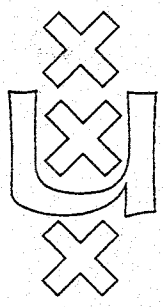
Process algebra with zero object and non-determinacy



University of Amsterdam
Faculty of Mathematics and Computer Sciences
Programming Research Group

Process algebra with zero object and non-determinacy

J.C.M. Baeten
J.A. Bergstra

Report P9002

March 1990

Jos Baeten

Department of Software Technology
Centre for Mathematics and Computer Science

CWI M331
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

P.O. Box 4079
1009 AB Amsterdam
The Netherlands

tel. +31 20 5924008

Jan Bergstra

Programming Research Group
Faculty of Mathematics and Computer Sciences
University of Amsterdam

NIKHEF K A109
Kruislaan 409
1098 SJ Amsterdam
The Netherlands

P.O. Box 41882
1009 DB Amsterdam
The Netherlands

tel. +31 20 5922013

Department of Philosophy
University at Utrecht

Transitorium II kr. 1026
Heidelberglaan 2
3584 CS Utrecht
The Netherlands

tel. +31 30 532761

Process Algebra with Zero Object and Non-Determinacy

J.C.M. Baeten
Dept. of Software Technology, Centre for Mathematics and Computer Science, P.O.Box 4079, 1009 AB Amsterdam, The Netherlands

J.A. Bergstra
Programming Research Group, University of Amsterdam, P.O.Box 41882, 1009 DB Amsterdam, The Netherlands

Department of Philosophy, State University of Utrecht, Heidelberglaan 2, 3584 CS Utrecht, The Netherlands

The object 0 acts as a zero for both sum and multiplication in process algebra. The constant δ, representing deadlock or inaction, is only a left zero for multiplication. We will call 0 predictable failure. Also, we look at a two-sorted structure of processes with history, and discuss non-determinacy.

1980 Mathematics Subject Classification (1985 revision): 68Q45, 68Q55, 68Q65, 68Q50.
1987 CR Categories: F.4.3, D.2.10, D.3.1, D.3.3.
Key words & Phrases: process algebra, zero, deadlock, inaction, failure, non-determinacy.
Note: Partial support received by ESPRIT basic research action 3006, CONCUR, and by RACE contract 1046, SPECS. This document does not necessarily reflect the views of the SPECS consortium.

1. INTRODUCTION.

The object 0 acts as a 0 for both sum and multiplication in process algebra. The constant δ representing deadlock or inaction is only a left zero for multiplication.

We will call 0 *predictable failure*. A predictable failure differs from deadlock (inaction) in the sense that it can be observed by the system. The axioms for 0 incorporate the intention of a system to avoid failure whenever possible. The axioms for δ (in particular $x \neq 0 \Rightarrow x + \delta = x$) incorporate the intention of a process to make progress if it can.

In fact 0 stands for a truly empty process, is execution is simply inconceivable. The process 0 also occurs in PONSE & DE VRIES [89] (but is called δ there).

A process specification involving 0 or renaming into 0 is not executable. It must be implemented, which means that one has to provide an equivalent specification not involving 0 or renaming into 0. Due to this observation 0 is a high level feature that plays a role in system design rather than in specification, implementation or verification.

Another setting where a zero object pays off is in trace theory, the brand of process theory that is based on collecting computational histories. We discuss trace theory with a zero object. It turns out that alternative composition on traces is *not* the same as on processes, because it represents uncertainty about the past, whereas on processes, it represents uncertainty about the future. We decide to use $\vee$ for alternative composition on histories. Finally, all features are combined in the sort HP of processes with histories.

2. AXIOMATIZATION.

2.1. BASIC PROCESS ALGEBRA WITH ZERO.

We start out from the theory of Basic Process Algebra as described in BERGSTRA & KLOP [84, 85, 86]. For a recent survey article see BERGSTRA & KLOP [89]. We have a set of **atomic actions**, A, that are constants in the theory. Further, we have two binary operators: + is **alternative composition** or sum, and · is **sequential composition** or product. For this signature, we have the first five axioms in table 1 below (A1-5), constituting BPA. Then we add the **zero** constant, obeying the next three axioms (Z1-3). Usually, we also have the constant δ in the theory, called **inaction**. The two axioms for inaction need conditions, in order to avoid clashes with the zero axioms ($A6^0$, $A7^0$). See the last two axioms below. We put $A_\delta = A \cup \{\delta\}$, $A_{0\delta} = A \cup \{0,\delta\}$, BPA_0 = BPA + Z1-3, $BPA_{0\delta}$ = BPA_0 + $A6^0$, $A7^0$. Note that we have in particular $\delta \cdot 0 = 0$. The explanation of this in philosophical terms is shallow. Of all operators, + will bind the weakest, and · the strongest. We often leave out the · sign.

$x + y = y + x$	A1
$x + (y + z) = (x + y) + z$	A2
$x + x = x$	A3
$(x + y)\cdot z = x\cdot z + y\cdot z$	A4
$(x\cdot y)\cdot z = x\cdot(y\cdot z)$	A5
$x\cdot 0 = 0$	Z1
$x + 0 = x$	Z2
$0\cdot x = 0$	Z3
$x \neq 0 \Rightarrow x + \delta = x$	$A6^0$
$x \neq 0 \Rightarrow \delta\cdot x = \delta$	$A7^0$

TABLE 1. $BPA_{0\delta}$.

2.2. PROJECTION.

We can extend the signature given above with the **projection operators** π_n (for $n>1$). The axioms are straightforward adaptations of the usual ones (see BERGSTRA & KLOP [84]). In table 2, $a \in A$.

$\pi_n(0) = 0$
$\pi_n(\delta) = \delta$
$\pi_n(a) = a$
$\pi_n(x + y) = \pi_n(x) + \pi_n(y)$
$\pi_{n+1}(ax) = a\cdot\pi_n(x)$

TABLE 2. Projection.

2.3. DEFINITION.

A useful unary predicate on processes is $\neq 0$, that determines whether or not a term can be proved equal to the zero process. Again, $a \in A$.

