Distributed Computing (1987) 2:1–12

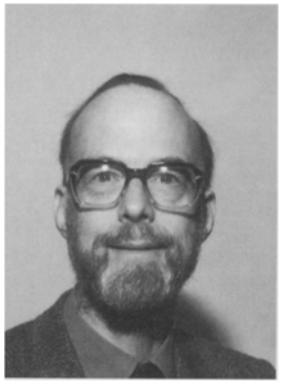
DISTRIBUTED COMPUTING

Algebraic specification and proof of a distributed recovery algorithm

He Jifeng and C.A.R. Hoare

Oxford University Computing Laboratory, Programming Research Group, 8–11 Keble Road, Oxford OX1 3QD, UK

He Jifeng *received the B.S. degree in mathematics from Fudan University, Shanghai (China), in 1965. Then he taught in the department of mathematics, Shanghai Normal University. In 1972 he moved to East China Normal University where he was a lecturer of computer science. Since September 1986 he has been a professor of computer science at East China Normal University. He is currently working at Oxford University Computing Laboratory. His major research interests are programming language semantics, software engineering and distributed computing.*



Tony Hoare *is Professor of Computation at the University of Oxford. He received his MA from Oxford in Classical Languages, Literature, History and Philosophy. He worked for eight years as programmer, manager and research scientist with a small computer manufacturer. He is the recipient of several honours for his contributions to the study of computer programming languages, and is generally famed for Hoare's Law: inside every large program there is a small program trying to get out.*

Abstract. 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 1985) and the proofs of correctness are conducted wholly by application of algebraic laws (together with the unique fixed point theorem).

Offprint requests to: C.A.R. Hoare

1 Introduction

Algebraic specifications have been formalized and used for a number of years (Guttag and Horning 1978; Goguen et al. 1978). A problem that has emerged in their practical application is the large numbers of mutually interactive equations, which cannot easily be seen to reflect the requirements. In this paper, we 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 1978; Burstall and Darlington 1977).

We shall use the notation of Communicating Sequential Processes (Hoare 1985) 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 $\lightning$ be a symbol standing for a catastrophic event. We specify $\hat{P}$ as a process which behaves like P until $\lightning$ occurs, and after each $\lightning$ behaves like $\hat{P}$ from the start again. Formally, $\hat{P}$ is defined in Hoare (1985) to satisfy the recursive equation

$$\hat{P} = P \,^\wedge (\lightning \longrightarrow \hat{P}).$$

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

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

where $s^\frown t$ 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 $\lightning$, i.e., on each occurrence of $\lightning$ 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 $\lightning$. This is just the behaviour of the process

$$P \parallel \mathrm{RUN}_{\lightning}$$

where

$$\mathrm{RUN}_{\lightning} = \mu X.(\lightning \longrightarrow X)$$

and

$\parallel$ is parallel composition.

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

$$(P \parallel \mathrm{RUN}_{\lightning})/s = (P \parallel \mathrm{RUN}_{\lightning})/s^\frown\langle\lightning\rangle = (P/(s \restriction \alpha P)) \parallel \mathrm{RUN}_{\lightning}.$$

Here Q/s (for $s \in \mathrm{traces}(Q)$) 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 $P \parallel \mathrm{RUN}_{\lightning}$. 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: $P \parallel \mathrm{STOP}_{\lightning}$ 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 $\lightning$ 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

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

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
>
> $\mathrm{PRE} \gg \hat{P} \gg \mathrm{POST} = (B \gg P \gg B) \parallel \mathrm{RUN}_{\lightning}$
>
> where B is the single-buffering process:
> $B = \mathrm{left}?x \longrightarrow \mathrm{right}!x \longrightarrow B.$

To simplify the solution, we suppose that $\lightning$ 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 $\lightning$ requires simultaneous participation of both its operands. Thus in the solution outlined above, whenever $\lightning$ 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 $\lightning$ 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 $\lightning$). Similarly, POST keeps a count of all messages output, and ignores that number of messages output by $\hat{P}$ after an occurrence of $\lightning$. The only subtle point is to ensure the correct outcome even when $\lightning$ occurs in the middle of the recovery procedure. The formal solution of this problem and its proof of correctness are given in Sect. 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 ©. The process $Ch(P)$ is defined to behave like $\hat{P}$, except that each occurrence of $\lightning$ sends it back to its state just after the most recent occurrence of ©, or to the beginning if © has not yet occurred. Any occurrence of © has no other effect on the behaviour of P. This operation is defined in Hoare

(1985) on deterministic processes by the following laws

L1 $Ch(P) = Ch2(P, P)$

L2 if $P = (x: B \longrightarrow P(x))$
then $Ch2(P, Q) = (x: B \longrightarrow Ch2(P(x), Q)$
$\Box \lightning \longrightarrow Ch2(Q, Q)$
$\Box \copyright \longrightarrow Ch2(P, P))$.

