PROCESS ALGEBRA WITH ASYNCHRONOUS COMMUNICATION MECHANISMS

J.A. BERGSTRA, J.W. KLOP
Centre for Mathematics and Computer Science, Kruislaan 413,
1098 SJ Amsterdam, The Netherlands.

J.V. TUCKER
Department of Computer Studies, University of Leeds
Leeds, LS2 9JT, England.

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 PA_δ (for *P*rocess *A*lgebra) 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 *asynchronous cooperation*, as each process may operate in connection with its own clock. The system PA_δ was first introduced in [7] together with an extension to an axiomatisation ACP (for *A*lgebra of *C*ommunicating *P*rocesses) of concurrent processes with a communication mechanism.

The laws for communication in ACP, like those in Milner's CCS, concern *synchronous communication*, requiring the synchronisation or simultaneous execution $a|b$ of so-called communication actions a,b. In research on laws for concurrency, while concurrent cooperation has been examined in its asychronous and synchronous cases, concurrent process communication in the asynchronous case has been neglected. In this paper we take up the idea of *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 PA_δ describing the free merge of processes; here δ 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/asychronous, and attempts a classification of formalisms such as CCS, CSP, MEIJE, 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_δ that analyses concurrent processes without communication. The system PA_δ 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 $\delta \epsilon A$. There are four process generating binary operations,

$+$	*alternative composition (sum)*
$\cdot$	*sequential composition (product)*
$\Vert$	*parallel composition (merge)*
$⌊⌊$	*left-merge*

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

1.1 Signature More formally, let Σ_{PA_δ} be the following signature :

S	:	P	*sorts*
F	:	$+: P \times P \rightarrow P$	*functions*
		$\cdot: P \times P \rightarrow P$	
		$\Vert: P \times P \rightarrow P$	
		$⌊⌊: P \times P \rightarrow P$	
C	:	a for all $a \in A$	*constants*

1.2 Axioms Let PA_δ be the set of equations over Σ_{PA_δ} in Table 1.

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x + \delta = x$	A6
$\delta x = \delta$	A7
$x \| y = x ⌊⌊ y + y ⌊⌊ x$	M1
$a ⌊⌊ x = ax$	M2
$(ax) ⌊⌊ y = a(x \| y)$	M3
$(x+y) ⌊⌊ z = x ⌊⌊ z + y ⌊⌊ z$	M4

Table 1

1.3 Semantics A Σ_{PA_δ}-structure P satisfying the axioms in PA_δ is a *process algebra with deadlock*; the class of all such algebras we denote $ALG(\Sigma_{PA_\delta}, PA_\delta)$.

In analogy with the theory of data type specifications, it is useful to consider the equational axiomatisation $(\Sigma_{PA_\delta}, PA_\delta)$ in two ways :

(i) as an *initial algebra specification* (in the sense of ADJ[1]) for the special structure A_ω of all finite processes with deadlock i.e. the initial algebra semantics of the specification is $I(\Sigma_{PA_\delta}, PA_\delta) \cong A_\omega$

(ii) as a general axiomatic specification of such concurrent process algebras with semantics $ALG(\Sigma_{PA_\delta}, PA_\delta)$.

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

Let $P \models PA_\delta$ be any process algebra. For $p \in P$ 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 α to be a *trace* of p:

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

$$p = p_1,\ p_i = a_i p_{i+1} + q_i \quad (i=1,\ldots,n-1)$$

$$p_n = a_n + q_n$$

(ii) If $\alpha = a_1 * a_2 * \ldots$ then we call α a *trace* of p if there are p_i, q_i such that

$$p = p_1,\ p_i = a_i p_{i+1} + q_i \quad (i\geq 1).$$

If $p \in P$ has an infinite trace, it is an *infinite* process; otherwise it is *finite*. The initial algebra $I(\Sigma_{PA_\delta}, PA_\delta)$ 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. COOPERATION AND COMMUNICATION

2.1 A Classification of Concurrency 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:

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 encorporated in SCCS [14,24], ASP [7], MEIJE [2,28].

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 encorporated in CSP [15-17], CCS [23], ACP [7], with restrictions determined by possible mutual interactions between processes, and in the system PA_δ, 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 :

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 *handshaking* and is encorporated in CSP, CCS, ACP, and Ada.