$\delta \neq 0$
$a \neq 0$
$x \neq 0, y \neq 0 \Rightarrow x \cdot y \neq 0$
$x \neq 0 \Rightarrow x + y \neq 0$

TABLE 3. Predicate $\neq 0$.

2.4 INFINITARY RULES.

AIP can be maintained as in BERGSTRA & KLOP [86]. It reads as follows:

for all n $\pi_n(x) = \pi_n(y) \quad \Rightarrow \quad x = y$.

This rule is valid for finitely branching processes only (a process is finitely branching if it has a representation as a finitely branching graph or tree, see further on).

Notice that the equation $x = a \cdot x$ has two solutions: 0 and a^ω. It follows that the principle RSP has to be relaxed: every guarded system of recursion equations not containing an occurrence of 0 has at most one solution different from 0. Thus more formally RSP^0 states:

Let E be a guarded recursive specification with k process names such that none of the equations of E involves either 0 or a renaming into 0 (see below). Such E is called **0-free**.
Let $X = (x_1, ..., x_k)$ and $Y = (y_1, ..., y_k)$ be process vectors of length k.
Then $x_1 \neq 0 \,\&\, ... \,\&\, x_k \neq 0 \,\&\, y_1 \neq 0 \,\&\, ... \,\&\, y_k \neq 0 \,\&\, X = E(X), Y = E(Y)$
implies $X = Y$.

Besides RSP^0 there is also RDP^0 which states that:

every 0-free guarded system of recursion equations possesses at least one solution vector consisting of processes different from 0.

2.5 RENAMING INTO ZERO.

0_b is an operator that substitutes 0 for $b \in A_\delta$. We have to look at the cases $b = \delta$ and $b \in A$ separately. First, we consider $b = \delta$. We will call 0_b (for $b \in A$) **failure prediction**, and 0_δ **deadlock prevention**. Let $a \in A$.

$0_\delta(0) = 0$
$0_\delta(\delta) = 0$
$0_\delta(a) = a$
$0_\delta(x + y) = 0_\delta(x) + 0_\delta(y)$
$0_\delta(ax) = a \cdot 0_\delta(x)$

TABLE 4. Deadlock prevention.

Next, let $b \in A$. Let $a \in A$ be any action different from b.

$0_b(0) = 0$
$0_b(\delta) = \delta$
$0_b(b) = 0$
$0_b(a) = a$
$0_b(x + y) = 0_b(x) + 0_b(y)$
$0_b(bx) = 0$
$0_b(ax) = a \cdot 0_b(x)$

TABLE 5. Failure prediction.

Note that the equation $0_b(x \cdot y) = 0_b(x) \cdot 0_b(y)$ leads to a problem as follows: Let $x = a^\omega$ (i.e. the unique solution different from 0 of the equation $z = a \cdot z$) and $y = b$, then $x \cdot y = x$ because of AIP, hence $x = 0_b(x) = 0_b(x \cdot y) = 0_b(x) \cdot 0_b(b) = 0_b(x) \cdot 0 = 0$, which contradicts the assumptions on x.

Note that also the equation $0_b(a \cdot x) = 0_b(a) \cdot 0_b(x)$ cannot be maintained for all $a \in A_\delta$, for that would imply $\delta = 0_b(\delta) = 0_b(\delta \cdot b) = 0_b(\delta) \cdot 0_b(b) = \delta \cdot 0 = 0$.

It is easy to generalize the 0_b operator to an operator 0_K, with K a set of atomic actions ($K \subseteq A$).

3. SEMANTICS.

3.1 TRANSITION RULES.

We define a **structured operational semantics** as in VAN GLABBEEK [87], on congruence classes of process expressions, i.e. a $\xrightarrow{a}$ relation holds between two terms iff they are provably equal to the format in the rules in table 6 below. Likewise, a term satisfies $\xrightarrow{a} \surd$ iff it can be written in the form $a + x$. For closed terms, the rules now determine an **action graph**. On action graphs, we will then define a notion of **bisimulation** as in BERGSTRA & KLOP [86].

$a + x \xrightarrow{a} \surd$
$x \neq 0 \Rightarrow a \cdot x + y \xrightarrow{a} x$

TABLE 6. Action rules.

We can also add rules in order to deal with recursive equations. In that case, however, the transition system may become undecidable because $x \neq 0$ is in general not decidable.

3.2 GRAPH MODEL.

We can also define a graph model directly. Let $\mathbb{T}$ be the set of all rooted labeled trees, where all edges are labeled with elements of A_δ and all endpoints labeled with an element of $\{\surd, \delta, 0\}$. The interpretation of the constants and operators of $BPA_{0\delta}$ is straightforward:

- $[\delta]$ is the one-node graph labeled with δ, $[0]$ is the one-node graph labeled with 0;
- $[a]$ is the two-node graph with one edge labeled a, and the endpoint labeled $\surd$;
- if g,h are in $\mathbb{T}$, then $g + h$ is obtained by identifying the roots of g and h. If one graph is $[0]$, the result is just the other graph. If one graph is $[\delta]$, the result is also the other graph, unless that graph is $[0]$ (in which case it is $[0]$);
- if g,h are in $\mathbb{T}$, then $g \cdot h$ is obtained by first making a copy of h for each $\surd$-endpoint of g. Then we remove the $\surd$-label, and identify the endpoint with the root of its copy.

We give some examples in fig. 1 below. We see the trees that represent the terms 0, $(a + b) \cdot c$, $a \cdot \delta + a \cdot 0$.

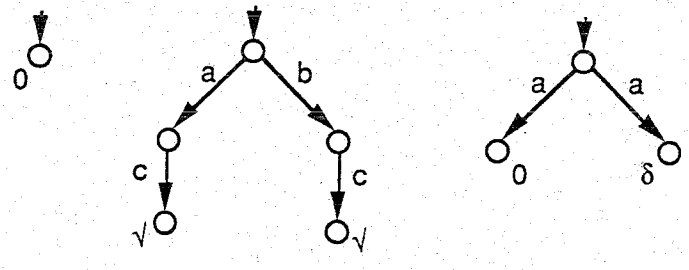


FIGURE 1.