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 $\lightning$, or superseded on the next occurrence of ©.

The improved solution can now be specified:

> Find a pair of pipes PRE and POST, containing both $\lightning$ and © in their alphabet, such that for all deterministic P
>
> $\text{PRE} \gg Ch(P) \gg \text{POST} = (B \gg P \gg B) \parallel \text{RUN}_{\{\lightning, \copyright\}}$.

The solution and its proof are given in Sect. 4.

Before embarking on the proofs, Sect. 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 (1985) because they apply only to deterministic processes, which are therefore free of divergence.

In future we shall use the following abbreviations:

$$B_x = B/\langle ?x \rangle \ (= !x \longrightarrow B)$$
$$BPB = B \gg P \gg B$$
$$B(P/s)\,B = B \gg (P/s) \gg B$$
$$B_x(P/s)\,B_y = B_x \gg (P/s) \gg B_y, \text{ etc.}$$

We also define

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

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

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 pipes to engage in events from a fixed alphabet A. Thus our definition is slightly more general than that given in Hoare (1985).

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 (1985).

(1) Law of general choice (Hoare 1985, 3.3.1 L5)

$$(x: A \longrightarrow P(x)) \Box (y: B \longrightarrow Q(y)) =$$
$$(z: (A \cup B) \longrightarrow (\text{if } z \in (A - B) \text{ then } P(z)$$
$$\text{else if } z \in (B - A) \text{ then } Q(z)$$
$$\text{else if } z \in (A \cap B) \text{ then } (P(z) \sqcap Q(z)).$$

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 1985, 1.8.3, L1–L3 and 2.6.1 L7)

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

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

(c) $(x: B \longrightarrow P(x))/\langle c \rangle = P(c)$ 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 1985, 2.3.1 L7 and 2.3.3 L2)

(a) Let

$$P=(x: B\longrightarrow P(x))$$

and

$$Q=(y: C\longrightarrow Q(y)).$$

Then

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

where

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

and

$$P'=P(z) \quad \text{if } z\in B$$
$$=P \quad \text{otherwise}$$

and

$$Q'=Q(z) \quad \text{if } z\in C$$
$$=Q \quad \text{otherwise}$$

and

αP denotes the alphabet of P.

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

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

(c) $(P\parallel Q)/s=(P/(s\uparrow\alpha P))\parallel(Q/(s\uparrow\alpha Q))$

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

(4) Laws of chaining

The introduction of Boolean guards (INMOS 1984) 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

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

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 (1985) 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

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

and

$$Q=(c1\&?x\longrightarrow Q1(x)\,\square$$
$$c2\&!f\longrightarrow Q2\,\square\ y: C\longrightarrow Q3(y)).$$

Then

$$P\gg Q=((T\,\square\,U)\sqcap U)\{b2\wedge c1\}T$$

where

$$T=(b1\&?x\longrightarrow(P1(x)\gg Q)$$
$$\square c2\&!f\longrightarrow(P\gg Q2)$$
$$\square y: B\cap C\longrightarrow(P3(y)\gg Q3(y)))$$

and

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

The first line of the definition of T describes the 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 $(T\,\square\,U)\sqcap U$ 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

$$P\gg Q=T\,\square\,U.$$

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

(b) 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

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

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.

