C.A.R. Hoare

A Theory of Synchrony and Asynchrony

He Jifeng, M.B. Josephs and C.A.R. Hoare
Programming Research Group
Oxford University Computing Laboratory

February 6, 1990

Abstract

Loosely-coupled (asynchronous) data flow networks are often constrasted to tightly-coupled (synchronous) systems. We present CSP [10] as a unified theory for both types of system, and deduce algebraic laws relating them. The theory may be useful in design and implementation of systems from parts which take advantage of both paradigms.

Keywords: Asynchrony, Synchrony, Communicating Process.

Contents

1 Introduction

A process can in general be defined by its behaviour, which evolves as a sequence of communications with its environment. Each communication is either an input of a message on a named input channel or the output of a message along a named output channel. When two processes are connected, output channels of each of them are connected to like-named input channels of the other. Networks of arbitrary complexity may thus be connected; but the behaviour of the network may be treated, in practice and in theory, as that of a single process.

For a synchronous process, each communication involves simultaneous participation both of the process and of the environment. If either of them is ready before the other, it must be delayed at least until the other is ready too. Then the communication takes place, and both process and environment may continue independently.

The problem with the implementation of synchronous networks is the delay and overhead of synchronisation. Recent breakthroughs in the architecture of communicating microprocessors [transputers] have reduced these almost to insignificance; but programmers using older operating systems [Unix] and communications technologies [ethernet] would be well advised to ensure that their programs still work in spite of unsynchronised communication. Our theory may help in this task.

For an asynchronous process, all channels are capable of buffering an arbitrary number of messages. Consequently, an output by the environment to the process is never delayed. Conversely, the environment may postpone indefinitely the acceptance of messages, without delaying the internal progress of the process.

The problem with the implementation of asynchronous processes is the provision of unbounded buffering on each channel. In practice, programmers must ensure that programs still work in spite of limited buffer size, and may have to complicate the program by feedback loops for flow control. Our theory may help in this transformation.

The method of the paper is to show how asynchronous processes may be modelled as a particular simple and attractive closed class of synchronous processes. This permits proof a number of elegant algebraic laws, which may be useful for transformation of programs expressed in one formalism for execution on a mechanism implementing the other. The theory as a whole may also be useful for designers and implementors of systems with components written and executing in different paradigms.

This paper can read as a companion paper to [7]; the latter presents a complete mathematical framework for the analysis and synthesis of asynchronous processes.

Summary

An infinite buffer may be expressed as a synchronous process BUF, which is always ready to input a message, and is ready to output the earliest waiting message whenever one exists.. If P is a synchronous process, a buffer can be attached to each of its input and output channels, giving a result defined as $\tilde{P}$. It is shown in section 3 that all asynchronous processes can be formed in this way.

Section 4 derives a number of algebraic laws, showing how the asynchronous operation $\tilde{}$ distributes through various synchronous operators

The results reported in this paper are part of a wider programme of research devoted to establish links between different notations, methods, and computational paradigms. The objectives are to aid in the transformation of specifications, designs and programs from simple mathematical abstractions to efficient computations in either hardware or software or both. Other possible applications include the design and construction of systems involving a mixture of language and computational paradigms.

Notation

The following CSP notion will be used in the later discussion. A communication is an event that is described by a pair $c.m$, where c is the name of the channel on which the communication takes place and m is the value of the message being passed. We will use Mes to stand for the set of messages.

Given P a process, $outch(P)$ is the set of output channel names of P, and $inch(P)$ the set of input channel names of P. $traces(P)$ is the set of finite sequences of communications which P can exchange with its environment. $failures(P)$ is the set of pairs (s, X), where s is a trace of P and X is a set of channel names on which P may refuse to engage in any communication after execution of the sequence s. $divergences(P)$ is the set of sequences of communications whose execution may cause the divergence of the process P. $F(P)$ is the set of traces after which P may refuse to output,

$$F(P) \stackrel{def}{=} \{s \mid (s, outch(P)) \in failures(P)\}$$

Note that

$$divergences(P) \subseteq F(P)$$

$out\_com(P)$ is used to denote the set of all output events of P, and $in\_com(P)$ the set of all input events, which are supposed to be finite. αP stands for the alphabet (in the sense of CSP) of process P, i.e.,

$$\alpha P = in\_com(P) \cup out\_com(P)$$

The process P' behaves the same as the process P except that all channel names are decorated with a dash. For any sequence s of communications, s' is the result of replacing channel names in all communications in s by the decorated names. The set A' is the result of replacing all symbols in A by the corresponding dashed symbols. We adopt the convention that

$$\begin{aligned} (P')' &= P \\ (s')' &= s \end{aligned}$$

We will use $\bigsqcup_n P_n$ to stands for the limit of an ascending chain $\{P_n\}$.

The expression $t \upharpoonright A$ denotes the sequence t when restricted to symbols in the set A. $\#t$ is the length of the sequence t. The set A^* is the set of all finite sequences (including the empty sequence) which are formed from symbols in the set A. We use $Chaos$ to denote the worst communicating process, whose behaviour is unpredictable and uncontrollable [10]. $Stop$ is the process which performs no action.

$\emptyset$ and $<>$ represent the empty set and the empty sequence respectively. Let u and v be sequences. Their concatenation is denoted $u \cdot v$. When u is a non-empty sequence, u_0 denotes its first element, and $tail(u)$ the sequence obtained by removing u_0 from u.

2 Preliminaries

The techniques we shall use for proving properties of asynchronous processes in the rest of this paper are based on algebraic calculations, which in practice amounts to no more than symbolic execution of input and output commands.

Two processes P and Q with $outch(P) = inch(Q)$ may be joined together so that the output channels of P are connected to the corresponding input channels of Q, and the sequence of messages output by P and input by Q along these lines are concealed from their common environment. The result of the connection is denoted

$$P >> Q$$

This definition is more general than that given in [10] because it simultaneously connects several channels. Formally it can be defined by

$$P >> Q \stackrel{def}{=} (P \parallel Q) \backslash outch(P)$$

where $\parallel$ stands for the concurrency operator in CSP [10], and $\backslash$ the concealment operator, and it is assumed that $inch(P)$ is disjoint from $outch(Q)$. For details we refer reader to [1, 10].

Now we intend to explore algebraic laws of the chaining operator $>>$, which are based on the following basic laws presented in [10].