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 encorpoated 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 The combinations SS and AS have been extensively studied in process theory; we refer to Austry & Boudol [2] and De Simone [28] for a comparison between MEIJE and SCCS, to Milner [23,24] and Hennessy [14] for CCS and SCCS, to Bergstra & Klop [7] for ACP and ASP, to De Bakker & Zucker [3,4] for uniform processes, and to Brookes [11,12], Winskel [30] for discussions about and comparisons between CSP and CCS. For CSP see Hoare [15,16] and Hoare, Brookes & Roscoe [17].

It might be puzzling why ASCCS, which gives according to Milner [24] a framework for "asychronous processes", is classified under SS. The reason is that it is a subcalculus of SCCS, 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 Koymans, 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_δ; 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 comparision with the construction of ACP as an extension of PA_δ.

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 $d \in D$ there are actions

$c \uparrow d$	*send data* d *via channel* c *considered as a potential action*
$c \boldsymbol{\uparrow} d$	*send data* d *via channel* c *considered as an actual action*
$c \downarrow d$	*receive data* d *via channel* c *considered as a potential action*
$c \boldsymbol{\downarrow} d$	*receive data* d *via channel* c *considered as an actual action*

The distinction between $c \uparrow d$ and $c \boldsymbol{\uparrow} d$ may be slightly unusual $c \uparrow d$ indicates an *internal, intended, potential,* or *future* action while $c \boldsymbol{\uparrow} d$ denotes an *external, realised, actual,* or *past* action; and similarly for $c \downarrow d$ and $c \boldsymbol{\downarrow} d$.

This distinction is implicit in the synchronous communication operator $|$ of ACP where a communication takes the form $a|b = c$ 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 $c{\uparrow}D = \{c{\uparrow}d \mid d \in D\}$ and likewise for $c\boldsymbol{\uparrow}D$, etc.

Now we define the alphabet to be

$$A = B \cup \{\delta\} \cup (c{\uparrow}D) \cup (c\boldsymbol{\uparrow}D) \cup (c{\downarrow}D) \cup (c\boldsymbol{\downarrow}D).$$

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',... is entirely unproblematic; in that case, B may also contain actions $c'{\uparrow}d$ 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* σ of data $d \in D$. The empty sequence is denoted by ε. Concatenation of sequences α,τ is denoted as $\alpha * \tau$; especially if $\alpha = <d_1,\ldots,d_n>$ $(n\geq 0)$ then $d*\sigma = <d,d_1,\ldots,d_n>$ and $\sigma * d = <d_1,\ldots,d_n,d>$. Further, if $n\geq 1$, last $(\sigma) = d_n$.

Now for each $\sigma \in D^*$ there is an *encapsulation operator* $\mu_c^\sigma : P \to P$ where 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 $\mu_c^\sigma(x)$ denotes *the process obtained by requiring that the channel* c *initially contains a data sequence* σ *and that no communications with* c *are performed outside* x. Thus, x and $\mu_c^\sigma(x)$ correspond to internal and external views of a system's behaviour, in some sense.

There are other relevant intuitions about encapsulation. The process $\mu_c^\sigma(x)$ can be viewed as the result of the *partial execution* of x with respect to c with initial contents σ. By execution we mean the transformation of internal or potential actions like $c\uparrow D$ into an external or actual actions like $c\boldsymbol{\Uparrow}d$, and their effect on processes (cf Remark 4.6). Encapsulation is formally defined by axioms MO1-9 below.

3.2 *Bag-like Channel.* For the bag-like channel the situation is very much the same except that a data sequence σ 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

$$\mu_c^M : P\to P$$

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 :

$PA_\delta(\mu_c^\sigma,B,D)$ in Table 3

$PA_\delta(\mu_c^M,B,D)$ in Talbe 4

for mail via a queue-like channel and mail via a bag-like channel, respectively. Here a varies over the alphabet $A = B \cup \{\delta\} \cup c\uparrow D \cup c\boldsymbol{\Uparrow}D \cup c\downarrow D \cup c\boldsymbol{\Downarrow}D$, and e varies over $E = B \cup \{\delta\} \cup c\uparrow D \cup c\downarrow D$.