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

where right.u and left.u are defined

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

where u_0 and u' denote 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

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

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

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

$$\begin{aligned} \text{PRE}(\langle\,\rangle, u) &= (?x \longrightarrow \text{PRE}(\langle x \rangle, u \,^\frown\, \langle x \rangle) \\ &\quad \square \not\longrightarrow \text{PRE}(u, u)) \end{aligned}$$

$$\begin{aligned} \text{PRE}(\langle x \rangle \,^\frown\, v, u) &= (!x \longrightarrow \text{PRE}(v, u) \\ &\quad \square \not\longrightarrow \text{PRE}(u, u)) \end{aligned}$$

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

$$\begin{aligned} \text{POST}(0, m) &= (?x \longrightarrow (!x \longrightarrow \text{POST}(0, m+1) \\ &\quad \square \not\longrightarrow \text{POST}(m, m)) \\ &\quad \square \not\longrightarrow \text{POST}(m, m)) \end{aligned}$$

$$\begin{aligned} \text{POST}(n+1, m) &= (?x \longrightarrow \text{POST}(n, m) \\ &\quad \square \not\longrightarrow \text{POST}(m, m)) \end{aligned}$$

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

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

where $\text{right}_{\cdot}u$ *is the trace consisting of outputs of all the messages in* u, *and* $\text{left}_{\cdot}v$ *is the trace consisting of all inputs of the messages in* v, *and* $\#u$ *is the length of the sequence* u.

Proof.

(a) By induction on the length of u:

(0) For $u = \langle\rangle$

$\text{PRE}(\langle\rangle, v)/\text{right}_{\cdot}\langle\rangle$

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

$= \text{PRE}(\langle\rangle, v)$ L2(a).

(1) Assume the inductive hypothesis.

$\text{PRE}(u, v)/\text{right}_{\cdot}u = \text{PRE}(\langle\rangle, v)$ for $\#u = n$

$\text{PRE}(\langle x\rangle^{\wedge}u, v)/\text{right}_{\cdot}(\langle x\rangle^{\wedge}u)$

$= \text{PRE}(\langle x\rangle^{\wedge}u, v)/(\langle \text{right}_{\cdot}x\rangle^{\wedge}\text{right}_{\cdot}u)$ def of right $(\langle x\rangle^{\wedge}u)$

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

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

$$\text{PRE} \gg \hat{P} \gg \text{POST} = BPB \parallel \text{RUN}_{\not\leftarrow}.$$

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 of s of P

$$A(s) = B(P/s)\,B \parallel \text{RUN}_{\not\leftarrow}$$

$$C(x, s) = B_x(P/s)\,B_y \parallel \text{RUN}_{\not\leftarrow}$$

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

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

$$BPB \parallel \text{RUN}_{\not\leftarrow} = A(\langle\rangle).$$

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.

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

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

In general we define for any trace s of P

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

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

Lemma 2.

$$BPB = (?x \longrightarrow (BP/\langle ?x\rangle\, B\,\{\!|r^?|\!\}\ (B_x P/\langle !e\rangle\, B_e\,\{\!|r^!|\!\}\ \text{STOP})) \ \Box\, r^! \,\&\, !e \longrightarrow (BP/\langle !e\rangle\, B)).$$

Proof.

$BPB = (?x \longrightarrow B_x PB \,\Box\, r^! \,\&\, BP/\langle !e\rangle\, B_e)$ L4(a)

$= (?x \longrightarrow (r^? \longrightarrow BP/\langle ?x\rangle\, B \ \Box\, r^! \longrightarrow B_x P/\langle !e\rangle\, B_e)$

$\Box\, r^! \,\&\, (?x \longrightarrow B_x P/\langle !e\rangle\, B_e$ L4(a)

$\Box\, !e \longrightarrow BP/\langle !e\rangle\, B))$

$= \text{RHS}.$ L1

Lemma 3.

$$B_x PB_y = (!y \longrightarrow (BP/\langle ?x\rangle\, B\,\{\!|r^?|\!\}\ (B_x P/\langle !e\rangle\, B_e\,\{\!|r^!|\!\}\ \text{STOP})) \ \Box\, r^? \,\&\, ?z \longrightarrow B_z P/\langle ?x\rangle\, B_y).$$

Proof.

Similar to Lemma 2.

Lemma 4.

(a) $A(s) = (?x \longrightarrow (A(s^{\wedge}\langle ?x\rangle)\,\{\!|r_s^?|\!\}\ (C(x, s^{\wedge}\langle !e\rangle)\,\{\!|r_s^!|\!\}\ \text{PRUN}_{\not\leftarrow}))$

$\Box\, r_s^! \,\&\, !e \longrightarrow A(s^{\wedge}\langle !e\rangle)$

$\Box\, \not\leftarrow \longrightarrow A(s))$ for $s \in \text{traces}(P)$

(b) $C(x, s) = (!y \longrightarrow (A(s^{\wedge}\langle ?x\rangle)\,\{\!|r_s^?|\!\}\ (C(x, s^{\wedge}\langle !e\rangle)\,\{\!|r_s^!|\!\}\ \text{PRUN}_{\not\leftarrow}))$

$\Box\, r_s^? \,\&\, ?z \longrightarrow C(z, s^{\wedge}\langle ?x\rangle)$

$\Box\, \not\leftarrow \longrightarrow C(x, s))$

for $s \in \text{traces}(P)$ and $\langle !y\rangle = \text{last}(s\restriction\{\text{right}\})$.

