INTRODUCTION

In this paper we present an algebraic analysis of concurrent processes with three types of asynchronous communication. Our starting point is an algebraic axiomatisation $\text{PA}_0$ (for Process Algebra) of concurrent processes without communication, in which the concurrency is that of the free merge, or arbitrary interleaving, of atomic actions. Such concurrent cooperation of processes is \textit{asynchronous cooperation}, as each process may operate in connection with its own clock. The system $\text{PA}_0$ was first introduced in [7] together with an extension to an axiomatisation $\text{ACP}$ (for Algebra of Communicating Processes) of concurrent processes with a communication mechanism.

The laws for communication in $\text{ACP}$, like those in Milner's CCS, concern \textit{synchronous communication}, requiring the synchronisation or simultaneous execution $a \mid b$ of so-called communication actions $a, b$. In research on laws for concurrency, while concurrent cooperation has been examined in its asynchronous and synchronous cases, concurrent process communication in the asynchronous case has been neglected. In this paper we take up the idea of \textit{asynchronous communication} wherein a communication by actions $a, b$ is consistent with $b$ being performed after $a$ (say). We have devised algebraic treatments of this idea based upon three models:

(i) mail via a queue-like channel;
(ii) mail via a bag-like channel;
(iii) causality in systems.

The plan of the paper is this: Section 1 introduces the axiom system $\text{PA}_0$ describing the free merge of processes; here 0 is a constant for process failure or deadlock. This axiom system underlies the three axiom systems we present. Section 2 is devoted to the distinctions been cooperation/communication and synchronous/ asynchronous, and attempts a classification of formalisms such as CCS, CSP, NEIJE, SCCS, CHILL and so forth. Section 3 presents the algebraic systems for (i) and (ii) above; and Section 4 presents the system for (iii) together with an involved example on the
control of a printer.

This paper is part of a long series of reports on process algebra and its applications, including [6,7]. The paper can be read independently, though knowledge of part of [7] may be helpful; in addition, [7] contains a discussion of related approaches to the algebraic theory of concurrency, including CCS and SCCS in Milner [23,24].

We thank Ms. Judith Thursby for her preparation of this typescript.

1. PROCESS ALGEBRA WITHOUT COMMUNICATION

As a point of departure we consider an algebraic axiom system PA_0 that analyses concurrent processes without communication. The system PA_0 is derived from [7] where a system PA was introduced for concurrent process algebra without communication, and the δ-laws for process deadlock were introduced in a system ACP for concurrent process algebra with synchronous communication.

Process algebra is concerned with concurrent processes made from a finite set A of atomic processes or actions, including a special failed or deadlocked process δ∈A. There are four process generating binary operations,

- alternative composition (sum)
- sequential composition (product)
- parallel composition (merge)
- left-merge

and these components satisfy a set PA_0 of axioms given below.

1.1 Signature

More formally, let Σ_{PA_0} be the following signature:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Sort</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>P</td>
<td>sorts</td>
</tr>
<tr>
<td>F</td>
<td>: P × P → P</td>
<td>functions</td>
</tr>
<tr>
<td></td>
<td>: P × P → P</td>
<td></td>
</tr>
<tr>
<td></td>
<td>: P × P → P</td>
<td></td>
</tr>
<tr>
<td></td>
<td>: P × P → P</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>a</td>
<td>for all a ∈ A</td>
</tr>
</tbody>
</table>

1.2 Axioms

Let PA_0 be the set of equations over Σ_{PA_0} in Table 1.
\[ x + y = y + x \quad \text{A1} \\
(x + y) + z = x + (y + z) \quad \text{A2} \\
x + x = x \quad \text{A3} \\
(x + y)z = xz + yz \quad \text{A4} \\
(xy)z = x(yz) \quad \text{A5} \\
x + \delta = x \quad \text{A6} \\
\delta x = \delta \quad \text{A7} \\
x||y = x||y + y||x \quad \text{M1} \\
a||x = ax \quad \text{M2} \\
(ax)||y = a(x||y) \quad \text{M3} \\
(x + y)||z = x||z + y||z \quad \text{M4} \]

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al</td>
</tr>
<tr>
<td>A4</td>
</tr>
<tr>
<td>A7</td>
</tr>
<tr>
<td>M3</td>
</tr>
</tbody>
</table>

1.3 **Semantics** A \( \pi_{\mathfrak{P}_{A_0}} \)-structure \( \mathcal{P} \) satisfying the axioms in \( \mathfrak{P}_{A_0} \) is a process algebra with deadlock; the class of all such algebras we denote \( \text{ALG}(\pi_{\mathfrak{P}_{A_0}}, \mathfrak{P}_{A_0}) \).

In analogy with the theory of data type specifications, it is useful to consider the equational axiomatisation \( (\pi_{\mathfrak{P}_{A_0}}, \mathfrak{P}_{A_0}) \) in two ways:

(i) as an initial algebra specification (in the sense of ADJ[1]) for the special structure \( A_\omega \) of all finite processes with deadlock i.e. the initial algebra semantics of the specification is \( I(\pi_{\mathfrak{P}_{A_0}}, \mathfrak{P}_{A_0}) \cong A_\omega \)

(ii) as a general axiomatic specification of such concurrent process algebras with semantics \( \text{ALG}(\pi_{\mathfrak{P}_{A_0}}, \mathfrak{P}_{A_0}) \).