$+$	*alternative composition (sum)*
$\cdot$	*sequential composition (product)*
$\parallel$	*parallel composition (merge)*
$\lfloor\!\lfloor$	*left-merge*
δ	*dead-lock or failure*
b	*atomic action* ε B*, independent from* c
$c\uparrow d$	*send* d *via channel* c*; internal view*
$c\boldsymbol{\Uparrow} d$	*send* d *via channel* c*; external view*
$c\downarrow d$	*receive* d *via* c*; internal view*
$c\boldsymbol{\Downarrow} d$	*receive* d *via* c*; external view*
μ_c^σ	*encapsulation w.r.t. queue-like channel* c
μ_c^M	*encapsulation w.r.t. bag-like channel* c

Table 2.

3.5 Semantics. The axiom systems $PA_\delta(\mu_c^\alpha,B,D)$ and $PA_\delta(\mu_c^M,B,D)$ determine initial algebras

$$A_\omega(+,\cdot,\parallel,\ \lfloor\!\lfloor,\ \delta,\ \mu_c^\sigma,B,D)$$

$$A_\omega(+,\cdot,\parallel,\ \lfloor\!\lfloor,\ \delta,\ \mu_c^M,B,D)$$

respectively. These are just enrichments of the initial algebra I(PA)denoted A_ω or

$$A_\omega(+,\cdot,\ \parallel,\ \lfloor\!\lfloor,\ \delta)$$

of PA_δ. 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

$$A^\infty(+,\cdot,\parallel,\ \lfloor\!\lfloor,\ \delta,\ \mu_c^\alpha,B,D)$$
$$A^\infty(+,\cdot,\parallel,\ \lfloor\!\lfloor,\ \delta,\ \mu_c^M,B,D)$$

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 :

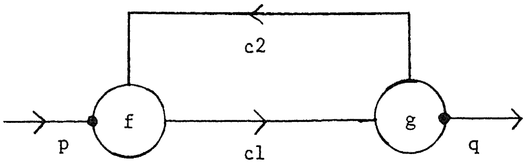


Figure 1.

$PA_\delta(\mu_c^\sigma, B, D)$

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x + \delta = x$	A6
$\delta x = \delta$	A7
$x \| y = x \underline{\|} y + y \underline{\|} x$	M1
$a \underline{\|} x = ax$	M2
$ax \underline{\|} y = a(x \| y)$	M3
$(x + y) \underline{\|} z = x \underline{\|} z + y \underline{\|} z$	M4
$\mu_c^\sigma(e) = e$	MO1
$\mu_c^\sigma(ex) = e.\mu_c^\sigma(x)$	MO2
$\mu_c^\sigma(c\uparrow d) = c\Uparrow d$	MO3
$\mu_c^\sigma(c\uparrow d.x) = c\Uparrow d.\mu_c^{d*\sigma}(x)$	MO4
$\mu_c^{\sigma*d}(c\downarrow d) = c\Downarrow d$	MO5
$\mu_c^{\sigma*d}(c\downarrow d.x) = c\Downarrow d.\mu_c^\sigma(x)$	MO6
$\mu_c^\sigma(c\downarrow d) = \delta$ if $d \neq last(\sigma)$ or $\sigma = \varepsilon$	MO7
$\mu_c^\sigma(c\downarrow d.x) = \delta$ if $d \neq last(\sigma)$ or $\sigma = \varepsilon$	MO8
$\mu_c^\sigma(x + y) = \mu_c^\sigma(x) + \mu_c^\sigma(y)$	MO9

Table 3 $(a \in A,\ e \in E,\ \sigma \in D^*)$

$PA_{\delta}(\mu_c^M, B, D)$

$x + y = y + x$	A1
$(x+y) + z = x + (y+z)$	A2
$x + x = x$	A3
$(x+y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x + \delta = x$	A6
$\delta x = \delta$	A7
$x \| y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x$	M1
$a \mathbin{\lfloor\!\lfloor} x = ax$	M2
$ax \mathbin{\lfloor\!\lfloor} y = a(x \| y)$	M3
$(x+y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$	M4
$\mu_c^M(e) = e$	MN01
$\mu_c^M(ex) = e.\mu_c^M(x)$	MN02
$\mu_c^M(c\uparrow d) = c\Uparrow d$	MN03
$\mu_c^M(c\uparrow d \;.\; x) = c\Uparrow d.\; \mu_c^{M\cup\{d\}}(x)$	MN04
$\mu_c^{M\cup\{d\}}(c\downarrow d) = c\Downarrow d$	MN05
$\mu_c^{M\cup\{d\}}(c\downarrow d.x) = c\Downarrow d \;.\; \mu_c^M(x)$	MN06
$\mu_c^M(c\downarrow d) = \delta$ if $d \notin M$	MN07
$\mu_c^M(c\downarrow d.x) = \delta$ if $d \notin M$	MN08
$\mu_c^M(x+y) = \mu_c^M(x) + \mu_c^M(y)$	MN09

