to prove that $SS \approx CASM$ where

$$\begin{aligned} SS &\stackrel{def}{=} (Square \mid Subone \mid Cell) \setminus \{down,\ out2\} \\ CASM &\stackrel{def}{=} (Copy \mid Addone \mid Subone \mid Mult) \setminus \{up,\ down,\ out1,\ out2\}. \end{aligned}$$

(b) Is it true that $SS \approx A$ where

$$A \stackrel{def}{=} in(x).\overline{out}(x^2-1).A.$$

If so find a bisimulation to establish this. If not show that no such bisimulation exists and modify the definition of A to obtain an agent observationally equivalent to SS. In this case construct a bisimulation to establish the observational equivalence of SS and the modification of A.

Exercise 2. Consider the following agents:

$$\begin{aligned} A &\stackrel{def}{=} in(x).\overline{send}(x).A1 \\ A1 &\stackrel{def}{=} in(x).\overline{send}(x).A2 + rec(y).\overline{out}(y).A \\ A2 &\stackrel{def}{=} rec(y).\overline{out}(y).A1 \\ B &\stackrel{def}{=} send(z).\overline{down}(z).up(w).\overline{rec}(w).B \\ C &\stackrel{def}{=} down(v).\overline{up}(hv).C \\ D &\stackrel{def}{=} (A \mid B \mid B \mid C) \setminus \{send,\ rec,\ down,\ up\} \end{aligned}$$

where h is some function defined on the set of values communicable at ports $down$ and up, and

$$\begin{aligned} E &\stackrel{def}{=} in(x).E1(x) \\ E1(x) &\stackrel{def}{=} in(y).E2(x,y) + \overline{out}(hx).E \\ E2(x,y) &\stackrel{def}{=} \overline{out}(hx).E1(y) + \overline{out}(hy).E1(x) \end{aligned}$$

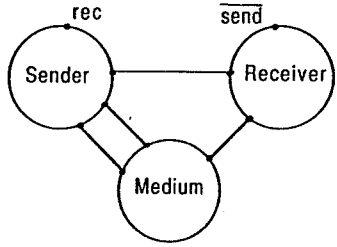
Investigate whether or not it is the case that $D \approx E$.

6. Divergence

To introduce the subject of this short, final section consider the following agents:

$$\begin{aligned}
Sender &\stackrel{def}{=} rec(x).\overline{sm}(x).Sender'(x) \\
Sender'(x) &\stackrel{def}{=} ms.\overline{sm}(x).Sender'(x) + rs.Sender \\
Medium &\stackrel{def}{=} sm(x).Medium'(x) \\
Medium'(x) &\stackrel{def}{=} \overline{mr}(x).Medium + \tau.\overline{ms}.Medium \\
Receiver &\stackrel{def}{=} mr(x).\overline{send}(x).\overline{rs}.Receiver \\
Protocol &\stackrel{def}{=} (Sender|Medium|Receiver)\backslash\{sm,\ ms,\ mr,\ rs\}.
\end{aligned}$$

This example is drawn from [P]. The agent *Protocol*, which may be represented by the picture below, is intended to provide a model of the behaviour of an extremely simple communications protocol, taking into account the possibility that a message may be lost during transmission.



Sender transmits to *Medium* any message which it receives at port *rec*. After receiving a message from *Sender*, *Medium* may either transmit the message to *Receiver*, or lose the message, an event whose occurrence we model as a τ-action, in which case it sends a "timeout" signal to *Sender* and the message is retransmitted. On receiving a message, *Receiver* may transmit it at port $\overline{send}$ and then send an acknowledgment signal directly to *Sender*. (We assume that there is no possibility of such an acknowledgment signal being lost.) After receiving such an acknowledgment, *Sender* may again accept a message at port *rec*.

If we define an agent *Buffer* by

$$Buffer \stackrel{def}{=} rec(x).\overline{send}(x).Buffer,$$

we may regard *Buffer* as providing a very high level description, or *service specification*, of the behaviour of a communications protocol. If we do so we may then ask whether *Protocol* satisfies the specification, that is, is it the case that $Buffer = Protocol$? Using the bisimulation proof technique of section 5, we establish below that *Buffer* and *Protocol* are observationally equivalent, i.e. that $Buffer \approx Protocol$. We may then observe that both

Buffer and *Protocol* are stable agents and thus appeal to the Stability Principle to deduce that *Buffer* = *Protocol*.