(1) Law of Concurrency
Let $P = (x : B \rightarrow P(x))$ and $Q = (y : C \rightarrow Q(y))$ Then $P \parallel Q = (z : D \rightarrow R \parallel S)$ where

$$\begin{array}{rcll} D & = & (B \cap C) \cup (B - \alpha Q) \cup (C - \alpha P) & \\ R & = & P(z) & if\ z \in B \\ & = & P & otherwise \\ S & = & Q(z) & if\ z \in C \\ & = & Q & otherwise \end{array}$$

(2) Laws of concealment
(a) $(P \backslash B) \backslash C = P \backslash (B \cup C)$
(b) $(x \rightarrow P) \backslash C = x \rightarrow (P \backslash C) \quad if\ x \notin C$
$\qquad = P \backslash C \quad otherwise$
(c) If $\alpha P \cap \alpha Q \cap C = \emptyset$ then $(P \parallel Q) \backslash C = (P \backslash C) \parallel (Q \backslash C)$.
(d) $P \backslash C = P$ if $C \cap (inch(P) \cup outch(P)) = \emptyset$

(3) Laws of chaining
The most useful algebraic property of $>>$ is associativity
(a) $(P >> Q) >> R = P >> (Q >> R)$

The chaining operator is also strict
(b) $Chaos >> Q = Chaos = P >> Chaos$

The chaining operator distributes with the non-determinic choice operator $\sqcap$
(c) $P >> (Q \sqcap R) = (P >> Q) \sqcap (P >> R)$
$(P \sqcap Q) >> R = (P >> R) \sqcap (Q >> R)$

$>>$ is a continuous operator:
(d) $P >> \bigsqcup_i Q_i \;=\; \bigsqcup_i (P >> Q_i)$
$\bigsqcup_i P_i >> Q \;=\; \bigsqcup_i (P_i >> Q)$

The following expansion law completes the algebraic definition of the chaining operator.
(e) The expansion law of $>>$
Let $X \subseteq inch(P)$ $Y \subseteq outch(P)$ $V \subseteq inch(P)$ and $W \subseteq outch(Q)$

$$\begin{array}{rcl} P & = & \Box_{a\in X}\,(a?x \rightarrow P_a(x)) \;\Box\; \Box_{b\in Y}\,(b!n_b \rightarrow P_b) \\ Q & = & \Box_{b\in V}\,(b?y \rightarrow Q_b(y)) \;\Box\; \Box_{c\in W}\,(c!m_c \rightarrow Q_c) \end{array}$$

Then

$$P >> Q \;=\; \textbf{if } Y \cap V \neq \emptyset \textbf{ then } (T\backslash\backslash U) \textbf{ else } T$$

where $\backslash\backslash$ stands for composite choice, and is defined by

$$T\backslash\backslash U \;=\; (T \;\Box\; U) \sqcap U$$

The processes T and U are given by

$$\begin{array}{rcl} T & = & \Box_{a\in X}\,(a?x \rightarrow (P_a(x) >> Q)) \;\Box \\ & & \Box_{c\in W}\,(c!m_c \rightarrow (P >> Q_c)) \\ U & = & \sqcap_{b\in Y\cap V}\,(P_b >> Q_b(n_b)) \end{array}$$

The first line of the definition of T describes the case when the external input by P takes places first; in the second line the external output by Q takes place first.

The definition of U describes the case in which the internal communication takes place first, so that the value n_b is transmitted through the channel b from P to Q, but the communication is concealed. In all three cases, the process or processes which engage in communication make the appropriate progress, and they continue to be chained by $>>$.

The main difficulty and complexity in the above law is the clause $T\backslash\backslash U$ which results from the hiding on an internal communication [10].

The infinite buffer with an input channel l and an output channel r behaves like a process $BUF_{l,r}$, which is at all times ready to accept a message on its input channel l, and (whenever possible) is ready to deliver to its output channel r the earliest message which has been input but not yet output. The state of this process can be identified with the sequence s of messages which it has input but not output. Each incoming message x is added to the right hand end of s (to give $s\cdot <x>$); and the next outgoing message is given up by s_0. The sequence s is initially empty. Formally the process $BUF_{l,r}$ can be defined by a system of mutually recursive equations, one for each value of s:

$$BUF_{l,r} \stackrel{def}{=} BUF_{l,r}(<>)$$

where

$$\begin{array}{llll} BUF_{l,r}(s) & \stackrel{def}{=} & l?x \rightarrow BUF_{l,r}(<x>) & \quad if\ s=<> \\ BUF_{l,r}(s) & \stackrel{def}{=} & (l?x \rightarrow BUF_{l,r}(s\cdot <x>) & \\ & & \Box\; (r!s_0 \rightarrow BUF_{l,r}(tail(s))) & \quad otherwise \end{array}$$

The following two laws presented in [2] are useful in deriving further properties of process $BUF_{l,r}$ and the chaining operator

(f) If any two of A, B, $A >> B$ are buffers, then so is the third.

(g) If $A_s >> C_s$ is a buffer with an input channel l and an output channel r for all $s \in S$, then for any function $g : Mes \rightarrow S$ the process

$$l?x \rightarrow (A_{g(x)} >> (r!x \rightarrow C_{g(x)}))$$

is a buffer.

If a buffer holds a message, then either an input from the input channel or the output of that stored message may happen first
(h) $BUF_{l,r}(<m>) \; = \; BUF_{l,a} >> (r!m \rightarrow \; BUF_{a,r})$
Proof:

$$\begin{array}{lll} & l?x \rightarrow (BUF_{l,a} >> (r!x \rightarrow BUF_{a,r})) & \\ = & BUF_{l,r} & \{law\ (f)\ and\ (g)\ of\ >>\} \\ = & l?x \rightarrow BUF_{l,r}(<x>) & \{def\ of\ BUF_{l,r}\} \end{array}$$

from which follows the conclusion.

The sequence of messages buffered up is immaterial when the messages are never read.
(i) $BUF_{l,r}(<m>) >> Stop \; = \; BUF_{l,r} >> Stop$
Proof: For any finite sequence s we define

$$\begin{array}{lll} A_s & \stackrel{def}{=} & BUF_{l,r}(<m>\cdot s) >> Stop \\ B_s & \stackrel{def}{=} & BUF_{l,r}(s) >> Stop \end{array}$$

Then one has

$$\begin{array}{lll} & A_s & \\ = & l?x \rightarrow (BUF_{l,r}(<m>\cdot s\cdot <x>) >> Stop) & \{by\ the\ expansion\ law\ of\ >>\} \\ = & l?x \rightarrow A_{s\cdot <x>} & \end{array}$$

$$\begin{array}{lll} & B_s & \\ = & l?x \rightarrow (BUF_{l,r}(s\cdot <x>) >> Stop) & \{by\ the\ expansion\ law\ of\ >>\} \\ = & l?x \rightarrow B_{s\cdot <x>} & \end{array}$$

which indicates that processes A and B satisfy the same guarded recursive equations. By appealing to the unique fixed point theorem we reach the conclusion.

If the right component of a chain is only willing to accept an input, and the left compontent $BUF_{l,r}$ has held a message, then the internal communication will take place instantaneously
(j) $BUF_{l,r}(<m>) >> (r?x \rightarrow P(x)) \; = \; BUF_{l,r} >> P(m)$
Proof:

$$\begin{array}{lll} & LHS & \\ = & BUF_{l,a} >> (r!m \rightarrow BUF_{a,r}) >> (r?x \rightarrow P(x)) & \{law\ (h)\ of\ >>\} \\ = & BUF_{l,a} >> BUF_{a,r} >> P(m) & \{the\ expansion\ law\ of\ >>\} \\ = & RHS & \{law\ (f)\ of\ >>\} \end{array}$$

Parallel composition of independent processes is interchangable with the chaining operator
(k) If P and Q have disjoint channels from R and Q, then

$$(P_1 >> Q_1) \parallel (P_2 >> Q_2) \;=\; (P_1 \parallel P_2) >> (Q_1 \parallel Q_2)$$