These views rest on the distinction between finite and infinite processes, which requires technical elaboration:

Let \( \mathcal{P} \models \mathfrak{P}_{A_0} \) be any process algebra. For \( p \in \mathcal{P} \) and \( \alpha \in A^* \cup A^\omega \), the set of finite or infinite sequences of actions from \( A \), we will define what it means for \( \alpha \) to be a trace of \( p \):

**Definition.** (i) If \( \alpha = a_1^*a_2^*...a_n \), where \( a_i \in A \) (\( i=1,\ldots,n \)) and * denotes concatenation, then \( \alpha \) is a trace of \( p \) if there are \( p_1,\ldots,p_n, q_1,\ldots,q_n,q_{n+1} \in \mathcal{P} \) such that

\[
p = p_1, \quad p_i = a_i p_{i+1} + q_i \quad (i=1,\ldots,n-1) \\
p_n = a_n + q_n \\
(i=1,\ldots,n)
\]

(ii) If \( \alpha = a_1^*a_2^*... \) then we call \( \alpha \) a trace of \( p \) if there are \( p_1, q_1 \) such that

\[
p = p_1, \quad p_i = a_i p_{i+1} + q_i \quad (i\geq1).
\]
If $p \in P$ has an infinite trace, it is an infinite process; otherwise it is finite. The initial algebra $I(\mathcal{C}_{PA_0}, PA_0)$ contains finite processes only.

There are various ways to construct process algebras that contain infinite processes, most of which have been developed for the more general case of communicating processes. The synchronisation trees (modulo observational equivalence or bisimulation) from Milner [23] (see also Winskel [31]) constitute such a model if one considers the degenerate case of the absence of synchronisation primitives. In De Bakker & Zucker [3,4] a topological construction is given via metric spaces, and in Bergstra & Klop [7] an equivalent algebraic construction using projective limits. Bergstra, Klop & Tucker [8] describes a direct algebraic construction by means of adjoining solutions of suitable fixed point equations. The solution of recursion equations is important in the theory because such equations constitute an important specification tool for process definition; these equations require infinite processes for their solution. The projective limit constructions and the topological constructions lead to models in which all guarded systems of equations can be solved.

2. \textbf{COOPERATION AND COMMUNICATION}

2.1 \textbf{A Classification ofConcurrency} Informally, one thinks of processes as logical configurations of atomic acts. A process $p$ is executed as follows: choose a first action, perform it; then choose a second action that is possible after the first action (according to the definition of the process), perform it; and so on. On thinking of the parallel execution of processes one involves notions to do with time and clocks. Informally, in the parallel execution of two processes $p,q$, two basic kinds of process cooperation can be distinguished:

\textit{Synchronous Cooperation}: the regime of synchronous cooperation allows $p,q$ to be executed in parallel with the same speed as measured by the same clock; this idea is incorporated in SCCS [14,24], ASP [7], MEIJE [2,28].

\textit{Asynchronous Cooperation}: the regime of asynchronous cooperation allows $p,q$ to proceed in parallel with their own speeds, as measured by their own independent clocks; this idea is incorporated in CSP [15-17], CCS [23], ACP [7], with restrictions determined by possible mutual interactions between processes, and in the system PA$_0^*$, where there are no interactions.

Now, in the interaction between the atomic actions of two processes $p,q$ two basic kinds of process communication can be distinguished:

\textit{Synchronous Communication}: the regime of synchronous communication requires that communication between actions $a,b$ can take place only if both are performed simultaneously; this type of communication is sometimes called \textit{handshaking} and is incorporated in CSP, CCS, ACP, and Ada.

\textit{Asynchronous Communication}: the regime of asynchronous communication allows communication between actions $a,b$ to be consistent with $b$ being performed after $a;$
this idea is encorporated in CHILL [9].

Combining the above regimes one arrives at four categories which can be used to classify models of concurrent processes, namely:

**SS**  
synchronous cooperation + synchronous communication  
SCCS, MEIJE, ASP, ASCCS

**SA**  
synchronous cooperation + asynchronous communication  
No example known to us.

**AS**  
asynchronous cooperation + synchronous communication  
CCS, CSP, ACP, Ada, Petri nets,  
uniform processes of [3,4]

**AA**  
asynchronous cooperation + asynchronous communication  
CHILL, data flow networks  
restoring circuit logic

2.2 Comments on Examples  

It might be puzzling why ASCCS, which gives according to Milner [24] a framework for "asynchronous processes", is classified under SS. The reason is that it is a subcalculus of SCOS, and hence also employs synchronous cooperation and synchronous communication - even though asynchronously cooperating processes may be interpreted in ASCCS.

The combination AA in studied for instance using temporal logic in Pnueli [26], Lamport [21] and Kojmans, Vytopil & de Roever [19], Kuiper & de Roever [20]. Moreover, trace theories are used to describe the semantics of data flow networks (see Kahn [18], Brock & Ackerman [10]) and the semantics of restoring circuit logic (see Ebergen [13], Rem [27] and Van de Snepscheut [29]. Restoring circuit logic is intended to describe the behaviour of circuits regardless of delays in the connecting wires. This delay insensitivity leads to the classification under AA.