<u>Table 4</u>. $(a \in A,\ e \in E,\ M$ a multiset over $D)$

with actions

$rp(d)$	*processor* f *reads value* d *at port* p
$wq(d)$	*processor* g *writes* d *at port* q

There are two order-preserving channels c1 and c2. Internally, the node f satisfies

$$f = \sum_{d\in D} (rp(d) + c2{\downarrow}d) \cdot c1{\uparrow}d \cdot f.$$

So, node f merges the inputs from p and c2 and emits these through c1. The node g is defined by

$$g = \sum_{d\in D} c1{\downarrow}d \cdot (i \cdot c2{\uparrow}\alpha(d) + i \cdot wq(d)) \cdot g$$

The effect of the internal step i is to make the choice nondeterministic, and $\alpha : D \to D$ is a transformation of the data; thus g obtains d from c1 and then chooses whether to 'recycle' $\alpha(d)$ via c2 or to output d via port q.

The network N is now externally described by

$$N = \mu^{\varepsilon}_{c1}\mu^{\varepsilon}_{c2}(f \| g).$$

Note that the actions $c1{\downarrow}d$, $c1{\uparrow}d$, $c1\boldsymbol{\downarrow}d$ and $c1\boldsymbol{\uparrow}d$ are unrelated to c2 and thereby work as b's in the definition for μ^{α}_{c2}. Conversely, the send and receive actions for c2 are unrelated to c1.

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

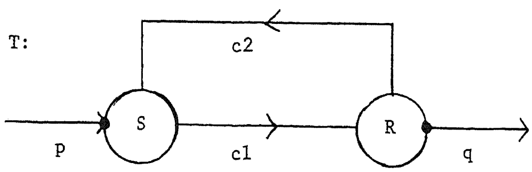


Figure 2

$$S = \sum_{d\in D} rp(d) \cdot c1{\uparrow}d \cdot c2{\downarrow}ack \cdot S$$

$$R = \sum_{d\in D} c1{\downarrow}d \cdot wq(d) \cdot c2{\uparrow}ack \cdot R$$

$$T = \mu^{\varepsilon}_{c1}\, \mu^{\varepsilon}_{c2}\, (S \| R).$$

In fact the protocol T satisfies the following recursion equation (as one easily computes from the axioms in $PA_{\delta}(\mu^{\alpha}_{c}, B, D))$:

$$T = \sum_{d\in D} rp(d) \cdot c1\boldsymbol{\uparrow}d \cdot c1\boldsymbol{\downarrow}d \cdot wq(d) \cdot c2\boldsymbol{\uparrow}ack \cdot c2\boldsymbol{\downarrow}ack \cdot T.$$

This recursion equation constitutes an external *specification* of T.

3.6.3 *Example for a bag-like channel* :

(i) $\mu_c^{\emptyset}(c\uparrow d \cdot c\downarrow d) = c\boldsymbol{\uparrow} d \cdot c\boldsymbol{\downarrow} d$

(ii) $\mu_c^{\emptyset}(c\uparrow d \cdot \sum_{u\in D} c\downarrow u) = c\boldsymbol{\uparrow} d \cdot c\boldsymbol{\downarrow} d$

(iii) $\mu_c^{\emptyset}(c\uparrow d \parallel c\downarrow d) = c\boldsymbol{\uparrow} d \cdot c\boldsymbol{\downarrow} d$

(iv) $\mu_c^{\emptyset}(c\uparrow d1 \cdot c\uparrow d2 \cdot \sum_{u\in D} c\downarrow u \cdot \sum_{u\in D} c\downarrow u) =$

$$c\boldsymbol{\uparrow} d1 \cdot c\boldsymbol{\uparrow} d2 \cdot (c\boldsymbol{\downarrow} d1 \cdot c\boldsymbol{\downarrow} d2 + c\boldsymbol{\downarrow} d2 \cdot c\boldsymbol{\downarrow} d1)$$

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

$$H = \left[\sum_{d\in D1} c1\downarrow d \cdot c2\uparrow d + \sum_{d\,\in D2} c1\downarrow d \cdot c3\uparrow d\right] \cdot H$$

Then H separates the D1 messages from the D2 messages.