Proof:

$$\begin{array}{lll} & LHS & \\ = & ((P_1 \parallel Q_1)\backslash outch(P)) \parallel ((P_2 \parallel Q_2)\backslash outch(P_2)) & \{def\ of\ >>\} \\ = & ((P_1 \parallel Q_1) \parallel (P_2 \parallel Q_2))\backslash(outch(P_1) \cup outch(P_2)) & \{law\ (c)\ (d)\ of\ the\ concealment\} \\ = & RHS & \{def\ of\ >>\} \end{array}$$

In the remainder of this paper we will use a useful function on sequences, **duplic**:

$$\begin{array}{rcll} \mathbf{duplic}(<>) & \stackrel{def}{=} & <> & \\ \mathbf{duplic}(< c.m > \cdot t) & \stackrel{def}{=} & < c.m,\ c'.m > \cdot\mathbf{duplic}(t) & if\ c \in inch \\ & \stackrel{def}{=} & < c'm,\ c.m > \cdot\mathbf{duplic}(t) & if\ c \in outch \end{array}$$

The function **duplic** maps the computation history of a process P to one of the possible computation histories of the buffered process $IN_P \parallel P' \parallel OUT_P$ where the interface processes IN_P and OUT_P are defined by

$$\begin{array}{rcl} IN_P & \stackrel{def}{=} & \|_{a \in inch(P)}\ BUF_{a,a'} \\ OUT_P & \stackrel{def}{=} & \|_{c \in outch(P)}\ BUF_{c',c} \end{array}$$

The subscript of the interfaces and buffers will be dropped if it is clear from the context.

Lemma 2.1

(1) $t \in traces(P) \;\Rightarrow\; \mathbf{duplic}(t) \in traces(IN_P \parallel P' \parallel OUT_P)$
(2) $t \in divergences(P) \;\Rightarrow\; \mathbf{duplic}(t) \in divergences(IN_P \parallel P' \parallel OUT_P)$
(3) $(t, outch(P)) \in failures(P) \;\Rightarrow$
$\quad(\mathbf{duplic}(t) \uparrow \alpha IN_P,\ inch(P')) \in failures(IN_P)$
$\quad\wedge\ (\mathbf{duplic}(t) \uparrow \alpha P',\ outch(P')) \in failures(P')$
$\quad\wedge\ (\mathbf{duplic}(t) \uparrow \alpha OUT_P,\ outch(P)) \in failures(OUT_P)$

Proof: Direct from the definition of **duplic** and $\parallel$.

Finally we introduce a binary relation on sequences of communications which represents the way in which buffering can reorder communications on distinct channels. Define $s \preceq t$ if there exists a sequence u such that

$$\begin{array}{l} s = u \uparrow (in\_com \cup out\_com)\ \wedge \\ u \uparrow (in\_com' \cup out\_com') \;=\; \mathbf{duplic}(t) \uparrow (in\_com' \cup out\_com')\ \wedge \\ \forall a \in inch \bullet u \uparrow \alpha\, BUF_{a,a'} \;=\; \mathbf{duplic}(t) \uparrow \alpha\, BUF_{a,a'}\ \wedge \\ \forall c \in outch \bullet u \uparrow \alpha\, BUF_{c',c} \;=\; \mathbf{duplic}(t) \uparrow \alpha\, BUF_{c',c} \end{array}$$

Lemma 2.2

(1) If both a and b are distinct input channel names then $< a.m,\ b.n > \preceq < b.n,\ a.m >$.
(2) If both c and d are distinct output channel names then $< c.m,\ d.n > \preceq < d.n,\ c.m >$.
(3) Let $a \in inch$ and $c \in outch$, then $< a.m,\ c.n > \preceq < c.n,\ a.m >$.
(4) $\preceq$ is respected by catenation: $s_1 \preceq t_1$ and $s_2 \preceq t_2$ implies $s_1 \cdot s_2 \preceq t_1 \cdot t_2$.

Proof: (1) Taking $u = < a.m,\ b.n,\ b'.n,\ a'.m >$.
(2) Taking $u = < d'.n,\ c'.m,\ c.m\ d.n >$.
(3) Taking $u = < a.m,\ c'.n,\ c.n,\ a'.m >$.
(4) From the fact that $\mathbf{duplic}(t_1 \cdot t_2) = \mathbf{duplic}(t_1) \cdot \mathbf{duplic}(t_2)$.

3 The asynchronous subset of CSP

We postulate that the asynchronous processes are a subset of the synchronous processes, namely, those processes P that satisfy the defining equation

$$P = IN >> P' >> OUT$$

We justify this in two complementary ways: first it will be shown that the computation history of the solutions of the defining equation actually meet those requirements on an asynchronous process that are given in the relevant literature [6, 7, 4, 14] (theorem 3.3); and second, any process satisfying those required properties is proved to be a solution of the defining equation (theorem 3.4). The defining equation is known as the Foam Rubber Wrapper postulate when used to characterize delay-insensitive circuits [14].

We start this section by exploring some simple facts about asynchronous processes:
Theorem 3.1
If $P = IN >> Q >> OUT$, then P is asynchronous.
Proof:

$$\begin{array}{lll} & IN >> P' >> OUT & \\ = & IN >> (IN >> Q >> OUT)' >> OUT & \{by\ the\ assumption\} \\ = & \|_{a \in inch(P)} (BUF_{a,i} >> BUF_{i,a'}) >> Q & \\ & >> \|_{c \in outch(P)} (BUF_{c',i} >> BUF_{i,c}) & \{law\ (k)\ of\ >>\} \\ = & IN >> Q >> OUT & \{law\ (f)\ of\ >>\} \\ = & P & \{by\ the\ assumption\} \end{array}$$

Theorem 3.2
If P is asynchronous then $P >> OUT = P = IN >> P$.
Proof: Similar to theorem 3.1.

Here we recall a reordering relation $\sqsubseteq$ on sequences of communications introduced in [7, 4], where $s \sqsubseteq t$ means s is obtainable from t by moving inputs before outputs, and by changing the interleaving of communications on distinct channels. Formally, $\sqsubseteq$ is defined as the smallest binary relation with the following properties:

1. It is a preorder.
2. It is respected by catenation: $s \sqsubseteq t$ and $u \sqsubseteq v$ implies $s \cdot u \sqsubseteq t \cdot v$.
3. The order in which the environment sends data along different input channels (say a and b) does not matter: $< a.m, b.n > \sqsubseteq < b.n, a.m >$.
4. Data on different output channels (say c and d) may be received by the user in any order: $< c.m, d.n > \sqsubseteq < d.n, c.m >$.
5. If the environment can receive data on an output channel c then it can still receive the same data after sending further data along an input channel a:
$< a.m, c.n > \sqsubseteq < c.n, a.m >$

In fact, the binary relation $\sqsubseteq$ is no more than the reflexive and transitive closure of the binary relation $\preceq$:
Lemma 3.1
$\sqsubseteq = \bigcup_{n \geq 0} \preceq^n$
Proof: From lemma [illegible]

As mentioned in section 2, the interfaces IN and OUT are introduced to store inputs which have been received from the environment, but have not been consumed, and outputs which have been produced, but have not been delivered to the environment, respectively. This fact can be formalised by the following lemma:

Lemma 3.2

$u \in traces(IN \parallel P' \parallel OUT) \Rightarrow$
$$\exists v \in out\_com(P)^*, w \in in\_com(P)^* \bullet (u \uparrow \alpha P) \cdot v \sqsubseteq (u \uparrow \alpha P')' \cdot w$$

Now comes one of the main results of this section:

Theorem 3.3

An asynchronous process P possesses the following properties

1. any trace can always be extended by an input event:
$$s \in traces(P) \Rightarrow s \cdot < a.m > \in traces(P)$$
for any $a.m \in in\_com(P)$.