If g is a tree in $\mathbb{T}$, and s a node in g, then we call s a **zero node** of g if every maximal path in g from s must end in a 0-labeled point. Now we can give the definition of bisimulation on these trees.

3.2.1 DEFINITION.
Let $g,h \in \mathbb{T}$. We say g,h are **bisimilar**, $g \leftrightarrows h$, if there exists a relation R (called a **bisimulation**) on nodes of g and h, such that

1. the domain of R consists of all non-zero nodes of g;
2. the range of R consists of all non-zero nodes of h;
3. either g,h are both the zero graph, or the roots of g,h are related;
4. if R(s,t) and $s \xrightarrow{a} s'$ and s' non-zero, then there is a non-zero t' such that $t \xrightarrow{a} t'$ and R(s',t');
5. if R(s,t) and $t \xrightarrow{a} t'$ and t' non-zero, then there is a non-zero s' such that $s \xrightarrow{a} s'$ and R(s',t');
6. if R(s,t) and s,t are endpoints, then they have the same label.

3.2.2 PROPOSITION
Bisimulation is a congruence on $\mathbb{T}$.

3.2.3 THEOREM
$\mathbb{T}/\leftrightarrows$ is a sound and complete model for $BPA_{0\delta}$.

The graph model has a substructure consisting of the processes $\neq 0$. That structure is a model of BPA_{δ}, the theory without zero, and with axioms A1-7 (no conditions on A6 and A7). Thus there is only one process in which 0 features in an essential way and that is 0 itself.

4. EXTENSIONS.

4.1 RENAMING.

Now we will look at renamings of atomic actions into atomic actions or δ. (Renaming into zero was discussed in 2.5.) Let $f: A_\delta \to A_\delta$ be any function that keeps δ fixed, i.e. $f(\delta) = \delta$. Then the **renaming operator** ρ_f is defined by the axioms in table 7, where $a \in A_\delta$.

$\rho_f(0) = 0$
$\rho_f(a) = f(a)$
$\rho_f(x + y) = \rho_f(x) + \rho_f(y)$
$\rho_f(x \cdot y) = \rho_f(x) \cdot \rho_f(y)$

TABLE 7. Renaming.

A very useful example of a renaming operator is the **encapsulation operator** ∂_H (for $H \subseteq A$) that is based on the function g given by:

$g(a) = \delta$ if $a \in H$ and $g(a) = a$ if $a \notin H$.

4.2 PARALLEL COMPOSITION.

We can extend the theory $BPA_{0\delta}$ with parallel composition as in the theory ACP of BERGSTRA & KLOP [84]. In order to axiomatize the parallel operator $\|$ (**merge**), we need two auxiliary operators $\lfloor\!\lfloor$ (**left-merge**) and $|$ (**communication merge**). The theory is parametrized by a **communication function** $|$, a binary function on the set of constants $A_{0\delta}$ that satisfies conditions C1-4 in table 8 below. Moreover, we have the encapsulation operator of 4.1, that is used to block communications

with the outside. The theory ACP_0 consists of $BPA_{0\delta}$ plus the axioms in table 8 (see below). In table 8, $a,b,c \in A_{0\delta}$.

$a \mid b = b \mid a$	C1
$(a \mid b) \mid c = a \mid (b \mid c)$	C2
$a \neq 0 \Rightarrow \delta \mid a = \delta$	C3
$0 \mid a = 0$	C4
$x \parallel y = x \mathbin{\mathbb{L}} y + y \mathbin{\mathbb{L}} x + x \mid y$	CM1
$a \mathbin{\mathbb{L}} x = a \cdot x$	CM2
$ax \mathbin{\mathbb{L}} y = a \cdot (x \parallel y)$	CM3
$(x + y) \mathbin{\mathbb{L}} z = x \mathbin{\mathbb{L}} z + y \mathbin{\mathbb{L}} z$	CM4
$ax \mid b = (a \mid b) \cdot x$	CM5
$a \mid bx = (a \mid b) \cdot x$	CM6
$ax \mid by = (a \mid b) \cdot (x \parallel y)$	CM7
$(x+y) \mid z = x \mid z + y \mid z$	CM8
$x \mid (y+z) = x \mid y + x \mid z$	CM9
$\partial_H(0) = 0$	D0
$\partial_H(a) = a$ if $a \notin H$	D1
$\partial_H(a) = \delta$ if $a \in H$	D2
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D3
$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$	D4

TABLE 8. ACP_0.

Note that we can prove that for all closed terms x we have $x \parallel 0 = x \mathbin{\mathbb{L}} 0 = x \mid 0 = 0 \mid x = 0$.

On the graph model, we can define parallel composition as follows:

- the node set of graph $g \parallel h$ is the cartesian product of the node sets of g and h;
- there is an edge $(s,t) \xrightarrow{a} (s',t)$ iff there is an edge $s \xrightarrow{a} s'$ in g;

likewise, there is an edge $(s,t) \xrightarrow{a} (s,t')$ iff there is an edge $t \xrightarrow{a} t'$ in h;

and there is an edge $(s,t) \xrightarrow{a} (s',t')$ iff there are edges $s \xrightarrow{b} s'$ in g and $t \xrightarrow{c} t'$ in h with $b \mid c = a$;

- an endpoint (s,t) has a 0-label iff either s or t has a 0-label;

an endpoint (s,t) has a δ-label if one has a δ-label, and the other a δ or $\surd$-label;

an endpoint (s,t) has a $\surd$-label if both s and t have a $\surd$-label (all labels in non-endpoints are dropped).

It can be proved that with this definition, bisimulation is also a congruence for parallel composition, and all axioms of ACP_0 hold in the graph model.

4.3 STATE OPERATOR.

We can add a **state operator** λ to the theory along the same lines as in BAETEN & BERGSTRA [88]. The process $\lambda_s(x)$ represents the process x in state s. The state operator is parametrized by two functions **action** and **effect**. We will allow that $a(s) = 0$ (the action function gives 0 for a certain constant and state), further one must assume that $s(0) = s$ and $0(s) = 0$ (effect and action remain fixed for 0), and $s(\delta) = s$ and either $\delta(s) = \delta$ or $\delta(s) = 0$. Then the state operator works just as well in the case of ACP. The axioms are displayed in table 9. We have $a \in A_{0\delta}$.