A discussion of the case AA in an algebraic setting is absent to our knowledge. In Milne [22] and Bergstra & Klop [6] the AA case is reduced to the AS case for switching circuits and data flow networks respectively. We are not aware of any "direct" algebraic descriptions of the AA case.

2.3 The AA Case  
One may imagine a wild variety of different mechanisms for asynchronous communication. We will now proceed to describe three mechanisms for asynchronous communication that are consistent with asynchronous cooperation.
The mechanisms are closely related to one another:

(i) Mail via an order-preserving channel (cf a queue)
(ii) Mail via a non-order-preserving channel (cf a bag)
(iii) A causal mechanism wherein one action causes another.

For each of these mechanisms we will present an algebraic notation based upon
(a) a special purpose alphabet of atomic actions;
(b) an appropriate encapsulation operator; and
(c) a set of axioms to specify the semantics of the mechanism.

In each case the axiom system is an extension of PA_0; cases (i) and (ii) we will complete in the next section while case (iii) we will treat in Section 4. It may be helpful to make a comparison with the construction of ACP as an extension of PA_0.

3. MAIL VIA A CHANNEL

We will treat the cases of mail via an order-preserving channel and mail via a non-order-preserving channel together since the syntax and axioms proposed for these mechanisms coincide to a large extent.

3.1 The alphabet. Let B be a finite set of actions. Let D be a finite set of data, and c a special symbol for channel. For all d ∈ D there are actions

\[ \mathrm{c} \cdot d \quad \text{send data } d \text{ via channel } c \text{ considered as a potential action} \]
\[ \mathrm{c} + d \quad \text{send data } d \text{ via channel } c \text{ considered as an actual action} \]
\[ \mathrm{c} \cdot d \quad \text{receive data } d \text{ via channel } c \text{ considered as a potential action} \]
\[ \mathrm{c} + d \quad \text{receive data } d \text{ via channel } c \text{ considered as an actual action} \]

The distinction between \( \mathrm{c} \cdot d \) and \( \mathrm{c} + d \) may be slightly unusual. \( \mathrm{c} \cdot d \) indicates an internal, intended, potential, or future action while \( \mathrm{c} + d \) denotes an external, realised, actual, or past action; and similarly for \( \mathrm{c} + d \) and \( \mathrm{c} \cdot d \).

This distinction is implicit in the synchronous communication operator of ACP where a communication takes the form \( a\mid b = c \) for atomic acts \( a, b, c \). By virtue of the equation, \( a, b \) can be seen as potential actions giving rise to the communication \( c \) as an actual action.

Let \( \mathrm{c} \cdot D = \{ \mathrm{c} \cdot d \mid d \in D \} \) and likewise for \( \mathrm{c} + D \), etc.

Now we define the alphabet to be

\[ A = B \cup \{ \mathrm{c} \cdot d \} \cup \{ \mathrm{c} + d \} \cup \{ \mathrm{c} \cdot d \} \cup \{ \mathrm{c} + d \}. \]

Note that the cardinality \( |A| = |B| + 4|D| + 1 \).

The actions \( b \in B \) are not related to channel \( c \). Although we specify syntax and axioms for one channel \( c \) only, the presence of several channels, \( c, c', \ldots \) is entirely unproblematic; in that case, \( B \) may also contain actions \( c' \cdot d \) etc. since these are not related to channel \( c \).
3.2 **Encapsulation Operator.** Here the situation divides into the cases of mail via an order-preserving channel (3.2.1) and mail via an non-order-preserving channel (3.2.2).

3.2.1 **Queue-like Channel.** Let $D^*$ be the set of sequences $\sigma$ of data $d \in D$. The empty sequence is denoted by $\varepsilon$. Concatenation of sequences $\sigma, \tau$ is denoted as $\sigma \tau$; especially if $\sigma = \langle d_1, \ldots, d_n \rangle$ then $\varepsilon \sigma = \langle d_1, \ldots, d_n \rangle$ and $\sigma \delta = \langle d_1, \ldots, d_n, \delta \rangle$. Further, if $n \geq 1$, last ($\sigma$) = $d_n$.

Now for each $\sigma \in D^*$ there is an encapsulation operator $\mu^\sigma_c : \mathcal{P} \rightarrow \mathcal{P}$ where $\mathcal{P}$ is a domain of processes (i.e. the elements of a process algebra satisfying the axioms below). Informally, if $x$ is a process, then $\mu^\sigma_c(x)$ denotes the process obtained by requiring that the channel $c$ initially contains a data sequence $\sigma$ and that no communications with care performed outside $x$. Thus, $x$ and $\mu^\sigma_c(x)$ correspond to internal and external views of a system's behaviour, in some sense.

There are other relevant intuitions about encapsulation. The process $\mu^\sigma_c(x)$ can be viewed as the result of the partial execution of $x$ with respect to $c$ with initial contents $\sigma$. By execution we mean the transformation of internal or potential actions like $c \triangleright d$ into an external or actual actions like $c \triangleright d$, and their effect on processes (cf Remark 4.6). Encapsulation is formally defined by axioms M01-9 below.

3.2 **Bag-like Channel.** For the bag-like channel the situation is very much the same except that a data sequence $\sigma$ is now a multiset of data. We denote a finite multiset of $d \in D$ by $M$. Now for all finite multisets $M$ over $D$ we introduce again an encapsulation or partial execution operator

$$\mu^M_c : \mathcal{P} \rightarrow \mathcal{P}$$

3.3 **The signature.** Although the various ingredients of the signature, both for the cases of mail via a queue-like channel and via a bag-like channel, have now all been introduced, we will display these signatures once more in Table 2.

