Process Algebra with a Zero Object

J.C.M. Baeten
Department 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.

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.
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.
Note: this article is a revision of BAETEN & BERGSTRA [90], but leaving out the material in section 6.

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. The purpose of this paper is to indtroduce a constant 0 in process algebra and discuss its properties.

We will call 0 *predictable failure*. A predictable failure differs from deadlock (inaction) in the sense that a system will actively try to avoid it. 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, its 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 (or better) 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 and specification, rather than in implementation.

2. AXIOMATIZATION

2.1. BASIC PROCESS ALGEBRA WITH ZERO

We start out from the theory of Basic Process Algebra as described in BERGSTRA & KLOP [84a, 85, 86]. For a recent survey article see BERGSTRA & KLOP [89].

We have a set of **atomic actions**, A. Each atomic action is a constant of the sort P, the sort of processes that the theory is about. Then, we have two binary operators on P: + 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. In table 1, x,y,z are arbitrary elements of P.

Then we add the **zero** constant, obeying the next three axioms (Z1-3). Usually, we also have the constant δ in the theory, called **deadlock** or **inaction**. The two axioms for inaction need conditions, in order to avoid clashes with the zero axioms ($A6^0$, $A7^0$). The conditions use a predicate on processes $\neq 0$, that determines whether or not a term can be proved equal to the zero process. This predicate is axiomatized in the last four axioms of table 1. There, we have $a \in A$. We put $A_\delta = A \cup \{\delta\}$, $A_{0\delta} = A \cup \{0,\delta\}$, $BPA_0 = BPA + Z1\text{-}3$, $BPA_{0\delta} = BPA_0 + A6^0, A7^0 + NZ1\text{-}4$.

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
$x \neq 0 \Rightarrow x + \delta = x$	$A6^0$
$x \neq 0 \Rightarrow \delta\cdot x = \delta$	$A7^0$
$\delta \neq 0$	NZ1
$a \neq 0$	NZ2
$x \neq 0, y \neq 0 \Rightarrow x\cdot y \neq 0$	NZ3
$x \neq 0 \Rightarrow x + y \neq 0$	NZ4

TABLE 1. $BPA_{0\delta}$

The difference between 0 and δ requires further comments:

i. δ is an inactive process, if a system reduces to δ it deadlocks. Of course the mere occurrence of δ in a process expression (like $a + b(\delta + c)$) need not indicate a deadlock.

Compare this to the expression $e = 2 + 0 \cdot (5 + 8)$; in no way the occurrence of 0 in e implies that e vanishes.
ii. $x + \delta = x$ is a progress rule: the system eventually discovers that δ is no option and proceeds with x (if possible).
iii. $x \cdot 0 = 0$ is a liveness rule: the system must, after completion of x, eventually execute 0, i.e. assert false. This form of liveness is not required for δ.
iv. $0 + \delta = \delta$ because δ is 'better than' 0.
v. $0 \cdot x = 0$ and $\delta \cdot x = \delta$ are explained by the non-executability of 0, resp. δ. Note that we have in particular $\delta \cdot 0 = 0$. Unfortunately, the explanation of this in philosophical terms is shallow.

2.2. PROJECTION

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

$\pi_n(0) = 0$
$\pi_n(\delta) = \delta$
$\pi_n(a) = a$
$x \neq 0 \Rightarrow \pi_1(ax) = a$
$\pi_{n+1}(ax) = a \cdot \pi_n(x)$
$\pi_n(x + y) = \pi_n(x) + \pi_n(y)$

TABLE 2. Projection

2.3 INFINITARY RULES

The axioms given above constitute a complete theory for finite processes, processes represented by closed terms, i.e. equality on finite processes (in a graph model to be presented further on) coincides with derivability from the axioms. When we are dealing with infinite processes however, processes specified by means of recursive equations, we need additional proof principles. In this paper, we will discuss 4 such principles.