$\lambda_s(a) = a(s)$	
$\lambda_s(a \cdot x) = 0$	if $a(s) = 0$
$x \neq 0 \Rightarrow \lambda_s(a \cdot x) = \delta$	if $a(s) = \delta$
$\lambda_s(a \cdot x) = a(s) \cdot \lambda_{s(a)}(x)$	if $a(s) \in A$
$\lambda_s(x + y) = \lambda_s(x) + \lambda_s(y)$	

TABLE 9. State operator.

4.4. PRIORITIES.

In examples in section 5 we will also make use of the **priority operator** of BAETEN, BERGSTRA & KLOP [87]. This operator gives some actions priority over others in a sum context. An auxiliary operator $\lhd$ (**unless**) is needed to give a finite axiomatization. We assume that a partial ordering $<$ is given on A (so 0 and δ are not ordered). Table 10 gives axioms on top of the axioms of ACP_0. We have $a,b \in A_\delta$.

$a \lhd b = a$	if *not* $a<b$
$a \lhd b = \delta$	if $a<b$
$0 \lhd x = 0$	
$x \lhd 0 = x$	
$z \neq 0 \Rightarrow x \lhd yz = x \lhd y$	
$x \lhd (y + z) = (x \lhd y) \lhd z$	
$xy \lhd z = (x \lhd z)y$	
$(x + y) \lhd z = x \lhd z + y \lhd z$	
$\theta(0) = 0$	
$\theta(a) = a$	
$\theta(xy) = \theta(x) \cdot \theta(y)$	
$\theta(x + y) = \theta(x) \lhd y + \theta(y) \lhd x$	

TABLE 10. Priority operator.

4.5. WEAVING.

In advance of an explanation of how to apply failure prediction in the design of (toy) control systems we will introduce a parallel composition operator that differs from the ACP merge. This operator is called **weaving**, because on trace sets it corresponds exactly to the weaving operator of trace theory, see REM [87]. It is denoted with $x \parallel_B y$ and has in failure semantics the same meaning as the corresponding operator of TCSP, see HOARE [85] from which the notation is taken. Our axioms explain it in terms of bisimulation semantics and therefore in terms of many more abstract semantic models. In table 11 we give an axiomatization on top of BPA_δ, so *not* considering the extra constant 0. We have $B \subseteq A$, $a,b \in A_\delta$. We can add 0 by putting: $0 \parallel_B x = x \parallel_B 0 = 0 \mathbin{\mathbb{L}}_B x = x \mathbin{\mathbb{L}}_B 0 = 0 \mid_B x = x \mid_B 0 = 0$.

$x \parallel_B y = x \mathbin{\mathbb{L}_B} y + y \mathbin{\mathbb{L}_B} x + x \mid_B y$	
$a \mathbin{\mathbb{L}_B} x = a \cdot x$	if $a \notin B$
$a \mathbin{\mathbb{L}_B} x = \delta$	if $a \in B$
$(a \cdot x) \mathbin{\mathbb{L}_B} y = a \cdot (x \parallel_B y)$	if $a \notin B$
$(a \cdot x) \mathbin{\mathbb{L}_B} y = \delta$	if $a \in B$
$(x + y) \mathbin{\mathbb{L}_B} z = x \mathbin{\mathbb{L}_B} z + y \mathbin{\mathbb{L}_B} z$	
$a \mid_B b = \delta$	if $a \notin B$ or $a \neq b$
$a \mid_B a = a$	if $a \in B$
$(a \cdot x) \mid_B b = (a \mid_B b) \cdot x$	
$a \mid_B (b \cdot x) = (a \mid_B b) \cdot x$	
$(a \cdot x) \mid_B (b \cdot y) = (a \mid_B b) \cdot (x \parallel_B y)$	
$(x + y) \mid_B z = (x \mid_B z) + (y \mid_B z)$	
$x \mid_B (y + z) = (x \mid_B z) + (y \mid_B z)$	

TABLE 11. Weaving.

Weaving is a parallel composition that uses action sharing: the actions named in the subscript B must occur in a shared fashion for both x and y simultaneously. The above equations describe weaving on the bisimulation model. It is possible to describe weaving in terms of the merge of ACP. Then it is necessary to introduce copies of the atomic actions, so let for every $b \in B$, b^c be a new copy different from all other actions in x and y and let c be a renaming function that renames every $b \in B$ to b^c and leaves all other atoms unchanged. As a communication function we have $b^c \mid b^c = b$, all other communications are trivial. Then the following identity holds for all finite closed process expressions:

$$x \parallel_B y = \partial_{Bc}(\rho_c(x) \parallel \rho_c(y)).$$

The reason to have weaving in addition to $\parallel$ of ACP is that in many cases the shared action communication mechanism is quite pleasant and one would prefer not to be burdened with its encoding in terms of the merge operator.

5. APPLICATIONS.

5.1. SYSTEMS CONTROL.

In order to apply failure prediction we start from a system S that may be operated with actions from a set B, a set of *buttons*. For simplicity we assume that S is perpetual (does not terminate). Every now and then an error e may occur ($e \notin B$). A controller allows the use of S. The functionality of this controller is as follows:

$$C = \sum_{b \in B} \mathrm{instr}(b) \cdot H(b) \cdot C,$$

where H(b) is a handler for the instruction b. H(b) may or may not perform the action b, meant as an instruction for S. We will choose the following equation for the handler:

$$H(b) = b \cdot \mathrm{done}(b) + \mathrm{not}(b)$$

The action not(b) denotes a signal from the controller that b may not be performed, the action done(b) is a controller signal indicating that b has successfully been performed. Both these actions are supposed not to occur in any other system component. Thus the external alphabet of the controller is $\mathrm{instr}(B) \cup \mathrm{done}(B) \cup \mathrm{not}(B)$ and none of these actions is supposed to occur in S.

The handler uses a simulation program SIM that simulates the action b as an instruction for S. If this simulation reveals a problem (a predictable failure) then b is not enforced on S, otherwise it will be. We have:

$SIM = \partial_e(S)$.