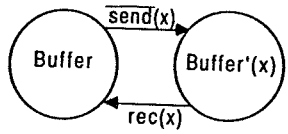
We prove that *Protocol* $\approx$ *Buffer* by constructing a bisimulation containing the pair (*Protocol*, *Buffer*). Suppose that D is the set of values which may be communicated at the ports *rec* and $\overline{send}$. If D is an infinite set then the derivation trees of *Protocol* and *Buffer* will contain infinitely many distinct agents. For example if $D = \mathrm{N}$, the set of nonnegative integers, then, introducing for each nonnegative integer i actions rec_i and $\overline{send}_i$ to represent the reception of i at port *rec* and the transmission of i at port $\overline{send}$ respectively, from the transition rule for action-prefixing we see that for each $i \in \mathrm{N}$

$$Buffer \xrightarrow{rec_i} Buffer'\{i/x\}$$

and

$$Buffer'\{i/x\} \xrightarrow{\overline{send}_i} Buffer.$$

However, the fact that the values are treated in a uniform way enables us to construct a finite transition graph representing the behaviour of *Buffer*:



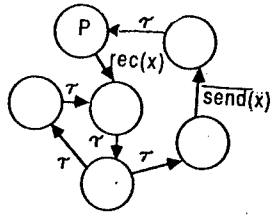
Similar remarks apply also for *Protocol*, and using the transition rules, or alternatively the Expansion Principle, we can construct a finite transition diagram representing the behaviour of *Protocol*. For if we let

$$\begin{aligned}
P &\stackrel{def}{=} Protocol \\
Q(x) &\stackrel{def}{=} (\overline{sm}(x).Sender'(x)|Medium|Receiver)\backslash L \\
R(x) &\stackrel{def}{=} (Sender'(x)|Medium'(x)|Receiver)\backslash L \\
S(x) &\stackrel{def}{=} (Sender'(x)|Medium|\overline{send}(x).\overline{rs}.Receiver)\backslash L \\
T(x) &\stackrel{def}{=} (Sender'(x)|\overline{ms}.Medium|Receiver)\backslash L \\
U(x) &\stackrel{def}{=} (Sender'(x)|Medium|\overline{rs}.Receiver)\backslash L
\end{aligned}$$

where $L = \{sm,\ ms,\ mr,\ rs\}$, then repeated use of the Expansion Principle yields

$$\begin{aligned}
P &= rec(x).Q(x) \\
&= rec(x).\tau.R(x) \\
&= rec(x).\tau.(\tau.S(x) + \tau.T(x)) \\
&= rec(x).\tau.(\tau.\overline{send}(x).U(x) + \tau.\tau.Q(x)) \\
&= rec(x).\tau.(\tau.\overline{send}(x).\tau.P + \tau.\tau.Q(x))
\end{aligned}$$

and thus we obtain the following transition diagram:



We can now construct the bisimulation. If D is infinite then the bisimulation will be infinite. However the uniform treatment of values enables us to give a finite presentation of it. Let $\mathcal{R}$ be the following relation (where we abbreviate *Buffer* by B):

$$\{(P, B), (U(v), B), (Q(v), B'(v)), (R(v), B'(v)), (S(v), B'(v)), (T(v), B'(v))\}$$

where v ranges over D.

That $\mathcal{R}$ is a bisimulation may be checked from the definition by considering each pair in turn. Consider, for example, $(U(v), B)$. Now $U(v) \xrightarrow{\tau} P$ so we must find an agent A reachable from B by zero or more τ-actions such that (P, A) is in $\mathcal{R}$. That agent is B. Also for $w \in D$, $B \xrightarrow{rec_w} B'(w)$ so we must find an agent V such that $U(v) \stackrel{rec_w}{\Longrightarrow} V$ and $(B'(v), V)$ is in $\mathcal{R}$. That agent is $Q(w)$, since $U(v) \xrightarrow{\tau} P$ and $P \xrightarrow{rec_w} Q(w)$. The arguments for the other pairs are similar. Note that for $v \in D$, each of the agents $Q(v)$, $R(v)$ and $S(v)$ in the "τ-loop" is related by $\mathcal{R}$ to $B'(v)$, and that the transition $B'(v \xrightarrow{\overline{send_v}} B$ can be "matched" by $Q(v) \stackrel{\overline{send_v}}{\Longrightarrow} U(v)$ and similarly for the other two agents.

