

ALGEBRAIC SPECIFICATION
AND PROOF OF PROPERTIES OF
COMMUNICATING SEQUENTIAL PROCESSES

by

C. A. R. Hoare

and

He Jifeng

Oxford University
Computing Laboratory
Programming Research Group-Library
8-11 Keble Road
Oxford OX1 3QD
Oxford (0865) 54141

Technical Monograph PRG-52

November 1985

Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford OX1 3QD
England

Copyright (C) 1985 C.A.R.Hoare, He Jifeng

Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford OX1 3QD
England

CONTENTS

Page

ALGEBRAIC SPECIFICATION AND PROOF OF THE PROPERTIES OF A MAIL SERVICE

Summary

1.	Introduction	1
2.	Proof Method	5
	2.1 Composite choice	5
	2.2 Laws of composite choice	8
	2.3 Properties of nondeterministic pipes	17
3.	Properties of BAG	21
	3.1 Composition in series	21
	3.2 Composition in parrallel	29
4.	Comparison	35
5.	Acknowledgements	38
	References	39
	Appendix	40

ALGEBRAIC SPECIFICATION AND PROOF OF A DISTRIBUTED RECOVERY ALGORITHM

Summary

1.	Introduction	45
2.	Deterministic pipes	50
3.	Recoverable processes	55
	Acknowledgements	67
	References	68
	Appendix	70

ALGEBRAIC SPECIFICATION AND
PROOF OF THE PROPERTIES OF A MAIL SERVICE

Hoare, C.A.R. and He, Jifeng

Summary

A mail service is a communications medium which can reorder messages between postage and delivery. This paper investigates the algebraic properties of such a service, using the mathematical theory of communicating sequential processes.

A communications link which delivers messages in the same order as they are transmitted can be modelled abstractly as an unbounded buffer. In [4, 4.2 X9] a buffer is defined as a process which is at all times ready to input a message on its left channel, and (whenever possible) is ready to output on its right channel the earliest message which it has input but not yet output. The state of a buffer can be identified with the sequence s of messages which it has input but not yet output. Each incoming message x is added to the right hand end of s (to give $s^{\wedge}\langle x \rangle$); and each outgoing message is removed from the left end of s (to give s'). The value of the message is that of the removed item s_0 . The sequence s is initially empty $\langle \rangle$.

This informal description is captured in the formal definition of a buffer by a system of mutually recursive equations, one for each value of s :

$$\text{BUFFER} = \text{BUF}(\langle \rangle)$$

$$\begin{aligned} \text{where } \text{BUF}(s) &= \text{left?}x \longrightarrow \text{BUF}(\langle x \rangle) && \text{if } s = \langle \rangle \\ &= (\text{left?}x \longrightarrow \text{BUF}(s^{\wedge}\langle x \rangle) \\ &\quad \square \text{right!}s_0 \longrightarrow \text{BUF}(s')) && \text{if } s \neq \langle \rangle \end{aligned}$$

The buffer described above can never refuse to input another message, so there is no upper bound on the number of messages it can store waiting for delivery. In contrast, a bounded buffer, when full, will refuse to input further messages. For example, a buffer which can store at most one message can be defined by simple recursion:

$$\text{COPY} = \text{left?}x \longrightarrow \text{right!}x \longrightarrow \text{COPY}$$

A buffer is an example of a pipe, i.e., a process with a single input channel on the left and a single output channel on the right. A pair of pipes P and Q can be assembled into a single longer pipe $(P \gg Q)$ by connecting the right channel of P to the left channel of Q, so that all messages output by P are simultaneously input by Q and vice versa. These internal messages are concealed, so that $(P \gg Q)$ is also a pipe, in which all external input goes to P and all external output comes from Q.

The algebraic properties of pipes in general and buffers in particular are investigated in [4, 4.4]. For a designer of communications services, the most important properties are that buffers can validly be composed in series by the chaining operator \gg :

$$\begin{aligned} \text{BUFFER} &= \text{BUFFER} \gg \text{COPY} \\ &= \text{COPY} \gg \text{BUFFER} \\ &= \text{BUFFER} \gg \text{BUFFER} \end{aligned}$$

In this paper we shall extend these results to a different kind of communication service, namely one which may non-deterministically reorder the messages before delivery. Such a service can be modelled abstractly as a bag (sometimes known as a collection or multiset, because each member may appear in it more than once). Like a buffer, a bag is at all times willing to input a message on its left channel. Whenever the bag is non-empty, it is prepared to output any of its stored messages on the right channel. The choice of message for output is made non-deterministically by the operator $\sqcap_{y \in t}$, which selects an arbitrary member y from a bag t. Thus a bag can be defined in a manner similar to the buffer, using the variable t to stand for the bag of values which have been input but not yet output:

$$\text{BAG} = B(\emptyset)$$

$$B(t) = \text{left?}x \rightarrow B(\langle x \rangle) \quad \text{if } t = \emptyset$$

$$= \text{left?}x \rightarrow B(t \langle + \rangle \langle x \rangle)$$

$$\coprod_{y \in t} (\prod \text{right!}y \rightarrow B(t \langle - \rangle \langle y \rangle)) \quad \text{if } t \neq \emptyset$$

where \emptyset is the empty bag

$\langle x \rangle$ is the bag containing only x

$\langle + \rangle \langle - \rangle$ are bag addition and subtraction

For a designer of communication services, it is an important fact that bags enjoy the same algebraic properties as those quoted above for buffers, so that they can be validly composed in series with other bags or with buffers

$$\begin{aligned} \text{BAG} = \text{BAG} \gg \text{COPY} &= \text{COPY} \gg \text{BAG} \\ &= \text{BAG} \gg \text{BUFFER} = \text{BUFFER} \gg \text{BAG} \\ &= \text{BAG} \gg \text{BAG} \end{aligned}$$

Furthermore, since a bag is non-deterministic, it is possible for the non-deterministic choice always to fall on the longest waiting message, and in this case a bag can behave exactly like a buffer. This means that a buffer is a perfectly valid implementation of a bag, or more formally

$$\text{BAG} \sqsubseteq \text{BUFFER}.$$

From the implementor's point of view, the most important distinction between buffers and bags is that bags can be composed not only in series but also in parallel with other bags and with buffers. Parallel composition in this case is represented by the interleaving operator \parallel [4, 3.6], which permits arbitrary interleaving of actions from its two

4.

operands. So consider the process $(BAG \parallel\parallel BAG)$. Each message it inputs is stored by one of its two operands, we know not which. Each message it outputs is output by one of the operands, we know not which. But we do know that each message output has been previously input and will not be output again. Thus the pair of bags behaves like a single bag, and it makes no difference which of the pair has actually carried each message. Thus we informally justify the law

$$BAG = BAG \parallel\parallel BAG$$

Other laws with similar justification are:

$$\begin{aligned} BAG &= BAG \parallel\parallel COPY \\ &= BAG \parallel\parallel BUFFER \end{aligned}$$

The algebraic laws quoted above are clearly ones which we would like to be true. It is the purpose of this paper to supply the necessary proofs. It is an equally important objective that these proofs should use simple methods such as algebraic calculation, which can be applied more generally to proofs of more complicated theorems needed for the design of successful protocols. Accordingly, the next section of the paper is devoted to quoting and explaining the relevant proof methods and algebraic properties of the operators in use. Many of these will be found in [4]. The proofs of the theorems are postponed to the third section. A final section is devoted to a comparison of our treatment with the original treatment of [1], which is based on an algebraic version of CCS, and which inspired us to work on the current paper.

2. Proof method

A general method of proving equations and inequations in Communicating Sequential Processes is to reduce each side to the same standard form. The standard form is a guarded expression defining a process by mutual recursion, and not containing parallel combinators. The theorem of unique solutions for guarded recursion then completes the proof. The reduction to standard form is achieved by symbolic execution of input and output commands, which can be formally justified by appeal to the relevant algebraic laws. The algebraic calculation is slightly simplified by introduction of some new compact but complicated notations. These are explained informally in section 2.1, and defined formally in terms of the more familiar operators of [4]. Section 2.2 lists the standard algebraic properties of the new operators, and proves them from their definitions. Section 2.3 gives the rather more complicated laws which are actually used in symbolic execution of expressions denoting pipes. These laws are strong enough to reduce all well-defined pipes to a standard guarded form, in which the parallel combinators \gg and \parallel are pushed as far inward as possible. This gives confidence that the laws are strong enough for all practical purposes. We will save space by omitting the channel 'left' and 'right' from input and output commands.

2.1 Composite choice

The technique we shall use for proving equations in the rest of this paper is simply based on algebraic calculations, which in practice amounts to no more than symbolic execution of input and output commands. In order to reduce the size of calculations and the number of laws needed, we introduce several composite choice operators.

If P and Q are processes, the process $P \bowtie Q$ is formally defined

$$P \bowtie Q = (P \sqcup Q) \sqcap Q \quad (\text{or equivalently, } (P \sqcap Q) \sqcup Q)$$

The composite choice \bowtie arises naturally from concealment of internal events. For instance, the following law describes the behaviour of a pipe $P \gg Q$ when P is ready for both external communication and internal communication, but Q is only willing to accept messages from P [4, 4.4], i.e.,

$$\text{if } P = (\exists x \rightarrow P1(x) \sqcup !e \rightarrow P2)$$

$$\text{and } Q = (\exists x \rightarrow Q1(x))$$

$$\text{then } P \gg Q = (\exists x \rightarrow (P1(x) \gg Q)) \bowtie (P2 \gg Q1(e)).$$

The left operand of \bowtie deals with the case when the external communication occurs first, and the right operand describes what happens after the concealed communication. That is why an asymmetric operator is needed.

We shall use an infix notation for conditions. If b is a boolean expression

$$\begin{aligned} P \dagger b \dagger Q &= P && \text{if } b \text{ is true} \\ &= Q && \text{if } b \text{ is false} \end{aligned}$$

Note that

$$(P1 \sqcap P2) \dagger b \dagger (Q1 \sqcap Q2) = (P1 \dagger b \dagger Q1) \sqcap (P2 \dagger b \dagger Q2)$$

Proof: consider the cases that b is true and false.

Similarly

$$(P1 \sqcup P2) \dagger b \dagger (Q1 \sqcup Q2) = (P1 \dagger b \dagger Q1) \sqcup (P2 \dagger b \dagger Q2).$$

Let b be a boolean expression, and S be a set. The guarded set $b \ \& \ S$ is defined

$$b \ \& \ S = S \ \nmid \ b \ \nmid \ \emptyset$$

where \emptyset is the empty set.

Similarly, if P is a process we define

$$b \ \& \ P = P \ \nmid \ b \ \nmid \ \text{STOP}$$

Another composite operator is denoted $\prod_{y \in t}$. Let P be a process, and $\{Q(y) \mid y \in t\}$ a set of processes. The process $P \prod_{y \in t} Q(y)$ is defined by

$$P \prod_{y \in t} Q(y) = P \ \nmid \ (t = \emptyset) \ \nmid \ (P \prod_{y \in t} Q(y))$$

where \emptyset is the empty set.

Thus a bag can be redefined from section 1:

$$\text{BAG} = B(\cup)$$

$$B(t) = (\exists x \rightarrow B(t \cup \{x\})) \prod_{y \in t} (\exists y \rightarrow B(t \cup \{y\}))$$

Let P be a process, and $\{Q(y) \mid y \in S\}$ a set of processes. The process $P \prod_{y \in t} Q(y)$ is defined in the same way as $\prod_{y \in t}$, but using \prod in place of \prod .

$$P \prod_{y \in t} Q(y) = P \ \nmid \ t = \emptyset \ \nmid \ (P \prod_{y \in t} Q(y))$$

An immediate advantage of the notations $\prod_{y \in t}$ and $\prod_{y \in t}$ is that they

enable us to reduce all well-defined pipes to a standard form, as shown in section 2.3.

8.

2.2 Laws of composite choice

This section is devoted to quoting and explaining algebraic properties of the operators in use, which are based on the following basic laws presented in [4].

(1) Laws of nondeterminism [4, 3.2]

\sqcap is idempotent, symmetric and associative.

(2) Laws of general choice [4, 3.3]

The general choice operator \sqcup enjoys algebraic properties similar to \sqcap . It is idempotent, symmetric, associative and distributes through \sqcap . STOP is the unit of \sqcup . Moreover \sqcap distributes through \sqcup .

A consequence of the last distributivity principle is that \sqcup is more deterministic than \sqcap ; this is expressed in the closure law

L1A (Closure)

$$P \sqcap Q = P \sqcap Q \sqcap (P \sqcup Q)$$

$$\begin{aligned} \text{Proof: RHS} &= (P \sqcap Q \sqcap P) \sqcup (P \sqcap Q \sqcap Q) && \sqcap \text{ distributes through } \sqcup \\ &= \text{LHS} && \text{properties of } \sqcap \end{aligned}$$

The following lemmas will be used immediately

$$(a) \text{ STOP } \sqcap (Q \sqcup R) = \text{STOP } \sqcap Q \sqcap R \sqcap (Q \sqcup R)$$

$$(b) \text{ STOP } \sqcap (Q \sqcup R) = \text{STOP } \sqcap Q \sqcap (Q \sqcup R)$$

Proof: (a) $LHS = (STOP \cap Q) \cap (STOP \cap R)$ \cap distributes through \cap
 $= (STOP \cap STOP) \cap (Q \cap STOP) \cap (STOP \cap R) \cap (Q \cap R)$
 $= RHS$ \cap distributes through \cap
 \cap is idempotent

(b) $RHS = Q \cap (STOP \cap (Q \cap R))$ properties of \cap
 $= LHS$ by (a) twice
and properties of \cap

The fundamental convexity law for \cap and \cap follows directly from the lemmas

L1B (Convexity)

(a) $P \cap (P \cap Q \cap R) = P \cap (P \cap Q) \cap (P \cap R) \cap (P \cap Q \cap R)$

(b) $P \cap (P \cap Q \cap R) = P \cap (P \cap Q) \cap (P \cap Q \cap R)$

Proof: (a) $LHS = P \cap (STOP \cap (Q \cap R))$ \cap distributes through \cap
and has unit $STOP$
 $= P \cap (STOP \cap Q \cap R \cap (Q \cap R))$ lemma (a)
 $= RHS$ \cap distributes through \cap

(b) similar

The closure and convexity laws generalise to any finite number of operands. Let \mathcal{T} be a finite nonempty family of sets and let \mathcal{T}' be an enlarged family such that $\mathcal{T} \subseteq \mathcal{T}'$

and $\forall T \in \mathcal{T}'. \exists U \in \mathcal{T}. U \subseteq T \subseteq \bigcup \mathcal{T}$

Thus \mathcal{T}' contains only sets which lie between a set of \mathcal{T} and the union of all sets in \mathcal{T} .

$$\bigcap_{T \in \mathcal{T}'} \left(\bigcap_{t \in T} P_t \right) = \bigcap_{T \in \mathcal{T}} \left(\bigcap_{t \in T} P_t \right)$$

10.

The convex closure of \mathcal{T} is the largest family satisfying the conditions of \mathcal{T}' given above.

Any expression in the operators \sqcap and \sqcup can be expanded to a normal form

$$\prod_{T \in \mathcal{T}'} \prod_{t \in T} p_t$$

in which \mathcal{T}' is its own convex closure. This method can be used to prove most of the theorems in this section.

When P and Q have the same initial event, $(P \sqcup Q)$ degenerates to nondeterministic choice $\underline{[4, 3.2.1]}$:

$$L2 \quad (a \rightarrow R) \sqcup (a \rightarrow S) = (a \rightarrow R) \sqcap (a \rightarrow S)$$

(3) Laws of \rightarrow $\underline{[4, 3.2.1]}$

L3 The prefixing operator \rightarrow distributes through \sqcap

$$(x:B \rightarrow (P(x) \sqcap Q(x))) = (x:B \rightarrow P(x)) \sqcap (x:B \rightarrow Q(x))$$

provided B is finite.