First, we consider the *Approximation Induction Principle* (AIP). AIP can be maintained in the present setting as in BERGSTRA & KLOP [86]. Roughly, this proof rule states that two processes should be identified if all their projections are equal. More explicitly:

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

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).

Next, we consider two principles that deal with solutions of recursive equations. The *Recursive Definition Principle* (RDP) states that every recursive specification has at least one solution, and the *Recursive Specification Principle* (RSP) states that every guarded recursive specification has at most one solution. Here, guarded roughly means that every variable occurring in the right-hand side of an equation must be preceded by an atomic action. For more details and formal definitions, see BERGSTRA & KLOP [86].

Now let us consider these rules in the present setting. Notice that the guarded equation $x = a\cdot x$ has two solutions: 0 and a^{ω} (the process that indefinitely performs a). 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 $X = (x_1, \ldots, x_k)$ and $Y = (y_1, \ldots, y_k)$ 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.
The last infinitary proof rule that we will consider is the *Limit Rule*. We describe this rule in 4.4.

2.4 Renaming into zero

0_δ is an operator that substitutes 0 for δ. We will call this operator **deadlock prevention.** Let $a \in A$.

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

TABLE 3. Deadlock prevention

An example: $0\delta(ab + c(de\delta + a\delta)) = ab$.

Note that the equation $0_\delta(x\cdot y) = 0_\delta(x)\cdot 0_\delta(y)$ 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 $y = \delta$, then $x\cdot y = x$ because of AIP, hence $x = 0_\delta(x) = 0_\delta(x\cdot y) = 0_\delta(x)\cdot 0_\delta(y) = 0_\delta(x)\cdot 0 = 0$, which contradicts the assumptions on x.

Also note that it is incorrect to rename an atomic action a into 0 by an operator 0_a, for that leads to $0 = 0_a(a) = 0_a(a + \delta) = 0_a(a) + 0_a(\delta) = 0 + \delta = \delta$.

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 4 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]. In this definition, we have to pay special attention to the separate status of the process 0. Note that we have to define the rules on congruence classes of process expressions, since we need the predicate $\neq 0$ in the definition. We cannot define action rules on terms in the format of GROOTE & VAANDRAGER [89] or GROOTE [90].

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

TABLE 4. 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. To see this notice that for every recursively enumerable set $W_e \subseteq \mathbb{N}$ with recursive index e and every $n \in \mathbb{N}$ a guarded recursive specification over ACP can be uniformly computed with solution $p(e,n)$ such that:

$$n \in W_e \Leftrightarrow \text{for some } k \;\; p(e,n) = i^k{\cdot}\delta$$
$$n \notin W_e \Leftrightarrow p(e,n) = i^\omega.$$

This construction can be found in BERGSTRA & KLOP [84b]. It follows that:

$$n \in W_e \Leftrightarrow 0_\delta(p(e,n)) = 0,$$

hence it is undecidable whether $X = 0$ for recursively specified X in general.

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, 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 $[\![\delta]\!]$);
- 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.

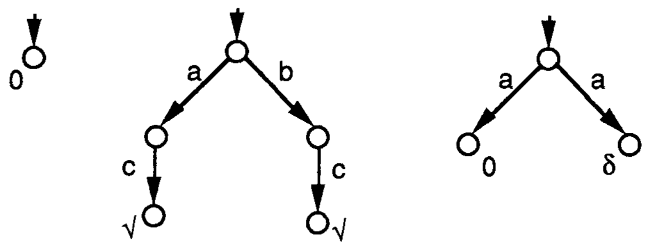

FIGURE 1

We see some examples in fig. 1 above. We see the trees that represent the terms 0, $(a + b)\cdot c$, $a\cdot\delta + a\cdot 0$. Recursively specified processes will in general have infinite trees.

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**, $g \leftrightarrow h$, 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 $\mathbb{T}$.

In order to prove that $\mathbb{T}/\leftrightarrow$ is a good model for the theory $BPA_{0\delta}$, we first need a couple of lemmas.