In earlier sections we have seen that observational congruence provides a very reasonable formalization of the idea of what it is for two agents "to have the same behaviour", and that in addition it enjoys many natural properties and has associated with it a manageable proof technique. However it is possible to argue that there is a feature of the behaviour of the agent *Protocol* which may lead us to consider that *Buffer* and *Protocol* do *not* "have the same behaviour". For it is possible that a message transmitted from *Sender* to *Medium* may be lost, that the retransmitted message may be lost, that the re-retransmitted message may be lost, and so on. In the transition diagram for *Protocol* this is represented by the τ-loop. Thus it is possible for *Protocol* to accept a message at $\overline{rec}$ and never again engage in any communication with its environment. The possibility that an agent P may engage in an endless sequence of silent internal communications and never again communicate with its environment, is expressed by saying that P is *divergent*. Thus although *Protocol* is not itself a divergent agent (since the only action it may engage in is an input at port *rec*), from *Protocol* it is possible to reach a divergent agent. From *Buffer*, however, it is not possible to reach a divergent state. Thus we see that in using observational congruence as our formalization of the notion of "having the same behaviour", we neglect the possibility of divergence, at least to the extent that it may lead us to regard as congruent two agents, exactly one of which is divergent.

It is possible to modify the definition of observational equivalence to preclude the possibility that a pair of agents exactly one of which is divergent be regarded as "equivalent". However the amended equivalence, $\approx'$, has the property that if $Div \stackrel{def}{=} \tau.Div$ and P is any agent which is not divergent, then $P|Div \not\approx' P$, that is P is *not* "equivalent" to the composition of P with an infinite τ-cycle, and this seems to be an undesirable property to hold in general.

It should be reiterated at this point that there do exist other notions of "equivalence" (and notions of "approximation") of agents, and that their treatments of the phenomenon of divergence vary. For example, in some cases divergence is regarded as *catastrophic*, and the behaviour of a system containing a divergent agent considered to be wholly unpredictable. In defence of the present definition of observational equivalence and observational congruence, it may be argued (a) that the theory is much simpler than one in which divergence is treated explicitly; (b) that in a theory in which two agents may be considered "equivalent" even when the proportion of their relative rates of progress may vary without bound, it is natural to admit the possibility that this proportion become infinite; and (c) that the possibility that an agent may be divergent, and the implications of such a possibility, can often be investigated by separate arguments in which features of the system being modelled other than those captured by the notion of observational equivalence are taken into consideration. In general how one chooses to regard the possibility of divergence of an agent may depend on features of the system modelled by that agent not captured within the calculus.

Exercises 6

Exercise 1. Consider the following agents:

$$\begin{aligned} A &\stackrel{def}{=} \tau.A + a.Nil \\ B &\stackrel{def}{=} a.C \\ C &\stackrel{def}{=} \tau.C. \end{aligned}$$

Is either of $A \approx B$ or $A = B$ true? In each case justify your answer. Which of the agents A, B and C is (or are) divergent?

Exercise 2. Let

$$D \stackrel{def}{=} (a.D + b.D \,|\, \overline{a}.Nil) \setminus a.$$

Examine the behaviour of D. Is D divergent? Does D have a finite transition diagram?

Exercise 3. (a) Modify the defintion of *Protocol* in the text to obtain an agent from which no divergent state is reachable, by ensuring that if a particular message is lost by *Medium* on two consecutive occasions, then no further attempt is made by *Sender* to transmit that message. Instead *Sender* may accept a new message for transmission.

(b) Modify the definition of *Buffer* in the text to obtain a simple description of the behaviour of the modified version of *Protocol*.

(c) Prove that the modified version of *Buffer* is observationally congruent to the modified version of *Protocol*.