Generalisation

If both B and C are finite, then

$$(x:B \rightarrow P(x)) \sqcap (x:C \rightarrow Q(x)) = (x:B \rightarrow R(x)) \sqcap (x:C \rightarrow R(x))$$

where	$R(x) = P(x) \sqcap Q(x)$	$x \in B \cap C$
	$= P(x)$	$x \in B - C$
	$= Q(x)$	$x \in C - B$

(4) Laws of \cap

\cap is idempotent, distributive and associative.

$$L4 \quad P \cap P = P$$

$$\begin{aligned} \text{Proof: LHS} &= (P \cap P) \cap P \\ &= \text{RHS} \end{aligned}$$

def of \cap
both \cap and \cap are idempotent

$$L5 \quad (a) \quad (P1 \cap P2) \cap Q = (P1 \cap Q) \cap (P2 \cap Q)$$

$$(b) \quad P \cap (Q1 \cap Q2) = (P \cap Q1) \cap (P \cap Q2)$$

$$\begin{aligned} \text{Proof: (a) LHS} &= ((P1 \cap P2) \cap Q) \cap Q && \text{def of } \cap \\ &= (P1 \cap Q) \cap (P2 \cap Q) \cap Q && \cap \text{ distributes through } \cap \\ &= ((P1 \cap Q) \cap Q) \cap ((P2 \cap Q) \cap Q) && \text{properties of } \cap \\ &= \text{RHS} && \text{def of } \cap \end{aligned}$$

(b) Similar to (a).

Corollary. If t is nonempty, then

$$P \cap_{y \in t} Q(y) = \bigcap_{y \in t} (P \cap Q(y))$$

$$\begin{aligned} \text{Proof: LHS} &= P \cap \left(\bigcap_{y \in t} Q(y) \right) && \text{def of } \bigcap_{y \in t} \\ &= \text{RHS} && \cap \text{ distributes through } \bigcap \end{aligned}$$

$$L6 \quad P \setminus (Q \sqcup R) = (P \setminus Q) \setminus R.$$

$$\begin{aligned}
 \text{Proof: LHS} &= P \setminus ((Q \sqcup R) \sqcap R) && \text{def of } \setminus \\
 &= (P \setminus (Q \sqcup R)) \sqcap (P \setminus R) && \setminus \text{ distributes through } \sqcap \\
 &= (P \sqcup Q \sqcup R) \sqcap (Q \sqcup R) \sqcap (P \sqcup R) \sqcap R && \text{def of } \setminus \\
 &= (P \sqcup Q \sqcup R) \sqcap (Q \sqcup R) \sqcap R && \text{law of convexity} \\
 &= ((P \setminus Q) \sqcup R) \sqcap R && \sqcup \text{ distributes through } \sqcap \\
 &= \text{RHS} && \text{def of } \setminus
 \end{aligned}$$

L7 \setminus also distributes through \sqcup

$$(a) \quad (P1 \sqcup P2) \setminus Q = (P1 \setminus Q) \sqcup (P2 \setminus Q)$$

$$(b) \quad P \setminus (Q1 \sqcup Q2) = (P \setminus Q1) \sqcup (P \setminus Q2)$$

$$\begin{aligned}
 \text{Proof: (a) LHS} &= (P1 \sqcup P2 \sqcup Q) \sqcap Q && \text{def of } \setminus \\
 &= (P1 \sqcup P2 \sqcup Q) \sqcap (P1 \sqcup Q) \sqcap (P2 \sqcup Q) \sqcap Q && \text{law of convexity} \\
 &= ((P1 \sqcup Q) \sqcap Q) \sqcup ((P2 \sqcup Q) \sqcap Q) && \sqcup \text{ distributes through } \sqcap \\
 &= \text{RHS} && \text{def of } \setminus
 \end{aligned}$$

.. (b) Similar to (a).

As expected, \sqcup and \sqcap distribute through \setminus .

$$L8 \quad (P \setminus Q) \sqcup R = (P \sqcup R) \setminus (Q \sqcup R)$$

$$\begin{aligned}
 \text{Proof: LHS} &= ((P \sqcup Q) \sqcap Q) \sqcup R && \text{def of } \setminus \\
 &= (P \sqcup Q \sqcup R) \sqcap (Q \sqcup R) && \sqcup \text{ distributes through } \sqcap \\
 &= \text{RHS} && \text{def of } \setminus
 \end{aligned}$$

$$L9 \quad (P \searrow Q) \cap R = (P \cap R) \searrow (Q \cap R)$$

$$\begin{aligned} \text{Proof: LHS} &= (P \sqcap Q) \cap Q \cap R && \text{def of } \searrow \\ &= ((P \cap R) \sqcup (Q \cap R)) \cap Q \cap R && \cap \text{ is idempotent and} \\ & && \text{distributes through } \sqcup \\ &= \text{RHS} && \text{def of } \searrow \end{aligned}$$

Furthermore, STOP is the left-unit of \searrow .

$$L10 \quad \text{STOP} \searrow Q = Q$$

$$\begin{aligned} \text{Proof: LHS} &= (\text{STOP} \sqcap Q) \cap Q && \text{def of } \searrow \\ &= Q \cap Q && \text{STOP is the unit of } \sqcap \\ &= \text{RHS} && \cap \text{ is idempotent} \end{aligned}$$

Finally we give an expansion theorem for \searrow , where the arguments are expressed in terms of general choice.

$$L11 \quad \text{Let } P = (x:B \rightarrow P(x))$$

$$\text{and } Q = (x:C \rightarrow Q(x))$$

If both B and C are finite, then

$$P \searrow Q = \bigsqcup_{C \subseteq S \subseteq (B \cup C)} R_S$$

$$\text{where } R_S = (x:S \rightarrow R(x))$$

$$\begin{aligned} \text{and } R(x) &= P(x) \cap Q(x) && x \in B \cap C \\ &= P(x) && x \in B - C \\ &= Q(x) && x \in C - B \end{aligned}$$

$$\begin{aligned} \text{Proof: LHS} &= ((x:B \rightarrow P(x)) \sqcup (x:C \rightarrow Q(x))) \cap (x:C \rightarrow Q(x)) && \text{def of } \searrow \\ &= (x:B \cup C \rightarrow R(x)) \cap (x:C \rightarrow R(x)) && \text{property of } \rightarrow \\ &= \text{RHS} && \text{law of convexity} \end{aligned}$$

(5) Laws of $\prod_{y \in S}$

$\prod_{y \in S}$ is distributive and associative.

$$L12 \quad (a) \quad (P1 \cap P2) \prod_{y \in S} Q(y) = (P1 \prod_{y \in S} Q(y)) \cap (P2 \prod_{y \in S} Q(y))$$

$$(b) \quad P \prod_{y \in S} (Q1(y) \cap Q2(y)) = (P \prod_{y \in S} Q1(y)) \cap (P \prod_{y \in S} Q2(y))$$

Proof: (a) LHS = $(P1 \cap P2) \prod_{y \in S} Q(y) \stackrel{\text{def of } \prod_{y \in S}}{=} ((P1 \cap P2) \prod_{y \in S} Q(y))$
 $= (P1 \cap P2) \prod_{y \in S} (Q(y) \cap Q(y)) \stackrel{\text{def of } \prod_{y \in S}}{=} ((P1 \prod_{y \in S} Q(y)) \cap (P2 \prod_{y \in S} Q(y)))$
 $\prod_{y \in S}$ distributes through \cap
 $=$ RHS def of $\prod_{y \in S}$ and property of $\prod_{y \in S}$

(b) Similar to (a).

$$L13 \quad (P \prod_{y \in S} Q(y)) \prod_{z \in S} R(z) = P \prod_{y \in S} (Q(y) \prod_{z \in S} R(z))$$

Proof: Case 1. $S = \emptyset$

$$\begin{aligned} \text{LHS} &= P \prod_{y \in S} Q(y) && \text{def of } \prod_{z \in \emptyset} \\ &= P && \text{def of } \prod_{y \in \emptyset} \\ &= \text{RHS} && \text{def of } \prod_{y \in \emptyset} \end{aligned}$$

Case 2. $S \neq \emptyset$

$$\begin{aligned} \text{LHS} &= (P \prod_{y \in S} Q(y)) \prod_{z \in S} R(z) && \text{def of } \prod_{y \in S} \\ &= P \prod_{y \in S} (Q(y) \prod_{z \in S} R(z)) && \text{the associative law of } \prod \\ &= P \prod_{y \in S} (Q(y) \prod_{z \in S} R(z)) && \prod \text{ distributes through } \cap. \\ &= P \prod_{y \in S} (Q(y) \prod_{z \in S} R(z)) && \text{def of } \prod_{z \in S} \\ &= \text{RHS} && \text{def of } \prod_{y \in S} \end{aligned}$$

Generalisation

$$(P \prod_{y \in S} Q(y)) \prod_{z \in T} R(z) = P \prod_{y \in S} (Q(y) \prod_{z \in T} R(z)) \quad \text{provided that } S = \emptyset \Rightarrow T = \emptyset$$

Proof: Similar to L13.

L14 $\prod_{y \in S}$ distributes left through \square .

$$(P1 \square P2) \prod_{y \in S} Q(y) = (P1 \prod_{y \in S} Q(y)) \square (P2 \prod_{y \in S} Q(y))$$

Proof: LHS = $(P1 \square P2) \not\leftarrow S = \emptyset \not\leftarrow ((P1 \square P2) \square \prod_{y \in S} Q(y))$ def of $\prod_{y \in S}$

$$= (P1 \square P2) \not\leftarrow S = \emptyset \not\leftarrow ((P1 \square \prod_{y \in S} Q(y)) \square (P2 \square \prod_{y \in S} Q(y)))$$

\square is idempotent and associative

$$= \text{RHS} \quad \text{property of } \not\leftarrow b \not\leftarrow \text{ and def of } \prod_{y \in S}$$

L15 $\prod_{y \in S}$ distributes through $\prod_{y \in S}$

$$(P \prod_{y \in S} Q(y)) \prod R = (P \prod R) \prod_{y \in S} (Q(y) \prod R)$$

Proof: LHS = $(P \not\leftarrow S = \emptyset \not\leftarrow (P \square \prod_{y \in S} Q(y))) \prod R$ def of $\prod_{y \in S}$

$$= (P \prod R) \not\leftarrow S = \emptyset \not\leftarrow ((P \square \prod_{y \in S} Q(y)) \prod R)$$

\prod distributes through $\not\leftarrow b \not\leftarrow$

$$= \text{RHS} \quad \prod \text{ distributes through } \square$$

L16 \square also distributes through $\prod_{y \in S}$

$$(P \prod_{y \in S} Q(y)) \square R = (P \square R) \prod_{y \in S} (Q(y) \square R)$$

Proof: Similar to L4.

L17 If t is nonempty, then

$$P \prod_{y \in t} Q(y) = \prod_{y \in t} (P \square Q(y))$$

Proof: LHS = $P \square (\prod_{y \in t} Q(y))$ def of $\prod_{y \in t}$

\square distributes through \prod

16.

$$L18 \quad P \searrow (Q \prod_{y \in S} R(y)) = (P \searrow Q) \prod_{y \in S} R(y)$$

$$\begin{aligned} \text{Proof: LHS} &= (P \sqcap (Q \prod_{y \in S} R(y))) \sqcap (Q \prod_{y \in S} R(y)) && \text{def of } \searrow \\ &= ((P \sqcap Q) \sqcap Q) \not\leq S = 0 \not\leq ((P \sqcap Q) \prod_{y \in S} R(y)) \sqcap (Q \prod_{y \in S} R(y)) && \text{def of } \prod_{y \in S} \\ &= (P \searrow Q) \not\leq S = 0 \not\leq ((P \searrow Q) \prod_{y \in S} R(y)) && \text{the distributive law of } \sqcap \\ &= \text{RHS} && \text{def of } \prod_{y \in S} \end{aligned}$$

finally, we have an expansion theorem for \searrow

$$L19 \quad \text{Let } P = (PL \prod_{y \in S} PR(y))$$

$$\text{and } Q = (QL \prod_{y \in T} QR(y))$$

If $S \subseteq T$, and $QR(y) \subseteq PR(y)$ for all $y \in S$, then

$$P \searrow Q = ((PL \searrow QL) \prod_{y \in T} QR(y))$$

Proof: Case 1. $S = 0$

$$\begin{aligned} \text{LHS} &= PL \searrow Q && \text{def of } \prod_{y \in 0} \\ &= \text{RHS} && L18 \text{ of } \prod_{y \in S} \end{aligned}$$

Case 2. $S \neq 0$

$$\begin{aligned} \text{LHS} &= \prod_{y \in S} \prod_{z \in T} (PL \sqcap QL \sqcap PR(y) \sqcap QR(z)) \sqcap \prod_{z \in T} (QL \sqcap QR(z)) && \text{the distributive law of } \sqcap \\ &= \prod_{y \in S} \prod_{z \in T} ((PL \sqcap QL \sqcap PR(y) \sqcap QR(z)) \sqcap (QL \sqcap QR(z))) \sqcap (PL \sqcap QL \sqcap QR(z)) && \text{law of convexity} \\ &= \prod_{y \in S} \prod_{z \in T} ((PL \sqcap QL \sqcap PR(y) \sqcap QR(z)) \sqcap (QL \sqcap QR(z))) && \\ &\quad \sqcap (PL \sqcap QL \sqcap QR(y) \sqcap QR(z)) \sqcap (PL \sqcap QL \sqcap QR(z)) && \text{law of convexity and since } S \subseteq T \\ &= \prod_{y \in S} \prod_{z \in T} ((PL \sqcap QL \sqcap QR(z)) \sqcap (QL \sqcap QR(z))) && \text{law of convexity and since } QR(y) \subseteq PR(y) \\ &= \text{RHS} \end{aligned}$$

2.3 Properties of nondeterministic pipes

The introduction of boolean guards and the composite choice operator $\bigsqcup_{y \in S}$ will reduce the number of laws needed to describe behaviour of nondeterministic pipes, since they permit all pipes to be expressed in the standard form

$$(b \ \& \ ?x \longrightarrow P1(x)) \bigsqcup_{y \in S} (!y \longrightarrow P2(y))$$

The special case when a pipe cannot initially input is dealt with by setting $b = \text{false}$. If it is not ready to output, then S is set to \emptyset . For the deadlocked pipe STOP, both $b = \text{false}$ and $S = \emptyset$.

Now we can give an expansion theorem for chaining of nondeterministic pipes.

$$L20 \quad \text{Let } P = (b1 \ \& \ ?x \longrightarrow P1(x)) \bigsqcup_{y \in S} (!y \longrightarrow P2(y))$$

$$\text{and } Q = (c1 \ \& \ ?x \longrightarrow Q1(x)) \bigsqcup_{y \in T} (!y \longrightarrow Q2(y))$$

$$\text{Then } P \gg Q = R \bigsqcup_{y \in c1 \ \& \ S} W(y)$$

$$\text{where } R = (b1 \ \& \ ?x \longrightarrow (P1(x) \gg Q)) \bigsqcup_{y \in T} (!y \longrightarrow (P \gg Q2(y)))$$

$$\text{and } W(y) = P2(y) \gg Q1(y)$$

The definition of R says that $P \gg Q$ may engage in an external communication if either P or Q (or both) does. When both are ready to communicate with each other, then the internal communication takes place, and the choice among the messages being transmitted from P to Q is left unspecified. In this subtle case, the behaviour of $P \gg Q$ is stated by L20 to be the composite choice $R \bigsqcup_{y \in c1 \ \& \ S} W(y)$.

