stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 222/83 FEBRUARI

J.A. BERGSTRA & J.W. KLOP

A PROCESS ALGEBRA FOR THE OPERATIONAL SEMANTICS OF STATIC DATA FLOW NETWORKS

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, Kruislaan 413, Amsterdam, The Netherlands.

The Mathematical Centre, founded 11th February 1946, is a non-profit institution for the promotion of pure and applied mathematics and computer science. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

1980 Mathematics subject classification: 68B10, 68C01, 68D25, 68F20

1982 CR. Categories: D.1.3, F.1.1, F.1.2, F.3.2

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

by

J.A. Bergstra & J.W. Klop

ABSTRACT

An algebra of communicating processes is used to provide an operational semantics of data flow networks with a static number of nodes and channels. A data flow network is modeled as a system of processes communicating by hand shaking. Nodes and channels are treated on equal footing: in both cases their semantics is derived as the solution of a fixed point equation in the process algebra.

KEY WORDS & PHRASES: *nondeterministic processes, process algebra, merge, concurrency, communication, synchronisation, hand shaking, operational semantics, data flow networks*

---

*) This report will be submitted for publication elsewhere.

0. INTRODUCTION

Process algebra is a useful and mathematically elegant tool for the description of processes. We feel that the proper semantics of computational processes (like data flow) should be given in terms of semantic objects which are richer in structure than sets of execution traces. Process algebra provides these objects together with an attractive mathematical structure. Therefore we pay attention to modeling data flow networks in terms of process algebra. To each data flow network a process can be assigned which in our view can be taken as its operational semantics.

Of course we are aware of the defects of this model: mainly, that it does restrict real concurrency. However, we are not aware of a more general model that still is technically attractive. (In the discussion at the end of the paper this issue is also considered.)

The contents of this document are as follows:

1. ALGEBRA OF COMMUNICATING PROCESSES (ACP)

Let A be a finite collection of atomic actions (events), $\delta \in A$ a distinguished symbol denoting deadlock. A^∞ denotes the set of (finite and infinite) processes over A. A^∞ is structured by the operations

$+$	*nondeterministic choice*
$\bullet$	*sequential composition*
$\|\|$	*merge (cooperation)*
$\lfloor\!\lfloor$	*left merge*
$\|$	*communication*
δ	*deadlock*

On atomic actions a function $.|. : A \times A \to A$ is given. $a|b$ is the action that results from simultaneous execution of a and b. We say that a and b communicate if $a|b \neq \delta$.

The following equational laws A1-A7, C1-C3, CM1-CM9 hold for A^∞. Here a,b,c range over A and x,y,z over A^∞.

$x + y = y + x$	A1
$x + (y + z) = (x + y) + z$	A2
$x + x = x$	A3
$(x + y).z = x.z + y.z$	A4
$(x.y).z = x.(y.z)$	A5
$x + \delta = x$	A6
$\delta.x = \delta$	A7
$a\|b = b\|a$	C1
$(a\|b)\|c = a\|(b\|c)$	C2
$\delta\|a = \delta$	C3

$x \| y = x \mathbin{\|\!\lfloor} y + y \mathbin{\|\!\lfloor} x + x \mid y$	CM1
$a \mathbin{\|\!\lfloor} x = a.x$	CM2
$(ax) \mathbin{\|\!\lfloor} y = a(x \| y)$	CM3
$(x + y) \mathbin{\|\!\lfloor} z = x \mathbin{\|\!\lfloor} z + y \mathbin{\|\!\lfloor} z$	CM4
$(ax) \mid b = (a \mid b).x$	CM5
$a \mid (bx) = (a \mid b).x$	CM6
$(ax) \mid (by) = (a \mid b).(x \| y)$	CM7
$(x + y) \mid z = x \mid z + y \mid z$	CM8
$x \mid (y + z) = x \mid y + x \mid z$	CM9

For each $H \subseteq A$ an operator Δ_H is introduced that views the $a \in H$ as deadlock:

$\Delta_H(a) = \begin{cases} a \text{ if } a \notin H \\ \delta \text{ if } a \in H \end{cases}$	Δ1
$\Delta_H(x + y) = \Delta_H(x) + \Delta_H(y)$	Δ2
$\Delta_H(x.y) = \Delta_H(x).\Delta_H(y)$	Δ3

Δ_H is a homomorphism on $(A^\infty, +, \bullet, \delta)$ (but not w.r.t. $\|$).