Proof.

(a) $A(s) = BP/sB \parallel \mathrm{RUN}_{\not\leftarrow}$ def of $A(s)$

$= (?x \longrightarrow (BP/s^\frown\langle ?x\rangle\, B\{r_s^?\}$

$(B_x\, P/s^\frown\langle !e\rangle\, B_e\{r_s^!\}\, \mathrm{STOP})) \parallel \mathrm{RUN}_{\not\leftarrow}$

$\Box r_s^! \,\&\, !e \longrightarrow BP/s^\frown\langle !e\rangle\, B \parallel \mathrm{RUN}_{\not\leftarrow}$

$\Box \not\leftarrow \longrightarrow BP/sB \parallel \mathrm{RUN}_{\not\leftarrow})$

Lemma 2 and L3(a)

$= \mathrm{RHS}$ L3(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 $\Box$, $\longrightarrow$, &. 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 the 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:

$$A'(s) = \mathrm{PRE}(\langle\,\rangle, \mathrm{ins}(s)) \gg (P/s^\frown(\not\leftarrow \longrightarrow \hat{P})) \gg \mathrm{POST}(0, \#\mathrm{outs}(s))$$

$$C'(x, s) = \mathrm{PRE}(\langle x\rangle, \mathrm{ins}(s)^\frown\langle x\rangle) \gg (P/s^\frown(\not\leftarrow \longrightarrow \hat{P})) \gg \mathrm{POST}_y(0, \#\mathrm{outs}(s) - 1)$$

where

$$\mathrm{POST}_y(0, n) = (!y \longrightarrow \mathrm{POST}(0, n+1) \;\Box\; \not\leftarrow \longrightarrow \mathrm{POST}(n, n))$$

and

$$\langle !y\rangle = \mathrm{last}(s\uparrow\{\mathrm{right}\}).$$

First, let us show that occurrence of $\not\leftarrow$ has no effect on the behaviour of the processes $A'(s)$ and $C'(x, s)$.

Lemma 5.

(a) $A'(s)/\langle\not\leftarrow\rangle = A'(s)$.

(b) $C'(x, s)/\langle\not\leftarrow\rangle = C'(x, s)$.

Proof.

(a) $\mathrm{LHS} = (\mathrm{PRE}(\langle\,\rangle, \mathrm{ins}(s))/\langle\not\leftarrow\rangle)$

$\gg ((P/S^\frown(\not\leftarrow \longrightarrow \hat{P}))/\langle\not\leftarrow\rangle)$

$\gg (\mathrm{POST}(0, \#\mathrm{outs}(s))/\langle\not\leftarrow\rangle)$ L4(b)

$= \mathrm{PRE}(\mathrm{ins}(s), \mathrm{ins}(s)) \gg \hat{P}$

$\gg \mathrm{POST}(\#\mathrm{outs}(s), \#\mathrm{outs}(s))$

L2(c) and definition of PRE, POST and P

$= (\mathrm{PRE}(\mathrm{ins}(s), \mathrm{ins}(s))/\mathrm{right}.\mathrm{ins}(s)) \gg (\hat{P}/s)$

$\gg (\mathrm{POST}(\#\mathrm{outs}(s), \#\mathrm{outs}(s))/\mathrm{left}.\mathrm{outs}(s))$ 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* $A'(s) = \mathrm{RUN}_{\not\leftarrow}$.

Proof.

$A'(s) = \not\leftarrow \longrightarrow \mathrm{PRE}(\mathrm{ins}(s), \mathrm{ins}(s)) \gg \hat{P}$

$\gg \mathrm{POST}(\#\mathrm{outs}(s), \#\mathrm{outs}(s))$ L4(b)

$= \not\leftarrow \longrightarrow A'(s)$ Lemma 5.