Simple special cases of this law are given below. They are best remembered as examples of symbolic execution of input and output commands.

$$(a) \text{ when } P = (\exists x \rightarrow P1(x)) \prod_{y \in S} (!y \rightarrow P2(y)) \text{ and } Q = (\exists x \rightarrow Q1(x))$$

$$P \gg Q = (\exists x \rightarrow (P1(x) \gg Q)) \prod_{y \in S} (P2(y) \gg Q1(y))$$

$$(b) \text{ when } P = (\exists x \rightarrow P1(x)) \prod_{y \in S} (!y \rightarrow P2(y)) \text{ and } Q = (!z \rightarrow R)$$

$$P \gg Q = (\exists x \rightarrow (P1(x) \gg Q)) \prod (!z \rightarrow (P \gg R))$$

$$(c) \text{ when } P = (\exists x \rightarrow P1(x)) \text{ and } Q = (\exists x \rightarrow Q1(x))$$

$$P \gg Q = \exists x \rightarrow (P(x) \gg Q)$$

$$(d) \text{ when } P = (!z \rightarrow R) \text{ and } Q = (\exists x \rightarrow Q1(x))$$

$$P \gg Q = R \gg Q1(z)$$

$$(e) \text{ when } P = (\exists x \rightarrow P1(x)) \text{ and } Q = (\exists x \rightarrow Q1(x)) \prod_{y \in T} (!y \rightarrow Q2(y))$$

$$P \gg Q = (\exists x \rightarrow (P1(x) \gg Q)) \prod_{y \in T} (!y \rightarrow (P \gg Q2(y)))$$

$$(f) \text{ when } P = (!z \rightarrow R) \text{ and } Q = (\exists x \rightarrow Q1(x)) \prod_{y \in T} (!y \rightarrow Q2(y))$$

$$P \gg Q = \prod_{y \in T} (!y \rightarrow (P \gg Q2(y))) \prod (R \gg Q1(z))$$

Proof: see appendix.

The expansion theorem for interleaving of pipes is

$$L21 \text{ Let } P = (b1 \ \& \ ?x \rightarrow P1(x)) \ \prod_{y \in S} (!y \rightarrow P2(y))$$

$$\text{and } Q = (c1 \ \& \ ?x \rightarrow Q1(x)) \ \prod_{z \in T} (!z \rightarrow Q2(z)).$$

$$\text{Then } P \parallel Q = (b1 \ \& \ ?x \rightarrow (P1(x) \parallel Q))$$

$$\ \prod_{c1 \ \& \ ?x \rightarrow (P \parallel Q1(x))}$$

$$\ \prod_{y \in S} !y \rightarrow (P2(y) \parallel Q)$$

$$\ \prod_{z \in T} !z \rightarrow (P \parallel Q2(z))$$

Proof: see appendix.

Finally, we have an expansion theorem for the composite choice of pipes.

$$L22 \text{ Let } P = (b1 \ \& \ ?x \rightarrow P1(x)) \ \prod_{y \in S} (!y \rightarrow P2(y))$$

$$\text{and } Q = (?x \rightarrow Q1(x)) \ \prod_{y \in T} (!y \rightarrow Q2(y))$$

If $S \subseteq T$, then

$$P \ \& \ Q = (?x \rightarrow R1(x)) \ \prod_{y \in T} (!y \rightarrow R2(y))$$

$$\text{where } R1(x) = (P1(x) \ \cap \ Q1(x)) \ \& \ b1 \ \& \ Q1(x)$$

$$\text{and } R2(y) = (P2(y) \ \cap \ Q2(y)) \ \& \ y \in S \ \& \ Q2(y)$$

$$\text{Proof: LHS} = ((b1 \ \& \ ?x \rightarrow R1(x)) \ \prod_{y \in S} (!y \rightarrow R2(y)))$$

$$\ \& \ ((?x \rightarrow R1(x)) \ \prod_{y \in T} (!y \rightarrow R2(y)))$$

property of \rightarrow

$$= ((b1 \ \& \ ?x \rightarrow R1(x)) \ \& \ (?x \rightarrow R1(x)))$$

$$\ \prod_{y \in T} (!y \rightarrow R1(y))$$

the expansion theorem of $\&$

$$= \text{RHS}$$

both $\&$ and \cap are idempotent.

There are simple special cases of this law.

(a) If $P = (\exists x \rightarrow P1(x))$

and $Q = (\exists x \rightarrow Q1(x)) \prod_{y \in T} (:y \rightarrow Q2(y)),$

then $P \wedge Q = (\exists x \rightarrow (P1(x) \wedge Q1(x))) \prod_{y \in T} (:y \rightarrow Q2(y))$

3. Properties of BAG

In this section we show that BAG enjoys a set of algebraic properties quoted in Section 1. The proofs are based on pure algebraic transformation and the unique fixed point theorem.

3.1 Composition in series

This subsection is devoted to showing the following results

$$\begin{aligned} \text{BAG} &= \text{BAG} \gg \text{COPY} = \text{COPY} \gg \text{BAG} \\ &= \text{BAG} \gg \text{BUFFER} = \text{BUFFER} \gg \text{BAG} \\ &= \text{BAG} \gg \text{BAG} \end{aligned}$$

We shall use the definition of BAG given in section 2.1.

The following properties of bags will be used in the later proofs.


(a) If $y \in t$, then

$$t \text{ (} \ominus \text{)}(y) \text{ (} \oplus \text{)}(x) = t \text{ (} \oplus \text{)}(x) \text{ (} \ominus \text{)}(y)$$

(b) If $y \in t$, then

$$(t \text{ (} \ominus \text{)}(y)) \text{ (} \oplus \text{)}(s \text{ (} \oplus \text{)}(y)) = t \text{ (} \oplus \text{)} s.$$

$$(c) t \text{ (} \oplus \text{)} s = \text{ (} \oplus \text{)} \iff (t = \text{ (} \oplus \text{)} \wedge s = \text{ (} \cup \text{)}) \vee (t = \text{ (} \cup \text{)} \wedge s = \text{ (} \oplus \text{)})$$

Readers will notice that the introduction of composite choices 

and $\prod_{y \in S}$ is a great help in reducing the size of calculations. The laws of composite choice play a key role in simplifying the proofs.

First we show that BAG can be composed in series with COPY.

Lemma 1

Let t denote a nonempty bag, then for any $y \in t$

$$(B(t) \gg \text{COPY}) \subseteq B(t \setminus \{y\}) \gg (!y \rightarrow \text{COPY})$$

$$\begin{aligned} \text{Proof: LHS} &= (\exists x \rightarrow (x \in t \wedge \{x\}) \gg \text{COPY}) \prod_{y \in t} (B(b \setminus \{y\}) \gg (!y \rightarrow \text{COPY})) \\ &\quad \text{expansion of } \gg \text{ case (a)} \\ &\subseteq \prod_{y \in t} (B(b \setminus \{y\}) \gg (!y \rightarrow \text{COPY})) \quad \text{def of } \prod_{y \in t} \\ &\subseteq \text{RHS} \quad \text{def of } \prod \end{aligned}$$

This lemma will be used in the form

$$\text{LHS} \sqcap \text{RHS} = \text{LHS}$$

Theorem 1

$$\text{BAG} \gg \text{COPY} = \text{BAG}$$

Proof: For any bag t we define

$$A(t) = B(t) \gg \text{COPY}$$

By taking $t = \cup$ we obtain

$$A(\cup) = \text{BAG} \gg \text{COPY}$$

We intend to show that the processes $A(t)$ and $B(t)$ meet the same guarded mutually recursive equations. There are two cases:

$$(0) \quad t = \cup$$

$$\begin{aligned} \therefore A(\cup) &= B(\cup) \gg \text{COPY} && \text{def of } A(\cup) \\ &= \exists x \rightarrow (B(\{x\}) \gg \text{COPY}) && \text{expansion of } \gg \text{ case (c)} \\ &= \exists x \rightarrow A(\{x\}) && \text{def of } A(\{x\}) \end{aligned}$$

(1) t is nonempty

$$A(t) = (?x \rightarrow B(t \cup \{x\}) \prod_{y \in t} !y \rightarrow B(t \setminus \{y\})) \gg (?y \rightarrow (!y \rightarrow \text{COPY}))$$

def A, B and COPY

$$= (?x \rightarrow (B(t \cup \{x\}) \gg \text{COPY}) \bigvee_{y \in t} (B(t \setminus \{y\}) \gg (!y \rightarrow \text{COPY})))$$

expansion of \gg case (a)

$$= \prod_{y \in t} (?x \rightarrow (B(t \cup \{x\}) \gg \text{COPY}) \bigvee B(t \setminus \{y\}) \gg (!y \rightarrow \text{COPY}))$$

since t is nonempty and \bigvee distributes through \prod .

$$= \prod_{y \in t} (?x \rightarrow (B(t \cup \{x\}) \gg \text{COPY}) \bigvee (?x \rightarrow (B(t \setminus \{y\} \cup \{x\}) \gg (!y \rightarrow \text{COPY}))) \bigvee !y \rightarrow (B(t \setminus \{y\}) \gg \text{COPY}))$$

expansion of \gg case (b)

$$= \prod_{y \in t} (?x \rightarrow ((B(t \cup \{x\}) \gg \text{COPY}) \prod (B(t \setminus \{y\} \cup \{x\}) \gg (!y \rightarrow \text{COPY}))) \prod !y \rightarrow (B(t \setminus \{y\}) \gg \text{COPY}))$$

expansion of the composite choice of pipes case (a)

$$= \prod_{y \in t} (?x \rightarrow A(t \cup \{x\}) \prod !y \rightarrow A(t \setminus \{y\}))$$

def A, lemma 1 and property (a) of bag.

$$= (?x \rightarrow A(t \cup \{x\}) \prod_{y \in t} !y \rightarrow A(t \setminus \{y\}))$$

since t is nonempty and \prod distributes through \prod .

24.

Combining these results we conclude that for any bag t

$$A(t) = B(t)$$

by the unique fixed point theorem. By taking $t = \langle \rangle$ we complete the proof.

Now we give a proof of the equality

$$BAG \gg BAG = BAG$$

Lemma 2

If t is a nonempty bag, then for any $y \in t$

$$(B(t) \gg B(s)) \subseteq (B(t \setminus \{y\}) \gg B(\{y\}))$$

Proof: Similar to lemma 1.

Lemma 3

$$B(u) \gg B(v) = (\exists x \rightarrow B(u \setminus \{x\}) \gg B(v)) \prod_{y \in u \setminus v} (!y \rightarrow (B(u) \gg B(v \setminus \{y\}) \wedge y \in v \rightarrow B(u \setminus \{y\}) \gg B(v)))$$

Proof: By induction on the size of the bag u

$$(0) \quad u = \cup$$

$$LHS = (\exists x \rightarrow B(\{x\}) \gg B(v)) \prod_{y \in v} (!y \rightarrow B(\cup) \gg B(v \setminus \{y\}))$$

expansion of \gg case (e)

(1) Assume that the conclusion is true when the size of u is m .

Now let us examine the case when the size of u is $m+1$.

$$\text{LHS} = (\exists x \rightarrow (B(u \cup \{x\}) \gg B(v))) \prod_{y \in v} !y \rightarrow (B(u) \gg B(v \cup \{y\}))$$

$$\prod_{z \in u} (B(u \cup \{z\}) \gg B(v \cup \{z\}))$$

expansion of \gg

$$= \prod_{z \in u} ((\exists x \rightarrow (B(u \cup \{x\}) \gg B(v))) \prod_{y \in v} !y \rightarrow (B(u) \gg B(v \cup \{y\})))$$

$$\prod (B(u \cup \{z\}) \gg B(v \cup \{z\}))$$

since u is nonempty and \prod distributes through \prod

$$= \prod_{z \in u} ((\exists x \rightarrow (B(u \cup \{x\}) \gg B(v))) \prod_{y \in v} !y \rightarrow (B(u) \gg B(v \cup \{y\})))$$

$$\prod (\exists x \rightarrow (B(u \cup \{z\} \cup \{x\}) \gg B(v \cup \{z\})))$$

$$\prod_{y \in u \cup \{z\}} !y \rightarrow ((B(u \cup \{z\}) \gg B(v \cup \{z\} \cup \{y\})) \wedge y \in v \rightarrow (B(u \cup \{z\} \cup \{y\}) \gg B(v \cup \{z\})))$$

$$(B(u \cup \{z\} \cup \{y\}) \gg B(v \cup \{z\})))$$

the inductive hypothesis and property (b) of bag

$$= \prod_{z \in u} (\exists x \rightarrow ((B(u \cup \{x\}) \gg B(v)) \wedge (B(u \cup \{z\} \cup \{x\}) \gg B(v \cup \{z\}))))$$

$$\prod_{y \in u \cup \{z\}} !y \rightarrow ((B(u) \gg B(v \cup \{y\})) \wedge y \in v \rightarrow ((B(u \cup \{y\}) \gg B(v))$$

$$\wedge y = z \rightarrow (B(u \cup \{z\} \cup \{y\}) \gg B(v \cup \{z\}))))$$

expansion of \prod

$$= \prod_{z \in u} (?x \rightarrow (B(u \uparrow x) \gg B(v)))$$

$$\prod_{y \in u \uparrow v} !y \rightarrow ((B(u) \gg B(v \uparrow y)) \wedge y \in v \wedge (B(u \uparrow y) \gg B(v)))$$

lemma 2 and property of \rightarrow

= RHS

\square is idempotent.

Theorem 2

$$BAG \gg BAG = BAG$$

Proof: For any bag t we define

$$A(t) = B(t) \gg BAG$$

It is easy to calculate that

$$\begin{aligned} A(\cup) &= ?x \longrightarrow B(\cup(x)) \gg \text{BAG} && \text{by taking } u = v = \cup \text{ in lemma 3} \\ &= ?x \longrightarrow A(\cup(x)) && \text{def of } A \end{aligned}$$

$$\begin{aligned} \text{and } A(t) &= (?x \longrightarrow (B(t \cup(x)) \gg \text{BAG})) \prod_{y \in t} !y \longrightarrow (B(t \cup(y)) \gg \text{BAG})) && \\ & && \text{by taking } v = \cup \text{ in lemma 3} \\ &= (?x \longrightarrow A(t \cup(x))) \prod_{y \in t} !y \longrightarrow A(t \cup(y)) && \\ & && \text{def of } A \text{ and } \prod_{y \in t} \end{aligned}$$

From the unique fixed point theorem it follows that for any bag t

$$A(t) = B(t)$$

By taking $t = \cup$, we have

$$\text{BAG} \gg \text{BAG} = A(\cup) = B(\cup) = \text{BAG}$$

Similarly we can prove

Theorem 3

$$\text{BAG} \gg \text{BUFFER} = \text{BAG}$$

Finally let us explore another property of BAG:

$$\text{COPY} \gg \text{BAG} = \text{BAG}$$

Lemma 4

$$(!y \longrightarrow \text{COPY}) \gg B(t) = \text{COPY} \gg B(t \cup(y))$$

This lemma states that though $B(t)$ is willing to output messages on its right channel, it does not matter if the internal communication between two operands takes place first; and the message y is transmitted to $B(t)$.