References

[M1] R. Milner, *A Calculus of Communicating Systems*, Springer-Verlag (1980) [out of print]

[M2] R. Milner, *Calculus for Communication and Concurrency*, handwritten notes (1986)

[P] J. Parrow, *Fairness Properties in Process Algebra*, Ph.D. thesis, Uppsala University, Uppsala, Sweden (1985)

Solutions to Exercises

Exercises 1.

Exercise 1. (a)

$$Copy \stackrel{def}{=} in(x).(\overline{out1}(x).\overline{out2}(x).Copy + \overline{out2}(x).\overline{out1}(x).Copy)$$

(b)

$$Split \stackrel{def}{=} in(x).\overline{out1}(x).in(x).\overline{out2}(x).Split$$

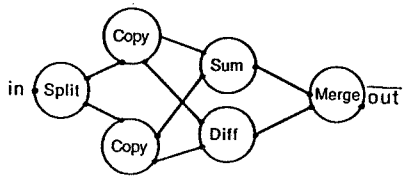
(c)

$$Merge \stackrel{def}{=} in1(x).\overline{out}(x).in2(y).\overline{out}(y).Merge$$

(d)

$$\begin{aligned} Sum &\stackrel{def}{=} in1(x).in2(y).\overline{out}(x+y).Sum + in2(y).in1(x).\overline{out}(x+y).Sum \\ Diff &\stackrel{def}{=} in1(x).in2(y).\overline{out}(x-y).Diff + in2(y).in1(x).\overline{out}(x-y).Diff \end{aligned}$$

(e)



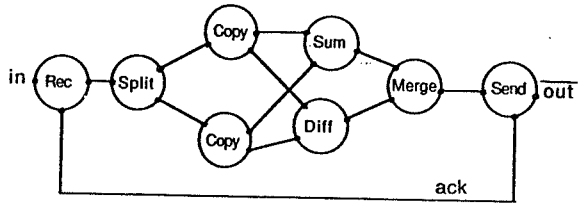
$$System \stackrel{def}{=} (Split' \mid Copy' \mid Copy'' \mid Sum' \mid Diff' \mid Merge') \setminus L$$

where

$$\begin{aligned} Split' &\stackrel{def}{=} Split[a/out1,\ b/out2] \\ Copy' &\stackrel{def}{=} Copy[a/in,\ c/out1,\ d/out2] \\ Copy'' &\stackrel{def}{=} Copy[b/in,\ e/out1,\ f/out2] \\ Sum' &\stackrel{def}{=} Sum[c/in1,\ e/in2,\ g/out] \\ Diff' &\stackrel{def}{=} Diff[d/in1,\ f/in2,\ h/out] \\ Merge' &\stackrel{def}{=} Merge[g/in1,\ h/in2] \end{aligned}$$

and $L = \{ a, b, c, d, e, f, g, h \}$.

(f) One possible solution is:



$$
\begin{array}{rcl}
Split'' & \stackrel{def}{=} & Split[rec/in] \\
Send & \stackrel{def}{=} & in(x).\overline{out}(x).\overline{ack}.Send \\
Rec & \stackrel{def}{=} & in(x).\overline{mid}(x).in(x).\overline{mid}(x).Rec' \\
Rec' & \stackrel{def}{=} & ack.in(x).\overline{mid}(x).in(x).\overline{mid}(x).Rec'
\end{array}
$$

Exercise 2. (a)

$$
\begin{array}{rcl}
Change & \stackrel{def}{=} & in(x).\ if\ x = 20\ then\ Change(20)\ else\ Change(10) \\
Change(n) & \stackrel{def}{=} & \overline{out}(10).Change(n-10) + \overline{out}(5).Change(n-5) \\
 & & + \overline{out}(2).Change(n-2) + \overline{out}(1).change(n-1)\ \ for\ 10 \leq n \leq 20 \\
Change(n) & \stackrel{def}{=} & \overline{out}(5).Change(n-5) + \overline{out}(2).Change(n-2) \\
 & & + \overline{out}(1).Change(n-1)\ \ for\ 5 \leq n < 10 \\
Change(n) & \stackrel{def}{=} & \overline{out}(2).Change(n-2) + \overline{out}(1).Change(n-1)\ \ for\ 2 \leq n < 5 \\
Change(1) & \stackrel{def}{=} & \overline{out}(1).Change(0) \\
Change(0) & \stackrel{def}{=} & Change.
\end{array}
$$