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) $A'(s) = (?x \longrightarrow \mathrm{PRE}(\langle x\rangle, \mathrm{ins}(s)^\frown\langle x\rangle)$

$\gg ((P/s)^\frown(\not\leftarrow \longrightarrow \hat{P}))$

$\gg \mathrm{POST}(0, \#\mathrm{outs}(s))$

$\Box \not\leftarrow \longrightarrow \mathrm{PRE}(\mathrm{ins}(s), \mathrm{ins}(s))$

$\gg \hat{P} \gg \mathrm{POST}(\#\mathrm{outs}(s), \#\mathrm{outs}(s))$

$\Box r_s^! \,\&\, \mathrm{PRE}(\langle\,\rangle, \mathrm{ins}(s))$

$\gg ((P/s^\frown\langle !e\rangle)^\frown(\not\leftarrow \longrightarrow \hat{P}))$

$\gg \mathrm{POST}_e(0, \#\mathrm{outs}(s))$ L4(b)

$= (?x \longrightarrow (r_s^? \,\&\, (\mathrm{PRE}(\langle\,\rangle, \mathrm{ins}(s)^\frown\langle x\rangle)$

$\gg ((P/s^\frown\langle ?x\rangle)^\frown(\not\leftarrow \longrightarrow \hat{P}))$

$\gg \mathrm{POST}(0, \#\mathrm{outs}(s)))$

$\Box r_s^! \,\&\, (\mathrm{PRE}(\langle x\rangle, \mathrm{ins}(s)^\frown\langle x\rangle)$

$\gg ((P/s^\frown\langle !e\rangle)^\frown(\not\leftarrow \longrightarrow \hat{P}))$

$\gg \mathrm{POST}_e(0, \#\mathrm{outs}(s)))$

$$\Box \not\in \longrightarrow \mathrm{PRE}(\mathrm{ins}(s)^\wedge\langle x\rangle, \mathrm{ins}(s)^\wedge\langle x\rangle) \gg \hat{P} \gg \mathrm{POST}(\#\mathrm{outs}(s), \#\mathrm{outs}(s)))$$

$$\Box \not\in \longrightarrow A'(s)$$

$$\Box r_s^! \,\&\, (?x \longrightarrow \mathrm{PRE}(\langle x\rangle, \mathrm{ins}(s)^\wedge\langle x\rangle) \gg ((P/s^\wedge\langle !e\rangle)^\wedge(\not\in \longrightarrow \hat{P})) \gg \mathrm{POST}_e(0, \#\mathrm{outs}(s)))$$

$$\Box !e \longrightarrow \mathrm{PRE}(\langle\rangle, \mathrm{ins}(s)) \gg ((P/s^\wedge\langle !e\rangle)^\wedge(\not\in \longrightarrow \hat{P}) \gg \mathrm{POST}(0, \#\mathrm{outs}(s)+1)$$

$$\Box \not\in \longrightarrow \mathrm{PRE}(\mathrm{ins}(s), \mathrm{ins}(s)) \gg \hat{P} \gg \mathrm{POST}(\#\mathrm{outs}(s), \#\mathrm{outs}(s))))$$

Lemma 5 and L4(b)

$$= (?x \longrightarrow (A'(s^\wedge\langle ?x\rangle)\ \{r_s^?\}\ (C'(x, s^\wedge\langle !e\rangle)\ \{r_s^!\}\ \mathrm{PRUN}_{\not\in})$$

$$\Box r_s^! \,\&\, !e \longrightarrow A'(s^\wedge\langle !e\rangle)$$

$$\Box \not\in \longrightarrow A'(s)$$

Corollary of Lemma 5 and L1.

(b) Similar to (a).

Theorem 2. $BPB \parallel \mathrm{RUN}_{\not\in} = \mathrm{PRE} \gg \hat{P} \gg \mathrm{POST}$.

Proof. From Lemma 6 and the unique fixed point theorem it follows that

$A'(s) = A(s)$ for $s \in \mathrm{traces}(P)$.