2. any trace can be extended by a finite sequence of output events so that it refuses to output:
$$s \in traces(P) \Rightarrow \exists t \in out\_com(P)^* \bullet s \cdot t \in F(P)$$

3. the set $F(P)$ is non-empty and closed wrt. the reordering $\sqsubseteq$:
$$F(P) \neq \emptyset \wedge (s \sqsubseteq t \wedge t \in F(P) \Rightarrow s \in F(P))$$

4. the set of $divergences(P)$ is also closed wrt. the reordering $\sqsubseteq$:
$$s \sqsubseteq t \wedge t \in divergences(P) \Rightarrow s \in divergences(P)$$

5. If P diverges after its environment has received data from it, so does P before the environment received that data:
$$s \cdot < c.m > \in divergences(P) \Rightarrow s \in divergences(P)$$
for all $c.m \in out\_com(P)$.

6. If P can engage in an unbounded amount of output before receiving an input from its environment, then it is diverging:
$$s \in traces(P) \wedge (\forall n, \exists t \in out\_com(P) \bullet \#t > n \wedge s \cdot t \in traces(P)) \Rightarrow s \in divergences(P)$$

7. $(s, X) \in failures(P)$ iff
$$\begin{aligned} & s \in traces(P) \\ \wedge\ & (s \in divergences(P) \\ & \vee X \subseteq outch(P) \wedge \exists t \in out\_com(P)^* \bullet s \cdot t \in F(P) \wedge t \uparrow (X \times Mes) = <>) \end{aligned}$$

Proof: See appendix.

The more important result is the inverse of theorem 3.3: the properties (1)–(7) are indeed characterised by the defining equation in the following sense:

Theorem 3.4

Any process that possesses the properties (1)–(7) in theorem 3.3 is asynchronous.
Proof: See appendix.

The defining equation gives us a complete subset of communicating sequential processes.

Theorem 3.5

The set of asynchronous processes is a complete partial order with least element *Chaos*.

Proof: Direct from the laws (c) and (d) of the chaining operator.

The chaining operator is to asynchronous processes what sequential composition is to imperative programs.

Theorem 3.6

If both P and Q are asynchronous processes, so is $P >> Q$.

Proof:

$$\begin{array}{lll} & P >> Q & \\ = & (IN_P >> P' >> OUT_P) >> (IN_Q >> Q' >> OUT_Q) & \{by\ the\ assumption\} \\ = & IN_P >> (P' >> OUT_P >> IN_Q >> Q') >> OUT_Q & \{law\ (a)\ of\ >>\} \\ = & IN_P >> (P >> Q)' >> OUT_Q & \{theorem\ 3.2\} \end{array}$$

Let C be a set of channel names, then $P \backslash C$ is a process which behaves like P except that each occurrence of any communication along the channels in C is concealed. Asynchronous processes are also closed wrt. the CSP concealment operator.

Theorem 3.7

If P is asynchronous, so is $P \backslash C$. When C contains input channel names then $P \backslash C = Chaos$.

Proof: Suppose that C contains input channel name a. From theorem 3.3 it follows that for any $n > 0$ there is a sequence t of communications occurring along the channel a such that $t \in traces(P)$, which implies

$$<> \in divergences(P \backslash C)$$

as required.

Now consider the case that C only contains output channel names. Let

$$\begin{array}{rcl} D & \stackrel{def}{=} & outch(P) - C \\ OUT_1 & \stackrel{def}{=} & \|_{d \in D}\ BUF_{d',d} \end{array}$$

Then one has

$$\begin{array}{lll} & P \backslash C & \\ = & (IN >> P' >> OUT) \backslash C & \{def\ of\ P\} \\ = & (IN \parallel P' \parallel OUT) \backslash (inch(P') \cup outch(P') \cup C) & \{def\ of\ >>\} \\ = & (IN \parallel (P' \parallel (\|_{c \in C}\ BUF_{c',c})) \backslash C' \backslash C) \parallel OUT_1) \backslash (inch(P') \cup D') & \\ & & \{law\ (c)\ of\ the\ concealment\} \\ = & IN >> (P' \backslash C') >> OUT_1 & \{theorem 3.2\} \end{array}$$

Putting two asynchronous processes with disjoint channels in parallel will produce an asynchronous network.

Theorem 3.8

Let P and Q be asynchronous processes with disjoint channels, then $P \parallel Q$ is also asynchronous.

Proof:

$$\begin{array}{lll} & P \parallel Q & \\ = & (IN_P >> P' >> OUT_P) \parallel (IN_Q >> Q' >> OUT_Q) & \{by\ the\ assumption\} \\ & (IN_P \parallel P' \parallel OUT_P) \backslash (inch(P') \cup outch(P')) \parallel & \end{array}$$

$$\begin{array}{lll} & (IN_Q \parallel Q' \parallel OUT_Q)\backslash(inch(Q') \cup outch(Q')) & \{def\ of\ >>\} \\ = & ((IN_P \parallel IN_Q) \parallel (P' \parallel Q') \parallel (OUT_P \parallel OUT_Q))\backslash & \\ & (inch(P') \cup outch(P') \cup inch(Q') \cup outch(Q')) & \{law\ (c)\ of\ the\ concealment\} \\ = & (IN_P \parallel IN_Q) >> (P' \parallel Q') >> (OUT_P \parallel OUT_Q) & \{def\ of\ >>\} \end{array}$$

Let f be an injective function which maps the set of channel names of P onto a set of channel names. We define the process $f(P)$ as one which engages in the communication $f(a).m$ whenever P would have engaged in $a.m$.

Theorem 3.9

If P is asynchronous, so is $f(P)$.

Proof: Trivial.

4 A CSP operator for asynchrony

In this section we show how a CSP process can be mapped to an asynchronous process, and explore the properties of this mapping. The theorems form the basis of an algebraic characterisation of a theory of asynchronous processes.

We transform a CSP process to an asynchronous process by chaining it in between two interfaces which store its input and output respectively. Formally the mapping is defined by

$$\tilde{P} \stackrel{def}{=} (IN >> P' >> OUT)$$

From theorem 3.1 it follows that the above function always delivers an asynchronous process as the result. For convenience the decoration $'$ will be dropped in the later discussion.

A buffer which can store at most one message can be defined by a simple recursion

$$Copy = l?x \rightarrow r!x \rightarrow Copy$$

It is not asynchronous since it will refuse to input when already storing a message. The function $\tilde{}$ maps this one-place buffer to an unbounded buffer:

$$BUF_{l,l'} >> Copy >> BUF_{r',r} = BUF_{l,r}$$

The process $\tilde{Stop}$ is always prepared to input, but never output and diverge, this is because

$$\begin{array}{lll} & IN >> Stop >> OUT & \\ = & IN >> Stop & \{the\ expansion\ law\ of\ >>\} \\ = & \square_{a\in inch}\, a?x \rightarrow (IN(<a.x>) >> Stop) & \{the\ expansion\ law\ of\ >>\} \\ = & \square_{a\in inch}\, a?x \rightarrow (IN >> (a'!x \rightarrow BUF_{a,a'}) >> Stop) & \\ & & \{law\ of\ (h)\ of\ >>\} \\ = & \square_{a\in inch}\, a?x \rightarrow (IN >> Stop) & \{the\ expansion\ law\ of\ >>\} \end{array}$$

where

$$IN(<a.x>) \stackrel{def}{=} BUF_{a,a'}(<x>) \parallel (\parallel_{b\in inch-\{a\}} BUF_{b,b'})$$

The asynchrony operator enjoys a number of algebraic properties.