Proof: By induction on the size of the bag t .

$$(0) \quad t = \cup$$

$$\text{LHS} = \text{COPY} \gg B(\cup) \qquad \text{expansion of } \gg \text{ case (d)}$$

(1) Assume that the conclusion is true when the size of t is m . Then, for any bag t with size $m+1$ we have

$$\begin{aligned} \text{LHS} &= \left(\prod_{z \in t} !z \rightarrow (!y \rightarrow \text{COPY}) \gg B(t \cup \{z\}) \right) \searrow (\text{COPY} \gg B(t \cup \{y\})) \\ & \qquad \qquad \qquad \text{expansion of } \gg \text{ case (f)} \\ &= \left(\prod_{z \in t} !z \rightarrow \searrow (\text{COPY} \gg B(t \cup \{z\} \cup \{y\})) \right) \searrow (\text{COPY} \gg B(t \cup \{y\})) \\ & \qquad \qquad \qquad \text{the inductive hypothesis} \\ &= \left(\prod_{z \in t} !z \rightarrow (\text{COPY} \gg B(t \cup \{z\} \cup \{y\})) \right) \\ & \searrow (!x \rightarrow (!!x \rightarrow \text{COPY}) \gg B(t \cup \{y\})) \prod_{z \in t \cup \{y\}} !z \rightarrow \searrow (\text{COPY} \gg B(t \cup \{y\} \cup \{z\})) \\ & \qquad \qquad \qquad \text{expansion of } \gg \text{ case (e)} \\ &= (!x \rightarrow (!!x \rightarrow \text{COPY}) \gg B(t \cup \{y\})) \prod_{z \in t \cup \{y\}} !z \rightarrow (\text{COPY} \gg B(t \cup \{y\} \cup \{z\})) \\ & \qquad \qquad \qquad \text{expansion of the composite choice of pipes and property (a) of bag} \\ &= \text{COPY} \gg B(t \cup \{y\}) \\ & \qquad \qquad \qquad \text{expansion of } \gg \text{ case (e)} \end{aligned}$$

Theorem 4

$$\text{COPY} \gg \text{BAG} = \text{BAG}$$

Proof: Define $A(t) = \text{COPY} \gg B(t)$ for any bag t . Then we have

$$\begin{aligned} A(\cup) &= \text{COPY} \gg \text{BAG} && \text{def of } A \\ &= !x \rightarrow \searrow (!x \rightarrow \text{COPY}) \gg \text{BAG} && \text{expansion of } \gg \text{ case (c)} \\ &= !x \rightarrow A(\{x\}) && \text{lemma 4 and def of } A. \end{aligned}$$

$$\begin{aligned}
 \text{and } A(t) &= (\exists x \rightarrow (\exists x \rightarrow \text{COPY}) \gg B(t) \prod_{y \in t} !y \rightarrow \text{COPY} \gg B(t \cup \{y\})) \\
 &\qquad \qquad \qquad \text{expansion of } \gg \text{ case (a)} \\
 &= (\exists x \rightarrow A(t \cup \{x\}) \prod_{y \in t} !y \rightarrow A(t \cup \{y\})) \\
 &\qquad \qquad \qquad \text{lemma 4 and def } \prod_{y \in t}
 \end{aligned}$$

which implies that $A(t) = B(t)$ by the unique fixed point theorem.

In particular, by taking $t = \cup$ we finish the proof

Theorem 5

$$\text{BUFFER} \gg \text{BAG} = \text{BAG}$$

Proof: Similar to theorem 4.

3.2 Composition in parallel

Bags can be composed not only in series as shown in Section 3.1, but also in parallel with bags and with buffers. In this subsection we shall prove that

$$\begin{aligned}
 \text{BAG} &= \text{BAG} \parallel \text{COPY} \\
 &= \text{BAG} \parallel \text{BUFFER} \\
 &= \text{BAG} \parallel \text{BAG}
 \end{aligned}$$

As in the previous subsection, both sides of the equation will be reduced to guarded mutually recursive expressions formulated in terms of primitive operators. The only subtle point is to choose the appropriate processes $A(t)$.

First let us show that

$$\text{BAG} = \text{BAG} \parallel \text{BAG}$$

Lemma 5

$$\begin{aligned}
 B(u) \parallel B(v) &= (\exists x \rightarrow (B(u \uparrow x) \parallel B(v) \cap B(u) \parallel B(v \uparrow x))) \\
 &\quad \prod_{z \in V} (\exists z \rightarrow B(u) \parallel B(v \uparrow z)) \\
 &\quad \prod_{y \in U} (\exists y \rightarrow B(u \uparrow y) \parallel B(v))
 \end{aligned}$$

Proof: LHS = $(\exists x \rightarrow B(u \uparrow x) \parallel B(v))$

$$\begin{aligned}
 &\quad \prod (\exists x \rightarrow B(u) \parallel B(v \uparrow x)) \\
 &\quad \prod_{z \in V} (\exists z \rightarrow B(u) \parallel B(v \uparrow z)) \\
 &\quad \prod_{y \in U} (\exists y \rightarrow B(u \uparrow y) \parallel B(v)) \qquad \text{expansion of } \parallel \\
 &= \text{RHS} \qquad \text{property of } \prod
 \end{aligned}$$

Theorem 6

$$BAG \parallel BAG = BAG$$

Proof: For any bag t , we define

$$A(t) = \prod_{u \uparrow v = t} (B(u) \parallel B(v))$$

$$\begin{aligned}
 \therefore A(\mathcal{U}) &= BAG \parallel BAG && \text{def of } A \text{ and } BAG \\
 &= (\exists x \rightarrow ((B(\uparrow x) \parallel BAG) \cap (BAG \parallel B(\uparrow x)))) && \text{lemma 5} \\
 &= \exists x \rightarrow A(\uparrow x) && \text{def of } A \text{ and property (c) of bag.}
 \end{aligned}$$

$$\begin{aligned}
 A(t) &= \prod_{u \uparrow v = t} ((\exists x \rightarrow (B(u \uparrow x) \parallel B(v) \cap B(u) \parallel B(v \uparrow x))) \\
 &\quad \prod_{z \in V} (\exists z \rightarrow B(u) \parallel B(v \uparrow z)) \\
 &\quad \prod_{y \in U} (\exists y \rightarrow B(u \uparrow y) \parallel B(v)))
 \end{aligned}$$

lemma 5

$$\begin{aligned}
&= \bigcap_{u \cup v = t} ((?x \rightarrow \bigcap_{u \cup v = t} ((B(u) \cup \{x\}) \parallel B(v)) \cap (B(u) \parallel B(v \cup \{x\})))) \\
&\quad \bigcap_{z \in v} (!z \rightarrow (\bigcap_{z \in v} (B(u) \parallel B(v \cup \{z\})) \cap \bigcap_{u \cup v = t} (B(v \cup \{z\}) \parallel B(v)))) \\
&\quad \bigcap_{y \in u} (!y \rightarrow (\bigcap_{y \in u} (B(u \cup \{y\}) \parallel B(v)) \cap \bigcap_{u \cup v = t} (B(u) \parallel B(v \cup \{y\})))) \\
&\hspace{20em} \text{property of } \rightarrow \\
&= (?x \rightarrow A(t \cup \{x\})) \bigcap_{y \in t} (!y \rightarrow A(t \cup \{y\})) \\
&\hspace{20em} \text{def of } A \text{ and since } \cap \text{ is idempotent}
\end{aligned}$$

By the unique fixed point theorem we conclude that

$$A(t) = B(t) \quad \text{for any bag } t$$

When $t = \cup$ we obtain the conclusion

Lemma 6

$$\begin{aligned}
B(t) \parallel (!e \rightarrow \text{COPY}) &= (?x \rightarrow B(t \cup \{x\})) \parallel (!e \rightarrow \text{COPY}) \\
&\quad \bigcap_{!e \rightarrow B(t)} \parallel \text{COPY} \\
&\quad \bigcap_{y \in t} (!y \rightarrow (B(t \cup \{y\}) \parallel (!e \rightarrow \text{COPY})))
\end{aligned}$$

Proof: Similar to lemma 5.

Lemma 7

$$\begin{aligned}
B(t) \parallel \text{COPY} &= (?x \rightarrow ((B(t \cup \{x\}) \parallel \text{COPY}) \cap (B(t) \parallel (!x \rightarrow \text{COPY})))) \\
&\quad \bigcap_{y \in t} (!y \rightarrow (B(t \cup \{y\}) \parallel \text{COPY}))
\end{aligned}$$

Proof: Similar to lemma 5.

Theorem 7

$$\text{BAG} \parallel \text{COPY} = \text{BAG}$$

Proof: We define

$$A(\top) = \text{BAG} \parallel \text{COPY}$$

$$\text{and } A(t) = (\text{B}(t) \parallel \text{COPY}) \cap \left(\prod_{y \in t} \text{B}(t \cup \{y\}) \parallel (!y \rightarrow \text{COPY}) \right)$$

Then similar to theorem 6, we have

$$A(\top) = (?x \rightarrow ((\text{B}(\top) \parallel \text{COPY}) \cap (\text{BAG} \parallel (!x \rightarrow \text{COPY})))) \quad \text{Lemma 7}$$

$$= ?x \rightarrow A(\top) \quad \text{def of } A$$

$$A(t) = (?x \rightarrow ((\text{B}(t \cup \{x\}) \parallel \text{COPY}) \cap (\text{B}(t) \parallel (!x \rightarrow \text{COPY}))))$$

$$\prod_{y \in t} (!y \rightarrow \text{B}(t \cup \{y\}) \parallel \text{COPY})$$

$$\cap \prod_{y \in t} ((?x \rightarrow \text{B}(t \cup \{y\} \cup \{x\}) \parallel (!y \rightarrow \text{COPY}))$$

$$\prod !y \rightarrow \text{B}(t \cup \{y\}) \parallel \text{COPY})$$

$$\prod_{z \in t \cup \{y\}} !z \rightarrow \text{B}(t \cup \{y\} \cup \{z\}) \parallel (!y \rightarrow \text{COPY}))$$

Lemma 6 and Lemma 7

$$= (?x \rightarrow A(t \cup \{x\})) \prod_{y \in t} (!y \rightarrow A(t \cup \{y\}))$$

$$\cap \prod_{y \in t} (?x \rightarrow A(t \cup \{x\}) \prod !y \rightarrow A(t \cup \{y\}) \prod_{z \in t \cup \{y\}} !z \rightarrow A(t \cup \{z\}))$$

def of A, property of \rightarrow and property (a) of bag

$$= (?x \rightarrow A(t \cup \{x\}) \prod_{y \in t} !y \rightarrow A(t \cup \{y\}))$$

law of convexity

The definition of BUFFER and BUF is given in Section 1. We can show

Theorem 8

$$\text{BAG} \sqcup \text{BUFFER} = \text{BAG}$$

Finally we show that a buffer is a valid implementation of a bag.

Theorem 9

$$\text{BAG} \sqsubseteq \text{BUFFER}$$

Proof: For any bag t we define

$$A(t) = \bigsqcup_{\text{bag}(s)=t} \text{BUF}(s)$$

where the function bag is defined

$$\text{bag}(s) = \bigcup \{ s' \mid s \leq s' \} \cup \{ s_0 \}$$

Then we have

$$\begin{aligned} A(\bigcup) &= \text{BUF}(\langle \rangle) && \text{def of } A(\bigcup) \\ &= \exists x \rightarrow \text{BUF}(\langle x \rangle) && \text{def of } \text{BUF}(\langle \rangle) \\ &= \exists x \rightarrow A(\bigcup x) && \text{def of } A(\bigcup x) \end{aligned}$$

When t is a nonempty bag, we have

$$\begin{aligned} A(t) &= \bigsqcup_{\bigcup x \text{ bag}(s)=t} \text{BUF}(\langle y \rangle \hat{\ } s) && \text{def of } A(t) \\ &= \bigsqcup_{\bigcup x \text{ bag}(s)=t} (\exists x \rightarrow \text{BUF}(\langle y \rangle \hat{\ } s \langle x \rangle)) \sqcap \{ y \rightarrow \text{BUF}(s) \} && \text{def of } \text{BUF}(\langle y \rangle \hat{\ } s) \\ &\sqsubseteq \bigsqcup_{\bigcup x \text{ bag}(s)=t} (\exists x \rightarrow A(t \bigcup x)) \sqcap \{ y \rightarrow A(t \bigcup y) \} && \text{def of } A \\ &= (\exists x \rightarrow A(t \bigcup x)) \bigsqcap_{\text{yet}} \{ y \rightarrow A(t \bigcup y) \} && \text{def of } \bigsqcap_{\text{yet}} \end{aligned}$$

34.

By the unique fixed point theorem we conclude that

$$B(t) \subseteq A(t)$$

for any bag t

When $t = \cup$ we get

$$BAG = B(\cup) \subseteq A(\cup) = BUFFER$$

4. Comparison

Many proof methods have been proposed for distributed systems [1] [3]. In this section we compare our method with that presented in [1].

The essence of treatments presented in [1] and this paper is an axiomatic framework for the description of processes; verifications are to be done on the basis of the algebraic laws. The axiomatic framework described in [1] is ACP_{τ} , the algebra of communicating processes including silent steps. The model of concurrency we concentrate on is based on the mathematical theory of Communicating Sequential Processes [4]. Algebraic laws presented in [1] and [4] allow us to prove the properties of distributed systems by symbolic execution of atomic events. The general method of proving an equation is to reduce both sides of the equation to the same guarded recursive expression. The theorem of unique solution for guarded recursion, called the Recursive Specification Principle in [2], plays a key role in both treatments.

The differences between these two methods are the following:

(1) In our method, the alphabet is a permanent, predefined property of a process. The choice of an alphabet usually involves a deliberate simplification. On the other hand, some processes may never engage in a particular event in their alphabet. For instance, $STOP_A$ is used to describe the behaviour of a broken object with alphabet A , though it never actually engages in any of the events of A . In general, two processes can behave identically, but have different alphabets.

In contrast, the alphabet of a process P in [1] is determined by its behaviour. Formally it is specified by the alphabet function $\alpha: \mathcal{P} \rightarrow \mathcal{A}$, where \mathcal{P} denotes the set of processes, and \mathcal{A} the set

of all subsets of events in the finite universe. Intuitively, $\alpha(P)$ contains exactly those events in which P will actually engage. As a consequence, the alphabet of the deadlocked process STOP is empty. In order to find the alphabet of an infinite process from its specification and in order to apply some conditional axioms involving the alphabet, a theory about alphabet, called α/β calculus, was introduced in [1]. In general equality in $\alpha(P)$ is undecidable.

The predefined alphabet simplifies the definition and laws for the \parallel operator in [4]. This paper has used the interleaving operator $\parallel\parallel$ instead of \parallel , and so casts no light on the significance of the constant alphabet.

(2) An explicit symbol τ , denoting a silent or internal event, was described in [1]. It has a natural meaning. The drawback is of course that we lose abstraction from internal activity.

In our model, the internal event is taken to be wholly invisible, and wholly irrelevant to the logical correctness of a process. As a result, we have more laws; and theorems involving internal actions become slightly easier to prove.

(3) In CCS, the timing of the resolution of internal non-determinism is significant; so in CCS it is not true that

$$(a \rightarrow P) \parallel (a \rightarrow Q) = (a \rightarrow (P \cap Q))$$

Such distinctions cannot be drawn in CSP. In the examples of this paper, the CSP approach seems advantageous.

(4) In order to simplify the algebraic calculation in section 3, we have developed a calculus adapted more to the specific needs of the present problem, and introduced some new compact but complicated notation

in section 2. The expansion theorem for nondeterministic pipes helps to abbreviate proofs, and is applicable to a class of similar problems. We think that it may be necessary to derive laws special to each application in order to control the complexity of the proofs.

In [1] all proofs were based on the set of axioms and conditional axioms for the primitive operators of CCS; the number of laws was small, and the proofs were correspondingly slightly more complex.

(5) The laws of CSP are proved from a mathematical model, in the traditional manner of mathematics. The laws of CCS are defined by an ingenious congruence relation over a syntactic or operational model. The distinction is more significant in non-algebraic reasoning, and so is not apparent in this paper.