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 $\not\in$ and © in their alphabet such that for all deterministic P

$$\mathrm{PRE} \gg Ch(P) \gg \mathrm{POST} = BPB \parallel \mathrm{RUN}_{\{\not\in, ©\}}.$$

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

The technique adopted in Sect. 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

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

$$\mathrm{PRE}(u, v) = (\not\in \longrightarrow \mathrm{PRE}(v, v) \;\Box\; © \longrightarrow \mathrm{PRE}(u, u) \;\Box\; u \neq \langle\rangle \,\&\, !u_0 \longrightarrow \mathrm{PRE}(u', v) \;\Box\; u = \langle\rangle \,\&\, ?x \longrightarrow \mathrm{PRE}(\langle x\rangle, v^\wedge\langle x\rangle))$$

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

$$\mathrm{POST}(u, n, m) = (\not\in \longrightarrow \mathrm{POST}(u, m, m) \;\Box\; © \longrightarrow \mathrm{POST}(u, n, n) \;\Box\; u \neq \langle\rangle \,\&\, !u_0 \longrightarrow \mathrm{POST}(\langle\rangle, n, m) \;\Box\; (u = \langle\rangle \vee n \neq 0) \,\&\, ?x \longrightarrow (\mathrm{POST}(u, n-1, m)\ \{n \neq 0\}\ \mathrm{POST}(\langle x\rangle, 0, m+1)))$$

where $n, m \geqq 0$ and u, v denote sequences of messages.

Furthermore, for any trace s of P we define

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

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

for $t \leqq s$ and $\langle !y\rangle = \mathrm{last}(s \restriction \{\mathrm{right}\})$

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

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

Similar to Lemma 1 and Lemma 5 we can show the following results.

Lemma 7.

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

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

Lemma 8.

(a) $Q'(s, t)/\langle \not\in \rangle = Q'(s, t)$

(b) $R'(x, s, t)/\langle \not\in \rangle = R'(x, s, t)$.

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

$$Q'(s, t) = \mathrm{PRUN}_{\not\in, ©}.$$

Lemma 9.

(a) $Q'(s,t) = (?x \longrightarrow (Q'(s^\frown\langle ?x\rangle, t)\ \{r_s^?\}$
$(R'(x, s^\frown\langle !e\rangle, t)\ \{r_s^!\}$
$\mathrm{PRUN}_{\not\leftarrow,\copyright}))$
$\Box r_s^! \,\&\, !e \longrightarrow Q'(s^\frown\langle !e\rangle, t)$
$\Box \not\leftarrow \longrightarrow Q'(s,t)$
$\Box \copyright \longrightarrow Q'(s,s))$

(b) $R'(x,s,t) = (!y \longrightarrow (Q'(s^\frown\langle ?x\rangle, t)\ \{r_s^?\}$
$(R'(x, s^\frown\langle !e\rangle, t)\ \{r_s^!\}$
$\mathrm{PRUN}_{\not\leftarrow,\copyright}))$
$\Box r_s^? \,\&\, ?z \longrightarrow R'(z, s^\frown\langle ?x\rangle, t)$
$\Box \not\leftarrow \longrightarrow R'(x,s,t)$
$\Box \copyright \longrightarrow R'(x,s,s)).$

Proof. Similar to Lemma 6.

Theorem 3. $BPB \parallel \mathrm{RUN}_{\not\leftarrow,\copyright} = \mathrm{PRE} \gg Ch(P) \gg \mathrm{POST}$

Proof. For any trace s of P we define

$$Q(s,t) = BP/sB \parallel \mathrm{RUN}_{\not\leftarrow,\copyright} \quad \text{for } t \leqq s$$

$$R(x,s,t) = B_x P/sB_y \parallel \mathrm{RUN}_{\not\leftarrow,\copyright} \quad \text{for } t \leqq s \text{ and } \langle !y\rangle = \mathrm{last}(s \upharpoonright \{\mathrm{right}\}).$$

From these definitions it follows that

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

and

$$R(x,s,t) = R(x,s,s) \quad \text{provided that } t \leqq s.$$

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

$$BPB \parallel \mathrm{RUN}_{\not\leftarrow,\copyright} = Q(\langle\,\rangle, \langle\,\rangle).$$