Theorem 4.1

It is idempotent, strict, distributive wrt. non-determinic choice and continuous

(1) $\tilde{\tilde{P}} = \tilde{P}$

(2) $\tilde{Chaos} = Chaos$

(3) $(P \sqcap Q)^{\tilde{}} = \tilde{P} \sqcap \tilde{Q}$
(4) $(\bigsqcup_n P_n)^{\tilde{}} = \bigsqcup_n \tilde{P}_n$
Proof:

$$
\begin{array}{lll}
 & (\tilde{\tilde{P}}) & \\
= & IN >> \tilde{P} >> OUT & \{def\ of\ \tilde{}\} \\
= & \tilde{P} & \{theorem\ 3.2\}
\end{array}
$$

The remaining conclusion follows from law (b), (c) and (d) of $>>$.

If P and Q have disjoint channels, then it does matter if we put interfaces on them independently or together:

Theorem 4.2

If P and Q have disjoint channels then $(P \parallel Q)^{\tilde{}} = \tilde{P} \parallel \tilde{Q}$.
Proof:

$$
\begin{array}{ll}
 & (P \parallel Q)^{\tilde{}} \\
= & \parallel_{a \in inch(P) \cup inch(Q)} BUF_{a,a'} >> (P \parallel Q) >> \parallel_{c \in outch(P) \cup outch(Q)} BUF_{c',c} \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{def\ of\ \tilde{}\} \\
= & (IN_P >> P >> OUT_P) \parallel (IN_Q >> Q >> OUT_Q) \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{law\ of\ (k)\ of\ >>\} \\
= & \tilde{P} \parallel \tilde{Q} \qquad\qquad\qquad\qquad\qquad\qquad\qquad \{def\ of\ \tilde{}\}
\end{array}
$$

The asynchrony operator $\tilde{}$ also distributes through the chaining operator.

Theorem 4.3

$(\tilde{P} >> Q)^{\tilde{}} = (P >> \tilde{Q})^{\tilde{}} = \tilde{P} >> \tilde{Q}$
Proof:

$$
\begin{array}{lll}
 & (\tilde{P} >> Q)^{\tilde{}} & \\
= & IN_P >> (\tilde{P} >> Q) >> OUT_Q & \{def\ of\ \tilde{}\} \\
= & IN_P >> (IN_P >> P >> OUT_P >> Q) >> OUT_Q & \{def\ of\ \tilde{}\} \\
= & IN_P >> (P >> (IN_Q >> Q >> OUT_Q) >> OUT_Q & \{inch(Q) = outch(P)\} \\
= & (P >> \tilde{Q})^{\tilde{}} & \{def\ of\ \tilde{}\} \\
= & (IN_P >> P >> OUT_P) >> (IN_Q >> Q >> OUT_Q) & \{inch(P) = outch(Q)\} \\
= & \tilde{P} >> \tilde{Q} & \{def\ of\ \tilde{}\}
\end{array}
$$

The nested application of $\tilde{}$ in a guarded process can be removed.

Theorem 4.4

(1) $(c!m \to P)^{\tilde{}} = (c!m \to \tilde{P})^{\tilde{}}$
(2) $(a?x \to P(x))^{\tilde{}} = (a?x \to P(x)^{\tilde{}})^{\tilde{}}$
(3) $([\!]_{a \in X}\ a?x \to P_a(x)\ [\!]\ [\!]_{c \in Y}\ c!m_c \to Q_b)^{\tilde{}} = ([\!]_{a \in X}\ a?x \to P_a(x)^{\tilde{}}\ [\!]\ [\!]_{c \in Y}\ c!m_c \to Q_c^{\tilde{}})^{\tilde{}}$
Proof:
(1) Define

$$OUT(< c.m >) \stackrel{def}{=} BUF_{c',c}(< m >) \parallel (\parallel_{d \in outch - \{c\}} BUF_{d',d})$$

$$
\begin{array}{lll}
 & LHS & \\
= & IN >> (c!m \to P) >> OUT & \{def\ of\ \tilde{}\} \\
= & IN >> P >> OUT(< c.m >) & \{the\ expansion\ law\ of\ >>\} \\
= & IN >> \tilde{P} >> OUT(< c.m >) & \{law\ (h)\ of\ >>\} \\
= & IN >> (c!m \to \tilde{P}) >> OUT & \{the\ expansion\ law\ of\ >>\} \\
 & RHS &
\end{array}
$$

(2) Similar to (1).
(3) Similar to (1).

The composite choice $P\backslash\backslash Q$ can select Q internally before the environment offers the choice; Q plays the same role as a *skip-guarded process* in a guarded choice [5]. It will not matter if Q has been chained between the interfaces or not.

Theorem 4.5
$(P\backslash\backslash Q)^{\sim} = (P\backslash\backslash \tilde{Q})^{\sim}$.
Proof: Similar to theorem 4.4.

The fact that the order in which an asynchronous process transmits messages on distinct channels does not determine the order in which the environment receives data is described by the following law.

Theorem 4.6
$(c!m \rightarrow d!n \rightarrow P)^{\sim} \;=\; (d!n \rightarrow c!m \rightarrow P)^{\sim}$.
Proof: Define

$$OUT(<c.m>, <d.n>) \stackrel{def}{=} BUF_{c',c}(<m>) \parallel BUF_{d'd}(<n>) \,|||_{r\in outch-\{c,d\}}\, BUF_{r',r}$$

$$\begin{array}{lll}
 & LHS & \\
= & (c!m \rightarrow (d!n \rightarrow P)^{\sim})^{\sim} & \{theorem\ 4.4\} \\
= & IN >> (d!n \rightarrow P)^{\sim} >> OUT(<c.m>) & \{the\ expansion\ law\ of\ >>\} \\
= & IN >> P >> OUT(<d.n>) >> OUT(<c.m>) & \{the\ expansion\ law\ of\ >>\} \\
= & IN >> P >> OUT(<c.m>, <d,n>) & \{law\ (h)\ of\ >>\} \\
= & RHS & \{by\ a\ mirror\ argument\}
\end{array}$$

The order in which an asynchronous process waits for messages from distinct input channels does not matter.

Theorem 4.7
$(a?x \rightarrow b?y \rightarrow P(x,y))^{\sim} \;=\; (b?y \rightarrow a?x \rightarrow P(x,y))^{\sim}$.
Proof: Similar to theorem 4.6.

If the right component of a chain is ready for an output, then that event can take place first.