3.2.3 LEMMA

Let t be a closed term over $BPA_{0\delta}$ such that $BPA_{0\delta} \nvdash t \neq 0$. Then $BPA_{0\delta} \vdash t = 0$.

PROOF: By induction on the structure of t. If t is a constant, this follows immediately from the axioms in table 1. If t is a sum, say $t = t_1 + t_2$, and $BPA_{0\delta} \nvdash t \neq 0$, it follows from axiom NZ4 that $BPA_{0\delta} \nvdash t_1 \neq 0$ and $BPA_{0\delta} \nvdash t_2 \neq 0$. By induction hypothesis, $BPA_{0\delta} \vdash t_1 = 0$ and $BPA_{0\delta} \vdash t_2 = 0$ and consequently $BPA_{0\delta} \vdash t = 0$. Finally, if t is a product, say $t = t_1\cdot t_2$, and $BPA_{0\delta} \nvdash t \neq 0$, it follows from axiom NZ3 that either $BPA_{0\delta} \nvdash t_1 \neq 0$ or $BPA_{0\delta} \nvdash t_2 \neq 0$. By induction hypothesis, we obtain either $BPA_{0\delta} \vdash t_1 = 0$ or $BPA_{0\delta} \vdash t_2 = 0$ In the first case, use Z3 and in the second case Z1 to obtain $BPA_{0\delta} \vdash t = 0$.

3.2.4 LEMMA

Let t be a closed term over $BPA_{0\delta}$ such that $BPA_{0\delta} \vdash t \neq 0$. Then t can be written without 0, i.e. there is a closed term s over $BPA_{0\delta}$ that does not contain 0, such that $BPA_{0\delta} \vdash t = s$.

PROOF: By induction on the structure of t. If t is a constant, this follows directly from the axioms in table 1. If t has the form of a product $t_1\cdot t_2$, it follows by the previous lemma that $t_1 \neq 0$ and $t_2 \neq 0$. Then use induction hypothesis. Finally, if t has the form of a sum $t_1 + t_2$, it is by the previous lemma enough to consider the following four cases.

<u>Case 1:</u> $t_1 \neq 0$, $t_2 \neq 0$. Use induction hypothesis.

Case 2: $t_1 = 0, t_2 = 0$. It follows that $t = 0$, contradiction.
Case 3: $t_1 = 0, t_2 \neq 0$. It follows that $t = t_2$, and we can use the induction hypothesis for t_2.
Case 4: $t_1 \neq 0, t_2 = 0$. Just like case 3.

3.2.3 THEOREM

$BPA_{0\delta}$ is a sound and complete axiomatization of the model $\mathbb{T}/\leftrightarrow$ (for closed terms).

PROOF: First note that 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.

Now soundness can be proved by inspection of the model. To prove completeness, consider two closed terms t,s over $BPA_{0\delta}$ and suppose $\mathbb{T}/\leftrightarrow \vdash t=s$. We use case distinction.
Case 1: $t \neq 0$, $s \neq 0$. By the previous lemma, t and s can be written as t^*, s^* without 0. By soundness $\mathbb{T}/\leftrightarrow \vdash t^* = s^*$. By the completeness theorem for BPA_δ, $BPA_\delta \vdash t^* = s^*$.
Case 2: $t = 0$, $s = 0$. Immediate.
Case 3: $t = 0$, $s \neq 0$. Write s as s^* without 0. The definition of $\leftrightarrow$ shows that $t \leftrightarrow s^*$ cannot hold, contradiction.
Case 4: $t \neq 0$, $s = 0$. Just like case 3.

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 5, 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 5. Renaming

It is straightforward to define the renaming operators on the graph model of 3.2. 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 $a \notin H$.

The composition of an encapsulation operator with the deadlock prevention operator ($0_\delta \circ \partial_H$) will prevent any action of H from occurring. We will see applications of this in the sequel.