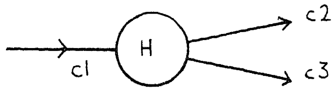


Figure 3.

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

$\mu_c^{\emptyset}$ $(c\uparrow \mathbf{d1} \cdot c\downarrow d2) = c\uparrow d1 \cdot \delta$

$\mu_c^{\emptyset}$ $(c\uparrow d1 \parallel c\downarrow d2) = c\uparrow d1 \cdot \delta$

$\mu_c^{\emptyset}$ $(c\downarrow d2 \;\lfloor\!\lfloor\; c\uparrow d1) = \delta$.

3.7 Remarks Notice that there is no guarantee that after a send action $c\uparrow d$ the corresponding receive action $c\downarrow d$ 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 :

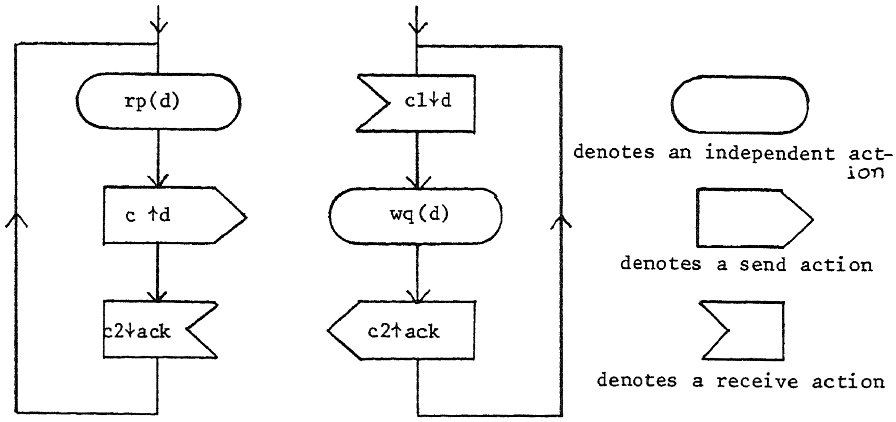


Figure 4

Here it is assumed that in each cycle d receives a value at rp(d) and c1↓d respectively. The μ-encapsulation of the protocol leads to the following SDL description :

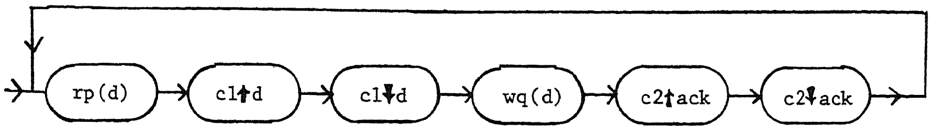


Figure 5

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 :

$$c!d \mid c?d = c\#d$$

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

$$c(d) \mid \bar{c}(d) = \tau.$$

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 : $c{\uparrow}d$ causes $c{\downarrow}d$. These concepts will be made explicit in the present section.

4.1 Actualisation. On the alphabet A we postulate an operator $\hat{}:A \to A$, such that $\hat{\delta} = \delta$ and $\hat{\hat{a}} = \hat{a}$ for all $a \in A$. The action $\hat{a}$ is called the *actualisation* of a. Writing $B = A-\hat{A}$, where $\hat{A} = \{\hat{a} \mid a \in A\}$, A is partitioned as follows :

$$A = B \cup \hat{A} .$$

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 :

Dom(R) for the *domain* of R, i.e. $\mathrm{Dom}(R) = \{b \mid \exists b' \quad b \Vdash b'\}$, and
Ran(R) for the *range* of R, i.e. $\mathrm{Ran}(R) = \{b \mid \exists b' \quad b' \Vdash b\}$.

