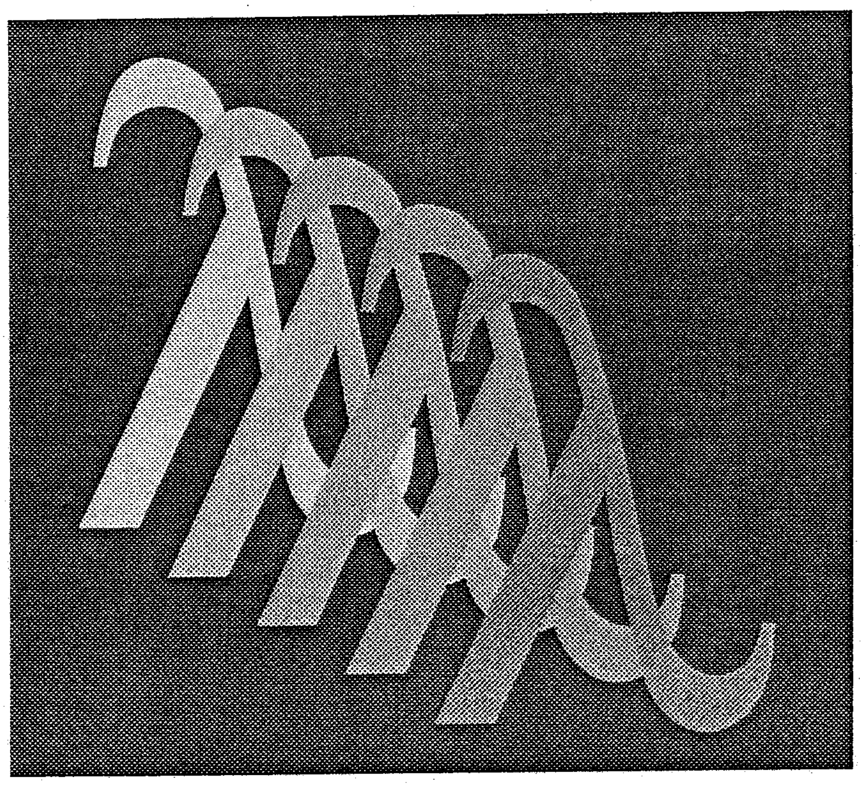
Recursive process definitions with the state operator

University of Amsterdam
Programming Research Group



Recursive Process Definitions with the State Operator

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

Report P8802

January 1988

University of Amsterdam
Department of Mathematics and Computer Science
Programming Research Group

Recursive process definitions with the state operator

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

Report P8802

January 1988

J.C.M. Baeten

Programming Research Group
Department of Mathematics and Computer Science
University of Amsterdam

NIKHEF H221b
Kruislaan 409
1098 SJ Amsterdam

P.O. Box 41882
1009 DB Amsterdam
The Netherlands

tel. +31 20 5926056

J.A. Bergstra

Programming Research Group
Department of Mathematics and Computer Science
University of Amsterdam

NIKHEF H221a
Kruislaan 409
1098 SJ Amsterdam

P.O. Box 41882
1009 DB Amsterdam

tel. +31 20 5925077

Centrale Interfaculteit
State University of Utrecht

Transitorium II kr. 1026
Heidelberglaan 2
3584 CS Utrecht

tel. +31 30 532761

Universiteit van Amsterdam, 1988

Recursive process definitions with the state operator

J.C.M. Baeten
Programming Research Group, University of Amsterdam,
P.O.Box 41882, 1009 DB 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.

We investigate the defining power of finite recursive specifications over the theory with + (non-determinate choice) and · (sequential composition) and λ (the state operator) over a finite set of states, and find that it is greater than that of the same theory without state operator. Thus, adding the state operator is an essential extension of BPA (the theory of processes over +,·). On the other hand, applying the state operator to a regular process again gives a regular process. As a limiting result in the other direction, we find that not all PA-processes (where also parallel composition ‖ is present) can be defined over BPA plus state operator.