Moreover we have

$Q(s,t) = (?x \longrightarrow (BP/s\langle ?x\rangle B\ \{r_s^?\}$
$(B_x P/s^\frown\langle !e\rangle B_e\ \{r_s^!\}\ \mathrm{STOP})) \parallel \mathrm{RUN}_{\not\leftarrow,\copyright}$
$\Box r_s^! \,\&\, !e \longrightarrow BP/s^\frown\langle !e\rangle B \parallel \mathrm{RUN}_{\not\leftarrow,\copyright}$
$\Box \not\leftarrow \longrightarrow BP/sB \parallel \mathrm{RUN}_{\not\leftarrow,\copyright}$
$\Box \copyright \longrightarrow BP/sB \parallel \mathrm{RUN}_{\not\leftarrow,\copyright}.$
Lemma 2 and L3(a)
$= (?x \longrightarrow (Q(s^\frown\langle ?x\rangle, t)\ \{r_s^?\}$
$(R(x, s^\frown\langle !e\rangle, t)\ \{r_s^!\}\ \mathrm{PRUN}_{\not\leftarrow,\copyright}))$
$\Box r_s^! \,\&\, !e \longrightarrow Q(s^\frown\langle !e\rangle, t)$
$\Box \not\leftarrow \longrightarrow Q(s,t)$
$\Box \copyright \longrightarrow Q(s,s))$ L3(b)(c) and since $Q(s,t) = Q(s,s)$.

Similarly we can show that

$R(x,s,t) = (!y \longrightarrow (Q(s^\frown\langle ?x\rangle, t)\ \{r_s^?\}$
$(R(x, s^\frown\langle !e\rangle, t)\ \{r_s^!\}\ \mathrm{PRUN}_{\not\leftarrow,\copyright}))$
$\Box r_s^? \,\&\, ?z \longrightarrow R(z, s^\frown\langle ?x\rangle, t)$
$\Box \not\leftarrow \longrightarrow R(x,s,t)$
$\Box \not\leftarrow \longrightarrow R(x,s,s)).$

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

$$BPB \parallel \mathrm{RUN}_{\not\leftarrow,\copyright} = Q(\langle\,\rangle, \langle\,\rangle) = Q'(\langle\,\rangle, \langle\,\rangle) = \mathrm{PRE} \gg Ch(P) \gg \mathrm{POST}.$$

Discussion

For the particular problem treated in this paper, algebraic methods seem much preferable to more familiar assertional techniques (Hoare and Zhou 1981). 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 F be a process with l input channels 1.left, ..., l.left, and m output channels 1.right, ..., m.right. Let G be a process with m

input channels 1.left, ..., m.left, and n output channels 1.right, ..., n.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.right, ..., m.right of P are connected to the input channels 1.left, ..., m.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.

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

Definition. Let P be a process with l input channels 1.left, ..., l.left, and m output channels 1.right, ..., m.right. We define

$$BPB = B_l \gg_l P \gg_m B_m$$

where

$$B_l = \underset{1 \leqq i \leqq l}{\|} i:B$$

and

$$B_m = \underset{1 \leqq i \leqq m}{\|} i:B$$

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

$$BPB \| \text{RUN}_{\not\in} = \text{PRE}_l \gg_l \hat{P} \gg_m \text{POST}_m.$$

In fact, the structures of PRE_l and POST_m are similar to those defined in Sect. 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.

$$\text{PRE}_l = \underset{1 \leqq i \leqq l}{\|} i:\text{PRE}$$

where PRE was defined in Sect. 3.

(2) Removal of the simplification that the $\not\in$ and © events are detected simultaneously by all processes.

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

When the specification is appropriately weakened, the implementation could be based on the idea of Lamport (1978).

(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 1984). The use of algebraic transformations for correctness proofs has been pioneered in Baeten et al. (1983).

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

References

Backus J (1978) Can programming be liberated from the von Neuman style? Commun ACM 21 (8):613–641

Baeten JCM, Bergstra JA, Klop JW (1983) Syntax and defining equations for an interrupt mechanism in process algebra. Report CS-R8503. Centre for Mathematics and Computer Science, Amsterdam, The Netherlands

Burstall RM, Darlington J (1977) A transformation system for developing recursive programs. J ACM 24 (1):44–67

Goguen JA, Thatcher JW, Wagner EG (1978) An initial algebra approach to the specification, correctness and implementation of abstract data types. Current Trends in Programming Methodology, vol IV. Data Structuring Yeh RT (ed), Prentice-Hall, Englewood Cliffs, pp 80–149

Guttag JV, Horning JJ (1978) The algebraic specification of abstract data type. Acta Inf 10 (1): 27–52

Hoare CAR (1985) Communicating sequential processes. Prentice Hall International Series in Computer Science

Hoare CAR, Zhou C. C. (1981) Partial correctness of communicating sequential processes. Proc Internat Conf on Distributed Computing

INMOS Ltd (1984) The occam programming manual. Prentice-Hall

Lamport L (1978) Time, clocks and the ordering of events in a distributed system. Commun ACM 21 (7):558–565

Prasad KVS (1984) Specification and proof of a simple fault tolerant system in CCS. Internal report CSR-178-84, University of Edinburgh

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 $P[d/c]$ behaves the same as P except that the channel c is renamed by d.