Let $<$ be the partial ordering of atomic actions that imposes $not(b) < b$ for all $b \in B$ and no other relations. Then the controller together with the simulated system work as follows:

$C\text{-}SIM = \theta_<(C \between_B SIM)$

The system C-SIM allows not(b) if it will not allow b. C-SIM allows b if after b, S can proceed with at least one infinite trace of actions not involving e. Of course, C-SIM must be implemented in a way that does not use the constant 0 or the 'real' system S.

Now, finally, the controller together with the system S is given by:

$C\text{-}S = C\text{-}SIM \between_B S$.

Due to the nature of the weaving operator, the occurring ternary communication can be described in a very compact way.

5.2. TELEPHONE NUMBERS.

Another example is the dialing of telephone numbers. After certain initial sequences, it is clear that no continuation can lead to a connection (e.g. because a switching board is used that is temporarily out of service). A specification can be given where an error occurs after the complete number has been dialed. Using failure prediction, the information becomes available as soon as it is derivable. In an implementation, this knowledge has to be incorporated in the nearest switching board.

5.3. NP-COMPLETENESS.

It is well-known that any NP-complete problem can be solved non-deterministically in polynomial time. For instance, the satisfiability problem can be solved by guessing the value of a proposition variable at every step. It is straightforward to code such an algorithm as a process. Using failure prediction, we can force such a process to only consider branches that do not lead to failure.

5.4. TRAFFIC LIGHT.

Let P be a point that travels on a one dimensional two way infinite discrete grid (i.e. the integers). At each moment in time the coordinates of the point are an integer pair (p, v) where p is the position on the grid and v is an integer denoting the velocity of P: if $v = -3$ this means that in one unit of time (say a second) P moves from p to $p - 3$. There are three actions for P and one of these is performed each second:

st	remain in the same state (keep the same speed in the same direction).
la	accelerate left: $v \rightarrow v - 1$,
ra	accelerate right: $v \rightarrow v + 1$.

Thus $P = (st + la + ra) \cdot tick \cdot P$ where tick marks the progress of a clock.

At the same time, there is a traffic light at position 10 on the grid. Every 3 seconds this light changes its colour, from green to red and back again:

$TL = green \cdot tick \cdot tick \cdot tick \cdot red \cdot tick \cdot tick \cdot tick \cdot TL$.

Here tick marks the progress of the same clock as for the moving object P.

We require the communication $tick \mid tick = t$. The composition of object and traffic light is $\partial_{\{tick\}}(P \between TL)$. The next step is that we have a state operator with triples consisting of an integer pair and a colour as states. The functions action and effect work as follows (p,v integers, c a colour):

effect	action
(p, v, c)(st) = (p, v, c)	st(p, v, c) = st
(p, v, c)(la) = (p, v-1, c)	la(p, v, c) = la
(p, v, c)(ra) = (p, v+1, c)	ra(p, v, c) = ra
(p, v, c)(red) = (p, v , red)	red(p, v, c) = red
(p, v, c)(green) = (p, v , green)	green(p, v, c) = green
(p, v, c)(t) = (p + v, v, c)	t(p, v, c) = δ if c = red & v > 0 & p ≤ 10 ≤ p + v
	t(p, v, c) = t otherwise.

Thus, for instance the second line of this table says that if action la is performed in state (p,v,c), then we see the action la occurring, and the resulting state is (p,v-1,c).
The process PTL = $\lambda_{(0,\ 0,\ green)}(\partial_{\{tick\}}(P \parallel TL))$ describes P starting in the position (0, 0) with the constraint that a deadlock occurs if P crosses the traffic light from left to right if it is green.

Next the process PTLC = 0_δ(PTL) describes P under the constraint that it will never cross the traffic light in red state from left to right. (C denotes correct functioning of the PTL combination).

Using the operator 0_δ it becomes possible to view all possible ways of correct behaviour as a process itself. Notice that if we view st, la, ra as control options for an agent that controls P then controlling P in the context PTL leaves the controlling agent all freedom of action (choice from st, la, ra at any moment). In contrast to this, the freedom of control in the context PTLC is limited.

We give examples of applying 0_δ in various states of PTL. For instance:
$0_\delta(\lambda_{(2,\ 5,\ green)}(\partial_{\{tick\}}(P \parallel red \cdot tick \cdot tick \cdot tick \cdot TL))) = 0$, but for no v we have
$0_\delta(\lambda_{(11,\ v,\ red)}(\partial_{\{tick\}}(P \parallel TL))) = 0$.

Of course this is just a toy example but one may imagine a more complex control system for which disastrous events have to be avoided. Then the freedom of a controlling agent has to be limited in order to avoid problems. Using the operation 0_δ it becomes possible to specify a control system that disallows actions that must inevitably lead to a problematic stage (i.e. 0). Of course the implementation of such a control system is quite a different matter. Already in the simple case with moving point and traffic light above, a specification of PTLC without the use of 0 is not so straightforward.

Using 0, one may cut down a process graph to correct (failure free) process executions only.

In terms of a control system as described in 5.1 we get the following:
B = {la, ra, st}
S = P
SIM = PTLC
C-SIM = $\theta_<$(C $\parallel_B$ SIM)
C-S = C $\parallel_B$ P.

6. PROCESSES WITH HISTORIES.

In this section we will look at a two-sorted structure for processes. We use this structure to describe a process that is still to be executed together with its history.

6.1 HISTORIES.

A **history** or **trace** is a sequence of actions. We have the following signature:

H	the set of histories, elements of A^* and elements of A^* ending in δ
ε	the empty history
0	impossible history

$\cdot$	binary operator, sequential composition
$\vee$	binary operator, **non-determinate composition** or disjunction
$\|$	binary operator, parallel composition
0_K	unary operators for $K \subseteq A$, failure prediction.