<u>EXAMPLE</u>. Let $c \mid c = c°$; all other communications are δ. Then $\Delta[(ab + ac) \| cd] =$

$\Delta[ab \mathbin{\|\!\lfloor} cd + ac \mathbin{\|\!\lfloor} cd + cd \mathbin{\|\!\lfloor} (ab+ac) + cd \mid ab + cd \mid ac] =$

$\Delta[a(b \| cd) + a(c \| cd) + c(d \| (ab+ac)) + (c \mid a)(d \| b) + (c \mid a)(d \| c)] =$

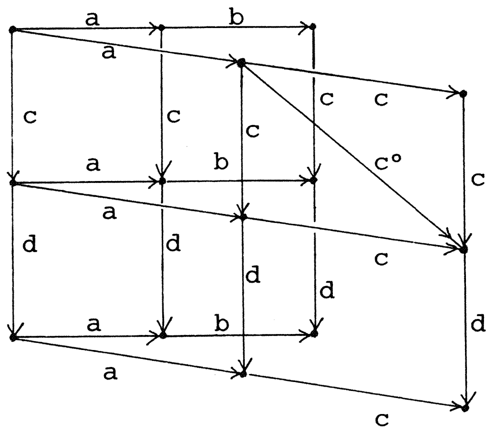
$\Delta[a(bcd + c(d \| b) + (b \mid c)d) + a(ccd + c(d \| c) + (c \mid c)d) + c(d \| (ab+ac)) +$

$+ \delta(d \| b) + \delta(d \| c)] =$

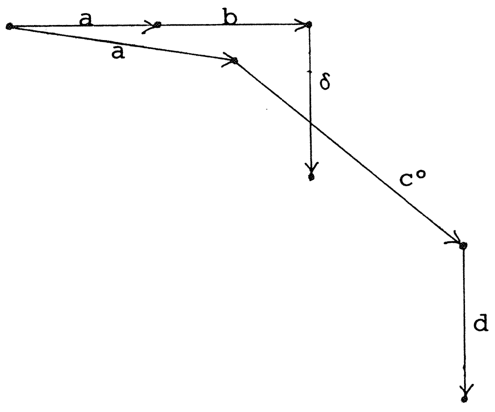
$\Delta[a(bcd + c(d \| b)) + a(ccd + c(d \| c) + c°d) + c(d \| (ab+ac))] =$

$ab\delta + ac°d$. (Here $\Delta = \Delta_{\{c\}}$.)

The following method using diagrams may be helpful: $(ab + ac) \| cd$ is the product graph



Diagrammatically, communications as c|c = c° are 'vector sums'.
After applying Δ the remains are:



that is, abδ +ac°d.

<u>Remarks on previous litterature</u>. In BERGSTRA & KLOP [7] A^∞ is defined; essentially one views ACP as an initial algebra specification of the finite processes, collected in A_ω, and then one enriches A_ω to A^∞ by some limit process. DE BAKKER & ZUCKER [4] use metric completion to obtain A^∞ ; however, they deal with infinite A.

In BERGSTRA & KLOP [7] technical information about ACP is provided, such as proofs of the commutativity and associativity of || and |.

To a large extent A^∞ can be considered a model of Milner's CCS (see [12]). The communication function .|. on atoms also appears in HENNESSY [10].

New in ACP is the extensive use made of $\mathbin{|\!\lfloor}$.

Adding the law x.(y + z) = x.y + x.z one obtains a model for trace theory as defined in REM [16]. In this setting there are also connections to NIVAT [13].

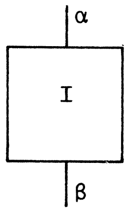
For further information we refer to the discussion at the end of this paper.

2. NODES AND CHANNELS

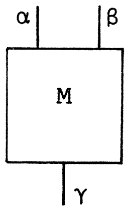
This section will provide examples of processes. The subsection 2.2 is essential for the rest of the paper.

2.1. Processes that provide the semantics for data processing nodes.

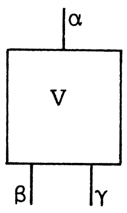
We start with a dozen of examples of some typical nodes in the data flow networks that will be described later on.



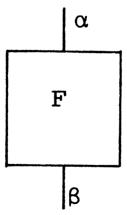
Identity



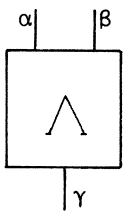
Merge



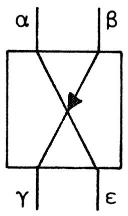
Nondeterministic choice



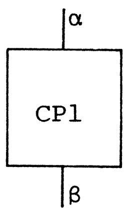
Function application



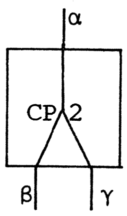
Pairing



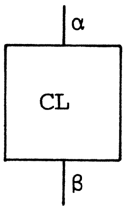
Switch



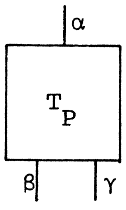
Copy 1



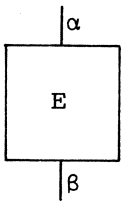
Copy 2



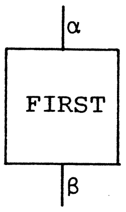
Clustering



Test for P



Even



First

We will provide processes that correspond to the intended semantics of these nodes. In all cases α,β,γ denote ports. A value passing at port α will be denoted by α^d ($d \in D$, a finite set of data). If there are just two ports α and β we can abbreviate α^d to d and β^d to $\underline{d}$. The processes are defined as the (unique) fixed points of (systems of) equations. (For an account of solving equations in process algebras see [6].)