3.4 **Axioms.** Suppose a set $B$ of actions, a set $D$ of data and a channel name $c$ are given. Then we have the following axiom systems :

$$\text{PA}^\sigma(c, B, D)$$ in Table 3

$$\text{PA}^M(c, B, D)$$ in Table 4

for mail via a queue-like channel and mail via a bag-like channel, respectively.

Here $a$ varies over the alphabet $A = B \cup \{\delta\} \cup c \triangleright D \cup c \triangleright D \cup c \triangleright D \cup c \triangleright D$, and $\varepsilon$ varies over $E = B \cup \{\delta\} \cup c \triangleright D \cup c \triangleright D$. 
3.5 **Semantics.** The axiom systems $PA_0(\mu_c^\alpha, B, D)$ and $PA_0(\mu_c^\beta, B, D)$ determine initial algebras

$$A_\omega^\alpha (+, \cdot, ||, \delta, \mu_c^\alpha, B, D)$$

$$A_\omega^\beta (+, \cdot, ||, \delta, \mu_c^\beta, B, D)$$

respectively. These are just enrichments of the initial algebra $I(PA)$ denoted $A_\omega$ of $PA_0$. Using a projective limit construction as with ACP in [7], or a topological completion as in [3,4], it is possible to construct larger models

$$A_\omega^\infty (+, \cdot, ||, \delta, \mu_c^\alpha, B, D)$$

$$A_\omega^\infty (+, \cdot, ||, \delta, \mu_c^\beta, B, D)$$

with infinite processes, in which all guarded systems of equations can be solved.

3.6 **Examples.** We will now give some examples both for the case of an order-preserving channel and the case of non-order-preserving channel.

3.6.1 **Example for a queue-like channel.** Consider the following very simple data flow network:

![Diagram](image)

**Figure 1.**
\[\mathcal{PA}_0(\mu^\sigma_{c}, B, D)\]

\[
\begin{align*}
    &x + y = y + x &\text{A1} \\
    &(x + y) + z = x + (y + z) &\text{A2} \\
    &x + x = x &\text{A3} \\
    &(x + y)z = xz + yz &\text{A4} \\
    &(xy)z = x(yz) &\text{A5} \\
    &x + \delta = x &\text{A6} \\
    &\delta x = \delta &\text{A7} \\
    &x\|y = x\|y + y\|x &\text{M1} \\
    &a\|x = ax &\text{M2} \\
    &ax \| y = a(x\| y) &\text{M3} \\
    &(x + y)\| z = x\| z + y\| z &\text{M4} \\
    &\mu^\sigma_c(e) = e &\text{M01} \\
    &\mu^\sigma_c(ex) = e\cdot\mu^\sigma_c(x) &\text{M02} \\
    &\mu^\sigma_c(c+d) = c\#d &\text{M03} \\
    &\mu^\sigma_c(c+d.x) = c\#d\cdot\mu^\sigma_c(x) &\text{M04} \\
    &\mu^\sigma_c(c\#d) = c\#d &\text{M05} \\
    &\mu^\sigma_c(c\#d.x) = c\#d\cdot\mu^\sigma_c(x) &\text{M06} \\
    &\mu^\sigma_c(c+d) = \delta \text{ if } d \neq \text{last } (\sigma) \text{ or } \sigma = \epsilon &\text{M07} \\
    &\mu^\sigma_c(c+d.x) = \delta \text{ if } d \neq \text{last } (\sigma) \text{ or } \sigma = \epsilon &\text{M08} \\
    &\mu^\sigma_c(x + y) = \mu^\sigma_c(x) + \mu^\sigma_c(y) &\text{M09}
\end{align*}
\]

**Table 3**  
\[a \in A, \ e \in E, \ \sigma \in D^*\]
\[ x + y = y + x \quad \text{A1} \]
\[ (x+y) + z = x + (y+z) \quad \text{A2} \]
\[ x + x = x \quad \text{A3} \]
\[ (x+y)z = xz + yz \quad \text{A4} \]
\[ (xy)z = x(yz) \quad \text{A5} \]
\[ x + \delta = x \quad \text{A6} \]
\[ \delta x = \delta \quad \text{A7} \]
\[ x \parallel y = x \parallel y + y \parallel x \quad \text{M1} \]
\[ a \parallel x = ax \quad \text{M2} \]
\[ ax \parallel y = a(x \parallel y) \quad \text{M3} \]
\[ (x+y) \parallel z = x \parallel z + y \parallel z \quad \text{M4} \]

\[
\begin{align*}
\mu_c^e(e) &= e \\
\mu_c^e(ex) &= e \cdot \mu_c^e(x) \\
\mu_c^{c+d} &= c \cdot d \\
\mu_c^{c+d,x} &= c \cdot d \cdot \mu_c^{d}(x) \\
\mu_c^{M+d} &= c \cdot d \\
\mu_c^{M+d,x} &= c \cdot d \cdot \mu_c^{M}(x) \\
\mu_c^{c+d} &= \delta \text{ if } d \notin M \\
\mu_c^{c+d,x} &= \delta \text{ if } d \notin M \\
\mu_c^{x+y} &= \mu_c^{x}(x) + \mu_c^{y}(y)
\end{align*}
\]