For this signature, we can give an axiomatization as follows, in table 12, where $a,b \in A$. This gives a standard axiomatization for so-called trace theory.

$$
\begin{array}{l}
x \vee y = y \vee x \\
(x \vee y) \vee z = x \vee (y \vee z) \\
x \vee x = x \\
(x \vee y)\cdot z = x\cdot z \vee y\cdot z \\
x\cdot(y \vee z) = x\cdot y \vee x\cdot z \\
(x\cdot y)\cdot z = x\cdot(y\cdot z) \\
\varepsilon\cdot x = x \\
x\cdot\varepsilon = x \\
x + 0 = x \\
0\cdot x = 0 \\
x\cdot 0 = 0 \\
\\
a \mid b = b \mid a \\
(a \mid b) \mid c = a \mid (b \mid c) \\
\\
\varepsilon \| x = x \\
x \| \varepsilon = x \\
0 \| x = 0 \\
x \| 0 = 0 \\
ax \| by = a(x \| by) \vee b(ax \| y) \vee (a \mid b)(x \| y) \\
(x \vee y) \| z = x \| z \vee y \| z \\
x \| (y \vee z) = x \| y \vee x \| z \\
\\
0_K(\varepsilon) = \varepsilon \\
0_K(0) = 0 \\
0_K(a) = 0 \qquad \text{if } a \in K \\
0_K(a) = a \qquad \text{if } a \notin K \\
0_K(x \vee y) = 0_K(x) \vee 0_K(y) \\
0_K(ax) = 0 \qquad \text{if } a \in K \\
0_K(ax) = a\cdot 0_K(x) \quad \text{if } a \notin K
\end{array}
$$

TABLE 12. Histories.

6.2 HISTORY AND PROCESS COMBINATION.

It is interesting now to combine a process with its history. We get the following signature.

H	sort of histories, with operators ε, 0, $\cdot$, $\vee$, $\|$, ∂_K
P	sort of processes, with operators δ, 0, $\cdot$, $+$, $\|$, $\lfloor\!\lfloor$, $\mid$, ∂_K
HP	sort of history and process combinations
$\varnothing \in HP$	non-existent combination
$\blacktriangleright : H \times P \to HP$	history and process combination operator

$\gg\surd: H \to HP$	history completion
$\_/a: HP \to HP$	external action function (for $a \in A$)
$\vee: HP \times HP \to HP$	non-determinate composition
$\|\|: HP \times HP \to HP$	parallel composition
$\partial_K: HP \to HP$	encapsulation (for $K \subseteq A$).

We display this signature in fig. 2.

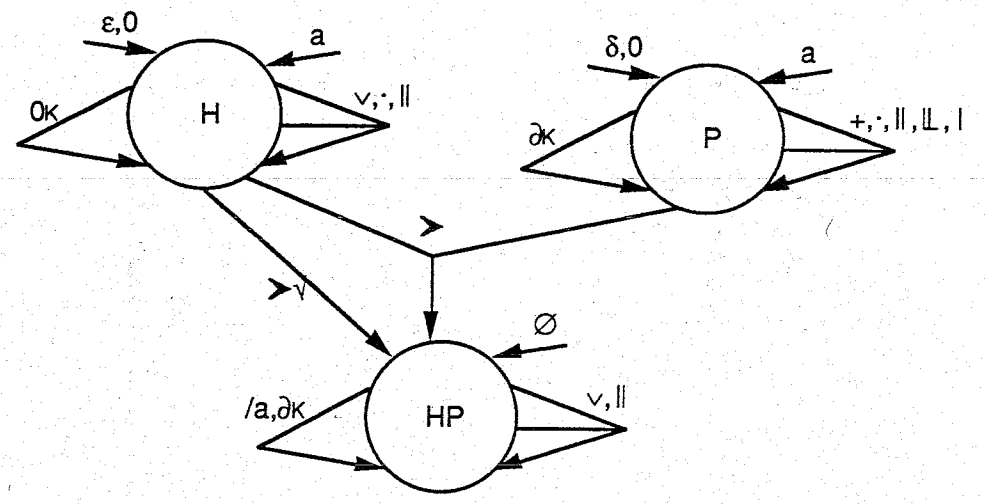


FIGURE 2.

6.3 CONSTRUCTORS.

We start off by giving the axioms for the constructors of HP. In the next table, we have $\alpha,\beta \in H$; $x \in P$; $X,Y,Z \in HP$.

$\alpha \gg 0 = \emptyset$
$0 \gg x = \emptyset$
$0 \gg \surd = \emptyset$
$(\alpha \vee \beta) \gg x = (\alpha \gg x) \vee (\beta \gg x)$
$(\alpha \vee \beta) \gg \surd = (\alpha \gg \surd) \vee (\beta \gg \surd)$
$X \vee Y = Y \vee X$
$(X \vee Y) \vee Z = X \vee (Y \vee Z)$
$X \vee X = X$
$X \vee \emptyset = X$

TABLE 13. Constructors of HP.

From these axioms, it follows that we can consider elements of HP as sets of elements of the form $\alpha \gg x$, with α not ending in δ and $x \neq 0$. In inductive definitions, we can use the fact that all elements of HP have one of the following four forms:

- $\emptyset$;
- $\alpha \gg \surd$, with α not ending in δ;
- $\alpha \gg x$, with $x \neq 0$ and α not ending in δ;
- $X \vee Y$.

We will use these forms in the following definitions. Notice that there is no point in using recursive equations on HP (there is not even a sequential composition operator), so that the constructors give all objects of HP indeed.

6.4 PARALLEL COMPOSITION.
First, we define parallel composition and encapsulation on HP. We have $\alpha,\beta \in H$; $x,y \in P$; $X,Y,Z \in HP$; $K \subseteq A$.

$$
\begin{aligned}
&X \| \varnothing = \varnothing \\
&\varnothing \| X = \varnothing \\
&(X \vee Y) \| Z = X \| Z \vee Y \| Z \\
&X \| (Y \vee Z) = X \| Y \vee X \| Z \\
&(\alpha \succ x) \| (\beta \succ y) = \alpha \| \beta \succ x \| y \\
&(\alpha \succ \surd) \| (\beta \succ x) = \alpha \| \beta \succ x \\
&(\alpha \succ x) \| (\beta \succ \surd) = \alpha \| \beta \succ x \\
&(\alpha \succ \surd) \| (\beta \succ \surd) = \alpha \| \beta \succ \surd \\
&\\
&\partial_K(\varnothing) = \varnothing \\
&\partial_K(X \vee Y) = \partial_K(X) \vee \partial_K(Y) \\
&\partial_K(\alpha \succ x) = 0_K(\alpha) \succ \partial_K(x) \\
&\partial_K(\alpha \succ \surd) = 0_K(\alpha) \succ \surd
\end{aligned}
$$

TABLE 14. Parallel composition and encapsulation.