So Dom(R) contains the *causes* or *stimuli* and 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* γ^E where $E \subseteq B$. The intuitive meaning of γ^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 γ^E are given in Table 5 below. Semantically, as with the previous axiomatisations, the equations specify an enrichment of the initial algebra $I(PA_\delta)$. 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 $a \Vdash d$, $c \Vdash b$ (see Figure 6(a)).

$$\gamma^{\emptyset}(ab \| cd) = \gamma^{\emptyset}(a(b \| cd)) + \gamma^{\emptyset}(c(d \| ab)) = \hat{a}\gamma^{\{d\}}(b \| cd) + \ldots =$$

$$\hat{a}\gamma^{\{d\}}(bcd+c(d \| b)) + \ldots = \hat{a}(\delta + \hat{c}\gamma^{\{d,b\}}(db+bd) + \ldots =$$

$$\hat{a}\hat{c}(\hat{d}\hat{b} + \hat{b}\hat{d}) + \hat{c}\hat{a}(\hat{b}\hat{d} + \hat{d}\hat{b}) = (\hat{a} \| \hat{c})(\hat{b} \| \hat{d}) .$$

$PA_\delta(\gamma,\hat{\ })$ over atoms A with causality relation R

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x + \delta = x$	A6
$\delta x = \delta$	A7
$x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$	M1
$a \lfloor\!\lfloor y = ay$	M2
$ax \lfloor\!\lfloor y = a(x \| y)$	M3
$(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	M4
$\hat{\delta} = \delta$	G1
$\hat{\hat{a}} = \hat{a}$	G2
$\gamma^E(a) = \hat{a}$ if $a \in E$ or $a \notin Ran(R)$	G3
$\gamma^E(a) = \delta$ if $a \notin E$ and $a \in Ran(R)$	G4
$\gamma^E(ax) = \gamma^E(a) \cdot \gamma^{(E-\{a\}) \cup R(a)}(x)$	G5
$\gamma^E(x + y) = \gamma^E(x) + \gamma^E(y)$	G6

Table 5.

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

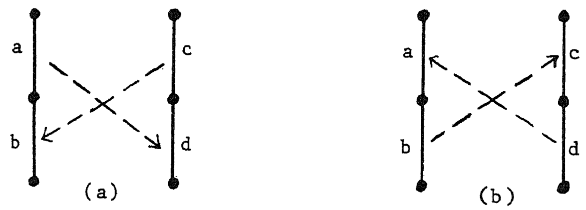


Figure 6.

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 $a \| \!- b$)

(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 $a \| \!- c$ and $d \| \!- b$. Then

$$\gamma^\emptyset(X \| Y) = \gamma^\emptyset(a(bX \| Y) + c(dY \| X)) = \hat{a}\gamma^{\{c\}}(bX \| Y) + \delta =$$

$$\hat{a}\gamma^{\{c\}}(b(X \| Y) + c(dY \| bX)) = \hat{a}(\delta + \hat{c}\gamma^\emptyset(dY \| bX)) =$$

$$\hat{a}\hat{c}(\partial\gamma^{\{b\}}(Y \| bX) + \delta) = \hat{a}\hat{c}\partial\gamma^{\{b\}}(b(X\| Y) + Y \lfloor\!\lfloor bX) =$$

$$\hat{a}\hat{c}\partial\hat{b}\gamma^{\emptyset}(X\| Y).$$

Hence $\gamma^{\emptyset}(X\| Y) = (\hat{a}\hat{c}\partial\hat{b})^{\omega}$.

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

$$\gamma^{\emptyset}(bbcc) = \hat{b}\gamma^{\{c\}}(bcc) = \hat{b}\hat{b}\gamma^{\{c\}}(cc) = \hat{b}\hat{b}\hat{c}\gamma^{\emptyset}(c) = \hat{b}\hat{b}\hat{c}\delta.$$

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

$$\mu_c^{\emptyset}(c{\uparrow}d \cdot c{\uparrow}d \cdot c{\downarrow}d \cdot c{\downarrow}d) = c{\uparrow}d \cdot c{\uparrow}d \cdot c{\downarrow}d \cdot c{\downarrow}d\ .$$

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 G1-6 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 $a_1,\ldots,a_n$:

$$a_1,\ldots,a_n \Vdash b,$$

meaning that all the a_i (i=1,...,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 ∂_H in ACP; μ_c^{σ}, μ_c^{M} in the mail mechanisms of Section 3; and the present γ^E for causality) and the "temporal" notion of execution. In some sense, one could say :

<u>encapsulation = execution</u>

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 Obbink [25] (Philips Research) suggested we should use a stimulus-response or causality mechamism 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 :

CM	*command module*
P	*printer*
D	*display*

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 b. So *italicized actions* are in B and are not yet completed, and *completed actions* are in $\hat{B}$ are in $\hat{B}$ are in usual print.