Table 4. \((a \in A, e \in E, M \text{ a multiset over } D)\)
with actions

\( rp(d) \) \hspace{0.5cm} \text{processor \( f \) reads value \( d \) at port \( p \)}

\( wq(d) \) \hspace{0.5cm} \text{processor \( g \) writes \( d \) at port \( q \)}

There are two order-preserving channels \( c_1 \) and \( c_2 \). Internally, the node \( f \) satisfies

\[
f = \sum_{d \in D} (rp(d) + c_2 + d) \cdot c_1 \cdot d \cdot f.
\]

So, node \( f \) merges the inputs from \( p \) and \( c_2 \) and emits these through \( c_1 \). The node \( g \) is defined by

\[
g = \sum_{d \in D} c_1 \cdot d \cdot (i \cdot c_2 \cdot a(d) + i \cdot wq(d)) \cdot g
\]

The effect of the internal step \( i \) is to make the choice nondeterministic, and \( a : D \rightarrow D \) is a transformation of the data; thus \( g \) obtains \( d \) from \( c_1 \) and then chooses whether to 'recycle' \( a(d) \) via \( c_2 \) or to output \( d \) via port \( q \).

The network \( N \) is now externally described by

\[
N = \mu^E_{c_1} \mu^E_{c_2} (f \parallel g).
\]

Note that the actions \( c_2 \cdot d \), \( c_1 \cdot d \), \( c_1 \cdot d \) and \( c_1 \cdot d \) are unrelated to \( c_2 \) and thereby work as \( b \)'s in the definition for \( \mu^E_{c_2} \). Conversely, the send and receive actions for \( c_2 \) are unrelated to \( c_1 \).

3.6.2 Example for a queue-like channel. Consider the very simple communication protocol as in Figure 2:

![Figure 2](image)

\[
T = \mu^E_{c_1} \mu^E_{c_2} (S \parallel R).
\]

In fact the protocol \( T \) satisfies the following recursion equation (as one easily computes from the axioms in \( PA_\delta(\mu^\alpha_{E,R,D}) \)):

\[
T = \sum_{d \in D} rp(d) \cdot c_1 \cdot d \cdot c_2 \cdot wq(d) \cdot c_2 \cdot \text{ack} \cdot c_2 \cdot \text{ack} \cdot T.
\]
This recursion equation constitutes an external specification of $T$.

3.6.3 Example for a bag-like channel:

(i) $\mu^\emptyset_c(\text{ctd} \cdot c+d) = c\downarrow d \cdot c\downarrow d$

(ii) $\mu^\emptyset_c(\text{ctd} \cdot \Sigma_{u \in D} c+u) = c\downarrow d \cdot c\downarrow d$

(iii) $\mu^\emptyset_c(\text{ctd} \parallel c+d) = c\downarrow d \cdot c\downarrow d$

(iv) $\mu^\emptyset_c(\text{ctd}1 \cdot \text{ctd}2 \cdot \Sigma_{u \in D} c+u \cdot \Sigma_{u \in D} c+u) = c\downarrow d1 \cdot c\downarrow d2 \cdot (c\downarrow d1 \cdot c\downarrow d2 + c\downarrow d2 \cdot c\downarrow d1)$

(v) Let $D = D1 \cup D2$, $D1 \cap D2 = \emptyset$, and

$$H = \left[ \sum_{d \in D1} c\downarrow d \cdot c\downarrow d + \sum_{d \in D2} c\downarrow d \cdot c\downarrow d \right] \cdot H$$

Then $H$ separates the $D1$ messages from the $D2$ messages.

Figure 3.

(vi) Let $d1 \neq d2$. Then:

$$\mu^\emptyset_c(\text{ctd}1 \cdot c+d2) = c\downarrow d1 \cdot \delta$$

$$\mu^\emptyset_c(\text{ctd}1 \parallel c+d2) = c\downarrow d1 \cdot \delta$$

$$\mu^\emptyset_c(c+d2 \parallel c\downarrow d1) = \delta.$$ 

3.7 Remarks Notice that there is no guarantee that after a send action $\text{ctd}$ the corresponding receive action $c+d$ will ever be performed. Thus the send action enables the receive action but does not force its execution. This holds for both mechanisms.

In the tele-communications area the design language SDL, used by CCITT, is quite popular. SDL mainly consists of a format for graphical notations for concurrent system descriptions with a send and receive mechanism. SDL leaves open the nature of the transmission protocol that supports the send and receive instructions. In SDL, example 3.6.2 can be depicted as follows:
Here it is assumed that in each cycle $d$ receives a value at $\text{rp}(d)$ and $\text{cl+d}$ respectively. The $\mu$-encapsulation of the protocol leads to the following SDL description:

$$
\text{rp}(d) \rightarrow \text{cl+d} \rightarrow \text{cl+d} \rightarrow \text{wq}(d) \rightarrow \text{c2+ack} \rightarrow \text{c2+ack}
$$

### Figure 4

### Figure 5

#### 3.8 Remark on synchronous communication.

A syntax for synchronous communication along a channel $c$, inspired by CSP and CCS, would be:

$c!d$  
$send d$

$c?d$  
$receive d$

$c#d$  
$communicate d$

In ACP [7] one introduces a communication function $|$ on actions. In this particular example, $|$ would work as follows:

$c!d | c?d = c#d$

Notice that we do not use variables: for example, $c?x\cdot P$ is modelled by $\sum_{d \in D} c?d\cdot P[d/x]$. This differs from CCS where one would have

$c(d) | \overline{c}(d) = \tau$.