Theorem 4.8
$(P >> (c!m \rightarrow Q))^{\sim} = (c!m \rightarrow (P >> Q))^{\sim}$
Proof:

$$\begin{array}{lll}
 & RHS & \\
= & IN_P >> (c!m \rightarrow (P >> Q)) >> OUT_Q & \{def\ of\ ^{\sim}\} \\
= & IN_P >> (P >> Q) >> OUT_Q(<c.m>) & \{the\ expansion\ law\ of\ >>\} \\
= & IN_P >> (P >> (c!m \rightarrow Q)) >> OUT_Q & \{the\ expansion\ law\ of\ >>\} \\
= & LHS & \{def\ of\ ^{\sim}\}
\end{array}$$

If both components of a chain of asynchronous processes are ready to communicate with each other, then the internal message will be transferred instantaneously.

Theorem 4.9
$(c!m \rightarrow P)^{\sim} >> (c?x \rightarrow Q(x))^{\sim} = \tilde{P} >> Q(m)^{\sim}$
Proof:

LHS

$$
\begin{aligned}
&= \tilde{P} >> OUT_P(< c.m >) >> IN_Q >> (c?x \rightarrow Q(x)) >> OUT_Q \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{\textit{the expansion law of } >>\} \\
&= \tilde{P} >> OUT_P(< c.m >) >> (c?x \rightarrow Q(x)) >> OUT_Q \qquad \{\textit{law } (h) \textit{ of } >>\} \\
&= \tilde{P} >> OUT_P >> Q(m) >> OUT_Q \qquad \{\textit{the expansion law of } >>\} \\
&= RHS \qquad \{\textit{theorem } 3.2\}
\end{aligned}
$$

If the left component of a chain is waiting for an input from the environment, and the right one is waiting to receive a message from the left one, then the chain will not make progress until the environment sends it data.

Theorem 4.10

$(a?x \rightarrow P(x))\tilde{} >> (c?y \rightarrow Q(y))\tilde{} = (a?x \rightarrow (P(x)\tilde{} >> (c?y \rightarrow Q(y)))\tilde{}$

Proof: Similar to theorem 4.9.

The effect of concealment is to allow any communication along the concealed channels to occur automatically and instantaneously, but make such occurrences totally invisible. Unconcealed communications will remain unchanged.

Theorem 4.11

(1) $(c!m \rightarrow P)\tilde{}\backslash C = \tilde{P}\backslash C$ *provided that* $c \in C$

(2) $(c!m \rightarrow P)\tilde{}\backslash C = (c!m \rightarrow P\backslash C)\tilde{}$ *provided that* $c \notin C$

(3) $(a?x \rightarrow P(x))\tilde{}\backslash C = (a?x \rightarrow (P(x)\backslash C))\tilde{}$ *provided that* $C \cap inch = \emptyset$

Proof:

(1) Let $OUT_1 \stackrel{def}{=} \|_{d \in outch(P)-\{c\}} BUF_{d',d}$.

$$
\begin{aligned}
& LHS \\
&= (IN >> P >> OUT(< c.m >))\backslash C \qquad \{\textit{the expansion law of } >>\} \\
&= (IN >> P >> (OUT_1 \parallel BUF_{c',c}(< m >)))\backslash C \qquad \{\textit{def of } OUT_1\} \\
&= (IN >> P >> (OUT_1 \parallel (BUF_{c',c}(< m >)\backslash\{c\})))\backslash C - \{c\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \{\textit{law } (c) \textit{ of concealment}\} \\
&= (IN >> P >> (OUT_1 >> (BUF_{c',i} >> (c!m \rightarrow BUF_{i,c})\backslash\{c\})))\backslash C - \{c\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \{\textit{law } (h) \textit{ of } >>\} \\
&= (IN >> P >> (OUT_1 >> (BUF_{c',c}\backslash\{c\})))\backslash C - \{c\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \{\textit{law } (b) \textit{ of the concealment}\} \\
&= RHS \qquad\qquad\qquad\qquad\qquad \{\textit{law } (c) \textit{ of the concealment}\}
\end{aligned}
$$

(2) Similar to (1).

(3) Similar to (1).

The operator $\tilde{}$ is interchangable with the renaming operator,

Theorem 4.12

$(f(P))\tilde{} = f(\tilde{P})$

5 Conclusion

Asynchronous processes (data flow networks) have been well-studied before [3, 4, 6, 8, 9, 12, 13]. Our main contribution is to show how they can be formally related to synchronous processes within the framework of CSP. This complements our earlier work in [7]. The advantages of our approach include

1. An axiomatic framework for the description of asynchronous processes. (Algebraic laws presented in [10, 11] with the unique fixed point theorem allow us to prove properties of asynchronous processes by symbolic execution of communications. The calculations have been simplified by prior development of a calculus of pipes in section 2. Algebraic methods seem often preferable to the direct manipulation of (finite and infinite) traces.)

2. The integration of asynchronous processes into the mathematical theory of Communicating Sequential Processes. (Both synchronous processes and asynchronous processes can be tackled in a unified conceptual framework. As a result, some techniques and tools (e.g., [11]) being developed for specification and implementation of CSP processes can be applied to design asynchronous systems effectively. In particular, the complete set of CSP laws and its proof system provide a way of transforming networks of processes into forms more suitable for sequential execution.)

Acknowledgement
This research was supported in part by the Science and Engineering Research Council of Great Britain and by Esprit Basic Research Actions.

References

[1] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe. *A theory of communicating sequential processes.* J.ACM 31 (7), (1984) 560–599.

[2] S.D. Brookes and A.W. Roscoe. *An improved failures model for communicating sequential processes.* LNCS 197, Springer-Verlag, (1984) 281–305.

[3] M. Broy. *Semantics of finite and infinite networks of concurrent communicating agents.* Distributed Computing 2 (1), (1987) 13–31.

[4] K.M. Chandy and J. Misra. *Reasoning about networks of communicating processes.* (Unpublished) Presented at INRIA Advanced NATO Study Institute on Logic and Models for Verification and Specification of Concurrent Systems, France (1984).

[5] INMOS Ltd. *Occam 2 Reference Manual.* Prentice-Hall International Series in Computer Sciences, (1985).

[6] B. Jonsson. *A model and proof system for asynchronous processes.* Proc. 4th ACM Symp. on Pricinple of Distributed Computing (1985), 49–58.

[7] M.B. Josephs, C.A.R. Hoare and He Jifeng. *A theory of asynchronous processes.* submitted to J.ACM.

[8] G. Kahn. *The semantics of a simple language for parallel processing.* In Rosenfeld JL (ed.) Information Processing 74. Proc of IFIP Congress 74, Amsterdam, North-Holland, (1974) 471–475.

[9] R.M. Keller and P. Panangaden. *Semantics of networks containing indeterminate operators.* Distributed Computing 1, (1986) 235–245.

[10] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International in Computer Sciences, (1985).

[11] A.W. Roscoe and C.A.R. Hoare. *The laws of occam programming.* TCS. 60 (2), (1988) 177–229.

[12] J. Misra. *Equational reasoning about non-deterministic processes.* (1989).

[13] J. Staples and V.N. Ngoyen. *A fixpoint semantics for nondeterministic data flow.* J.ACM 32 (2), (1985) 411–444.

[14] J.T.Udding. *Classification and Composition of Delay-Insensitive Circuits.* Ph.D. Thesis, Eindhoven University of Technology (1984).