Identity. $I = \sum_d d.\underline{d}.I$

Merge. $M = \sum_d (\alpha^d + \beta^d).\gamma^d.M$

Nondeterministic choice. $V = \sum_d \alpha^d.(\beta^d + \gamma^d).V$

Function application. $F = \sum_d d.\underline{f(d)}.F$

Pairing. $\bigwedge = \sum_{d,e} (\alpha^d.\beta^e + \beta^e.\alpha^d).\gamma^{d\wedge e}.\bigwedge$

Switch. Along β there are values 0 and 1. Switch transports the α's to γ as long as the last value observed along β was 0 and turns to ε if a 1 is observed; after the next 0 the switch is back in its original position. There are two states: SW_0 and SW_1.

$$SW_0 = \sum_d (\alpha^d.\gamma^d.SW_0 + \beta^0.SW_0 + \beta^1.SW_1)$$

$$SW_1 = \sum_d (\alpha^d.\varepsilon^d.SW_1 + \beta^0.SW_0 + \beta^1.SW_1)$$

Copy 1. $CP1 = \sum_d d.\underline{d}.\underline{d}.CP1$

Copy 2. $CP2 = \sum_d \alpha^d.(\beta^d \| \gamma^d).CP2$

$$(= \sum_d \alpha^d.(\beta^d.\gamma^d + \gamma^d.\beta^d).CP2)$$

Clustering. $CL2 = \sum_{d,e} d.e.\underline{d}.\underline{e}.CL2$

$$CL3 = \sum_{d,e,f} d.e.f.\underline{d}.\underline{e}.\underline{f}.CL3$$

Test for P. $T_P = (\sum_{d\in P} \alpha^d.\beta^d + \sum_{d\notin P} \alpha^d.\gamma^d).T_P \qquad (P \subseteq D)$

Even. $E = \sum_d d.(\sum_e e.\underline{e}.E)$

First. $FIRST = \sum_d d.\underline{d}$

(Here $\sum_d$ stands for $\sum_{d\in D}$.)

1.2. Processes that provide the semantics for data transport channels.

The next processes that we consider play a key role in the data flow networks which are treated in Section 4. In a data flow network the connections or channels between the various nodes, as they appeared in the preceding subsection, may be of various nature; e.g. in such a channel values may retain their input order while being transmitted (as in a queue) or values may overtake each other (as in a bag). It is important to realize that such channels are processes themselves.

All channels that we discuss here have two ports. Intuitively, one of the ports acts as an input port, the other one as an output port. This distinction however plays no role in the formal theory.

We consider BAG, QUEUE, SET and STACK. (Bounded queues and stacks can equally easy be defined, but we will not do so here.) In order to specify QUEUE we use an operator $x \mathbin{/\!\!\!\wedge} \underline{a}$ that merges $\underline{a}$ into x, but in such a way that $\underline{a}$ will precede all underlined steps from x. Here the process x has action alphabet $A \cup \underline{A}$. The rules for $\mathbin{/\!\!\!\wedge}$ contain the auxiliary operator $\mathbin{/\!\!\!\triangle}$.
$x \mathbin{/\!\!\!\triangle} \underline{a}$ is like $x \mathbin{/\!\!\!\wedge} \underline{a}$ but will take its first step from x (if possible and deadlock otherwise.)

$x \mathbin{/\!\!\!\wedge} \underline{a} = \underline{a}.x + x \mathbin{/\!\!\!\triangle} \underline{a}$	QM1
$b \mathbin{/\!\!\!\triangle} \underline{a} = b.\underline{a}$	QM2
$\underline{b} \mathbin{/\!\!\!\triangle} \underline{a} = \delta$	QM3
$bx \mathbin{/\!\!\!\triangle} \underline{a} = b(x \mathbin{/\!\!\!\wedge} \underline{a})$	QM4
$\underline{b}x \mathbin{/\!\!\!\triangle} \underline{a} = \delta$	QM5
$(x + y) \mathbin{/\!\!\!\triangle} \underline{a} = x \mathbin{/\!\!\!\triangle} \underline{a} + y \mathbin{/\!\!\!\triangle} \underline{a}$	QM6