### 4. CAUSALITY

In the previous section, the action $c+d$ is the executed or actualised form of $c+d$ and likewise $c+d$ is $c+d$ after execution or actualisation. Moreover, in some
sense a casual effect is involved: \( c \vdash d \) causes \( c \vdash d \). These concepts will be made explicit in the present section.

4.1 **Actualisation.** On the alphabet \( A \) we postulate an operator \( \hat{\cdot} : A \to A \), such that \( \hat{a} = \hat{\circ} \) and \( \hat{a} = \hat{a} \) for all \( a \in A \). The action \( \hat{a} \) is called the actualisation of \( a \).

Writing \( B = A - \hat{\cdot} \), where \( \hat{\cdot} = \{ \hat{a} \mid a \in A \} \), \( A \) is partitioned as follows:

\[
A = B \cup \hat{\cdot}.
\]

4.2 **Causal relations.** On the set \( B \) of not yet completed actions we have a binary relation \( R \) encoding the casual relations between such actions. Instead of \( (a,b) \in R \), we write:

\[
a \vdash b,
\]

in words: "\( a \) causes \( b \)". Further notations are:

- \( \text{Dom}(R) \) for the domain of \( R \), i.e. \( \text{Dom}(R) = \{ b \mid \exists b' \ b \vdash b' \} \), and
- \( \text{Ran}(R) \) for the range of \( R \), i.e. \( \text{Ran}(R) = \{ b \mid \exists b' \ b' \vdash b \} \).

So \( \text{Dom}(R) \) contains the causes or stimuli and \( \text{Ran}(R) \) the effects or responses.

Note that an action can be both a cause and an effect. Finally, write \( R(b) = \{ b' \mid b \vdash b' \} \) for the set of effects of \( b \).

4.3 **Encapsulation Operator.** Let \( b \in B \). Performing \( b \) has two consequences: \( b \) is now changed into \( \hat{b} \), and all \( b' \in R(b) \), actions caused by \( b \) are now enabled. The operator which takes care of the execution of \( b \) (or, in another phrasing which changes the view from "internal" to "external") and which takes into account which actions are enabled, is the encapsulation operator \( \gamma^E \) where \( E \subseteq B \). The intuitive meaning of \( \gamma^E \) is: \( \gamma^E(x) \) is the process where initially all actions of \( E \) are enabled and all casual effects take place within \( x \), i.e. actions within \( x \) are neither enabled or disabled by actions outside \( x \) and conversely.

4.4 **Axioms and Semantics.** The axioms for the operations \( \gamma^E \) are given in Table 5 below. Semantically, as with the previous axiomatisations, the equations specify an enrichment of the initial algebra \( I(PA_0) \). And again it is possible to enrich the important model constructions for infinite processes to permit the solution of guarded systems of equations.

4.5 **Examples**

(i) Suppose \( a \vdash b, c \vdash b \) (see Figure 6(a)).

\[
\begin{align*}
\gamma^g(ab \vdash cd) &= \gamma^g(a(b \vdash cd)) + \gamma^g(c(d \vdash ab)) = \hat{a} \gamma^d(b \vdash cd) + \ldots = \\
\hat{a} \gamma^d(bcd+c(d \vdash b)) + \ldots &= \hat{a}(\delta + \hat{\gamma}^d(b))(db+bd) + \ldots = \\
\hat{a}(\delta + \hat{\gamma}^d(b))(db+bd) + \ldots &= (a \|| b) (b \| b).
\end{align*}
\]
\( \mathbf{PA}_R(\gamma, \cdot) \) over atoms \( A \) with causality relation \( R \)

\[
\begin{align*}
x + y &= y + x & \text{A1} \\
(x + y) + z &= x + (y + z) & \text{A2} \\
x + x &= x & \text{A3} \\
(x + y)z &= xz + yz & \text{A4} \\
(xy)z &= x(yz) & \text{A5} \\
x + \delta &= x & \text{A6} \\
\delta x &= \delta & \text{A7} \\
x|y &= x|y + y|x & \text{M1} \\
a|y &= ay & \text{M2} \\
a|x|y &= a(x|y) & \text{M3} \\
(x + y)_{|z} &= x_{|z} + y_{|z} & \text{M4}
\end{align*}
\]

\( \hat{\delta} = \delta \)

\( \hat{\delta} = \delta \)

\[
\begin{align*}
\gamma^E(a) &= \hat{\delta} \text{ if } a \in E \text{ or } a \notin \text{Ran}(R) & \text{G3} \\
\gamma^E(a) &= \delta \text{ if } a \notin E \text{ and } a \in \text{Ran}(R) & \text{G4} \\
\gamma^E(ax) &= \gamma^E(a) \cdot \gamma^E(E^{-a}) \cup R(a)(x) & \text{G5} \\
\gamma^E(x + y) &= \gamma^E(x) + \gamma^E(y) & \text{G6}
\end{align*}
\]

Table 5.