CM

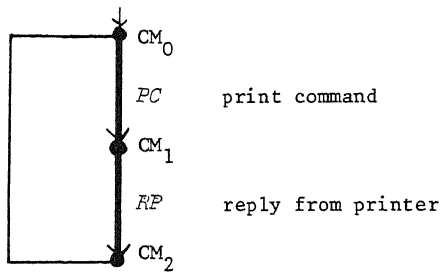


Figure 7(a).

$CM = CM_0$ $\quad$ $CM_0 = PC.CM_1$

$CM_1 = RP.CM_2$ $\quad$ $CM_2 = CM_0$

$$CM = PC.RP.CM$$

P

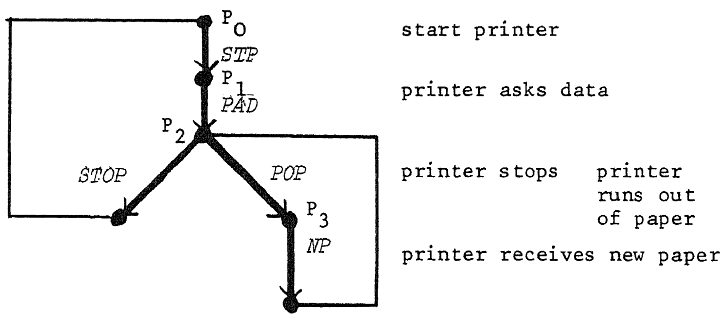
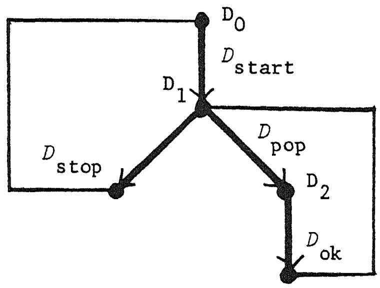


Figure 7(b).

$$P = \mathit{STP}.\mathit{PAD}.P_2$$

$$P_2 = \mathit{STOP}.P + \mathit{POP}.\mathit{NP}.P_2$$

D



display "start"

display "stop" *display "pop"*

display "OK"

Figure 7(c).

$$D = D_{start} . D_1$$

$$D_1 = D_{stop} . D + D_{pop}\, D_{ok} . D_1$$

In these diagrams, fat arrows represent actions; the other lines identify control points and have no direction.

The casual relations in R are listed below :

PC	$\Vdash$	STP	STP	$\Vdash$	D_{start}
$STOP$	$\Vdash$	D_{stop}	D_{stop}	$\Vdash$	RP
POP	$\Vdash$	D_{pop}	D_{pop}	$\Vdash$	NP
NP	$\Vdash$	D_{ok}			

The entire system S is now described externally by :

$$S = \gamma^{\emptyset}(CM_0 \| P_0 \| D_0) \ .$$

Further, let S^* be the subsystem that starts with the exceptional case of no paper :

$$S^* = \gamma^{\{NP\}}(CM_1 \| P_3 \| D_2) \ .$$

It can be shown that S and S^* satisfy the following specification by means of recursion equations :

$$\begin{aligned} S = {} & PC \cdot STP \cdot [D_{start} \cdot PAD \cdot \{STOP \cdot D_{stop} \cdot RP \cdot S + POP \cdot D_{pop} \cdot S^*\} \\ & + PAD \cdot \{D_{start} \cdot (STOP \cdot D_{stop} \cdot RP \cdot S + POP \cdot D_{pop} \cdot S^*) + \\ & + STOP \cdot D_{stop} \cdot RP \cdot S + POP \cdot D_{pop} \cdot S^*\}] \end{aligned}$$

$$\begin{aligned} S^* = {} & NP \cdot [STOP \cdot D_{OK} \cdot D_{stop} \cdot RP \cdot S + POP \cdot D_{OK} \cdot D_{pop} \cdot S^* \\ & + D_{OK} \cdot (STOP \cdot D_{stop} \cdot RP \cdot S + POP \cdot D_{pop} \cdot S^*)] \end{aligned}$$

REFERENCES

[1] ADJ (GOGUEN, J.A., THATCHER, J.W., WAGNER, E.G. & J.B. WRIGHT),
Initial algebra semantics and continuous algebras,
JACM Vol. 24, Nr. 1, p.68-95 (1975).