(b)

$$\begin{aligned} Source &\stackrel{def}{=} \overline{give}.Source \\ Sink &\stackrel{def}{=} take(x).Sink. \end{aligned}$$

(c)

$$\begin{aligned} Man &\stackrel{def}{=} give.\overline{in_{20}}.Hold(\langle\rangle) \\ Hold(s) &\stackrel{def}{=} \textit{if } \underline{sum}(s) \geq 20 \textit{ then } Send(s) \\ &\quad\ \textit{else } out(x).Hold(x \underline{\textit{ insert }} s) \\ Send(\langle v_1 \ldots v_{n-1}, v\rangle) &\stackrel{def}{=} \overline{take}(v).Send(\langle v_1 \ldots v_{n-1}\rangle) \\ Send(\langle\rangle) &\stackrel{def}{=} Man \end{aligned}$$

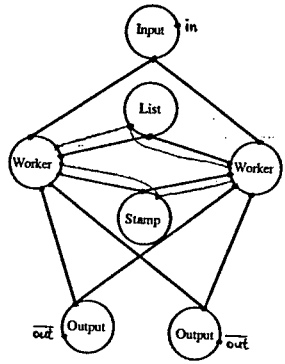
where $\underline{sum}$(s) is the sum of the values in the sequence s, x $\underline{\textit{insert}}$ s is the sequence obtained by inserting x in s according to the order $\leq$, and where if n=1 then $\langle v_1 \ldots v_{n-1}\rangle = \langle\rangle$.

(d)

$$\begin{aligned} Woman &\stackrel{def}{=} give.\overline{in_{20}}.Hold(\langle\rangle) \\ Hold(s) &\stackrel{def}{=} \textit{if } \underline{sum}(s) \geq 20 \textit{ then } Check(s) \\ &\quad\ \textit{else } out(x).Hold(x \underline{\textit{ insert }} s) \\ Check(s) &\stackrel{def}{=} \textit{if } s = \langle v_1 \ldots v_{n-1}, 10\rangle \textit{ then } \overline{in_{10}}.Hold(\langle v_1 \ldots v_{n-1}\rangle) \\ &\quad\ \textit{else } Send'(s) \\ Send'(\langle v, v_1, \ldots, v_{n-1}\rangle) &\stackrel{def}{=} \overline{take}(v).Send'(\langle v_1 \ldots v_{n-1}\rangle) \\ Send'(\langle\rangle) &\stackrel{def}{=} Woman \end{aligned}$$

(e) *System* may exhibit the behaviour described: if *Change* gives a 10 pence coin in change to *Woman* then *Woman* will repeatedly try to change it for coins of smaller value. But *Change* may give a single 10 pence coin in change for a 10 pence coin, so there is the possibility of infinite internal communication between *Change* and *Woman*. Notice that the environment can not observe this communication as the ports through which *Change* and *Woman* communicate are restricted.

Exercise 3. (a) The flow-graph is as follows:



(b) *Processor* may deadlock: if each *Worker* receives an item from *Input*, then each will attempt to acquire both *List* and *Stamp*. Since the acquisitions may occur in either order it is possible that one of the *Workers* acquires the *List* and the other the *Stamp*. The *Processor* in then deadlocked as neither *Worker* is capable of releasing the object it is holding until it has acquired the object held by the other *Worker*.
(c) One solution is to redefine $Worker'(x)$ as follows:

$$Worker'(x) \stackrel{def}{=} \overline{getlist}.\overline{getstamp}.\overline{receive}_{d(x)}(x).Worker''$$