BAG, or B for short, after consuming d can produce one instance of $\underline{d}$ (that is d at its output port). Thus after the action d, B is $\underline{d} \| B$ (basically B but with an additional option $\underline{d}$). QUEUE, or Q for short, after input action d is $\underline{d} \mathbin{/\!\!\!\wedge} Q$ which is basically Q but such that a $\underline{d}$ is preceding all outputs.
S^+ is a stack which can terminate whenever it has become empty. S as in the table below is the usual stack. Sing(d) is a process which represents the singleton {d}, i.e. an entity in which d can be stored whenever it is empty and from which d can be fetched ($\underline{d}$) whenever it is not empty. Finally, SET

is just the cooperation of all singletons. The defining equations are:

$B = \sum_{d \in D} d(\underline{d} \| B)$	BAG
$Q = \sum_{d \in D} d(\underline{d} \wedge Q)$	QUEUE
$S^{+} = \sum_{d \in D} (d + d\,S^{+}).(\underline{d} + \underline{d}\,S^{+})$	TERMINATING STACK
$S = S^{+}.S$	STACK
$Sing(d) = d.\underline{d}.Sing(d)$	SINGLETON
$SET = \Vert_{d \in D}\ Sing(d)$	SET

A further comment on BAG: define B(d) as a bag which can contain only occurrences of d, as follows.

$$B(d) = d(\underline{d} \| B(d)).$$

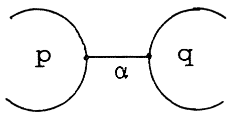
Then B(d) is also a counter, counting d's. Now the following identity holds:

$$B = \Vert_{d \in D}\ B(d).$$

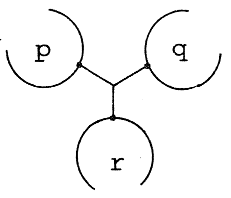
An account of B(d) is contained in [5].

3. NETWORKS OF PROCESSES THAT COMMUNICATE BY HAND SHAKING

3.1. <u>A domain of actions</u>. At the basis of this formalism lies a fixed finite set $\mathbb{P}$ of port names, $\mathbb{P} = \{\alpha, \beta, \gamma, \ldots\}$. Ports are binary connections between processes. Port connections should be distinguished from channels which are processes themselves; a port connection may be thought of as a coincidence of port α in process p and α in process q:



$(\alpha \in \mathbb{P})$

(Though n-ary port connections, as e.g. in 

, can be described just as well in the formalism of ACP, we will refrain from doing so since it involves quite some detail.)

To each port a set D_α of data (values) is assigned. Two sets of actions are associated with α:

$$E_\alpha = \{(\alpha,d) \mid d \in D_\alpha\}$$

$$E^\circ_\alpha = \{(\alpha,d)^\circ \mid d \in D_\alpha\}.$$

(Above, (α,d) was written as α^d.) E_α is called the set of α-*communications* and E°_α is called the set of α-*transactions*.

Further, U is a set of *primitive* atomic actions. Now let $E = \bigcup_{\alpha\in\mathbb{P}} E_\alpha$, $E^\circ = \bigcup_{\alpha\in\mathbb{P}} E^\circ_\alpha$. Then

$$A = E \cup E^\circ \cup U \cup \{\delta\} .$$

We write $A = A(\mathbb{P},\{D_\alpha\},U)$ if A is constructed in this way from $\mathbb{P}$,U and the D_α . Summarizing in a diagram, A is partitioned as follows:

	E_α	E_β	E_γ	....	E *(communications)*
I *(noncommunicating or internal actions)*	E°_α	E°_β	E°_γ	....	E°*(transactions)*
	U				$\cup\{\delta\}$ *(primitive events)*

3.2. <u>A communication function</u>. Given $A(\mathbb{P},\{D\}_\alpha,U)$ a communication function $.|.$ is defined by

$$a|a = a^\circ \text{ for } a \in E.$$

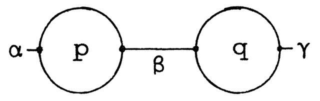
(I.e. $(\alpha,d)|(\alpha,d) = (\alpha,d)^\circ$ for $\alpha \in \mathbb{P}$ and $d \in D_\alpha$.) All other communications yield δ. The basic axioms C1-C3 are immediately satisfied by this definition.

The set I of noncommunicating actions is $U \cup E^\circ \cup \{\delta\}$.