(ii) Suppose \( d|\sim a, b|\sim c \) (see Figure 6(b)). Then \( \gamma^\emptyset(ab|cd) = \delta \).

\[\text{Figure 6.}\]

Note that circular causal relations, such as in this example (ii), yield deadlock.

Here an action \( a \) must be considered to cause the actions accessible from \( a \) or 'later' than \( a \). (Indeed, we have \( a \cdot b = \gamma^\emptyset(a||b) \) for \( a|\sim b \))

(iii) Let \( X \) and \( Y \) be the two infinite processes recursively defined by \( X = abX \) and \( Y = cdY \); so \( X = (ab)^\omega \) and \( Y = (cd)^\omega \). Suppose \( a|\sim c \) and \( d|\sim b \). Then

\[\begin{align*}
\gamma^\emptyset(X|Y) &= \gamma^\emptyset(a(bX||Y) + c(dY||X)) = \hat{\delta} \gamma^\emptyset(c(bX||Y)) + \delta = \\
&= \hat{\delta} \gamma^\emptyset(c(bX||Y) + c(dY||bX)) = \hat{\delta} + \hat{\delta} \gamma^\emptyset(dY||bX) = \hat{\delta} + \hat{\delta}
\end{align*}\]
\[ \mathcal{A}(\Delta Y \{b\}) (Y || bX) + \delta = \mathcal{A}(\Delta Y \{b\}) (b(X || Y) + Y || bX) = \]

\[ \mathcal{A}(\Delta b) Y (X || Y) . \]

Hence \( Y (X || Y) = (\mathcal{A}(\Delta b))^\omega \).

4.6 **Remarks**

It should be noted that however often an action \( b \) has been enabled, after being performed it is again disabled. For instance if \( b \vdash c \), then

\[ Y^\omega (bbcc) = \Delta Y^\omega (c)(bcc) = \Delta Y^\omega (c) = \Delta b^2 = b \Delta b . \]

Thanks to the interpretation of causality as introducing an obligation (which has no multiplicity), the mail via an unordered channel mechanism differs from the present mechanism. For, in the setting of Section 3, we have

\[ \mu^\omega_c (ctd \cdot ctd \cdot c+ \cdot c+ d) = ctd \cdot ctd \cdot c+ \cdot c+ d . \]

It is, however, easy to specify the variant of the causality mechanism above such that the obligations form a multiset rather than a set: axioms GL-6 from Table 5 carry over to that case unaltered, with only the stipulation that \( E \) is a multiset.

It is also simple to generalise the above causality relation to the case where an effect \( b \) may have several causes \( a_1, \ldots, a_n \):

\[ a_1, \ldots, a_n \vdash \vdash b, \]

meaning that all the \( a_i \) (\( i=1, \ldots, n \)) have to be executed in order to enable \( b \).

Finally, let us remark that there is an interesting connection between the "spatial" notion of encapsulation (as represented by the operators \( \partial_H \) in ACP; \( \mu^H_c, \mu^M_c \) in the mail mechanisms of Section 3; and the present \( \gamma^E \) for causality) and the "temporal" notion of execution. In some sense, one could say:

**encapsulation = execution**

Indeed, an encapsulated process can be thought to be already executed since no further interactions with an environment are possible.

4.7 **Printer Example**

As a finale we will examine a somewhat involved example. This example of the control of a printer constitutes an abstract version of the highest level of a specification case study reported in [5]. Henk Ommink [25] (Philips Research) suggested we should use a stimulus-response or causality mechanism at the highest specification level. An important motivation for the present paper is to present a proper foundation for such a causality mechanism in process algebra. In fact, mail via order-preserving or non-order-preserving channels turn out to be modifications of this same idea (with the advantage of having better syntax).

Let us consider a configuration of three components:
The only command that CM can issue is to start the printer; the printer will stop by itself. If the printer runs out of paper, a message to this effect must be displayed where upon new paper will be provided, and printing proceeds. When printing has finished this is reported to CM.

The behaviour of the components is defined by equations and depicted in the diagrams in Figure 7(a), (b), (c). From now on, we adopt the following Convention. We will use the following typographical convention: instead of denoting actions as \( b \), \( \hat{b} \) we will write, respectively, \( b \) and \( \hat{b} \). So italicized actions are in \( B \) and are not yet completed, and completed actions are in \( \hat{B} \) are in usual print.

![Figure 7(a)](image1)

\[
\begin{align*}
CM &= CM_0 \\
CM_1 &= RP.CM_2 \\
CM_0 &= PC.CM_1 \\
CM_2 &= CM_0 \\
CM &= PC.RP.CM
\end{align*}
\]

![Figure 7(b)](image2)

\[
\begin{align*}
P &= STP.PAD.P_2 \\
P_2 &= STOP.P + POP.NP.P_2
\end{align*}
\]
In these diagrams, fat arrows represent actions; the other lines identify control points and have no direction.