4.2 PARALLEL COMPOSITION

We can extend the theory $BPA_{0\delta}$ with parallel composition as in the theory ACP of BERGSTRA & KLOP [84a]. In order to axiomatize the parallel operator $\|$ **(merge)**, we need two auxiliary operators $⌊\!⌊$ **(left-merge)** and $\mid$ **(communication merge)**. The theory is parametrized by a **communication function** $\mid$, 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 6 below. In table 6, $a,b,c \in A_{0\delta}$.

$a \mid b = b \mid a$	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
$x \parallel y = x ⌊\!⌊ y + y ⌊\!⌊ x + x \mid y$	CM1
$a ⌊\!⌊ x = a \cdot x$	CM2
$ax ⌊\!⌊ y = a \cdot (x \parallel y)$	CM3
$(x + y) ⌊\!⌊ z = x ⌊\!⌊ z + y ⌊\!⌊ z$	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
$\partial_H(a) = a \quad \text{if } a \notin H$	D1
$\partial_H(a) = \delta \quad \text{if } a \in H$	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 6. ACP_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 DISCUSSION

Note that we can derive from ACP_0 that for all finite closed terms x we have

$0 \| x = 0 \llcorner\!\llcorner x = x \llcorner\!\llcorner 0 = x \mid 0 = 0 \mid x = 0.$

Now let us consider this for recursively defined processes. Look at the process B that is the non-zero solution of $B = b \cdot B$. We calculate: $0 \| B = 0 \llcorner\!\llcorner B + B \llcorner\!\llcorner 0 + 0 \mid B = 0 \cdot B + bB \llcorner\!\llcorner 0 + bB \mid 0 = 0 + b \cdot (B \| 0) + (b \mid 0) \cdot B = b \cdot (B \| 0) = b \cdot b \cdot (0 \| B)$.

We also have $B = b \cdot b \cdot B$, so by RSP^0 we have either $0 \| B = B$ or $0 \| B = 0$.

Both options are consistent. Our choice is to put $0 \| B = 0$ and in general $0 \| x = 0$. We can motivate this if we use the *Limit Rule* of BAETEN & BERGSTRA [88]. We describe this rule next.

4.4 LIMIT RULE

Let $FCPE_0$ be the class of finite closed process expressions over ACP_0. Let $p(x_1,...,x_n) = q(x_1,...,x_n)$ be an equation over ACP0. The limit rule (LR) is as follows:

$$\frac{\text{for all } t_1,...,t_n \in FCPE_0 \quad p(t_1,...,t_n) = q(t_1,...,t_n)}{p(x_1,...,x_n) = q(x_1,...,x_n)} \quad \text{LR.}$$

The identity $0 \| x = 0$ follows from LR, because, as remarked above, $0 \| t = 0$ holds for all $t \in FCPE_0$.

4.5 PROJECTION AXIOMS

The limit rule has other applications of equal importance. Consider the projection operator π_n. The following hold for the projection operators: for $x,y \in FCPE_0$

$$\pi_n(x \| y) = \pi_n(\pi_n(x) \| \pi_n(y))$$
$$\pi_n(x \cdot y) = \pi_n(x) \cdot \pi_n(y)$$
$$\pi_n(\partial_H(x)) = \partial_H(\pi_n(x))$$
$$\pi_n(x \llcorner\!\llcorner y) = \pi_n(\pi_n(x) \llcorner\!\llcorner \pi_n(y))$$
$$\pi_n(x \mid y) = \pi_n(\pi_n(x) \mid \pi_n(y)).$$

We call this set of equations EP. It follows from the limit rule that these identities are valid for all processes. Now consider once more the process B from 4.3. Then $\pi_n(0 \| B) = \pi_n(\pi_n(0) \| \pi_n(B)) = \pi_n(0 \| b^n) = 0$. This holds for all n and so by AIP $0 \| B = 0$. More generally, using AIP and EP we can prove that an identity $p(x_1,...,x_n) = q(x_1,...,x_n)$ holds for all recursively specified processes as soon as it holds for all finite processes. The proof proceeds just like in BAETEN & VAN GLABBEEK [87].