Thus we obtain the process algebra

$$A^\infty = A(\mathbb{P},\{D_\alpha\},U)^\infty = (A^\infty,+,\cdot,\|,\mathbin{|\!\lfloor},|,\delta) .$$

3.3. Example. Let $D_\alpha = D_\beta = D_\gamma = D$ and $p = \sum_{d\in D} \alpha^d.\beta^d.p$ and $q = \sum_{d\in D} \beta^d.\gamma^d.q$.
(We write α^d for $(\alpha,d) \in E_\alpha$.)



Here p and q can communicate along port β by both performing actions β^d.
A successful communication $\beta^d|\beta^d = (\beta^d)°$ can be viewed as a value passing from p to q.

Now p||q is the process corresponding to this small network. Taking into account the fact that single actions β^d cannot be performed (they have to communicate) the process

$$\Delta_{E_\beta}(p||q)$$

describes the process as it can be seen by an observer who can communicate at ports α and γ only. In fact:

$$\Delta_E(p||q) = r \text{ with } r = \sum_{d\in D} \alpha^d.(\beta^d)°.r_d \text{ where}$$

$$r_d = \gamma^d.r + \sum_{e\in D} \alpha^e.\gamma^d.(\beta^e)°.r_e.$$

Here the actions $(\beta^d)°$ are internal actions of r. Note that r is a buffer with capacity 2.

3.4. Process signatures.

A *process signature* is a tuple $\sigma = \{\alpha_1,\ldots,\alpha_k\}$ of ports from $\mathbb{P}$. A process is of signature σ if it is in

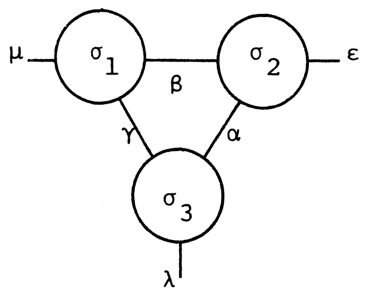
$$(I \cup E_{\alpha_1} \cup \ldots \cup E_{\alpha_k})^\infty.$$

Typically, a process of signature σ is the possible semantics of a component that can communicate (at most) at ports $\alpha_1,\ldots,\alpha_k$.

3.5. Network signatures.

A *network signature* Σ is a family $\{\sigma_1,\ldots,\sigma_k\}$ of process signatures subject to the following condition: each port $\alpha \in \mathbb{P}$ is contained in at most two process signatures σ_1,σ_2 of Σ. Example:

Σ:



3.6. Network abstraction.

For a network signature Σ we find a process signature $\sigma(\Sigma)$ containing all ports shared by exactly one of the process signatures in Σ. In the example above:

$$\sigma(\Sigma) = \{\mu,\lambda,\varepsilon\}.$$

$\sigma(\Sigma)$ is the signature that corresponds to the network Σ if we abstract from the internal communication structure.

3.7. Semantics.

Let $\Sigma = (\sigma_1,\ldots,\sigma_k)$ be a network signature. Let for each $\sigma_i \in \Sigma$ a process p_i of signature σ_i be given. We construct a process $p = \Sigma(p_1,\ldots,p_k)$ of signature $\sigma(\Sigma)$ that corresponds to the network Σ with the p_i substituted for the σ_i-parameters. The process p is denoted by:

$$p = \Delta_H(p_1 \| \ldots \| p_k)$$

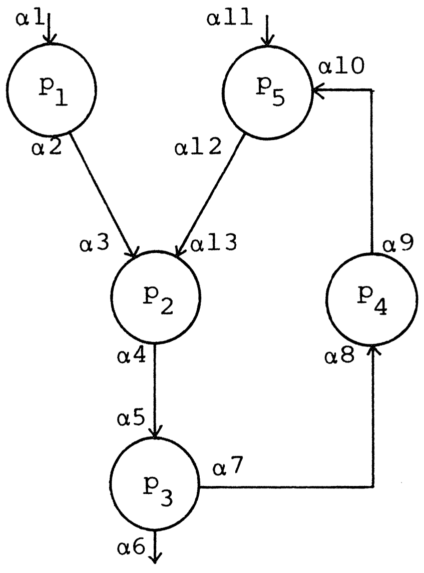
where

$$H = \bigcup\{E_\alpha \mid \alpha \text{ is a port occurring in two process signatures of } \Sigma\}.$$

H contains the communications that should have succeeded within $\Sigma(p_1,\ldots,p_k)$, and Δ_H eliminates the communication actions that failed.

4. DATA FLOW NETWORKS

As an application of communication networks we now turn to pure data flow. Consider a network like:



Here the p_i denote nodes (subprocesses) of the data flow network; the α_i are ports of the nodes.