[2] AUSTRY, D. & G. BOUDOL,
Algèbre de processus et synchronisation,
Theoretical Computer Science 30 (1984), p.91-131.

[3] DE BAKKER, J.W. & J.I. ZUCKER,
Denotational semantics of concurrency,
Proc. 14th ACM Symp. on Theory of Computing, p.153-158, 1982.

[4] DE BAKKER, J.A. & J.I. ZUCKER,
Processes and the denotational semantics of concurrency,
Information and Control, Vol. 54, No.1/2, p.70-120, 1982.

[5] BERGSTRA, J.A., HEERING, J., KLINT, P. & J.W. KLOP,
Een analyse van de case-study HμP,
mimeographed notes, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[6] BERGSTRA, J.A. & J.W. KLOP,
A process algebra for the operational semantics of static data flow networks,
Report IW 222/83, Mathematisch Centrum, Amsterdam 1983.

[7] BERGSTRA, J.A. & J.W. KLOP,
Process algebra for communication and mutual exclusion. Revised version,
Report CS-R8409, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[8] BERGSTRA, J.A., KLOP, J.W. & J.V. TUCKER,
Algebraic tools for system construction,
in : Logics of Programs, Proceedings 1983 (eds. E. Clarke and D. Kozen), Springer LNCS 164, 1984.

[9] BRANQUART, P., LOUIS, G. & P. WODON,
An analytical description of CHILL, the CCITT High Level Language,
Springer LNCS 128, 1982.

[10] BROCK, J.D. & W.B. ACKERMAN,
Scenarios : A model of non-determinate computation,
in : Proc. Formalization of Programming Concepts (eds. J. Diaz and I. Ramos), p.252-259, Springer LNCS 107, 1981.

[11] BROOKES, S.D.,
On the relationship of CCS and CSP,
Proc. 10th ICALP, Barcelona 1983 (ed. J. Diaz), Springer LNCS 154, p.83-96, 1983.

[12] BROOKES, S.D. & W.C. ROUNDS,
Behavioural equivalence relations induced by programming logics,
Proc. 10th ICALP, Barcelona 1983, Springer LNCS 154 (ed. J. Diaz), p.97-108, 1983.

[13] EBERGEN, J.,
On VLSI design,
NGI-SION Proceedings 1984, p.144-150, Nederlands Genootschap voor Informatica, Amsterdam 1984.

[14] HENNESSY, M.,
A term model for synchronous processes,
Information and Control 51, p.58-75 (1981).

[15] HOARE, C.A.R.,
Communicating Sequential Processes,
C. ACM <u>21</u> (1978), p.666-677.

[16] HOARE, C.A.R.,
A model for Communicating Sequential Processes
in : "On the construction of programs" (eds. R.M. McKeag and A.M. McNaughton), Cambridge University Press (1980), p.229-243.

[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,
in : PROC. IFIP 74, North-Holland, Amsterdam 1974.

[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 Concepts, Proc., Garmisch 1982.

[21] LAMPORT, L.
'Sometime' is sometimes 'NOT never',
Tutorial on the temporal logics of programs, SRI International CSL-86, 1979.

[22] MILNE, G.J.,
CIRCAL : A calculus for circuit description,
Integration, Vol.1, No. 2 & 3, 1983, p.121-160.

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

[24] MILNER, R.,
Calculi for synchrony and asynchrony,
Theor. Comp. Sci. 25 (1983), p.267-310.

[25] OBBINK, H., personal communication, April 1984.

[26] PNUELI, A.,
The temporal logic of programs,
in : Proc. 19th Ann. Symp. on Foundations of Computer Science, IEEE, p.46-57, 1977.

[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 159, p.1-44, Mathematisch Centrum, Amsterdam 1983.

[28] DE SIMONE, R.,
On MEIJE and SCCS : infinite sum operators vs. non-guarded definitions,
Theoretical Computer Science 30 (1984), p.133-138.

[29] VAN DE SNEPSCHEUT, J.L.A.,
Trace Theory and VLSI Design,
Ph.D. Thesis, Eindhoven University of Technology, 1983.

[31] WINSKEL, G.,
Event structure semantics for CCS and related languages,
Proc. ICALP 82, Springer LNCS 140, p.561-576, 1982.

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