Summarizing the discussion, we have that our model satisfied LR, AIP, EP. The logical relationships are: LR $\vdash$ EP, and AIP + EP $\vdash$ LR for recursively specified processes only.

4.6 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 write $a(s)$ for $action(a,s)$ and $s(a)$ for $effect(a,s)$ (a an action, s a state). When an action a is to be executed, $a(s)$ gives the resulting action, and $s(a)$ the resulting state. We will not allow that $a(s) = 0$, further one must assume that $s(0) = s$ and $0(s) = 0$, and $s(\delta) = s$, $\delta(s) = \delta$. Then the state operator works just as well as in the case of ACP. The axioms are displayed in table 7. We have $a \in A_\delta$. It is straightforward to define the state operator on the graph model of 3.2.

$$\lambda_s(0) = 0$$
$$\lambda_s(a) = a(s)$$
$$\lambda_s(a \cdot x) = a(s) \cdot \lambda_{s(a)}(x)$$
$$\lambda_s(x + y) = \lambda_s(x) + \lambda_s(y)$$

TABLE 7. State operator

4.7. 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 $\triangleleft$ (**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 8 gives axioms on top of the axioms of ACP_0. We have $a,b \in A_\delta$.

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

TABLE 8. Priority operator

The priority operator can be defined on the graph model of 3.2 similarly as in BAETEN, BERGSTRA & KLOP [87]: we prune away every branch that splits off at a node where there is a 'brother' edge with higher priority, that leads to a non-zero node.

4.8. 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 other abstract semantic models. In table 9 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 $0 \mathbin{|\!\lfloor}_B x = x \mathbin{|\!\lfloor}_B 0 = 0 \mid_B x = x \mid_B 0 = 0$.

$x \parallel_B y = x \mathbin{|\!\lfloor}_B y + y \mathbin{|\!\lfloor}_B x + x \mid_B y$	
$a \mathbin{|\!\lfloor}_B x = a \cdot x$	if $a \notin B$
$a \mathbin{|\!\lfloor}_B x = \delta$	if $a \in B$
$(a \cdot x) \mathbin{|\!\lfloor}_B y = a \cdot (x \parallel_B y)$	if $a \notin B$
$(a \cdot x) \mathbin{|\!\lfloor}_B y = \delta$	if $a \in B$
$(x + y) \mathbin{|\!\lfloor}_B z = x \mathbin{|\!\lfloor}_B z + y \mathbin{|\!\lfloor}_B z$	
$a \mid_B b = \delta$	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 9. 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:

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

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

$$H(b) = b \cdot done(b) + not(b)$$

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 $instr(B) \cup done(B) \cup not(B)$ 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:

$$SIM = 0_\delta \circ \partial_{\{e\}}(S).$$

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:

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

The system $C\text{-}SIM$ allows $not(b)$ if it will not allow b. $C\text{-}SIM$ allows b if after b, S can proceed with at least one infinite trace of actions not involving e. Of course, $C\text{-}SIM$ 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:

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

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

5.2. Specifying a path through a combinatorial explosion

It is well-known that any NP-complete problem can be solved non-deterministically in polynomial time. Essentially, this is done by non-deterministically guessing a value at each step. We formalize this as follows. Let P be a computable predicate on sequences of length k of natural numbers in the range $\{1,...,n\}$. The set of states is $S = \{\langle j,\sigma\rangle : 1 \le j \le k,\ \sigma$ a sequence of length j from $\{1,...,n\}\}$. We have atomic actions $guess(i)$ (for $1 \le i \le n$). The action and effect functions are as follows:

- $guess(i)(\langle j,\sigma\rangle) = skip$ if $j < k$
- $guess(i)(\langle k,\sigma\rangle) = exit$ if $P(\sigma)$
- $guess(i)(\langle k,\sigma\rangle) = \delta$ if $\neg P(\sigma)$
- $\langle j,\sigma\rangle(guess(i)) = \langle j+1, \sigma{*}i\rangle$ if $j < k$
- $\langle k,\sigma\rangle(guess(i)) = \langle k, \sigma\rangle$
- all other actions a are inert, i.e. $a(s) = a$ and $s(a) = s$ for all states s.