An important observation is that e.g. $\alpha 2$ and $\alpha 3$ denote <u>different</u> ports, since, although passing the value a at $\alpha 2$ will eventually invoke that a is passed at $\alpha 3$ as well, there may be a long temporal interval in between. In fact, the semantics of "$\alpha 2 \longrightarrow \alpha 3$" is not obvious. Two possibilities arise naturally: "$\longrightarrow$" denotes a queue or a bag (in the latter case values can overtake each other). Let us say that $\alpha 2 \longrightarrow \alpha 3$ and also the other connections $\alpha 4 \longrightarrow \alpha 5$, etc., denote unbounded queues. In the diagram below these processes are drawn as

$$\alpha 2 \text{---}\boxed{\qquad\qquad Q \qquad\qquad}\text{---}\alpha 3$$

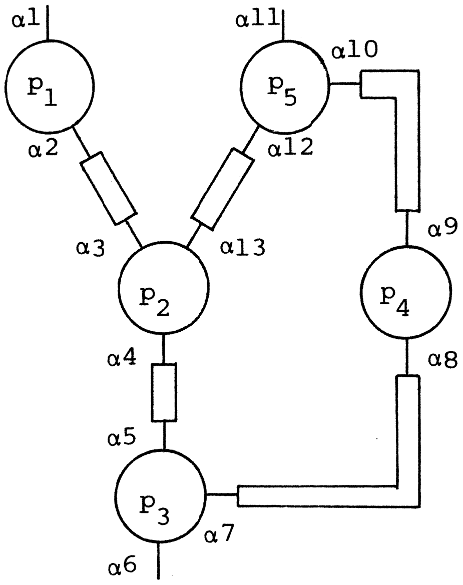
etc. Here the semantics of Q is given by

$$Q^{\alpha 2,\alpha 3} = \sum_{a\in A} a.(\underline{a} \mathbin{/\!\!\!\wedge} Q^{\alpha 2,\alpha 3})$$

where $/\!\!\!\wedge$ is as defined in Section 1.2, 'a' denotes passing a value at port $\alpha 2$, and '$\underline{a}$' denotes passing a value at port $\alpha3$.

(To be more precise: $A = E_{\alpha2} = \{(\alpha2,d) \mid d \in D_{\alpha2}\}$ and $A = E_{\alpha3} = \{(\alpha3,d) \mid d \in D_{\alpha3}\}$. Here $D_{\alpha2} = D_{\alpha3}$, and if $a = (\alpha2,d)$, then $\underline{a} = (\alpha3,d)$.)

In this way the data flow network is turned into a communication network with binary communications:



The semantics of this network has been described in the previous section. Thus, if $P_1,\ldots,P_5$ are processes that describe the nodes $p_1,\ldots,p_5$, then

$$P = P_1 \| \ldots \| P_5 \| Q^{\alpha2,\alpha3} \| Q^{\alpha4,\alpha5} \| \ldots \| Q^{\alpha12,\alpha13}$$

describes the process that corresponds to the data flow network.

We may now view the network as a process with ports $\alpha1,\alpha6,\alpha11$ and with internal actions

$$\bigcup_{i=1}^{5} I(P_i) \cup \bigcup_{i\in J} E^{\circ}_{\alpha i} \quad \text{where } J = \{1,\ldots,13\} - \{1,6,11\}.$$

That process is described by

$$\Delta_H(P), \text{ with } H = \bigcup_{i\in J} E_{\alpha i}.$$

The external actions of this process are $E_{\alpha1} \cup E_{\alpha6} \cup E_{\alpha11}$.

5. DISCUSSION

5.1. Hiding internal steps.

One may wish to forget entirely the internal actions of the data flow network. This may not be possible while staying in the realm of processes. The following however is easy:

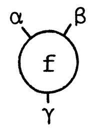
Let p be a process in A^∞. The set *trace*(p) of traces in p is defined as in [5] (essentially by $\mathit{trace}(\Sigma x_i) = \Sigma(\mathit{trace}(x_i))$, $\mathit{trace}(a) = \{a\}$, $\mathit{trace}(\delta) = \lambda$, $\mathit{trace}(ax) = a\cdot\mathit{trace}(x)$.) Let I be a subset of A that we want to hide, i.e. we are interested in p's actions outside I. Let ϕ_I be the homomorphism

$$\phi_I(a) = \begin{cases} a & \text{if } a \notin I \\ \lambda & \text{otherwise.} \end{cases}$$

Then $\phi_I(\mathit{trace}(x))$ gives information about the behaviour of x, not taking into account I.

5.2. Multiple (simultaneous) inputs.

Suppose we have a data flow node as in

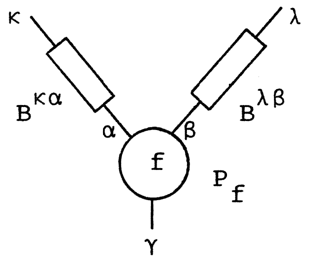


where $f\colon D\times D \to D$ and P_f describes f. We want, intuitively, that P_f returns the value f(a,b) at port γ after obtaining inputs a,b at ports α and β.