5. Acknowledgements

The original work on use of algebraic transformations for correctness proofs in [1] inspired us to work on this paper. The research of this paper was supported in part by the Science and Engineering Research Council of Great Britain.

References

- [1] J.C.M. Baeten, J.A. Bergstra, J.W. Klop
Conditional axioms and ω/β calculus in process algebra.
Report CS-R8502. Centre for Mathematics and Computer Science,
(1985). Amsterdam, The Netherlands.
- [2] J.A. Bergstra, J.W. Klop
Algebra of communicating processes with abstraction.
Report CS-R8403. Centrum voor Wiskunde en Informatica (1984).
Amsterdam, The Netherlands.
- [3] C.A.R. Hoare
A Calculus of Total Correctness for Communicating Processes.
SCP I (1981).
- [4] C.A.R. Hoare
Communicating Sequential Processes.
Prentice-Hall (1985).

Appendix

Definition

Let P and Q both be pipes. The process $P \gg Q$ is formally defined

$$P \gg Q = (P \text{ [mid/right] } \parallel Q \text{ [mid/left] }) \setminus \{ \text{mid} \}$$

where \setminus denotes the concealment operator, and the process $P \text{ [d/c]}$ behaves the same as P except that the channel c is renamed by d .

The following law of concealment in [4, 3.5] is useful in exploring the property of nondeterministic pipe:

(a) If $B \cap C$ is finite, then

$$(x:B \rightarrow P(x)) \setminus C = (x:B-C \rightarrow P(x) \setminus C) \coprod_{x \in B \cap C} (P(x) \setminus C)$$

We will refer the following law [4, 2.3.1 L7] in the proof of L2D of nondeterministic pipe

(b) Let $P = (x:A \rightarrow P(x))$

and $Q = (y:B \rightarrow Q(y))$

Then $(P \parallel Q) = (z:C \rightarrow P' \parallel Q')$

where $C = (A \cap B) \cup (A - \alpha Q) \cup (B - \alpha P)$

and $P' = P(z)$ if $z \in A$

$= P,$ otherwise

and $Q' = Q(z)$ if $z \in B$

$= Q$ otherwise

Now we offer a proof for L20 of nondeterministic pipe quoted in section 2.3

$$\text{L20 Let } P = (b1 \ \& \ \text{left?}x \longrightarrow P1(x)) \prod_{y \in S} (\text{right!}y \longrightarrow P2(y))$$

$$\text{and } Q = (c1 \ \& \ \text{left?}x \longrightarrow Q1(x)) \prod_{y \in T} (\text{right!}y \longrightarrow Q2(y))$$

$$\text{Then } P \gg Q = R \prod_{y \in C1 \ \& \ S} W(y)$$

$$\text{where } R = (b1 \ \& \ \text{left?}x \longrightarrow P1(x) \gg Q) \prod_{y \in T} (\text{right!}y \longrightarrow P \gg Q2(y))$$

$$\text{and } W(y) = P2(y) \gg Q1(y)$$

Proof:

$$\text{LHS} = ((b1 \ \& \ \text{left?}x \longrightarrow P1(x) \text{[mid/right]}) \prod_{y \in S} \text{mid!}y \longrightarrow P2(y) \text{[mid/right]})$$

$$\| (c1 \ \& \ \text{mid?}x \longrightarrow Q1(x) \text{[mid/left]}) \prod_{y \in T} \text{right!}y \longrightarrow Q2(y) \text{[mid/left]}) \setminus \{ \text{mid} \}$$

def of \gg

$$= (b1 \ \& \ \text{left?}x \longrightarrow P1(x) \text{[mid/right]}) \| Q \text{[mid/left]}$$

$$\prod_{y \in T} \text{right!}y \longrightarrow P \text{[mid/right]} \| Q2(y) \text{[mid/left]}$$

$$\prod_{y \in C1 \ \& \ S} \text{mid!}y \longrightarrow P2(y) \text{[mid/right]} \| Q1(y) \text{[mid/left]} \setminus \{ \text{mid} \}$$

by (b)

= RHS

by (e)

The following properties of the interleaving operator $\| \|$ [4, 3.6] will be used in the proof of the expansion theorem for interleaving of pipes.

(a) $\| \|$ distributes through \sqcap

(b) If $P = (x:A \rightarrow P(x))$

and $Q = (y:B \rightarrow P(y))$

then $P \| \| Q = (x:A \rightarrow (P(x) \| \| Q) \sqcap y:B \rightarrow (P \| \| Q(y)))$

L21 Let $P = (b1 \ \& \ \text{left?}x \rightarrow P1(x)) \prod_{y \in S} (!y \rightarrow P2(y))$

and $Q = (c1 \ \& \ \text{left?}x \rightarrow Q1(x)) \prod_{z \in T} (!z \rightarrow Q2(z))$

Then $P \| \| Q = (b1 \ \& \ \text{left?}x \rightarrow (P1(x) \| \| Q)$

$\sqcap c1 \ \& \ \text{left?}x \rightarrow (P \| \| Q1(x))$

$\prod_{y \in S} \text{right!}y \rightarrow (P2(y) \| \| Q)$

$\prod_{z \in T} \text{right!}z \rightarrow (P \| \| Q2(z))$

Proof: There are four different cases

(0) $S = T = \emptyset$

LHS = $(b1 \ \& \ \text{left?}x \rightarrow P1(x)) \| \| (c1 \ \& \ \text{left?}x \rightarrow Q1(x))$

def of $\prod_{y \in \emptyset}$

= RHS

property (b) of $\| \|$

(1) $S = \emptyset$ and $T \neq \emptyset$

$$\begin{aligned}
 \text{LHS} &= (b1 \ \& \ \text{left?}x \rightarrow p1(x)) \ || \ || \ \bigcap_{z \in T} (c1 \ \& \ \text{left?}x \rightarrow q1(x)) \ \bigcap \ \text{right!}z \rightarrow q2(z)) \\
 &\hspace{15em} \text{def of } \bigcap_{y \in \emptyset} \text{ and since } T \neq \emptyset \\
 &= \bigcap_{z \in T} ((b1 \ \& \ \text{left?}x \rightarrow p1(x)) \ || \ || \ (c1 \ \& \ \text{left?}x \rightarrow q1(x)) \ \bigcap \ \text{right!}z \rightarrow q2(z)) \\
 &\hspace{15em} \text{property (a) of } \ || \ || \\
 &= \bigcap_{z \in T} (b1 \ \& \ \text{left?}x \rightarrow (p1(x) \ || \ || \ (c1 \ \& \ \text{left?}x \rightarrow q1(x)) \ \bigcap \ \text{right!}z \rightarrow q2(z))) \\
 &\quad \bigcap \ c1 \ \& \ \text{left?}x \rightarrow (p \ || \ || \ q1(x)) \\
 &\quad \bigcap \ \text{right!}z \rightarrow (p \ || \ || \ q2(z)) \\
 &\hspace{15em} \text{property (b) of } \ || \ || \\
 &= \bigcap_{z \in T} (b1 \ \& \ \text{left?}x \rightarrow \bigcap_{z \in T} (p1(x) \ || \ || \ (c1 \ \& \ \text{left?}x \rightarrow q1(x)) \ \bigcap \ \text{right!}z \rightarrow q2(z))) \\
 &\quad \bigcap \ c1 \ \& \ \text{left?}x \rightarrow (p \ || \ || \ q1(x)) \\
 &\quad \bigcap \ \text{right!}z \rightarrow (p \ || \ || \ q2(z)) \\
 &\hspace{15em} \text{property of } \rightarrow \\
 &= \text{RHS} \hspace{15em} \text{property (a) of } \ || \ ||
 \end{aligned}$$

For the following cases

(2) $S \neq \emptyset$ and $T = \emptyset$

(3) $S \neq \emptyset$ and $T \neq \emptyset$

the reasoning process is similar to (1).

This completes the proof.

ALGEBRAIC SPECIFICATION AND PROOF
OF A DISTRIBUTED RECOVERY ALGORITHM

He, Jifeng and C.A.R. Hoare

Summary

An algebraic specification is given of an algorithm for recovery from catastrophe by a deterministic process. A second version of the algorithm also includes check-points. The algorithms are formulated in the notations of Communicating Sequential Processes [Hoare], and the proofs of correctness are conducted wholly by application of algebraic laws (together with the unique fixed point theorem).

1. Introduction

Algebraic specifications have been formalized and used for a number of years [Guttag and Horning] [Goguen, Thatcher and Wagner]. A problem that has emerged in their practical application is that it involves large numbers of mutually interactive equations, which cannot easily be seen to reflect the requirements. In this paper, present a problem in distributed computing, for which the specification can be clearly expressed by a single algebraic equation. Furthermore, the correctness of its solution is established by purely algebraic transformations similar to those proposed for functional programs [Backus] [Burstall and Darlington].

We shall use the notation of Communicating Sequential Processes [Hoare] to define the problem and its solution.

The problem is defined as follows: Let P be a deterministic pipe, i.e., a process which only has two channels in its alphabet, namely an input channel 'left' and an output channel 'right'. Let \downarrow be a symbol standing for a catastrophic event. We specify \hat{P} as a process which behaves like P until \downarrow occurs, and after each \downarrow behaves like \hat{P} from the start again. Formally, \hat{P} is defined in [Hoare] to satisfy the recursive equation.

$$\hat{P} = P \wedge (\downarrow \rightarrow \hat{P})$$

The infix \wedge denotes the interrupt operator, whose traces are simply defined

$$\text{traces}(P \wedge Q) = \left\{ s \wedge t \mid s \in \text{traces}(P) \wedge t \in \text{traces}(Q) \right\}$$

where $s \wedge t$ is the catenation of the two sequences s and t .

In general, for a long-lasting process P , a return to the start is not the most pleasant way to deal with catastrophe. It would be much better to return to the state just before the occurrence of \downarrow , i.e. on each occurrence of \downarrow P just carries on from where it has reached so far. The behaviour that we wish for can be specified as the arbitrary interleaving of the behaviour of P and that of a process which just engages in a series of \downarrow . This is just the behaviour of the process

$$P \parallel \text{RUN}_{\downarrow}$$

$$\text{where } \text{RUN}_{\downarrow} = \mu X. (\downarrow \rightarrow X)$$

and \parallel is parallel composition

In order to lend plausibility to $P \parallel \text{RUN}_{\downarrow}$ as a specification of what we want, we can prove using laws given in section 2 that whenever \downarrow occurs it behaves exactly the same afterwards as before, i.e., for all its traces s

$$(P \parallel \text{RUN}_{\downarrow})/s = (P \parallel \text{RUN}_{\downarrow})/s \wedge \langle \downarrow \rangle = (P/(s \hat{\alpha} P)) \parallel \text{RUN}$$

Here Q/s (for $s \in \text{traces}(Q)$) describes the behaviour of Q after engaging in the events recorded in the trace s .

Parallel composition is the same as interleaving in this case, since the alphabets of its operands are disjoint.

In summary, what we have been given is \hat{P} and what we want is $P \parallel \text{RUN}$. So the problem is to find some function F transforming the first into the second. In theory, the function F would be easy to define. (hint: $P \parallel \text{STOP}$ behaves the same as P). In practice, there are additional constraints to be satisfied, which will rule out such formal tricks as preventing the occurrence of \hat{P} altogether. In this particular case, the additional constraint is that \hat{P} be chained in between two pipes PRE and POST , which filter its input and output respectively. So the solution must take the form

$$\text{PRE} \gg \hat{P} \gg \text{POST}$$

where the chaining operator \gg is defined to link the output channel of its left operand to the input channel of its right operand, and to conceal the communications which pass on this internal channel.

Readers will notice that the introduction of the input interface PRE and the output interface POST involves at least one level of buffering on the input and output of \hat{P} . Therefore this fact must be reflected in our definitive statement of the problem:

Find processes PRE and POST such that for all deterministic P

$$\text{PRE} \gg \hat{P} \gg \text{POST} = (B \gg P \gg B) \parallel \text{RUN}$$

where B is the single - buffering process:

$$B = \text{left?x} \rightarrow \text{right!x} \rightarrow B$$

To simplify the solution, we suppose that \hat{P} is in the alphabet of the processes PRE and POST ; and we extend the definition of \gg in a natural way to ensure the occurrence of \hat{P} requires simultaneous participation of both its operands. Thus in the solution outlined above, whenever \hat{P} occurs, all three processes PRE , \hat{P} and POST know about it at once. Furthermore, we may assume that all pipes being examined are deterministic - a fact of which our solution will take advantage.

The solution method we adopt is one that is fairly widely used in interactive systems. PRE records all messages input; after an occurrence of \downarrow and before inputting any further messages, PRE feeds all these messages on again to \hat{P} (which has restarted as a result of the same occurrence of \downarrow). Similarly, POST keeps a count of all messages output, and ignores that number of messages output by \hat{P} after an occurrence of \downarrow . The only subtle point is to ensure the correct outcome even when \downarrow occurs in the middle of the recovery procedure. The formal solution of this problem and its proof of correctness are given in Section 3.

For a long-lasting process, the solution described above suffers from two severe drawbacks:

1. The delay involved in recovery grows linearly with the passage of time.
2. The storage required by PRE also grows linearly (and POST logarithmically).

The solution to these problems is to introduce a checkpoint facility, triggered by a special event \odot . The process $\text{Ch}(P)$ is defined to behave like \hat{P} , except that each occurrence of \downarrow sends it back to its state just after the most recent occurrence of \odot , or to the beginning if \odot has not yet occurred. Any occurrence of \odot has no other effect on the behaviour of P . This operation is defined in [Hoare] on deterministic processes by the following laws

$$\text{L1 } \text{Ch}(P) = \text{Ch}_2(P, P)$$

$$\text{L2 } \text{if } P = (x:B \rightarrow P(x))$$

$$\text{then } \text{Ch}_2(P, Q) = (x:B \rightarrow \text{Ch}_2(P(x), Q))$$

$$\boxed{\downarrow} \rightarrow \text{Ch}_2(Q, Q)$$

$$\boxed{\odot} \rightarrow \text{Ch}_2(P, P)$$

Here L2 is suggestive of the standard implementation method. P is the current process and Q is the checkpointed process, waiting to be reinstated on the next occurrence of \downarrow , or superseded on the next occurrence of \odot .

The improved solution can now be specified:

Find a pair of pipes PRE and POST, containing both \downarrow and \uparrow in their alphabet, such that for all deterministic P

$$\text{PRE} \gg \text{Ch}(P) \gg \text{POST} = (\text{B} \gg P \gg \text{B}) \parallel \text{RUN} \left\{ \downarrow, \uparrow \right\}$$

The solution and its proof are given in section 4.

Before embarking on the proofs, section 2 contains a summary of the algebraic laws which will be used. Some of them are somewhat simpler and/or stronger than those of [Hoare] because they apply only to deterministic processes, which are therefore free of divergence.

In future we shall use the following abbreviations:

$$B_x = B / \langle ?x \rangle \quad (= !x \rightarrow B)$$

$$\text{BPB} = B \gg P \gg B$$

$$B(P/s)B = B \gg (P/s) \gg B$$

$$B_x(P/s)B_y = B_x \gg (P/s) \gg B_y, \text{ etc}$$

We also define

$$\text{PRUN} \triangleq \text{RUN} \parallel \text{STOP} \{ \text{left}, \text{right} \}$$

This is the pipe which has channels left and right in its alphabet, but never uses them; it only engages (forever) in the \downarrow event

Oxford University
 Computing Laboratory
 Programming Research Group-Library
 8-11 Keble Road
 Oxford OX1 3QD
 Oxford (0965) 54141

2. Deterministic Pipes

In this section we are concerned with processes which input only on a channel named 'left' and output only on a channel named 'right'. Such processes are called pipes. We also allow pipe to engage in events from a fixed alphabet A . Thus our definition is slightly more general than that given in [Hoare].