6 Appendix

In the later proof we will use the following properties of the failure set of communicating sequential processes in [10]:

$$\begin{array}{rcl} (s, X) \in failures(P) \ \wedge \ Y \subseteq X & \Rightarrow & (s, Y) \in failures(P) \\ s \in divergences(P) \ \wedge \ t \in (\alpha P)^* & \Rightarrow & s \cdot t \in divergences(P) \end{array}$$

Proof of Theorem 3.3
(1)

$$\begin{array}{lll} & s \in traces(P) & \\ \Rightarrow & s \in traces(IN >> P' >> OUT) & \{P = IN >> P' >> OUT\} \\ \Rightarrow & \exists u \bullet s = u \uparrow \alpha P \ \wedge & \\ & \quad u \uparrow \alpha IN \in traces(IN) \ \wedge & \\ & \quad u \uparrow \alpha OUT \in traces(OUT) \ \wedge & \\ & \quad u \uparrow \alpha P' \in traces(P') & \{def\ of\ >>\} \\ \Rightarrow & \exists u \bullet s \cdot < a.m >= (u \cdot < a.m >) \uparrow \alpha P \ \wedge & \\ & \quad (u \cdot < a.m >) \uparrow \alpha IN = (u \uparrow \alpha IN) \cdot < a.m > \in traces(IN) \ \wedge & \\ & \quad (u \cdot < a.m >) \uparrow \alpha OUT = u \uparrow \alpha OUT \in traces(OUT) \ \wedge & \\ & \quad (u \cdot < a.m >) \uparrow \alpha P' = u \uparrow \alpha P' \in traces(P') & \{def\ of\ BUF\} \\ \Rightarrow & s \cdot < a.m > \in traces(IN >> P' >> OUT) & \{def\ of\ >>\} \\ \Rightarrow & s \cdot < a.m > \in traces(P) & \{P = IN >> P' >> OUT\} \end{array}$$

(2) Let $u =$ **duplic**(s). Then from lemma 2.1 one has

$$\begin{array}{l} s = u \uparrow \alpha P \ \wedge \ u \uparrow \alpha IN \in traces(IN) \ \wedge \\ u \uparrow \alpha OUT \in traces(OUT) \ \wedge \ u \uparrow \alpha P' \in traces(P') \end{array}$$

Now consider two cases:
(a) After the process P' engages in $u \uparrow \alpha P'$ it will be able to deliver an unbounded amount of output before receiving an input from its environment. In this case one has

$$\begin{array}{lll} & \forall n, \exists v \in out\_com(P') \bullet \#v > n \ \wedge \ (u \uparrow \alpha P') \cdot v \in traces(P') & \\ \Rightarrow & \forall n, \exists v \in out\_com(P') \bullet & \\ & \quad \#v > n \ \wedge & \\ & \quad (u \cdot v) \uparrow \alpha IN = u \uparrow \alpha IN \in traces(IN) \wedge & \\ & \quad (u \cdot v) \uparrow \alpha OUT = (u \uparrow \alpha IN) \cdot v \in traces(OUT) \wedge & \\ & \quad (u \cdot v) \uparrow \alpha P' = (u \uparrow \alpha P') \cdot v \in traces(P') & \{def\ of\ BUF\} \\ \Rightarrow & s \in divergences(IN >> P' >> OUT) & \{def\ of\ >>\} \\ \Rightarrow & s \in divergences(P) & \{P = IN >> P' >> OUT\} \\ \Rightarrow & s \in F(P) & \{def\ of\ F(P)\} \end{array}$$

(b) Otherwise there is a finite sequence v of outputs such that

$$((u \uparrow \alpha P') \cdot v, \ outch(P')) \in failures(P')$$

which implies that $s \cdot v' \in F(P)$ as required.
(3) Since $<> \in traces(P)$, from (2) it follows that

$$\exists t \in out\_com(P)^* \bullet t \in F(P)$$

i.e., $F(P)$ is nonempty.

$$
\begin{array}{ll}
& s \preceq t \ \wedge \ t \in F(P) \\
\Rightarrow & s \preceq t \ \wedge \ (t,\ outch(P)) \in failures(P) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{def\ of\ F(P)\} \\
\Rightarrow & s \preceq t \ \wedge \\
& \exists w \bullet w = \mathbf{duplic}(t) \ \wedge \ w \in traces(IN \parallel P' \parallel OUT) \ \wedge \\
& \qquad (w,\ outch(P) \cup inch(P') \cup outch(P')) \in failures(IN \parallel P' \parallel OUT) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{lemma\ 2.1\} \\
\Rightarrow & \exists u, w \bullet \ s = u \uparrow \alpha P \ \wedge \ w = \mathbf{duplic}(t) \ \wedge \\
& \qquad \forall a \in inch(P) \bullet u \uparrow \alpha BUF_{a,a'} \ = \ w \uparrow \alpha BUF_{a,a'} \ \wedge \\
& \qquad u \uparrow \alpha P' \ = \ w \uparrow \alpha P' \ \wedge \\
& \qquad \forall c \in outch(P) \bullet u \uparrow \alpha BUF_{c',c} \ = \ w \uparrow \alpha BUF_{c',c} \ \wedge \\
& \qquad (w,\ outch(P) \cup inch(P') \cup outch(P')) \in failures(IN \parallel P' \parallel OUT) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{def\ of\ \preceq\} \\
\Rightarrow & \exists u \bullet \ s = u \uparrow \alpha P \ \wedge \\
& \quad (u,\ outch(P) \cup inch(P') \cup outch(P')) \in failures(IN \parallel P' \parallel OUT) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{def\ of\ \parallel\} \\
\Rightarrow & (s,\ outch(P)) \in failures(P) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad \{def\ of\ >>\} \\
\Rightarrow & s \in F(P) \qquad\qquad\qquad\qquad\qquad\qquad \{def\ of\ F(P)\}
\end{array}
$$

from which and lemma 2.2 we reach the conclusion

$$s \sqsubseteq t \ \wedge \ t \in F(P) \ \Rightarrow \ s \in F(P)$$

(4) Similar to (3).
(5) Similar to (3).
(6) Similar to (2).
(7)

$$
\begin{array}{ll}
& (\exists X \subseteq outch(P), t \in out\_com(P)^* \bullet \ s \cdot t \in F(P) \ \wedge \ t \uparrow (X \times Mes) \ = <>) \\
\Rightarrow & (\exists X \subseteq outch(P), t \in out\_com(P)^* \bullet s \cdot t \in F(P) \ \wedge \ t \uparrow (X \times Mes) \ = <> \wedge \\
& \qquad\qquad\qquad\qquad \mathbf{duplic}(s) \cdot t' \in traces(IN \parallel P' \parallel OUT)) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{lemma\ 2.1\} \\
\Rightarrow & (\exists X \subseteq outch(P), t \in out - com(P)^* \ \bullet \ t \uparrow (X \times Mes) \ = <> \wedge \\
& \ ((\mathbf{duplic}(s) \cdot t') \uparrow \alpha IN,\ inch(P')) \in failures(IN) \ \wedge \\
& \ s \cdot t \in F(P) \ \wedge \\
& \ ((\mathbf{duplic}(s) \cdot t') \uparrow \alpha OUT,\ X) \in failures(OUT)) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{def\ of\ \parallel\ and\ BUF\} \\
\Rightarrow & (\mathbf{duplic}(s) \cdot t',\ inch(P') \cup outch(P') \cup X) \in failures(IN \parallel P' \parallel OUT) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{def\ of\ \parallel\} \\
\Rightarrow & ((\mathbf{duplic}(s) \cdot t') \uparrow \alpha P,\ X) \in failures(IN >> P' >> OUT) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{def\ of\ concealment\} \\
\Rightarrow & (s,\ X) \in failures(P) \qquad\qquad\qquad\qquad \{P = IN >> P' >> OUT\}
\end{array}
$$

It is easy to establish the remaining part of (7).