Now define the process Q,R by:

$$Q = \lambda_{\langle 0,\varepsilon\rangle}\Big(\big(\sum_{i=1}^{n} guess(i)\big)^{k+1}\Big), \quad R = 0_\delta(Q).$$

R will equal 0 iff no sequence σ with $P(\sigma)$ exists; otherwise, a sequence of length k will be accepted by R such that P holds.

5.3. 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 $\partial_{\{tick\}}(P \parallel TL)$. 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
$(p, v, c)(st) = (p, v, c)$	$st(p, v, c) = st$
$(p, v, c)(la) = (p, v-1, c)$	$la(p, v, c) = la$
$(p, v, c)(ra) = (p, v+1, c)$	$ra(p, v, c) = ra$
$(p, v, c)(red) = (p, v, red)$	$red(p, v, c) = red$
$(p, v, c)(green) = (p, v, green)$	$green(p, v, c) = green$
$(p, v, c)(t) = (p + v, v, c)$	$t(p, v, c) = \delta$ if $c = red$ & $v>0$ & $p \leq 10 \leq p+v$
	$t(p, v, c) = t$ 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 $PTLC = 0_\delta(PTL)$ 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:

$B = \{la, ra, st\}$
$S = P$
$SIM = PTLC$
$C\text{-}SIM = \theta_<(C \parallel_B SIM)$
$C\text{-}S = C \parallel_B P$.

6. CONCLUDING REMARKS

6.1. 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.)

6.2 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).

6.3 IMPLEMENTATION

Implementation of a recursive ACP0 specification first of all involves an elimination of 0. Now it must be noticed that interesting use of 0 happens just in those cases where elimination of 0 is possible but at a very high cost. In the examples 5.1, 5.2 and 5.3 this elimination is possible if the state operator is allowed. Elimination of 0 is also possible if in addition to ACP abstraction (τ_I) may be used.

In all of these examples it is not known to us whether an equivalent specification in ACP can be given (i.e. whether the state operator or abstraction operator are necessary strengthenings for an elimination of 0 and 0_δ).

6.4 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.

ACKNOWLEDGEMENTS

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. We thank the referees for their careful review and many helpful comments.

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 [90], *Process algebra with zero object and non-determinacy,* report P9002, Programming Research Group, University of Amsterdam 1990.

J.C.M. BAETEN & R.J. VAN GLABBEEK [87], *Merge and termination in process algebra,* in: Proc. 7th FST&TCS, Pune (K.V. Nori, ed.), Springer LNCS 287, 1987, pp. 153-172.

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 [84a], *Process algebra for synchronous communication,* I&C 60, 1984, pp. 109-137.

J.A. BERGSTRA & J.W. KLOP [84b], *The algebra of recursively defined processes and the algebra of regular processes,* in: Proc. 11th ICALP, Antwerpen (J. Paredaens, ed.), Springer LNCS 172, 1984, pp. 82-95.

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.

J.F. GROOTE & F.W. VAANDRAGER [89], *Structured operational semantics and bisimulation as a congruence,* extended abstract in: Proc. ICALP 89, Stresa (G. Ausiello, M. Dezani-Ciancaglini & S. Ronchi Della Rocca, eds.), Springer LNCS 372, 1989, pp. 423-438. Full version to appear in I&C.

J.F. GROOTE [90], *Transition system specifications with negative premises,* report CS-R8950, Centre for Math. & Comp. Sci. 1990. To appear in Proc. CONCUR'90, Springer LNCS.

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.