1980 Mathematical Subject Classification (1985 revision): 68Q10, 68Q70, 68Q40.
1987 CR Categories: F.1.2, F.4.3, F.4.2, F.3.1.
Key words & Phrases: process algebra, state operator, defining power, recursive definition.
Note: Partial support received from the European Communities under ESPRIT contract 432, An Integrated Formal Approach to Industrial Software Development (Meteor).

1. INTRODUCTION.

The theory BPA (Basic Process Algebra) is the starting point for a whole range of theories for concurrent communicating processes (see e.g. BERGSTRA & KLOP [BK2]), that can be classified as an algebraic and axiomatic approach to concurrency (in the vein of CCS, see MILNER [M] or CSP, see HOARE [H]).

BPA has two binary operators: + is alternative composition (non-deterministic choice, as in CCS), and · is sequential composition (as ; in CSP), and consists of just five simple axioms (see below). We add the constant δ for deadlock, with two extra axioms. In addition, we allow systems of recursive equations over BPA_δ (compare the μ-operator in CCS or CSP). The defining power of such recursive specifications was studied in BERGSTRA & KLOP [BK1]. There, it was found that a wider class of processes can be defined than the class of regular processes, and that the addition of the parallel operator ‖ (merge, giving the theory PA) increases the defining power further.

The state operator λ was introduced in BAETEN & BERGSTRA [BB]. It can be used to describe actions that have a side effect on a state space, and showed itself useful in a range of applications, e.g. for the translation of computer programs into process algebra (for instance, see VAANDRAGER [V]). Now the question arises if the defining power of BPA is increased by the addition of the state operator. Of course, we have to limit ourselves to a finite state space, for otherwise any process becomes definable (see the example of the queue in [BB]). In this paper, we answer this question positively.

We obtain the theory of regular processes (finite automata) if we limit ourselves to *linear* specifications over BPA. We show that applying the state operator to a regular process again yields a regular process. On the other hand, if we are allowed to use the state operator in the recursion, then all processes that are definable over BPA_δ, are definable by a linear specification over $BPA_\delta + \lambda$. Even some processes that are not definable over BPA_δ, are definable by a linear specification over $BPA_\delta + \lambda$. On the other hand, not all PA-definable processes are definable over $BPA_\delta + \lambda$.

2. STATING THE PROBLEM.

2.1 BASIC PROCESS ALGEBRA.

The axiom system BPA consists of the axioms in table 1 below. The signature of BPA consists of a set A = {a,b,c,...} of constants, called *atomic actions*, and the operators + (alternative composition) and · (sequential composition. Often the dot · and parentheses will often be suppressed. · binds stronger than +. By a *process* we mean an element of some algebra satisfying the axioms of BPA; the x,y,z in table 1 vary over processes. Such an algebra is a *process algebra* (for BPA), e.g. the initial algebra of BPA is one.

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5

Table 1. BPA.

2.2 EXAMPLE.

a(b + c)d denotes the process whose first action is a followed by a choice between b and c and concluding with d. By axioms A1 and A4 we see that a(b + c)d = a(cd + bd). Note, however, that BPA does not enable us to prove that a(cd + bd) = acd + abd.

2.3 DEADLOCK.

We distinguish one special constant in A, namely δ. We use this constant to denote deadlock, reached when no action is possible anymore, the absence of an alternative to proceed. The constant $\delta \in A$ has two special axioms, displayed in table 2 below. We denote the theory BPA + δ, with axioms A1-7, by BPA_δ.

$x + \delta = x$	A6
$\delta \cdot x = \delta$	A7

Table 2. Deadlock.

Now we consider recursive specifications over BPA_δ. We give some definitions.

2.4 DEFINITIONS.

1. A *system of recursion equations* or *recursive specification* (over BPA_δ) is a finite set of equations

$$E = \{X_i = s_i(X_0,...,X_n) : i = 0,...,n\},$$

where the $s_i(\vec{X})$ are process expressions in the signature of BPA_δ, possibly containing occurrences of the recursion variables in $\vec{X}$. The variable X_0 is the *root variable.* Usually we will omit mentioning the root variable when presenting a system of recursion equations, with the understanding that it is the first variable in the actual presentation.