In most cases the proper semantics P_f of this situation is found by taking P_f as a process with internal states:

$$\left[\begin{array}{l} P_f = \sum_{d\in D} \alpha^d.P_\alpha^d + \sum_{d\in D} \beta^d.P_\beta^d \\ P_\alpha^d = \sum_{e\in D} \beta^e.\gamma^{f(d,e)}.P_f \\ P_\beta^d = \sum_{e\in D} \alpha^e.\gamma^{f(d,e)}.P_f. \end{array}\right.$$

However, if one <u>really</u> means that inputs d and e must be passed simultaneously to $\textcircled{f}$, then ternary communications have to be used. Let us assume that the channels to be connected at α,β act like bags, since for queues there is no real objection against the previous solution. So let us take into account the channels that connect to $\textcircled{f}$ at ports α,β:

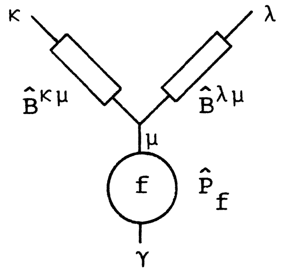


In Section 1.2 it was explained that these input channels are specified by the equations

$$B^{\kappa\alpha} = \sum_{d\in D} \kappa^d.(\alpha^d \| B^{\kappa\alpha})$$

$$B^{\lambda\beta} = \sum_{d\in D} \lambda^d.(\beta^d \| B^{\lambda\beta}).$$

In order to describe the desired behaviour, we replace this subnetwork by:



where a ternary port connection μ is used and where the variants $\hat{P}_f$, $\hat{B}^{\kappa\mu}$, $\hat{B}^{\lambda\mu}$ are given by:

$$\hat{B}^{\kappa\mu} = \sum_{d\in D} \kappa^d.(\sum_{e\in D} \mu^{d\wedge e} \| \hat{B}^{\kappa\mu})$$

$$\hat{B}^{\lambda\mu} = \sum_{e\in D} \lambda^e.(\sum_{d\in D} \mu^{d\wedge e} \| \hat{B}^{\lambda\mu})$$

$$\hat{P}_f = \sum_{d,e\in D\times D} \mu^{d\wedge e}.\gamma^{f(d,e)}.\hat{P}_f\ .$$

Here $d\wedge e$ is a coding of the pair (d,e), and $D\wedge D$ is the port alphabet of μ.

The channel $\hat{B}^{\kappa\mu}$ reacts on an input d at κ by introducing the potential communication $d\wedge e$ at μ. At port μ, communication works as follows:

$$(\mu^{d\wedge e} \mid \mu^{d\wedge e} \mid \mu^{d\wedge e}) = (\mu^{d\wedge e})\circ$$

and this completes the description of the desired simultaneous input behaviour.

It requires an essentially straightforward, but notationally quite nontrivial exposé to discuss n-ary ports in full generality.

5.3. Some other process models.

We will make a few remarks about the process models described in BROCK & ACKERMAN [9], PRATT [15], PARK [14] and BACK & MANNILA [2].

As demonstrated in [9], there is no straightforward generalization of "Kahn's principle" (see KAHN [11]), as it is called in PARK [14], to the case of nondeterministic data flow networks. That is, a semantics for such networks in terms of *history relations* (as a generalization of Kahn's history functions for deterministic networks) turns out not to be adequate (i.e. 'abstract'), by lack of sufficient information of causality (or better, precedence) relations between the events occurring in a network.

This extra information is supplied by PARK [14] by the artificial device of inserting 'hiatons' (silent moves) into the traces of events in order to bring out the necessary precedence relations between events.

BROCK & ACKERMAN [9] use *scenario's* instead: these are, as in PRATT [15], partially ordered multisets of events. (Scenario's in [9] are more restricted than in [15]: e.g. in a scenario as in [9] no precedence relations occur between events at different input ports of the same node.) In compositions of processes the restrictions in the scenario's have to be respected. A process is now a *set of scenario's*. A scenario can be best understood as a generalization of a linear *trace* of events. In fact, as BACK & MANNILA [2] show, trace sets are already an abstract semantics for processes; the extra nondeterminism embodied by the partial order in a scenario is not necessary to ensure abstractness. But it may be necessary, as PRATT [15] points out, to describe processes where some events cannot be temporarily related, e.g. in a situation where parts of the processes involved have their own 'local' time and where a global time is lacking. Or, there may be situations in which events have some duration in time and overlap each other. In this respect scenario's can describe processes which the process model which we use cannot: events for us are points in a global time. However, the processes above are not simply sets of traces: they contain moreover the information of the nondeterministic choices which are made by the process. E.g. the processes a(b + c) and ab + ac have the same trace sets but are unequal since they differ in the timing of their choices. This difference cannot be overlooked since it results in a different deadlock behaviour when embedded in a context. The processes above may be infinite, other than in PRATT [15] who considers only finite scenario's, or in BACK & MANNILA [2] where prefix closed sets of finite traces are used. As Back and Mannila remark, their model of processes therefore does not describe deadlock behaviour and termination behaviour.