The following laws of concealment in (Hoare 1985, 3.5) are useful in exploring the properties of the chaining operator

(1) If $B \cap C = \{\ \}$

then

$$(x{:}B \longrightarrow P(x)) \setminus C = (x{:}B \longrightarrow (P(x) \setminus C)).$$

(2) If $B \cap C \neq \{\ \}$ and is finite

then

$$(x{:}B \longrightarrow P(x)) \setminus C = Q \sqcap (Q \,\Box\, (x{:}B - C \longrightarrow P(x)))$$

where

$$Q = \prod_{x:B \cap C} P(x) \setminus C$$

(3) $(P \setminus C)/s = (\prod_{t \in T} P/t) \setminus C$

where

$$T = \text{traces}(P) \cap \{t \mid t \restriction (\alpha P - C) = s\}$$

provided that T is finite and $s \in \text{traces}(P \setminus C)$.

Here we offer proofs for laws of chaining quoted in Sect. 2.

(a) Let

$$P = (b1 \,\&\, ?x \longrightarrow P1(x) \,\Box\, b2 \,\&\, !e \longrightarrow P2 \;\Box\; y{:}B \longrightarrow P3(y))$$

and

$$Q = (c1 \,\&\, ?x \longrightarrow Q1(x) \,\Box\, c2 \,\&\, !f \longrightarrow Q2 \;\Box\; y{:}C \longrightarrow Q3(y))$$

then

$$P \gg Q = (T \,\Box\, U) \sqcap U \not\leftarrow b2 \wedge c1 \not\rightarrow T$$

where

$$T = (b1 \,\&\, ?x \longrightarrow P1(x) \gg Q \;\Box\; c2 \,\&\, !f \longrightarrow P \gg Q2 \;\Box\; y{:}B \cap C \longrightarrow P3(y) \gg Q3(y))$$

and

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

Proof.

$$\begin{aligned} LHS &= (P[\text{mid/right}] \parallel Q[\text{mid/left}]) \setminus \{\text{mid}\} \\ &= (b1 \,\&\, ?x \longrightarrow P1(x)[\text{mid/right}] \parallel Q[\text{mid/left}] \\ &\quad \Box c2 \,\&\, !f \longrightarrow P[\text{mid/right}] \parallel Q2[\text{mid/left}] \\ &\quad \Box y{:}B \cap C \longrightarrow P3(y)[\text{mid/right}] \\ &\qquad \parallel Q(y)[\text{mid/left}] \\ &\quad \Box (b2 \wedge c1) \,\&\, \text{mid}\,!e \longrightarrow P2[\text{mid/right}] \\ &\qquad \parallel Q1(e)[\text{mid/left}]) \setminus \{\text{mid}\} \\ &\qquad\qquad \text{L3(c) in Sect. 2} \\ &= RHS \quad \text{Law 1.2 of concealment.} \end{aligned}$$

(b)

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

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

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

Proof. We define

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

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

$$\begin{aligned} (P \gg Q)/\langle x \rangle &= \prod_{t \in T} (P[\text{mid/right}] \\ &\qquad \parallel Q[\text{mid/left}]/t) \setminus \{\text{mid}\} \\ &\qquad\qquad \text{Law 3 of concealment} \\ &= ((P[\text{mid/right}] \\ &\qquad \parallel Q[\text{mid/left}])/\langle x \rangle) \setminus \{\text{mid}\} \\ &\qquad\qquad \text{since } P \gg Q \text{ is deterministic} \\ &= (P[\text{mid/right}]/\langle x \rangle \\ &\qquad \parallel Q[\text{mid/left}]/\langle x \rangle) \setminus \{\text{mid}\} \\ &\qquad\qquad \text{L3(d) in Sect. 2} \\ &= (P/\langle x \rangle) \gg (Q/\langle x \rangle) \quad \text{def of } \gg. \end{aligned}$$

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

$$P \gg Q = (P/\text{right}.u) \gg (Q/\text{left}.u)$$

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

Proof. We define

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

From the assumption we have

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

and

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

which implies that

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

Thus we conclude that

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

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 (b).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 (b).2.