2. We will also on occasion use *infinitary* recursive specifications

$$E = \{X_i = s_i(\vec{X}) : i \in \mathbb{N}\},$$

but will always state explicitly when that is the case.

3. A process p_0 (in a certain process algebra) is a *solution* of a specification E if there are processes p_1,... in this process algebra such that substituting processes p_i for variables X_i yields only true statements.

4. Suppose that the right hand side of a recursion equation $X_i = s_i(\vec{X})$ is in normal form w.r.t. applications (from left to right) of axioms A4 and A5 in table 1. Such a recursion equation is *guarded* if every occurrence of every X_j ($j = 0,...,n$) in $s_i(\vec{X})$ is preceded (*guarded*) by an atom from A; more precisely, every occurrence of X_j is in a subexpression of the form $a \cdot s'$ for some atom a and expression s'. For instance, the equation $X = aX + YbY$ is not guarded, as the first occurrence of Y is unguarded; but the recursion equation $X = c(aY + ZbX)$ is guarded.

If the right hand side of an equation is not in normal form w.r.t. A4 and A5, it is said to be guarded if it is so after bringing the right hand side into normal form.

Now we can use guarded recursive specifications to define processes. It is obvious that not every specification can be used to determine a process (as *every* process satisfies the equation $X = X$), but guardedness is a sufficient criterion to guarantee unique solutions in several algebras (see e.g. BERGSTRA & KLOP [BK2]). We will assume in the sequel that every guarded recursive specification has a unique solution (also for infinitary specifications!), and we say this process is *defined* by the specification.

2.5 TRACE CONSISTENCY.

We will also need a way to tell when two process expression cannot give the same process. Certainly, two processes that are equal, must be able to perform the same sequences of actions (must have the same *traces*). Actually, this criterion is sufficient for our purposes. We will now give an operational semantics for process expressions that yields the traces of such an expression. This semantics is given by means of *action rules* (first given for process algebra in VAN GLABBEEK [G], but appearing earlier in many places, see e.g. PLOTKIN [P]).

2.6 ACTION RULES.

For each $a \in A$, we define two predicates on process expressions: $\xrightarrow{a}$ is a binary relation, and $\xrightarrow{a} \surd$ is a unary relation. Their intuitive meaning is as follows:

- $x \xrightarrow{a} y$ means that x can perform an a-step and evolve into y;
- $x \xrightarrow{a} \surd$ means that x can perform an a-step and terminate successfully.

The formal definition of these predicates is given in the following table 3. The last lines give rules for recursion: the idea is that if we know that an action relation holds for the right-hand side of an equation, we can infer it holds for the left-hand side, the recursion variable. A more exact treatment can be found in [G].

$a \xrightarrow{a} \surd$	
$x \xrightarrow{a} x' \Rightarrow x+y \xrightarrow{a} x'$	$x \xrightarrow{a} \surd \Rightarrow x+y \xrightarrow{a} \surd$
$y \xrightarrow{a} y' \Rightarrow x+y \xrightarrow{a} y'$	$y \xrightarrow{a} \surd \Rightarrow x+y \xrightarrow{a} \surd$
$x \xrightarrow{a} x' \Rightarrow x \cdot y \xrightarrow{a} x' \cdot y$	$x \xrightarrow{a} \surd \Rightarrow x \cdot y \xrightarrow{a} y$
$s_i \xrightarrow{a} y \Rightarrow X_i \xrightarrow{a} y$	$s_i \xrightarrow{a} \surd \Rightarrow X_i \xrightarrow{a} \surd$

Table 3. Action rules for BPA_δ + recursion.

2.7 STATE OPERATOR.

Now we add the state operator to the signature of BPA_δ. This operator was introduced and used in BAETEN & BERGSTRA [BB]. Let S be some *finite* set (the state space). Then λ_s is a unary operator on processes, for each $s \in S$. If x is some process, then $\lambda_s(x)$ denotes process x in state s. Then, if x is able to execute an action a, the result will be a certain action, and it will have a certain effect on the state. Thus, the state operator comes with two functions:

action: $A \times S \to A$, that gives the result of the execution of an action;

effect: $A \times S \to S$, that gives the state resulting from the execution of an action.