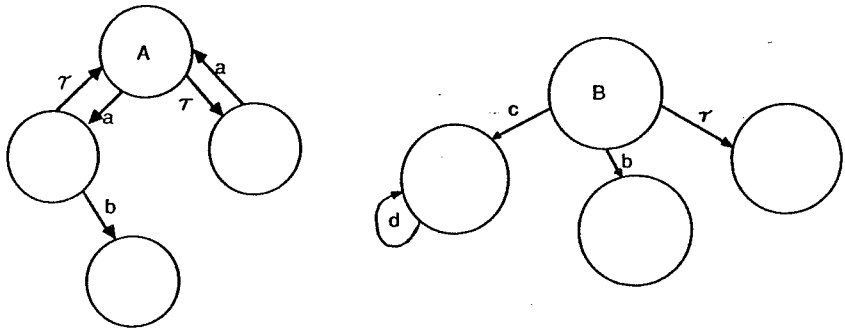
(d) P and the deadlock-free *Processor* do not exhibit the same behaviour since the order in which P outputs items is the same as that in which they are accepted, but this need not be so for *Processor*, since one item may "overtake" another during processing.

Exercises 2

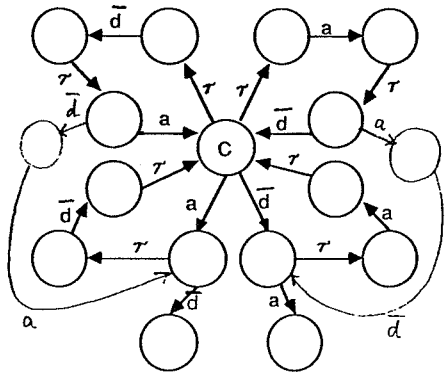
Exercise 1. The transitions are as follows:

$$\begin{aligned}
B &\xrightarrow{c} ((B \mid (\overline{b}.Nil + a.B))\,[f]) \setminus b \\
B &\xrightarrow{\overline{c}} (((\overline{a}.Nil + b.B) \mid Nil)\,[f]) \setminus b \\
B &\xrightarrow{\tau} ((B \mid Nil)\,[f]) \setminus b \\
B &\xrightarrow{\tau} ((Nil \mid B)\,[f]) \setminus b
\end{aligned}$$

Exercise 2.



Exercise 3.



$$
\begin{aligned}
D \;=\; & \tau.\overline{d}.\tau.(\overline{a}.D + \overline{d}.a.(\overline{d}.Nil + \tau.\overline{d}.\tau.D)) \\
& + \tau.a.\tau.(\overline{d}.D + a.\overline{d}.(\overline{d}.Nil + \tau.\overline{d}.\tau.D)) \\
& + a.(\overline{d}.Nil + \tau.\overline{d}.\tau.D) + \overline{d}.(a.Nil + \tau.\overline{d}.\tau.D)
\end{aligned}
$$

Exercises 3

Exercise 1.

$$\begin{aligned} A &\stackrel{def}{=} a.(\tau.\tau.A + b.Nil) + \tau.b.Nil \\ B &\stackrel{def}{=} a.(a.B + b.Nil) + b.Nil + \tau.b.Nil \\ C &\stackrel{def}{=} a.C + a.b.Nil + b.Nil \\ D &\stackrel{def}{=} a.\tau.D + a.b.Nil + \tau.b.Nil \\ E &\stackrel{def}{=} a.E' + \tau.b.Nil \\ E' &\stackrel{def}{=} a.E' + b.Nil + \tau.b.Nil \end{aligned}$$

The only equivalence is $D \approx E$.

Exercise 2. (a) $R \not\approx S$, since R may buffer up to two items whereas S may hold only one item.

(b) $T \not\approx W$. Later we will see how to prove this rigorously.

Exercises 4

Exercise 1. (a) Using the Expansion Principle and the facts that $(Nil \mid Nil) = Nil$ and $Nil \setminus c = Nil$, we see that

$$\begin{aligned} S &= a.(Q|R)\backslash c + \tau.(Nil|b.Nil)\backslash c \\ &= a.S + \tau.b.(Nil|Nil)\backslash c \\ &= a.S + \tau.b.Nil \end{aligned}$$

Now consider the equation $X = a.X + \tau.b.Nil$. This equation has a solution unique up to $=$ by Law 9A and both P and S are solutions, so $P = S$.