6.5 EXTERNAL ACTION OPERATOR.
Next, we look at the **external action operator**. X/a denotes X, after a has been executed. Such an operator can only be described in the present setting. In table 15, we have $a \in A$; $b \in A_{0\delta}$; $\alpha \in H$; $x,y \in P$; $X,Y \in HP$.

$$
\begin{aligned}
&\varnothing/a = \varnothing \\
&(X \vee Y)/a = X/a \vee Y/a \\
&(\alpha \succ \surd)/a = \varnothing \\
&(\alpha \succ a)/a = \alpha a \succ \surd \\
&(\alpha \succ b)/a = \varnothing \qquad \text{if } b \neq a \\
&(\alpha \succ ax)/a = \alpha a \succ x \\
&(\alpha \succ bx)/a = \varnothing \qquad \text{if } b \neq a \\
&(\alpha \succ (x + y))/a = (\alpha \succ x)/a \vee (\alpha \succ y)/a
\end{aligned}
$$

TABLE 15. External action operator.

The external action operator can be extended to histories, and X/α ($\alpha \in A^*$) denotes X after the sequence of actions α has been executed. It is straightforward to give axioms for this extension.

6.6 LEMMA.
1. $(X \| Y)/a = (X/a) \| Y \vee X \| (Y/a) \vee \bigvee_{b,c \text{ with } b \mid c=a} (X/b) \| (Y/c)$;
2. $\partial_K(X)/a = \partial_K(X/a)$.

6.7 COMMENT
The difference in intuition between + and $\vee$ in HP is as follows: $X \vee Y$ describes lack of information (non-determinacy) about the past, wheras + represents lack of information (option to choose) about the future.

In addition, it should be noticed that it is difficult to define the operator $\vee$ on P. Indeed, one would expect $(x \vee y) + z = (x + z) \vee (y + z)$ in that case. But then: $a \vee b = (a \vee b) + (a \vee b) = (a + a) \vee (b + a) \vee (a + b) \vee (b + b) = a \vee (a + b) \vee b$, which is not a plausible equation. Introducing the sort HP avoids this difficulty.

6.8 EXAMPLE.
Suppose we have two connected one-bit buffers, i.e. we have:

$$B_{1,2} = \sum_{d\in D} r_1(d)\cdot s_2(d)\cdot B_{1,2}$$

$$B_{2,3} = \sum_{d\in D} r_2(d)\cdot s_3(d)\cdot B_{2,3}.$$

As usual, we have the communication $r_2(d) \mid s_2(d) = c_2(d)$ and $H = \{r_2(d), s_2(d) : d \in D\}$. Then

$$(\varepsilon \gg \partial_H(B_{1,2} \parallel B_{2,3})) / r_1(d)c_2(d) = \partial_H(r_1(d)s_2(d) \gg B_{1,2} \parallel r_2(d) \gg s_3(d)\cdot B_{2,3}).$$

This system is determinate, in the sense that there is never any uncertainty which system we are dealing with (no disjunctions occur).

6.9 EXAMPLE.
Now let us consider two connected buffers, that may lose data, i.e.:

$$B^*_{1,2} = \sum_{d\in D} r_1(d)\cdot(i\cdot s_2(d) + i)\cdot B^*_{1,2},$$

$$B^*_{2,3} = \sum_{d\in D} r_2(d)\cdot(i\cdot s_3(d) + i)\cdot B^*_{2,3}.$$

With communication and encapsulation as above we get

$$(\varepsilon \gg \partial_H(B^*_{1,2} \parallel B^*_{2,3})) / r_1(d)\cdot i\cdot c_2(d)\cdot r_1(e)\cdot i\cdot s_3(d) =$$

$$\partial_H(r_1(d) \gg (i\cdot s_2(d) + i)\cdot B^*_{1,2} \parallel \varepsilon \gg B^*_{2,3}) / i\cdot c_2(d)\cdot r_1(e)\cdot i\cdot s_3(d) =$$

$$\partial_H(\{r_1(d)\cdot i \gg s_2(d)\cdot B^*_{1,2}\} \vee \{r_1(d)\cdot i \gg B^*_{1,2}\} \parallel \varepsilon \gg B^*_{2,3}) / c_2(d)\cdot r_1(e)\cdot i\cdot s_3(d) =$$

$$\partial_H(r_1(d)\cdot i\cdot s_2(d) \gg B^*_{1,2} \parallel r_2(d) \gg (i\cdot s_3(d) + i)\cdot B^*_{2,3}) / r_1(e)\cdot i\cdot s_3(d) =$$

$$\partial_H(r_1(d)\cdot i\cdot s_2(d)\cdot r_1(e) \gg (i\cdot s_2(e) + i)\cdot B^*_{1,2} \parallel r_2(d) \gg (i\cdot s_3(d) + i)\cdot B^*_{2,3}) / i\cdot s_3(d) =$$

$$\partial_H(\{r_1(d)\cdot i\cdot s_2(d)\cdot r_1(e)\cdot i \gg s_2(e)\cdot B^*_{1,2}\} \vee \{r_1(d)\cdot i\cdot s_2(d)\cdot r_1(e)\cdot i \gg B^*_{1,2}\} \parallel r_2(d) \gg (i\cdot s_3(d) + i)\cdot B^*_{2,3})$$

$$\vee \partial_H(r_1(d)\cdot i\cdot s_2(d)\cdot r_1(e) \gg (i\cdot s_2(e) + i)\cdot B^*_{1,2} \parallel \{r_2(d)\cdot i \gg s_3(d)\cdot B^*_{2,3}\} \vee \{r_2(d)\cdot i \gg B^*_{2,3}\})$$

$$/ s_3(d) =$$

$$\partial_H(r_1(d)\cdot i\cdot s_2(d)\cdot r_1(e) \gg (i\cdot s_2(e) + i)\cdot B^*_{1,2} \parallel r_2(d)\cdot i\cdot s_3(d) \gg B^*_{2,3}).$$

This system is not determinate, since disjunctions do occur in the calculations, but nevertheless, this system is eventually deterministic. What this means, is defined next.