We will always require that $action(\delta,s) = \delta$ and $effect(\delta,s) = s$, for any $s \in S$ (i.e. δ is *inert*).

For examples and more motivation, see [BB] or [V].

The state operator has axioms SO1-3, displayed in table 4. Here $s \in S$, $a \in A$ and x,y are arbitrary processes.

$\lambda_s(a) = action(a,s)$	SO1
$\lambda_s(ax) = action(a,s) \cdot \lambda_{effect(a,s)}(x)$	SO2
$\lambda_s(x+y) = \lambda_s(x) + \lambda_s(y)$	SO3

Table 4. State Operator.

We note that in [BB] also a *generalized* state operator is defined. We remark that the results in this paper could also have been obtained using the generalized state operator. We also give the action rules for the state operator, in table 5.

$x \xrightarrow{a} x'$, $action(a,s) \neq \delta$	$\Rightarrow$	$\lambda_s(x) \xrightarrow{action(a,s)} \lambda_{effect(a,s)}(x')$
$x \xrightarrow{a} \surd$, $action(a,s) \neq \delta$	$\Rightarrow$	$\lambda_s(x) \xrightarrow{action(a,s)} \surd$

Table 5. Action rules for the state operator.

2.8 THE QUESTION.

Now we can state the central question of this paper as follows:

> Let E be a guarded recursive specification over BPA_δ with root variable X.
> Let a finite state space S with action, effect be given, and let $s \in S$.
> Is $\lambda_s(X)$ again the solution of a guarded recursive specification over BPA_δ ?

We will show in section 3 that the answer is no, and so the state operator increases the defining power of BPA_δ. Moreover, we will show in section 4:

> Let E be a *linear* recursive specification over BPA_δ with root variable X.
> Let a finite state space S with action, effect be given, and let $s \in S$.
> Then $\lambda_s(X)$ is again the solution of a linear recursive specification over BPA_δ.

Thus, the state operator applied to a regular process again gives a regular process (a ***regular*** process is a process defined by a linear specification). On the other hand, we will show:

> *All* BPA_δ-definable processes, and some that are not even BPA_δ-definable, can be defined by a *linear* recursive specification over $BPA_\delta + \lambda$.

Not *all* processes can be defined over $BPA_\delta + \lambda$, however, as we will also show:

> There is a PA-definable process that is not $BPA_\delta + \lambda$-definable.

We see that the defining power of $BPA_\delta + \lambda$ does not give all of the defining power of PA.

3. SOLVING THE PROBLEM.

3.1 DEFINITION.

Let $a,b \in A$ be two distinct atoms different from δ, and consider the following guarded recursive specification:

$$C = a \cdot D \cdot C$$
$$D = b + a \cdot D \cdot D.$$

This is a well-known specification (see e.g. BERGSTRA & KLOP [BK2]) which has as solution the *counter* C (interpret a as "add one" and b as "subtract one"). Note that this process has infinitely many different subprocesses, since subprocess $D^n \cdot C$, reached after executing a n times, has a trace beginning with n b's, but no trace beginning with n+1 b's. This observation immediately gives the following lemma.

3.2 LEMMA.

Not every guarded recursive specification over BPA gives a regular process.

3.3 MERGE.

In order to define the processes we want to discuss in the sequel, it will be useful to extend the theory BPA with the *merge* operator $\|$, parallel composition. As a semantics for merge we use *arbitrary interleaving*. In order to give a finite axiomatisation of merge, we use an auxiliary operator $⫽$ (*left-merge*). Now, $x ⫽ y$ means the same as $x \| y$ (the parallel, but interleaved, execution of x and y), but with the restriction that the first step must come from x. For more about these issues, see e.g. BERGSTRA & KLOP [BK2].

The theory PA has operators $+, \cdot, \|, ⫽$ and adds axioms M1-4 of table 4 below to the axioms A1-5 of BPA. The theory PA_δ adds constant δ and axioms A6-7 to this.