(b)

$$\begin{aligned} U &= \tau.(d.T|S)\backslash a + \tau.(T|b.Nil)\backslash a \\ &= \tau.(d.U + \tau.(d.T|b.Nil)\backslash a) + \tau.b.(T|Nil)\backslash a \\ &= \tau.(d.U + \tau.(d.b.(T|Nil)\backslash a + b.d.(T|Nil)\backslash a)) \\ &\quad + \tau.b.(T|Nil)\setminus a \end{aligned}$$

Also $(T \mid Nil) \setminus a = Nil$ so

$$U = \tau.(d.U + \tau.(d.b.Nil + b.d.Nil)) + \tau.b.Nil.$$

Hence $U = V$ by Law 9A.

Exercise 2. (a) The result follows from Law 9A, the Expansion Principle and Law 2(a) which yield (letting L denote the set of restricted actions)

$$\begin{aligned} D &= a.(\overline{b}.c.A \mid B \mid C) \setminus L \\ &= a.\tau.(c.A \mid d.\overline{e}.B \mid C) \setminus L \\ &= a.\tau.d.(c.A \mid \overline{e}.B \mid C) \setminus L \\ &= a.\tau.d.\tau.(c.A \mid B \mid f.\overline{c}.C) \setminus L \\ &= a.\tau.d.\tau.f.(c.A \mid B \mid \overline{c}.C) \setminus L \\ &= a.\tau.d.\tau.f.\tau.D \\ &= a.d.f.D \end{aligned}$$

where $L = \{ b, c, e\}$.

(b) By the Expansion Principle we have

$$\begin{aligned} G &= a.(d.f.D \mid F) \setminus d + a'.(D \mid \overline{d}.f'.F) \setminus d \\ &= a.a'.(d.f.D \mid \overline{d}.f'.F) \setminus d + a'.a.(d.f.D \mid \overline{d}.f'.F) \setminus d \end{aligned}$$

and also

$$\begin{aligned} (d.f.D \mid \overline{d}.f'.F) \setminus d &= \tau.(f.D \mid f'.F) \setminus d + f'.(f.D \mid F) \setminus d \\ &= \tau.(f.(a.f'.a'.(d.f.D \mid \overline{d}.f'.F) \setminus d + f'.(D \mid F) \setminus d) \\ &\quad + f'.(f.(D \mid F) \setminus d + a'.f.a.(d.f.D \mid \overline{d}.f'.F) \setminus d)). \end{aligned}$$

Hence $G = a.a'.G1 + a'.a.G1$ where $G1 \stackrel{def}{=} (d.f.D \mid \overline{d}.f'.F) \setminus d$ and

$$G1 = \tau.(f.(a.f'.a'.G1 + f'.G) + f'.(f.G + a'.f.a.G1)).$$

Now consider the equations

$$\begin{aligned} X &= a.a'.X1 + a'.a.X1 \\ X1 &= \tau.(f.(a.f'.a'.X1 + f'.X) + f'.(f.X + a'.f.a.X1)). \end{aligned}$$

The result follows by Law 9B.

G is equivalent to the modified H. This may be proved using the bisimulation proof technique of section 5.

Exercises 5

Exercise 1. (a) We use the following abbreviations:
$Sq = Square$, $Sb = Subone$, $C = Cell$, $A = Addone$, $M = Mult$ and $Cy = Copy$. Moreover we set $Sq' = \overline{down}.Sq$, $C' = \overline{out}.C$, $Cy' = \overline{up}.Cy''$, $Cy'' = \overline{down}.Cy$, $Sb' = \overline{out2}.Sb$, $M' = out2.M''$, $M'' = \overline{out}.M$ and $A' = \overline{out1}.A$. (Here we omit the values parameterising the labels.) Then from the transition diagrams of SS and $CASM$ we can see that the following relation is a bisimulation (where the agents are grouped into four 'equivalence classes' any two agents in any one of these being observationally equivalnet):

$$Sq|Sb|C \quad (Cy|A|S|M)$$

$$\begin{array}{ll} Sq'|Sb|C & (Cy'|A|S|M),\ (Cy''|A'|S|M) \\ Sq|Sb'|C & (Cy|A'|S'|M),\ (Cy''|A|S|M'),\ (Cy|A|S'|M') \\ Sq|Sb|C' & (Cy|A|S|M'') \end{array}$$