The causal relations in $\mathbb{R}$ are listed below:

$$
\begin{align*}
PC & \not\rightarrow STOP & STOP & \not\rightarrow D_{\text{start}} \\
STOP & \not\rightarrow D_{\text{stop}} & D_{\text{stop}} & \not\rightarrow RF \\
POP & \not\rightarrow D_{\text{pop}} & D_{\text{pop}} & \not\rightarrow NP \\
NP & \not\rightarrow D_{\text{ok}}
\end{align*}
$$

The entire system $\mathcal{S}$ is now described externally by:

$$
\mathcal{S} = \gamma(\mathcal{C}_0 || P_0 || D_0).
$$

Further, let $\mathcal{S}^*$ be the subsystem that starts with the exceptional case of no paper:

$$
\mathcal{S}^* = \gamma(\mathcal{C}_1 || P_3 || D_2).
$$

It can be shown that $\mathcal{S}$ and $\mathcal{S}^*$ satisfy the following specification by means of recursion equations:

$$
\begin{align*}
\mathcal{S} &= PC \cdot \text{STOP} \cdot \{D_{\text{start}} \cdot \text{PAD} \cdot \{\text{STOP} \cdot D_{\text{stop}} \cdot \text{RP} \cdot \mathcal{S} + \text{POP} \cdot D_{\text{pop}} \cdot \mathcal{S}^*\} \\
&\quad + \text{PAD} \cdot \{D_{\text{start}} \cdot \{\text{STOP} \cdot D_{\text{stop}} \cdot \text{RP} \cdot \mathcal{S} + \text{POP} \cdot D_{\text{pop}} \cdot \mathcal{S}^*\} \\
&\quad + \text{STOP} \cdot D_{\text{stop}} \cdot \text{RP} \cdot \mathcal{S} + \text{POP} \cdot D_{\text{pop}} \cdot \mathcal{S}^*\} \\
\mathcal{S}^* &= NP \cdot \{\text{STOP} \cdot D_{\text{OK}} \cdot D_{\text{stop}} \cdot \text{RP} \cdot \mathcal{S} + \text{POP} \cdot D_{\text{OK}} \cdot D_{\text{pop}} \cdot \mathcal{S}^* \\
&\quad + D_{\text{OK}} \cdot \{\text{STOP} \cdot D_{\text{stop}} \cdot \text{RP} \cdot \mathcal{S} + \text{POP} \cdot D_{\text{pop}} \cdot \mathcal{S}^*\} \}
\end{align*}
$$
REFERENCES

Initial algebra semantics and continuous algebras,

[2] AUSTRY, D. & G. BOUDOL,
Algèbre de processus et synchronisation,

Denotational semantics of concurrency,

Processes and the denotational semantics of concurrency,
Information and Control, Vol. 54, No.1/2, p.70-120, 1982.

Een analyse van de case-study HuP,

A process algebra for the operational semantics of static data flow networks,

[7] BERGSTRA, J.A. & J.W. KLOP,
Process algebra for communication and mutual exclusion. Revised version,

[8] BERGSTRA, J.A., KLOP, J.W. & J.V. TUCKER,
Algebraic tools for system construction,

[9] BRANQUART, P., LOUIS, G. & P. WODON,
An analytical description of CHILL, the CCITT High Level Language,

[10] BROCK, J.D. & W.B. ACKERMAN,
Scenarios : A model of non-determinate computation,

[11] BROOKES, S.D.,
On the relationship of CCS and CSP,

[12] BROOKES, S.D. & W.C. ROUNDS,
Behavioural equivalence relations induced by programming logics,

[13] EBERGEN, J.,
On VLSI design,

[14] HENNESSY, M.,
A term model for synchronous processes,

[15] HOARE, C.A.R.,
Communicating Sequential Processes,
[16] HOARE, C.A.R.,
   A model for Communicating Sequential Processes
   in: "On the construction of programs" (eds. R.M. McKeag and

[17] HOARE, C., BROOKES, S. & W. ROSCOE,
   A theory of communicating sequential processes,
   Programming Research Group, Oxford University (1981). To appear
   in JACM.

[18] KAHN, G.,
   The semantics of a simple language for parallel programming,

[19] KOYMANS, R., VYTOPIL, J. & W.P. DE ROEVER,
   Real-time programming and asynchronous message passing,
   in: Proc. of the Second Annual ACM Symposium on Principles of
   Distributed Computing, Montreal, 1983.

[20] KUIPER, R. & W.P. DE ROEVER,
   Fairness assumptions for CSP in a temporal logic framework,
   TC2 Working Conference on the Formal Description of Programming

[21] LAMPORT, L.,
   'Sometime' is sometimes 'NOT never',
   Tutorial on the temporal logics of programs, SRI International

[22] MILNE, G.J.,
   CIRCAL : A calculus for circuit description,

[23] MILNER, R.,
   A Calculus of Communicating Systems,
   Springer LNCS 92, 1980.

[24] MILNER, R.,
   Calculi for synchrony and asynchrony,


[26] PNUELI, A.,
   The temporal logic of programs,

[27] REM, M.,
   Partially ordered computations, with applications to VLSI design,
   Proc. 4th Advanced Course on Foundations of Computer Science, Part 2
   (eds. J.W. de Bakker and J. van Leeuwen), Mathematical Centre Tracts

[28] DE SIMONE, R.,
   On MEIJE and SCCS : infinite sum operators vs. non-guarded definitions,

[29] VAN DE SNEPSCHEUT, J.L.A.,
   Trace Theory and VLSI Design,

[30] WINSKEL, G.,
   Event structure semantics for CCS and related languages,

[31] WINSKEL, G.,
   Synchronization trees,
   Proc. 10th ICALP (ed. J. Diaz), Barcelona 1983, Springer LNCS 154,
   p.695-711.