Proof of Theorem 3.4

Suppose that Q possesses the properties (1)–(7). Define

$$S \stackrel{def}{=} IN \parallel Q' \parallel OUT$$
$$P \stackrel{def}{=} IN >> Q' >> OUT$$

First we want to prove that

$$divergences(P) = divergences(Q)$$

(a)

$$\begin{array}{lll} & s \in divergences(Q) & \\ \Rightarrow & \mathbf{duplic}(s) \in divergences(S) & \{lemma\ 2.1\} \\ \Rightarrow & s = \mathbf{duplic}(s) \uparrow \alpha Q \in divergences(P) & \{def\ of\ the\ concealment\} \end{array}$$

On the other hand, let $s \in divergences(P)$, one has either

(b)

$$\begin{array}{lll} & \exists u \bullet (s = u \uparrow \alpha Q \ \wedge \ u \in divergences(S)) & \\ \Rightarrow & \exists u \bullet (s = u \uparrow \alpha Q \ \wedge \ u \in traces(S) \ \wedge & \\ & \quad u \uparrow \alpha Q' \in divergences(Q')) & \{def\ of\ S\ and\ \parallel\} \\ \Rightarrow & \exists u, v, w \bullet (v \in out\_com(Q)^* \ \wedge \ w \in in\_com(Q)^* \ \wedge & \\ & \quad s \cdot v \sqsubseteq (u \uparrow \alpha Q')' \cdot w \ \wedge & \\ & \quad u \uparrow \alpha Q' \in divergences(Q')) & \{lemma\ 3.2\} \\ \Rightarrow & \exists u, v, w \bullet (v \in out\_com(Q)^* \ \wedge \ w \in in\_com(Q)^* \ \wedge & \\ & \quad s \cdot v \sqsubseteq (u \uparrow \alpha Q')' \cdot w \ \wedge & \\ & \quad (u \uparrow \alpha Q') \cdot w' \in divergences(Q')) & \\ & & \{def\ of\ divergences(Q)\} \\ \Rightarrow & \exists v \bullet (v \in out - com(Q)^* \ \wedge s \cdot v \in divergences(Q)) & \{property\ (4)\} \\ \Rightarrow & s \in divergences(Q) & \{property\ (5)\} \end{array}$$

or

(c)

$$\begin{array}{lll} & \exists u \bullet (s = u \uparrow \alpha Q \ \wedge \ u \in traces(S) \ \wedge & \\ & \quad \forall n, \exists v \in out\_com(Q')^* \bullet \#v > n \ \wedge \ u \cdot v \in traces(S)) & \\ \Rightarrow & \exists u \bullet (s = u \uparrow \alpha Q \ \wedge \ u \in traces(S) \ \wedge & \\ & \quad \forall n, \exists v \in out\_com(Q')^* \bullet \#v > n \ \wedge \ (u \uparrow \alpha Q') \cdot v \in traces(Q')) & \\ & & \{def\ of\ S\} \\ \Rightarrow & \exists u, w \bullet (s = u \uparrow \alpha Q \ \wedge \ u \in traces(S) \ \wedge & \\ & \quad u \uparrow \alpha Q' \in divergences(Q')) & \{property\ (6)\} \\ \Rightarrow & s \in divergences(Q) & \{see\ (b)\} \end{array}$$

Combine (a), (b) and (c) we conclude that $divergences(P) = divergences(Q)$.

Now we wish to prove that

$$failures(Q) \quad failures(P)$$

From (a)–(c) and property (7) we only need consider those failures (s, X) with

$$s \notin divergences(Q) \ \wedge \ X \subseteq outch(Q)$$

(d)

$$
\begin{array}{ll}
 & (s,\ X) \in failures(Q) \\
\Rightarrow & \exists t \in out\_com(Q)^* \bullet\ s \cdot t \in F(Q) \ \wedge \ t \uparrow (X \times Mes) = <> \\
 & \hfill \{property\ (7)\} \\
\Rightarrow & \exists t \in out\_com(Q)^* \bullet\ \mathbf{duplic}(s) \in traces(S) \ \wedge \ t \uparrow (X \times Mes) = <> \wedge \\
 & \mathbf{duplic}(s) \cdot t' \in traces(S) \ \wedge \ s \cdot t \in F(Q) \hfill \{lemma\ 2.1\} \\
\Rightarrow & \exists t \in out\_com(Q)^* \bullet \\
 & (\mathbf{duplic}(s) \uparrow \alpha IN,\ inch(Q')) \in failures(IN) \ \wedge \\
 & (s' \cdot t',\ outch(Q')) \in failures(Q') \ \wedge \\
 & ((\mathbf{duplic}(s) \uparrow \alpha OUT) \cdot t',\ X) \in failures(OUT) \\
 & \hfill \{lemma\ 2.1\} \\
\Rightarrow & \exists t \in out\_com(Q)^* \bullet \\
 & (\mathbf{duplic}(s) \cdot t',\ inch(Q') \cup outch(Q') \cup X) \in failures(S) \quad \{def\ of\ \|\} \\
\Rightarrow & \exists t \in out\_com(Q)^* \bullet\ ((\mathbf{duplic}(s) \cdot t') \uparrow \alpha Q,\ X) \in failures(P) \\
 & \hfill \{def\ of\ the\ concealment\} \\
\Rightarrow & (s,\ X) \in failures(P) \hfill \{property\ of\ \mathbf{duplic}\}
\end{array}
$$

(e)

$$
\begin{array}{ll}
 & (s,\ X) \in failures(P) \ \wedge \ s \notin divergences(P) \\
\Rightarrow & \exists u \bullet (s = u \uparrow \alpha Q \ \wedge \\
 & \quad (u \uparrow \alpha IN,\ inch(Q')) \in failures(IN) \ \wedge \\
 & \quad (u \uparrow \alpha Q',\ outch(Q')) \in failures(Q') \ \wedge \\
 & \quad (u \uparrow \alpha OUT,\ X) \in failures(OUT)) \qquad \{def\ of\ P\ and\ property\ (7)\} \\
\Rightarrow & \exists u, v \bullet (v \in out\_com(Q)^* \ \wedge v \uparrow (X \times Mes) = <> \wedge \\
 & \qquad s = u \uparrow \alpha Q \ \wedge \\
 & \qquad s \cdot v \sqsubseteq (u \uparrow \alpha Q')' \ \wedge \\
 & \qquad (u \uparrow \alpha Q',\ outch(Q')) \in failures(Q')) \hfill \{lemma\ 3.2\} \\
\Rightarrow & \exists v \bullet (v \in out\_com(Q)^* \ \wedge \ v \uparrow (X \times Mes) = <> \wedge \ s \cdot v \in F(Q)) \\
 & \hfill \{property\ (3)\} \\
\Rightarrow & (s,\ X) \in failures(Q) \hfill \{property\ (7)\}
\end{array}
$$

This completes the proof.