The distinctive feature between the approaches considered above and our approach is the possibility of *algebraically* manipulating and specifying processes and networks. This algebraization of process theory starts, of course, with MILNER [12].

Another point is that we have not distinguished between input and output ports in processes and networks of processes, in contrast with e.g. PRATT [15] and in accordance with BACK & MANNILA [2]. Indeed, it turns out that there is a pleasing symmetry between the supposed input and output behaviour of a network: the distinction between input and output ports is one of view, and not inherent to the network. In the model of data flow above the flow of data can just as well be seen to be in the 'reverse' direction; in fact, the intuition of 'flow' is rather deceptive and a more apt intuition would be that of a cellular automaton.

For an operational semantics of nondeterministic data flow networks involving nodes (with internal states) that are given by sets of 'reduction rules', see ARNOLD [1]. The reduction rules specify from what subset of the set of input ports of a node, values are simultaneously taken to be processed and how these input values are processed. It seems essential in this model that the channels are unbounded queues. Using multiple inputs, as in Section 5.2, the Arnold model can be modeled in terms of ACP (although it would be notationally tedious to do so).

Finally, for a denotational semantics of *dynamic* data flow networks, where nodes can be created (in contrast with our static networks), see BÖHM & DE BRUIN [8].

REFERENCES

[1] ARNOLD, A.,
Sémantique des processus communicants,
R.A.I.R.O. Informatique théorique/Theoretical Informatics, Vol.15, No.2, p.103-139 (1981).

[2] BACK, R.J.R. & H. MANNILA,
A refinement of Kahn's semantics to handle non-determinism and communication,
Extended abstract, Preprint University of Helsinki, 1982.

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

[4] DE BAKKER, J.W. & J.I. ZUCKER,
Processes and the denotational semantics of concurrency,
Department of Computer Science Technical Report IW 209/82, Mathematisch centrum, Amsterdam 1982.

[5] DE BAKKER, J.W., J.A. BERGSTRA, J.W. KLOP & J.-J.CH. MEYER,
Linear time and branching time semantics for recursion with merge,
Department of Computer Science Technical Report IW 211/82, Mathematisch Centrum, Amsterdam 1982.

[6] BERGSTRA, J.A. & J.W. KLOP,
Fixed point semantics in process algebras,
Department of Computer Science Technical Report IW 206/82, Mathematisch Centrum, Amsterdam 1982.

[7] BERGSTRA, J.A. & J.W. KLOP,
Process algebra for communication and mutual exclusion,
Department of Computer Science Technical Report IW 218/83, Mathematisch Centrum, Amsterdam 1983.

[8] BÖHM, A.P.W. & A. DE BRUIN,
Dynamic networks of parallel processes,
Department of Computer Science Technical Report IW 192/82, Mathematisch centrum, Amsterdam 1982.

[9] BROCK, J.D. & W.B. ACKERMAN,
Scenarios: a model of non-determinate computation,
Proc. Formalization of Programming concepts (J. Díaz & I. Ramos, eds.), p.252-259, Springer LNCS 107, 1981.

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

[11] KAHN, G.,
The semantics of a simple language for parallel programming,
Proc. IFIP 74, North-Holland, 1974.

[12] MILNER, R.,
A Calculus for Communicating Systems,
Springer LNCS 92, 1980.

[13] NIVAT, M.,
Infinite words, infinite trees, infinite computations,
Foundations of Computer Science III.2 (J.W. de Bakker & J. van Leeuwen, eds.) pp.3-52, Mathematical Centre Tracts 109, Mathematisch Centrum, Amsterdam 1979.

[14] PARK, D.,
Nondeterministic networks: notes on some anomalies,
To appear in the Proc. of the 4th Advanced Course on Foundations of Computer Science, Amsterdam, June 1982.

[15] PRATT, V.R.,
On the composition of processes,
Proc. 9th ACM Symp. on Principles of Programming Languages, p.213-223, 1982.

[16] REM, M.,
Partially ordered computations, with applications to VLSI design,
To appear in the Proc. of the 4th Advanced Course on Foundations of Computer Science, Amsterdam, June 1982.

69D13
69F11
69F12
69F32