Two pipes P and Q may be joined together so that the output channel of P is connected to the input channel of Q , and the sequence of messages output by P and input by Q on this internal channel is concealed from their common environment. Furthermore any event in A requires simultaneous participation of both P and Q . The result of connection is denoted

$$P \gg Q$$

We will save space by omitting the channel names 'left' and 'right' from input and output commands. We also suppose that all pipes being examined in the rest of this paper are deterministic, and do not diverge.

Now we intend to explore algebraic laws of the chaining operator \gg , which are based on the following basic laws presented in [Hoare].

(1) Law of general choice [Hoare, 3.3.1 L5]

$$\begin{aligned} (x:A \rightarrow P(x)) \square (y:B \rightarrow Q(y)) = \\ (z:(A \cup B) \rightarrow (\text{if } z \in (A-B) \text{ then } P(z) \\ \text{else if } z \in (B-A) \text{ then } Q(z) \\ \text{else if } z \in (A \cap B) \text{ then } (P(z) \sqcap Q(z))) \end{aligned}$$

In particular, when the left-hand side is known to be a deterministic process, then the term $(P(z) \sqcap Q(z))$ in the right-hand side can be replaced by either $P(z)$ or $Q(z)$. (If $A \cap B$ is nonempty, $P(z)$ equals $Q(z)$, since otherwise the left hand side would be nondeterministic.)

(2) Laws for the after operator [Hoare, 1.8.3, L1-L3 and 2.6.1 L7]

(a) $P/\langle \rangle = P$

(b) $P/(s \hat{\ } t) = (P/s)/t$

(c) $(x:B \rightarrow P(x))/\langle c \rangle = P(\langle c \rangle)$ provided that $c \in B$

where P/s is the behaviour of P after engaging in the events of the trace s , and it is undefined if s is not a trace of P .

(d) $f(P)/F^*(s) = f(P/s)$

where f is an injection from the alphabet of P onto a set of symbols S , and the process $f(P)$ is defined as one which engages in the event $f(c)$ whenever P would have engaged in c . The starred function F^* is defined by the following laws

$F^*(\langle \rangle) = \langle \rangle$

$F^*(\langle x \rangle \hat{\ } u) = \langle f(x) \rangle \hat{\ } F^*(u)$

(3) Laws of concurrency [Hoare, 2.3.1 L7 and 2.3.3 L2]

(a) Let $P = (x:B \rightarrow P(x))$

and $Q = (y:C \rightarrow Q(y))$

Then $(P \parallel Q) = (z:D \rightarrow P' \parallel Q')$

where $D = (B \cap C) \cup (B - \alpha Q) \cup (C - \alpha P)$

and $P' = P(z)$ if $z \in B$

$= P$ otherwise

and $Q' = Q(z)$ if $z \in C$

$= Q$ otherwise

and αP denotes the alphabet of P .

(b) $(P \{b\} Q) \parallel R = (P \parallel R) \{b\} (Q \parallel R)$

where $P \{b\} Q = \text{if } b \text{ then } P \text{ else } Q$

(c) $(P \parallel Q)/s = (P/(s \hat{\ } \alpha P)) \parallel (Q/(s \hat{\ } \alpha Q))$

where the expression $(s \hat{\ } B)$ denotes the trace s when restricted to events in the set B .

(4) Laws of chaining

The introduction of Boolean guards [INMOS] is a great help in reducing the number of laws needed and the size of the calculations which use them. A boolean guarded command is simply defined

$$b \& P = P \{b\} \text{ STOP}$$

An immediate advantage of this notation is that it enables us to represent every deterministic pipe P in a fixed representation

$$P = (b1 \& ?x \rightarrow P1(x)) \square (b2 \& !e \rightarrow P2) \square (y:B \rightarrow P3(y))$$

The special case when P cannot initially input is dealt with by setting $b1$ to false, so that the first clause reduces to STOP , which disappears because it is a unit of \square . Similarly initial output is prevented by setting $b2$ false, and initial participation is an event not possible when B is empty.

In [Hoare] there are eight laws for chaining (4.4.1). Use of Boolean guards enable these to be reduced to a single expansion law, which serves as an algebraic definition of the chaining operator.

- (a) Let $P = (b1 \& ?x \rightarrow P1(x)) \square (b2 \& !e \rightarrow P2) \square (y:B \rightarrow P3(y))$
 and $Q = (c1 \& ?x \rightarrow Q1(x)) \square (c2 \& !f \rightarrow Q2) \square (y:C \rightarrow Q3(y))$

Then $P \gg Q = ((T \square U) \square (b2 \wedge c1) \{T$

where $T = (b1 \& ?x \rightarrow (P1(x) \gg Q)$

$$\square (c2 \& !f \rightarrow \cancel{P \gg Q2})$$

$$\square (y:B \cap C \rightarrow \cancel{P3(y) \gg Q3(y)})$$

and $U = (b2 \wedge c1) \& (P2 \gg Q1(e))$

The first line of the definition of T describes case when the external input by P takes place first; in the second line the external output by Q takes place first; and the third line describes simultaneous participation by P and Q in an external event in which both are ready to engage.

The definition of U describes the case in which the internal communication takes place first, so that the value of e is transmitted from P to Q , but the communication is concealed. In all four cases, the process or processes which engage in the initial event make the appropriate progress, and they continue to be chained by \gg . A proof of this law is given in the appendix.

The main difficulty and complexity in the above law is the clause $(T \parallel U) \sqcap U$ which results from the hiding of an internal event [Hoare, 3.5.1 L10]. Fortunately, if $P \gg Q$ is known to be deterministic (and therefore free from divergence) we can simplify the statement of the law to

$$P \gg Q = T \parallel U$$

Proof: when $b2 \wedge c1$ is false, $U = \text{STOP}$ and $T \parallel U = T$. In the other case, $(T \parallel U) \sqcap U = T \parallel U$, since (because of determinism) the two operands of \sqcap are equal.

- (c) We come next to laws which show how the after operator distributes through chaining. The proofs of these laws are given in appendix.

A deterministic pipe $P \gg Q$ may engage in an external event x if both P and Q are ready for it. In this case, both operands of \gg make the appropriate progress, and continue to be connected by \gg . Formally, this is described by the law.

$$(P \gg Q) / (x) = (P / \langle x \rangle) \gg (Q / \langle x \rangle) \quad \text{provided that } x \in A \text{ and } \langle x \rangle \in \text{traces}(P) \cap \text{traces}(Q)$$

If P is ready to output a sequence of messages u to Q , and Q is willing to accept this sequence from P , then the internal communications take place, so that the messages in u are transmitted from P to Q , but the communications are concealed. The following law is just an obvious formalization of the informal description in terms of symbolic execution.

$$P \gg Q = (P / \text{right}.u) \gg (Q / \text{left}.u) \quad \text{provided that } \text{right}.u \in \text{traces}(P) \text{ and } \text{left}.u \in \text{traces}(Q)$$

where $\text{right}.u$ and $\text{left}.u$ are defined

$$\begin{aligned} \text{right}.u &= \langle \rangle && \text{if } u = \langle \rangle \\ &= \langle \text{right}.u_0 \rangle \wedge \text{right}.(u') && \text{otherwise} \\ \text{left}.u &= \langle \rangle && \text{if } u = \langle \rangle \\ &= \langle \text{left}.u_0 \rangle \wedge \text{left}.(u') && \text{otherwise} \end{aligned}$$

where u_0 and u' denotes the head and the tail of the sequence u respectively.

The laws given above for a deterministic chain generalise to three operands

$$(P \gg Q \gg R) / \langle x \rangle = (P / \langle x \rangle) \gg (Q / \langle x \rangle) \gg (R / \langle x \rangle)$$

provided that $x \in A$ and $\langle x \rangle \in \text{traces}(P) \cap \text{traces}(Q) \cap \text{traces}(R)$

If s is a trace of Q , and P is willing to offer those input messages in s to Q , and R is ready to accept those output messages in s from Q , then the internal communications may take place. Furthermore, after three operands make progress, they will still be connected by \gg . This informal description is most succinctly formalized in the law

$$(P \gg Q \gg R) = (P/\text{right.ins}(s)) \gg (Q/s) \gg (R/\text{left.outs}(s))$$

provided that $s \in \text{traces}(Q)$ and $\text{right.ins}(s) \in \text{traces}(P)$ and $\text{left.outs}(s) \in \text{traces}(R)$, where $\text{ins}(s) = s \downarrow \left\{ \text{left} \right\}$ is the sequence of values input in the trace s , and $\text{outs}(s) = s \downarrow \left\{ \text{right} \right\}$ is the sequence of values output in the trace s .

3. Recoverable Processes

We return now to the first recovery problem, that of finding pipes PRE and POST. PRE is conveniently defined by mutual recursion with two parameters:

- u the sequence of all values input so far
 - v the sequence of values that must be output before the next input takes place
- Similarly, POST maintains two counts,
- n the number of all outputs so far
 - m the number of inputs that must be ignored before the next output is copied.

The processes PRE and POST are defined:

$$PRE = PRE (\langle \rangle, \langle \rangle)$$

$$PRE (\langle \rangle, u) = (?x \longrightarrow PRE (\langle x \rangle, u \wedge \langle x \rangle)) \square \downarrow \longrightarrow PRE (u, u)$$

$$PRE (\langle x \rangle \wedge v, u) = (!x \longrightarrow PRE (v, u)) \square \downarrow \longrightarrow PRE (u, u)$$

$$POST = POST (0, 0)$$

$$POST (0, m) = (?x \longrightarrow (!x \longrightarrow POST (0, m+1))) \square \downarrow \longrightarrow POST (m, m)$$

$$\square \downarrow \longrightarrow POST (m, n)$$

$$POST (n+1, m) = (?x \longrightarrow POST (n, m)) \square \downarrow \longrightarrow POST (m, m)$$

where u and v denote the sequence of messages.

The purpose of PRE(v,u) is first to output the messages recorded in v, and then to behave like PRE(⟨⟩,u). Similarly, the purpose of POST(n,m) is to input and ignore any sequence of n messages and then behave like POST(0,m). These facts can be formalized and proved as simple lemmas

Lemma 1

$$(a) \quad PRE(u,v)/right.u = PRE (\langle \rangle, v)$$

$$(b) \quad POST(\#v, \#u)/left.u = POST(0, \#u)$$

where right.u is the trace consisting of outputs of all the messages in v, and left.v is the trace consisting of all the messages in v, and #u is the length of the sequence u.

Proof:

$$(a) \quad \text{By induction on the length of } u$$

$$(0) \quad \text{For } u = \langle \rangle$$

$$PRE (\langle \rangle, v) / right.\langle \rangle = PRE (\langle \rangle, v) / \langle \rangle \quad \text{def of } right.\langle \rangle$$

$$= PRE (\langle \rangle, v)$$

L2(a)

(1) Assume the inductive hypothesis

$$\text{PRE}(u, v) / \text{right}.u = \text{PRE}(\langle \rangle, v)$$

for $\#u = n$

$$\therefore \text{PRE}(\langle x \rangle^{\wedge} u, v) / \text{right}.(\langle x \rangle^{\wedge} u) = \text{PRE}(\langle x \rangle^{\wedge} u, v) / (\langle \text{right}.x \rangle^{\wedge} \text{right}.u)$$

def of $\text{right}(\langle x \rangle^{\wedge} u)$

$$= (\text{PRE}(\langle x \rangle^{\wedge} u, v) / \langle \text{right}.x \rangle) / \text{right}.u \quad \text{L2(b)}$$

$$= \text{PRE}(u, v) / \text{right}.u$$

L2(c) and definition of
PRE

$$= \text{PRE}(\langle \rangle, v)$$

the inductive
assumption

(b) Similar to (a)

In the following part of this section, we intend to show that

$$\text{PRE} \gg \hat{P} \gg \text{POST} = \text{BPB} \parallel \text{RUN} \downarrow$$

The technique of the proof is one of quite general applicability: we show that each side of the equation is a solution of the same set of guarded mutually recursive definitions. In order to formulate these equations, we need to choose an appropriate set of indices, where there is at least one index for each 'state' of the process. A common strategy is to use the traces of the process itself as an indexing set, or as its main component. We deal with the right hand side of the equation first

3.1 The right hand side of the equation

First of all we define for any trace s of p

$$A(s) = B(P/s) B \parallel \text{RUN} \downarrow$$

$$C(x, s) = B_x(P/s) B_y \parallel \text{RUN} \downarrow$$

$$\text{for } \langle !y \rangle = \text{last}(s \wedge \{\text{right}\})$$

By taking $s = \langle \rangle$ we get the initial equation

$$\text{BPB} \parallel \text{RUN} \downarrow = A(\langle \rangle)$$

For convenience we introduce, for any pipe P a pair of predicates $r^?$ and $r^!$ to indicate whether P is ready for input or output.

$$r^? = \exists m. \langle ?m \rangle \in \text{traces}(P)$$

$$r^! = \exists m. \langle !m \rangle \in \text{traces}(P)$$

In general we define for any trace s of P

$$r_s^? = \exists m. s \wedge \langle ?m \rangle \in \text{traces}(P)$$

$$r_s^! = \exists m. s \wedge \langle !m \rangle \in \text{traces}(P)$$

Lemma 2

$$\text{BPB} = \{ ?x \rightarrow \langle \text{BP} / \langle ?x \rangle \text{B} \uparrow r^2 \uparrow \langle \text{B}_x \text{P} / \langle !e \rangle \text{B}_e \uparrow r^1 \uparrow \text{STOP} \rangle \} \\ \uparrow r^1 \&!e \rightarrow \langle \text{BP} / \langle !e \rangle \text{B} \rangle$$

Proof:

$$\text{BPB} = \{ ?x \rightarrow \text{B}_x \text{PB} \uparrow r^1 \quad \& \quad \text{BP} / \langle !e \rangle \text{B}_e \} \quad \text{L4(b)} \\ = \{ ?x \rightarrow \langle r^2 \rightarrow \text{BP} / \langle ?x \rangle \text{B} \uparrow r^1 \rightarrow \text{B}_x \text{P} / \langle !e \rangle \text{B}_e \} \quad \uparrow r^1$$

$$\uparrow r^1 \& \{ ?x \rightarrow \text{B}_x \text{P} / \langle !e \rangle \text{B}_e \} \uparrow !e \rightarrow \text{BP} / \langle !e \rangle \text{B} \} \quad \text{L4(b)} \\ = \text{RBS} \quad \text{L1}$$

Lemma 3

$$\text{B}_x \text{PB}_y = \{ !y \rightarrow \langle \text{BP} / \langle ?x \rangle \text{B} \uparrow r^2 \uparrow \langle \text{B}_x \text{P} / \langle !e \rangle \text{B}_e \uparrow r^1 \uparrow \text{STOP} \rangle \} \\ \uparrow r^2 \& ?z \rightarrow \text{B}_z \text{P} / \langle ?x \rangle \text{B}_y$$

Proof:

Similar to lemma 2.