$$\begin{array}{ll} Sq'|Sb'|C & (Cy'|A'|S'|M),\ (Cy'|A|S|M'),\ (Cy''|A'|S|M') \\ Sq'|Sb|C' & (Cy'|A|S'|M'),\ (Cy|A'|S'|M) \\ Sq|Sb'|C' & (Cy''|A'|S|M''),\ (Cy|A'|S'|M'') \end{array}$$

$$Sq'|Sb'|C' \quad (Cy'|A'|S'|M'),\ (Cy'|A'|S|M''),\ (Cy'|A'|S'|M'').$$

(b) It is not the case that $SS \approx A$. Modify the definition of A to obtain B as follows:

$$\begin{array}{rcl} B & \stackrel{def}{=} & in(x).B'(x) \\ B'(x) & \stackrel{def}{=} & \overline{out}(x^2-1).B + in(y).B''(x,y) \\ B''(x,y) & \stackrel{def}{=} & \overline{out}(x^2-1).B'(y) + in(z).B'''(x,y,z) \\ B'''(x,y,z) & \stackrel{def}{=} & \overline{out}(x^2-1).B''(y,z). \end{array}$$

The following relation is a bisimulation (again we indicate the pairings and omit the values):

$$\begin{array}{ll} B & SS \\ B' & (Sq'|Sb|C),\ (Sq|Sb'|C),\ (Sq|Sb|C') \\ B'' & (Sq'|Sb'|C),\ (Sq'|Sb|C'),\ (Sq|Sb'|C') \\ B''' & Sq'|Sb'|C'. \end{array}$$

Exercise 2. This example raises an interesting issue. With the definition of $\approx$ used in these notes E and D are not observationally equivalent, since any bisimulation containing the pair (E, D) would have to contain also the pair $(E2(v,w), (\overline{out}(hv).A1 \mid B \mid up(x).\overline{rec}(x).B \mid \overline{up}(hw).C) \setminus L)$ for any pair of values v and w, and the second of these is committed to outputting hv before hw whereas the first is capable of ouputting these values in either order. If, however, there is only one value, then D and E are observationally equivalent. However, it is possible to argue that E and D could be regarded as "equivalent" independently of the size of the set of values, since to distinguish between them an external observer would have to have the ability to accept some values and yet refuse to accept others at a certain

port, and this may seem an unnatural property for an agent to have. The technical details of this modified equivalence have not, so far as I am aware, been investigated.

Exercises 6

Exercise 1. $A \approx B$ since $\{ (A, B), (Nil, C) \}$ is a bisimulation. $A \not\approx B$ since $A \xrightarrow{\tau} A$ but there is no C such that $B \stackrel{\tau}{\Longrightarrow} C$. A and C are divergent; B is not.

Exercise 2. D is divergent since

$$\begin{array}{rl} D & \xrightarrow{\tau} (D \mid Nil) \setminus a \\ & \xrightarrow{\tau} ((D \mid Nil) \setminus a) \mid Nil) \setminus a \\ & \xrightarrow{\tau} \ldots \end{array}$$

The derivation tree of D contains infinitely many distinct agents, so there is no finite transition diagram in which each node is marked with the agents it represents.

Exercise 3. (a) Redefine *Sender* as follows:

$$\begin{array}{rcl} Sender'(x) & \stackrel{def}{=} & ms.\overline{sm}(x).Sender''(x) + rs.Sender \\ Sender''(x) & \stackrel{def}{=} & ms.Sender + rs.Sender \end{array}$$

(b) One possible modification is the following:

$$\begin{array}{rcl} B & \stackrel{def}{=} & rec(x).B'(x) \\ B'(x) & \stackrel{def}{=} & \tau.\overline{send}(x).B + \tau.B \end{array}$$

(c) Since both the modified *Protocol* and the modified *Buffer* are stable agents, by the Stability Principle it is enough to prove that they are observationally equivalent. This may be achieved by exhibiting a bisimulation containing the pair (*Buffer*, *Protocol*).

Note added June 1988

I am indebted to a number of people who have detected mistakes in solutions to exercises and typographical errors in the original version of these notes. All known inaccuracies have been corrected.