6.10 DEFINITIONS.
1. A system P in HP is called **determinate** if it is of the form $\alpha \gg x$ or $\alpha \gg \surd$ (for $\alpha \in A^*$, $x \in P$).

2. A system P in HP is **deterministic** if for all $\alpha \in A^*$, P/α is determinate.
3. A system P in HP is **eventually deterministic** if for each sequence of actions $a_0, a_1, a_2, \ldots$ from A, there are infinitely many n such that $P/a_0 \ldots a_n$ is determinate.

We see that the system in example 6.8 is not deterministic, but is eventually deterministic. Also, the systems $\varepsilon \blacktriangleright P$ and $\varepsilon \blacktriangleright Q$, with $P = a \cdot b \cdot d^\omega + a \cdot c \cdot d^\omega$ and $Q = a \cdot b \cdot d \cdot Q + a \cdot c \cdot d \cdot Q$, are examples of eventually deterministic systems (that are not deterministic).

7. CONCLUDING REMARKS.

7.1. RELATION BETWEEN ACP_0 AND ACP.

ACP_0 is a generalization of ACP. The mechanism of generalization can be compared to the case in which one takes the positive rational numbers which combine a multiplicative group structure and an additive semi-group and adds 0 to it. One adds a single object and several laws become invalid. (Let us assume that one has defined $p / 0$ as 1 in order to avoid partial functions.)

7.2 EFFECTIVE COMPUTABILITY.

The reason not to have 0 as a member of the core system ACP is that it is not effectively computable. That is to say that if we have a finite guarded recursive specification of a process X over ACP_0, it may be impossible to compute its finite projections π_n in a uniform way. The central axiom systems BPA, PA, ACP and its extensions in concrete process algebra all have the property that finite projections of finitely recursively specified processes can be determined in a uniform mechanical way. This simply means that ACP and its extensions in concrete process algebra can be viewed as an executable programming language. This is why we propose not to consider 0 a part of concrete process algebra (just as the empty step ε and the silent step τ are not part of concrete process algebra).

7.3 RELATED WORK.

In MILNER [89], a process 0 is introduced that replaces the constant NIL of CCS of MILNER [80]. This is just a notational matter and does not introduce semantic modifications as such. Nevertheless the notation differs from ours considerably in the sense that Milner's 0 definitely corresponds to our constant δ and not to our constant 0. Similarly the constant STOP of TCSP of OLDEROG & HOARE [86] corresponds to δ and not to (our) 0. We use δ because that makes the notation consistent with other papers about ACP (e.g. BERGSTRA & KLOP [89]). Because 0 is more truly a zero in process algebra than δ we preferred not to adapt our notation to the notation of Milner.

Of course Milner's restriction must be compared to our encapsulation operator and not to a substitution of (our) 0 for some actions. Thus $x / \{a, b\}$ in the notation of MILNER [80] corresponds to $\partial_{\{a, b\}}(x)$ in the case of ACP.

CSP of HOARE [85] contains two forms of alternative composition: internal and external choice. This distinction is quite a different one; both of Hoare's operators are close to our $+$ and unrelated to our $\vee$.

ACKNOWLEDGEMENT.

Hans Mulder and Sjouke Mauw have contributed to this paper through several critical remarks including the view that an undisputable zero in ACP must satisfy the law $x \cdot 0 = 0$ and not just $a \cdot 0 = 0$ for atomic actions a. In addition, the second author acknowledges extended discussions with C.A.R. Hoare about various aspects of this paper.

REFERENCES.

J.C.M. BAETEN & J.A. BERGSTRA [88], *Global renaming operators in concrete process algebra,* I&C 78, 1988, pp. 205-245.

J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP [86], *Syntax and defining equations for an interrupt mechanism in process algebra,* Fund. Inf. IX, 1986, pp. 127-168.

J.A. BERGSTRA & J.W. KLOP [84], *Process algebra for synchronous communication,* I&C 60, 1984, pp. 109-137.

J.A. BERGSTRA & J.W. KLOP [85], *Algebra of communicating processes with abstraction,* TCS 37, 1985, pp. 77-121.

J.A. BERGSTRA & J.W. KLOP [86], *Process algebra: specification and verification in bisimulation semantics,* in: Math. & Comp. Sci. II (M. Hazewinkel, J.K. Lenstra & L.G.L.T. Meertens, eds.), CWI Monograph 4, North-Holland, Amsterdam, 1986, pp. 61-94.

J.A. BERGSTRA & J.W. KLOP [89], *Process theory based on bisimulation semantics,* in: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency (J.W. de Bakker, W.-P. de Roever & G. Rozenberg, eds.), Springer LNCS 354, 1989, pp. 50-122.

R.J. VAN GLABBEEK [87], *Bounded nondeterminism and the approximation induction principle in process algebra,* in: Proc. STACS 87 (F.J. Brandenburg, G. Vidal-Naquet & M. Wirsing, eds.), Springer LNCS 247, 1987, pp. 336-347.

C.A.R. HOARE [85], *Communicating sequential processes,* Prentice Hall International, 1985.

R. MILNER [80], *A calculus for communicating systems,* Springer LNCS 92, 1980.

R. MILNER [89], *Communication and concurrency,* Prentice Hall International, 1989.

E.-R. OLDEROG & C.A.R. HOARE [86], *Specification-oriented semantics for communicating processes,* Acta Informatica 23, 1986, pp. 9-66.

A. PONSE & F.-J. DE VRIES [89], *Strong completeness for Hoare logics of recursive processes: an infinitary approach,* report CS-R8957, Centre for Math. & Comp. Sci., Amsterdam 1989.

M. REM [87], *Trace theory and systolic computations,* in: Proc. PARLE Vol. I (J.W. de Bakker, A.J. Nijman & P.C. Treleaven, eds.), Springer LNCS 258, 1987, pp. 14-33.