Lemma 4

$$(a) \text{A}(s) = \{ ?x \rightarrow \langle \text{A}(s \wedge \langle ?x \rangle) \uparrow r_s^2 \uparrow \langle \text{C}(x, s \wedge \langle !e \rangle) \uparrow r_s^1 \uparrow \text{PRUN} \rangle \} \\ \uparrow r_s^1 \&!e \rightarrow \text{A}(s \wedge \langle !e \rangle) \\ \uparrow \downarrow \rightarrow \text{A}(s) \quad \text{for } s \in \text{traces}(P)$$

$$(b) \text{C}(x, s) = \{ !y \rightarrow \langle \text{A}(s \wedge \langle ?x \rangle) \uparrow r_s^2 \uparrow \langle \text{C}(x, s \wedge \langle !e \rangle) \uparrow r_s^1 \uparrow \text{PRUN} \rangle \} \\ \uparrow r_s^2 \& ?z \rightarrow \text{C}(z, s \wedge \langle ?x \rangle) \\ \uparrow \downarrow \rightarrow \text{C}(x, s) \quad \text{for } s \in \text{traces}(P) \\ \text{and } \langle !y \rangle = \text{last}(s \wedge \{\text{right}\})$$

Proof (a)

$$\text{A}(s) = \text{BP} / s \text{B} \uparrow \text{RUN} \downarrow \quad \text{def of A}(s) \\ = \{ ?x \rightarrow \langle \text{BP} / s \wedge \langle ?x \rangle \text{B} \uparrow r_s^2 \uparrow \langle \text{B}_x \text{P} / s \wedge \langle !e \rangle \text{B}_e \uparrow r_s^1 \uparrow \text{STOP} \rangle \} \uparrow \text{RUN} \downarrow \\ \uparrow r_s^1 \&!e \rightarrow \text{BP} / s \wedge \langle !e \rangle \text{B} \uparrow \text{RUN} \downarrow \\ \uparrow \downarrow \rightarrow \text{BP} / s \text{B} \uparrow \text{RUN} \downarrow \quad \text{lemma 2 and L3(a)} \\ = \text{RHS} \quad \text{L1(b)}$$

(b) Similar to (a)

We have now reduced the right hand side of our equation to a set of guarded mutually recursive definitions of A and C. As is quite usual, these equations are formulated in terms of elementary operators $\square, \rightarrow, \&$. They do not contain \parallel, \gg or hiding, so they conceal the original process structure of the formula. That is why they are useful in proving identity of formulae with radically differing process structures.

The time has come to apply the same technique to the left hand side of the equation. If we can derive the same set of mutually recursive definitions, then an appeal to the unique fixed point theorem completes the proof of the solution.

3.2 The left hand side

Similar to A(s) and C(x,s), the processes A'(s) and C'(x,s) are defined for any trace s of P:

$$A'(s) = \text{PRE}(\langle \rangle, \text{ins}(s)) \gg (P/s^{\wedge} (\hat{y} \rightarrow \hat{P})) \gg \text{POST}(0, \# \text{outs}(s))$$

$$C'(x,s) = \text{PRE}(\langle x \rangle, \text{ins}(s)^{\wedge}(x)) \gg (P/s^{\wedge} (\hat{y} \rightarrow \hat{P})) \gg \text{POST}_Y(0, \# \text{outs}(s)-1)$$

where $\text{POST}_Y(0,n) = (\{y \rightarrow \text{POST}(0,n+1)\})^{\wedge} \hat{y} \rightarrow \text{POST}(n,n)$

and $\langle y \rangle = \text{last}(s \uparrow \{\text{right}\})$

First, let us show that occurrence of \hat{y} has no effect on the behaviour of the processes A'(s) and C'(x,s)

Lemma 5

(a) $A'(s) / \langle \hat{y} \rangle = A'(s)$

(b) $C'(x,s) / \langle \hat{y} \rangle = C'(x,s)$

Proof:

(a) LHS = $(\text{PRE}(\langle \rangle, \text{ins}(s)) / \langle \hat{y} \rangle) \gg ((P/s^{\wedge} (\hat{y} \rightarrow \hat{P})) / \langle \hat{y} \rangle) \gg (\text{POST}(0, \# \text{outs}(s)) / \langle \hat{y} \rangle)$

L4(C)

= $\text{PRE}(\text{ins}(s), \text{ins}(s)) \gg \hat{P} \gg \text{POST}(\# \text{outs}(s), \# \text{outs}(s))$

L2(C) and definition

PRE, POST and P

= $(\text{PRE}(\text{ins}(s), \text{ins}(s)) / \hat{y} \uparrow \text{ins}(s)) \gg (P/s) \gg (\text{POST}(\# \text{outs}(s), \# \text{outs}(s)) / \text{left_out}(s))$

L4(C)

= A'(s)

Lemma 1 and definition of \hat{P}

(b) Similar to (a)

Corollary

If neither $r_s^?$ nor $r_s^!$ is true, then

$$A'(s) = \text{RUN}_{\emptyset}^!$$

Proof

$$\begin{aligned} A'(s) &= \begin{cases} \xrightarrow{\emptyset} \text{PRE}(\text{ins}(s), \text{ins}(s)) \gg \hat{P} \gg \text{POST}(\#\text{outs}(s), \#\text{outs}(s)) & \text{L4 (b)} \\ \xrightarrow{\emptyset} \Lambda'(s) & \text{Lemma 5} \end{cases} \end{aligned}$$

Now is the time to reduce the left hand side of the equation to a set of guarded mutually recursive definitions of A' and C' , and to show that both sides of the equation meet the same set of recursive definitions.

Lemma 6

The processes $A'(s)$ and $C'(x, s)$ meet the same guarded recursive equations as $A(s)$ and $C(s)$.

Proof: (a)

$$\begin{aligned} A'(s) &= (?x \rightarrow \text{PRE}(\langle x \rangle, \text{ins}(s) \hat{\langle x \rangle}) \gg \{(P/s) \hat{\langle \emptyset \rangle} \rightarrow P\} \gg \text{POST}(0, \#\text{outs}(s)) \\ &\quad \sqcup \xrightarrow{\emptyset} \text{PRE}(\text{ins}(s), \text{ins}(s)) \gg \hat{P} \gg \text{POST}(\#\text{outs}(s), \#\text{outs}(s)) \\ &\quad \sqcup r_s^! \& \text{PRE}(\langle \rangle, \text{ins}(s)) \gg \{(P/s \hat{\langle !e \rangle}) \hat{\langle \emptyset \rangle} \rightarrow \hat{P}\} \gg \text{POST}_e(0, \#\text{outs}(s)) \\ &\hspace{15em} \text{L4 (b)} \\ &= (?x \rightarrow (r_s^? \& (\text{PRE}(\langle \rangle, \text{ins}(s) \hat{\langle x \rangle}) \gg \{(P/s \hat{\langle ?x \rangle}) \hat{\langle \emptyset \rangle} \rightarrow \hat{P}\}) \gg \text{POST}(0, \#\text{outs}(s)) \\ &\quad \sqcup r_s^! \& (\text{PRE}(\langle x \rangle, \text{ins}(s) \hat{\langle x \rangle}) \gg \{(P/s \hat{\langle !e \rangle}) \hat{\langle \emptyset \rangle} \rightarrow \hat{P}\}) \gg \text{POST}_e(0, \#\text{outs}(s)) \\ &\quad \sqcup \xrightarrow{\emptyset} \text{PRE}(\text{ins}(s) \hat{\langle x \rangle}, \text{ins}(s) \hat{\langle x \rangle}) \gg \hat{P} \gg \text{POST}(\#\text{outs}(s), \#\text{outs}(s))) \\ &\quad \sqcup \xrightarrow{\emptyset} A'(s) \\ &\quad \sqcup r_s^! \& (?x \rightarrow \text{PRE}(\langle x \rangle, \text{ins}(s) \hat{\langle x \rangle}) \gg \{(P/s \hat{\langle !e \rangle}) \hat{\langle \emptyset \rangle} \rightarrow \hat{P}\}) \gg \text{POST}_e(0, \#\text{outs}(s)) \\ &\quad \quad \sqcup !e \rightarrow \text{PRE}(\langle \rangle, \text{ins}(s)) \gg \{(P/s \hat{\langle !e \rangle}) \hat{\langle \emptyset \rangle} \rightarrow \hat{P}\} \gg \text{POST}(0, \#\text{outs}(s)+1) \\ &\quad \quad \sqcup \xrightarrow{\emptyset} \text{PRE}(\text{ins}(s), \text{ins}(s)) \gg \hat{P} \gg \text{POST}(\#\text{outs}(s), \#\text{outs}(s))) \\ &\hspace{15em} \text{Lemma 5 and L4 (b)} \\ &= (?x \rightarrow (A'(s \hat{\langle ?x \rangle}) \{r_s^? \} (C'(x, s \hat{\langle !e \rangle}) \{r_s^! \} \text{PRUN}_{\emptyset}^{\emptyset})) \end{aligned}$$

$$\sqcup r_s^! \& !e \rightarrow \Lambda'(s \hat{\langle !e \rangle})$$

$$\sqcup \xrightarrow{\emptyset} A'(s)$$

Corollary of lemma 5 and L1

Theorem 2

$$\text{BFB} \mid \text{RUN} \downarrow = \text{PRE} \gg \hat{P} \gg \text{POST}$$

Proof:

From lemma 6 and the unique fixed point theorem it follows that

$$A'(s) = A(s) \quad \text{for } s \in \text{traces}(P)$$

By taking $s = \langle \rangle$, we complete the proof

4. Recoverable Processes with Checkpoints

This section is devoted to the second recovery problem, that of finding a pair of pipes of PRE and POST, containing both \downarrow and \odot in their alphabet such that for all deterministic P

$$\text{PRE} \gg \text{Ch}(P) \gg \text{POST} = \text{BPB} \parallel \text{RUNG}(\downarrow, \odot)$$

Here PRE has the same parameters as that defined in Section 3. But POST is with an extra parameter u, which records the sequence of messages which has input but not yet output.

The technique adopted in Section 3 will be used again; we will show that each side of the equation is a solution of the same set of mutually recursive equations, and choose the traces of P as the main part of an indexing set for the equations.

Definition

The processes PRE and POST are defined

$$\text{PRE} = \text{PRE}(\langle \rangle, \langle \rangle)$$

$$\begin{aligned} \text{PRE}\{u, v\} = & (\downarrow \rightarrow \text{PRE}(v, v) \\ & \parallel \odot \rightarrow \text{PRE}(u, u) \\ & \parallel u \neq \langle \rangle \ \& \ !u \rightarrow \text{PRE}(u', v) \\ & \parallel u = \langle \rangle \ \& \ ?u \rightarrow \text{PRE}(\langle x \rangle, v \wedge \langle x \rangle)) \end{aligned}$$

$$\text{POST} = \text{POST}(\langle \rangle, 0, 0)$$

$$\begin{aligned} \text{POST}(u, n, m) = & (\downarrow \rightarrow \text{POST}(u, m, m) \\ & \parallel \odot \rightarrow \text{POST}(u, n, n) \\ & \parallel u \neq \langle \rangle \ \& \ !u \rightarrow \text{POST}(\langle \rangle, n, m) \\ & \parallel (u = \langle \rangle \ \vee \ n \neq 0) \ \& \ ?x \rightarrow (\text{POST}(u, n-1, m) \{n \neq 0\} \text{POST}(\langle x \rangle, 0, m+1))) \end{aligned}$$

where $n, m \geq 0$ and u, v denote the sequence of messages.

Furthermore, for any trace s of P we define

$$Q'(s, t) = \text{PRE}(\langle \rangle, \text{ins}(s-t)) \gg \text{Ch}_2(P/s, P/t) \gg \text{POST}(\langle \rangle, 0, \#\text{outs}(s-t))$$

for $t \leq s$

$$R'(x, s, t) = \text{PRE}(\langle x \rangle, \text{ins}(s-t) \wedge \langle x \rangle) \gg \text{Ch2}(P/s, P/t) \gg \text{POST}(\langle y \rangle, 0, \#\text{outs}(s-t)!)$$

For $t \leq s$

and $\langle !y \rangle = \text{last}(s \uparrow \{\text{right}\})$

where $s-t$ is the suffix of s obtained by removing t from s .

By taking $s = t = \langle \rangle$ we get the initial equation

$$\text{PRE} \gg \text{Ch}(P) \gg \text{POST} = Q'(\langle \rangle, \langle \rangle)$$

Similar to lemma 1 and lemma 5 we can show the following results

Lemma 7

$$(a) \text{PRE}(u, v) / \text{right}.u = \text{PRE}(\langle \rangle, v)$$

$$(b) \text{POST}(S, \#u, \#v) / \text{left}.u = \text{POST}(S, 0, \#v)$$

Lemma 8

$$(a) Q'(s.t) / \langle ! \rangle = Q'(s.t)$$

$$(b) R'(x, s, t) / \langle ! \rangle = R'(x, s, t)$$

Corollary

If neither $r_s^?$ nor $r_s^!$ is true, then

$$Q'(s, t) = \text{PRUN} \langle ! \rangle, \textcircled{C}$$

Lemma 9

$$(a) Q'(s, t) = (?x \rightarrow (Q'(s \wedge \langle ?x \rangle, t) \uparrow r_s^? \uparrow (R'(x, s \wedge \langle !e \rangle, t) \uparrow r_s^! \uparrow \text{PRUN} \langle ! \rangle, \textcircled{C})))$$

$$\boxed{r_s^! \wedge !e \rightarrow Q'(s \wedge \langle !e \rangle, t)}$$

$$\boxed{\langle ! \rangle \rightarrow Q'(s, t)}$$

$$\boxed{\textcircled{C} \rightarrow Q'(s, s)}$$

$$(b) R'(x, s, t) = (!y \rightarrow (Q'(s \wedge \langle ?x \rangle, t) \uparrow r_s^? \uparrow (R'(x, s \wedge \langle !e \rangle, t) \uparrow r_s^! \uparrow \text{PRUN} \langle ! \rangle, \textcircled{C})))$$

$$\boxed{r_s^? \wedge ?z \rightarrow R'(z, s \wedge \langle ?x \rangle, t)}$$

$$\boxed{\langle ! \rangle \rightarrow R'(x, s, t)}$$

$$\boxed{\textcircled{C} \rightarrow R'(x, s, s)}$$

Proof

Similar to lemma 6

Theorem 3

$$BPB \parallel \text{RUN}_{\downarrow}, \odot = \text{PRE} \gg \text{Ch}(P) \gg \text{POST}$$

Proof:

For any trace s of P we define

$$Q(s, t) = BP/s \ B \parallel \text{RUN}_{\downarrow}, \odot \quad \text{for } t \leq s$$

$$R(x, s, t) = B_x P/s \ B_y \parallel \text{RUN}_{\downarrow}, \odot \quad \text{for } t \leq s$$

$$\text{and } \langle !y \rangle = \text{last} (s! \{ \text{right} \})$$

From these definitions it follows that

$$Q(s, t) = Q(s, s)$$

and $R(x, s, t) = R(x, s, s)$ provided that $t \leq s$

When $s = t = \langle \rangle$ we obtain

$$BPB \parallel \text{RUN}_{\downarrow}, \odot = Q(\langle \rangle, \langle \rangle)$$

Moreover we have

$$Q(s, t) = (\exists x \rightarrow (BP/s \ \langle ?x \rangle B \{ r_s^? \} (B_x P/s \ \langle !e \rangle B_e \{ r_s^! \} \text{STOP})) \parallel \text{RUN}_{\downarrow}, \odot)$$

$$\boxed{r_s^! \& !e \rightarrow BP/s \ \langle !e \rangle B \parallel \text{RUN}_{\downarrow}, \odot}$$

$$\boxed{\downarrow \rightarrow BP/s \ B \parallel \text{RUN}_{\downarrow}, \odot}$$

$$\boxed{\odot \rightarrow BP/s \ B \parallel \text{RUN}_{\downarrow}, \odot}$$

lemma 2 and L3(a)

$$= (\exists x \rightarrow (Q(s^{\wedge} \langle ?x \rangle, t) \{ r_s^? \} (R(x, s^{\wedge} \langle !e \rangle, t) \{ r_s^! \} \text{PRUN}_{\downarrow}, \odot)))$$

$$\boxed{r_s^! \& !e \rightarrow Q(s^{\wedge} \langle !e \rangle, t)}$$

$$\boxed{\downarrow \rightarrow Q(s, t)}$$

$$\boxed{\odot \rightarrow Q(s, s)}$$

L3(b)(C) and since

$$Q(s, t) = Q(s, s)$$

Similarly we can show that

$$R(x, s, t) = (\exists y \rightarrow (Q(s^{\wedge} \langle ?x \rangle, t) \{ r_s^? \} (R(x, s^{\wedge} \langle !e \rangle, t) \{ r_s^! \} \text{PRUN}_{\downarrow}, \odot)))$$

$$\boxed{r_s^? \& ?z \rightarrow R(z, s^{\wedge} \langle ?x \rangle, t)}$$

$$\boxed{\downarrow \rightarrow R(x, s, t)}$$

$$\boxed{\downarrow \rightarrow R(x, s, s)}$$

Thus we conclude that the processes $Q(s, t)$ and $R(x, s, t)$ meet the same guarded recursive equations as $Q'(s, t)$ and $R'(x, s, t)$ and they must be the same.

In particular, by taking $s = t = \langle \rangle$, we obtain

$$BPB \parallel \text{RUN}_{\downarrow}, \odot = Q(\langle \rangle, \langle \rangle) = Q'(\langle \rangle, \langle \rangle) = \text{PRE} \gg \text{Ch}(P) \gg \text{POST}$$

Discussion

For the particular problem treated in this paper, algebraic methods seem much preferable to more familiar assertional techniques [Hoare, Zhou]. There is insufficient experience to generalise this conclusion; perhaps in some cases a mixed approach would be the most effective.

Nevertheless, even for a grossly over-simplified problem, the algebraic calculations are non-trivial. This probably has to be accepted as inevitable in any serious application of mathematics to engineering. The calculations can be simplified by prior development of a calculus adapted more to the specific needs of a problem. It will be interesting to see how far such calculi are applicable to more general classes of problems; but it seems quite likely that they will not. Again, we may have to accept that each application will require derivation of specialised laws to control its complexity.

It would be interesting to explore more realistic problems and solutions. For example:

- (1) Extension of the present solution beyond pipes to any number of input and output channels. Here the problem and its solution are sketched out.

Let P be a process with ℓ input channels $1.\text{left}, \dots, \ell.\text{left}$, and m output channels $1.\text{right}, \dots, m.\text{right}$. Let Q be a process with m input channels $1.\text{left}, \dots, m.\text{left}$, and n output channels $1.\text{right}, \dots, n.\text{right}$. We suppose that they are also allowed to engage in events from a fixed alphabet A .

P and Q can be joined together so that the output channels $1.\text{right}, \dots, m.\text{right}$ of P are connected to the input channels $1.\text{left}, \dots, m.\text{left}$ of Q respectively, and the sequences of messages output by P and input by Q on these internal channels are concealed from the common environment. Moreover, any event in A requires simultaneous participation of both P and Q . The result of connection is denoted by

$$P \gg_m Q$$

Whenever we connect P and Q , we assume that these connected channels are capable of transmitting the same kind of messages.

$$\alpha_{i.\text{right}}(P) = \alpha_{i.\text{left}}(Q) \quad \text{for } 1 \leq i \leq m$$

Definition

Let P be a process with ℓ input channels $1.\text{left}, \dots, \ell.\text{left}$, and m output channels $1.\text{right}, \dots, m.\text{right}$. We define

$$BPB = P_{\ell} \gg_{\ell} P \gg_m R_m$$

where $B_{\ell} = \parallel_{1 \leq i \leq \ell} i:B$

and $B_m = \parallel_{1 \leq i \leq m} i:B$

and the process $i:B$ is defined as one that engages in the communication i.c.v whenever B would have engaged in the communication c.v.

We are required to find the suitable processes PRE_{ℓ} and $POST_m$ such that

$$BPB \parallel \text{RUN}_y = PRE_{\ell} \gg_{\ell} \hat{P} \gg_m POST_m$$

In fact, the structures of PRE_{ℓ} and $POST_m$ are similar to those defined in Section 3. Here we only formalize the process PRE_{ℓ} , and leave the definition of $POST_m$ and the detailed proof as an exercise for the interested reader.

$$PRE_{\ell} = \parallel_{1 \leq i \leq \ell} i:PRE$$

where PRE was defined in Section 3.

- (2) Removal of the simplification that the ζ and \odot events are detected simultaneously by all processes.

In this case, it is not possible to implement check pointing exactly, because PRE and/or $POST$ may continue to input or output after the \odot but before the checkpointing message reaches them. The best that can be done is to guarantee recovery to some point soon after \odot .

When the specification is appropriately weakened, the implementation could be based on the idea of [Lamport].

- (3) But perhaps the most serious simplification is the assumption that all processes are deterministic. For general non-deterministic processes, a full recovery using the techniques of this paper is not possible; so the specification must be somehow weakened.

(4) Another related problem is selective recovery in a distributed transaction processing system. Here an operator at a console may deliberately wish to fall back to his most recent checkpoint; but this should have minimal effect on the behaviour of the system as observed by operators at other consoles. It would be interesting if the methods used in this paper could be extended to throw light on this intractable problem.

Acknowledgements

The distributed recovery problem was suggested by K.V.S. Prasad, and solved by him in the framework of CCS [Prasad]. The use of algebraic transformations for correctness proofs has been pioneered in [Bacon, Bergstra and Klop].

The research on this paper was supported in part by the Science and Engineering Research Council of Great Britain.

References

J. Backus

Can Programming be liberated from the von Neuman style?

Comm. ACM 21.8 (1978) pp 613-641

J.C.M. Baeten, J.A. Bergstra and J.W. Klop

Syntax and defining equations for an interrupt mechanism in process algebra.

Report CS-R8503. Centre for Mathematics and Computer Science, Amsterdam,
The Netherlands (1983)

R.M. Burstall and J. Darlington

A Transformation System for Developing Recursive Programs.

J.ACM 24.1 (1977) pp 44-67

J.A. Goguen, J.W. Thatcher and E.G. Wagner

An initial algebra approach to the specification, correctness and
implementation of abstract data types.

Current Trends in Programming Methodology Vol IV: Data Structuring

R.T. Yeh (Ed.), Prentice-Hall, Englewood Cliffs, (1978) pp 80-149

J.V. Guttag and J.J. Horning

The algebraic specification of abstract data type.

Acta Informatica 10.1 (1978) pp 27-52

C.A.R. Hoare

Communicating Sequential Processes.

Prentice Hall International Series in Computer Science (1985)

C.A.R. Hoare and Chao Chen Zhou

Partial correctness of Communicating Sequential Processes.

Proc. International Conference on Distributed Computing (1981)

INMOS Ltd.

The Occam Programming Manual.

Prentice-Hall (1984)

L. Lamport

Time, Clocks and the Ordering of Events in a Distributed System.

Communications of the ACM 21.7 (1978) pp 558-565

K.V.S. Prasad

Specification and proof of a simple fault tolerant system in CCS.

Internal report CSR-178-84

University of Edinburgh (1984)

Appendix

Definition

Let P and Q both be pipes. The process $P \gg Q$ is formally defined

$$P \gg Q = (P[\text{mid}/\text{right}] \parallel Q[\text{mid}/\text{left}]) \setminus \{\text{mid}\}$$

where \setminus denotes the concealment operator, and the process $P[d/c]$ behaves the same as P except that the channel c is renamed by d .

The following laws of concealment in [Hoare, 3.5] are useful in exploring the properties of the chaining operator

- (1) If $B \cap C = \{\}$
then $(x:B \rightarrow P(x)) \setminus C = (x:B \rightarrow (P(x) \setminus C))$
- (2) If $B \cap C \neq \{\}$ and is finite
then $(x:B \rightarrow P(x)) \setminus C = Q \sqcap (Q \sqcup (x:B \rightarrow P(x)))$
where $Q = \bigsqcap_{x:B \cap C} P(x) \setminus C$
- (3) $(P \setminus C) / s = (\bigsqcap_{t \in T} P/t) \setminus C$
where $T = \text{traces } (P) \cap \{t \mid t \uparrow (\nrightarrow P-C) = s\}$
provided that T is finite and $s \in \text{traces } (P \setminus C)$

Here we offer proofs for laws of chaining quoted in Section 2

- (a) Let $P = (b1 \& ?x \rightarrow P1(x)) \parallel b2 \& !e \rightarrow P2 \parallel y:B \rightarrow P3(y)$
and $Q = (c1 \& ?x \rightarrow Q1(x)) \parallel c2 \& !f \rightarrow Q2 \parallel y:C \rightarrow Q3(y)$
then $P \gg Q = \text{if } b2 \wedge c1 \text{ then } (T \sqcup U) \sqcap U \text{ else } T$
where $T = (b1 \& ?x \rightarrow P1(x) \gg Q \parallel c2 \& !f \rightarrow P \gg Q2 \parallel y:B \cap C \rightarrow P3(y) \gg Q3(y))$
and $U = (b2 \wedge c1) \& (P2 \gg Q1(e))$

Proof:

$$\begin{aligned} \text{LHS} &= (P[\text{mid}/\text{right}] \parallel Q[\text{mid}/\text{left}]) \setminus \{\text{mid}\} \\ &= (b1 \& ?x \rightarrow P1(x) [\text{mid}/\text{right}] \parallel Q[\text{mid}/\text{left}] \\ &\quad \parallel c2 \& !f \rightarrow P[\text{mid}/\text{right}] \parallel Q2[\text{mid}/\text{left}] \\ &\quad \parallel y:B \cap C \rightarrow P3(y) [\text{mid}/\text{right}] \parallel Q(y) [\text{mid}/\text{left}]) \\ &\quad \setminus (b2 \wedge c1) \& \text{mid} \& !e \rightarrow P2[\text{mid}/\text{right}] \parallel Q1(e) [\text{mid}/\text{left}]) \setminus \{\text{mid}\} \end{aligned}$$

L3(c) in Section 2

= RHS

Law 1.2 of concealment a

(b) Let $P = (b1 \ \& \ ?x \longrightarrow P1(x) \parallel b2 \ \& \ !e \longrightarrow P2 \parallel y:B \longrightarrow P3(y))$
 and $Q = (c1 \ \& \ ?x \longrightarrow Q1(x) \parallel c2 \ \& \ !f \longrightarrow Q2 \parallel y:C \longrightarrow Q3(y))$
 and $R = (d1 \ \& \ ?x \longrightarrow R1(x) \parallel d2 \ \& \ !y \longrightarrow R2 \parallel y:D \longrightarrow R3(y))$

If $P \gg Q \gg R$ is deterministic, then

$$\begin{aligned}
 P \gg Q \gg R &= (b1 \ \& \ ?x \longrightarrow P1(x) \gg Q \gg R \\
 &\parallel d2 \ \& \ !y \longrightarrow P \gg Q \gg R2 \\
 &\parallel y: (B \wedge C \wedge D) \longrightarrow P3(y) \gg Q3(y) \gg R3(y) \\
 &\parallel (b2 \wedge c1) \ \& \ (P2 \gg Q1(e) \gg R) \\
 &\parallel (c2 \wedge d1) \ \& \ (P \gg Q2 \gg R1(f))
 \end{aligned}$$

Proof:

Similar to (a)

(c)

1. If $P \gg Q$ is deterministic, then

$$\begin{aligned}
 (P \gg Q) / \langle x \rangle &= (P / \langle x \rangle) \gg (Q / \langle x \rangle) && \text{provided that } x \in A \\
 &&& \text{and } \langle x \rangle \in \text{traces}(P) \cap \text{traces}(Q)
 \end{aligned}$$

Proof:

We define

$$T = \{t \mid t \in \text{traces}(P[\text{mid}/\text{right}]) \parallel Q[\text{mid}/\text{left}] \wedge t \uparrow (\{\text{left}, \text{right}\} \cup A) = \langle x \rangle\}$$

From the assumption it follows that $\langle x \rangle \in T$

$$\begin{aligned}
 \therefore (P \gg Q) / \langle x \rangle &= \bigsqcup_{t \in T} ((P[\text{mid}/\text{right}]) \parallel Q[\text{mid}/\text{left}]/t) \setminus \{\text{mid}\} \\
 &&& \text{Law 3 of concealment} \\
 &= ((P[\text{mid}/\text{right}]) \parallel Q[\text{mid}/\text{left}]/\langle x \rangle) \setminus \{\text{mid}\} \\
 &&& \text{Since } P \gg Q \text{ is deterministic} \\
 &= ((P[\text{mid}/\text{right}]/\langle x \rangle) \parallel (Q[\text{mid}/\text{left}]/\langle x \rangle)) \setminus \{\text{mid}\} \\
 &&& \text{L3(d) in Section 2} \\
 &= (P/\langle x \rangle) \gg (Q/\langle x \rangle) && \text{def of } \gg
 \end{aligned}$$

2. If $P \gg Q$ is deterministic, then

$$\begin{aligned}
 P \gg Q &= (P/\text{right}.u) \gg (Q/\text{left}.u) && \text{provided that } \text{right}.u \in \text{traces}(P) \\
 &&& \text{and } \text{left}.u \in \text{traces}(Q)
 \end{aligned}$$

Proof:

We define

$$T = \{t \mid t \in \text{traces}(P[\text{mid}/\text{right}]) \parallel Q[\text{mid}/\text{left}] \wedge t \uparrow (\{\text{left}, \text{right}\} \cup A) = \langle \rangle\}$$

From the assumption we have

$$\text{mid}.u \in \text{traces}(P[\text{mid}/\text{right}])$$

$$\text{mid}.u \in \text{traces}(Q[\text{mid}/\text{left}])$$

$$\text{and } \text{mid}.u \uparrow (\{\text{left}, \text{right}\} \cup A) = \langle \rangle$$

which implies that

$$\text{mid}.u \in T$$

Thus we conclude that

$$\begin{aligned}
 P \gg Q &= (P \gg Q) / \langle \rangle && \text{L2(a) in Section 2} \\
 &= \bigsqcup_{t \in T} ((P[\text{mid}/\text{right}] \parallel Q[\text{mid}/\text{left}]) / t) \setminus \{\text{mid}\} && \text{Law 3 of concealment} \\
 &= ((P[\text{mid}/\text{right}] \parallel Q[\text{mid}/\text{left}]) / \text{mid.u}) \setminus \{\text{mid}\} && \text{since } P \gg Q \text{ is deterministic} \\
 &= (P[\text{mid}/\text{right}] / \text{mid.u}) \parallel (Q[\text{mid}/\text{left}] / \text{mid.u}) \setminus \{\text{mid}\} && \text{L3(d) in Section 2} \\
 &= |P/\text{right.u} \gg \{Q/\text{left.u}\} && \text{def of } \gg
 \end{aligned}$$

3. If $P \gg Q \gg R$ is deterministic, then

$$(P \gg Q \gg R) / \langle x \rangle = (P / \langle x \rangle) \gg (Q / \langle x \rangle) \ll (R / \langle x \rangle)$$

provided that $x \in A$ and $\langle x \rangle \in \text{traces}(P) \cap \text{traces}(Q) \cap \text{traces}(R)$

Proof:

Similar to (c).1

4. If $P \gg Q \gg R$ is deterministic, then

$$P \gg Q \gg R = (P/\text{right.ins}(s)) \gg (Q/s) \gg (R/\text{left.outs}(s))$$

provided that $s \in \text{traces}(Q)$ and $\text{right.ins}(s) \in \text{traces}(P)$ and $\text{left.outs}(s) \in \text{traces}(R)$

Proof:

Similar to (c).2