$x \| y = x ⌊\!⌊ y + y ⌊\!⌊ x$	M1
$a ⌊\!⌊ x = a\cdot x$	M2
$ax ⌊\!⌊ y = a(x \| y)$	M3
$(x+y) ⌊\!⌊ z = x ⌊\!⌊ z + y ⌊\!⌊ z$	M4

Table 6. PA.

We also give an operational semantics for PA, by means of the action rules in table 7.

$x \xrightarrow{a} x' \Rightarrow x \| y \xrightarrow{a} x' \| y$	$x \xrightarrow{a} \surd \Rightarrow x \| y \xrightarrow{a} y$
$y \xrightarrow{a} y' \Rightarrow x \| y \xrightarrow{a} x \| y'$	$y \xrightarrow{a} \surd \Rightarrow x \| y \xrightarrow{a} x$

Table 7. Action rules for PA.

3.4 DEFINITION.

Now let C be the process defined in 3.1, and let $d \in A$ be different from a,b,δ. Define the process P by:

$$P = C \| d.$$

P is just like the counter, except that *once* in its existence, it can do the action d. The moment, when this action will be executed, is completely undetermined, however. In the sequel, we will show that P cannot be defined over BPA_δ, but can be defined over $BPA_\delta + \lambda$.

3.5 THEOREM.

P can be defined over $BPA_\delta + \lambda$.

PROOF: Consider the following guarded recursive specification over BPA:

$$C_1 = a\cdot D_1\cdot C_1 + d\cdot C_1$$
$$D_1 = b + a\cdot D_1\cdot D_1 + d\cdot D_1$$

This specification always adds a d-possibility to the one in 3.1, and the solution can be seen to be $C \| d^\omega$, where d^ω is the solution of $X = d\cdot X$.

Now, define $S = \{0,1\}$, and let the functions action and effect be trivial (i.e. action(a,s) = a & effect(a,s) = s) except in two cases:

1. action(d,0) = δ;
2. effect(d,1) = 0.

CLAIM: $P = \lambda_1(C_1)$.

PROOF: First we establish that $\lambda_0(C_1) = C$:

$$\lambda_0(C_1) = a\cdot\lambda_0(D_1\cdot C_1) + \delta\cdot\lambda_0(C_1) = a\cdot\lambda_0(D_1\cdot C_1), \text{ and}$$
$$\lambda_0(D_1^{n+1}\cdot C_1) = b\cdot\lambda_0(D_1^{n}\cdot C_1) + a\cdot\lambda_0(D_1^{n+2}\cdot C_1) + \delta\cdot\lambda_0(D_1^{n+1}\cdot C_1) =$$
$$= b\cdot\lambda_0(D_1^{n}\cdot C_1) + a\cdot\lambda_0(D_1^{n+2}\cdot C_1), \text{ for each } n \in \mathbb{N}.$$

Thus, $\lambda_0(C_1)$ and C are both solutions of the same infinitary guarded recursive specification, and must be equal.

Then we establish the claim:

$$\lambda_1(C_1) = a\cdot\lambda_1(D_1\cdot C_1) + d\cdot\lambda_0(C_1), \text{ and}$$

$$\lambda_1(D_1^{n+1}\cdot C_1) = b\cdot\lambda_1(D_1^{n}\cdot C_1) + a\cdot\lambda_1(D_1^{n+2}\cdot C_1) + d\cdot\lambda_0(D_1^{n+1}\cdot C_1), \text{ for each } n \in \mathbb{N}.$$

On the other hand, we find

$$P = C \| d = C ⫍ d + d ⫍ C = (a\cdot D\cdot C) ⫍ d + d\cdot C = a\cdot(D\cdot C \| d) + d\cdot C, \text{ and}$$

$$D^{n+1}\cdot C \| d = (b\cdot D^n\cdot C + a\cdot D^{n+2}\cdot C) ⫍ d + d ⫍ D^{n+1}\cdot C =$$

$$= b\cdot(D^n\cdot C \| d) + a\cdot(D^{n+2}\cdot C \| d) + d\cdot D^{n+1}\cdot C, \text{ for each } n \in \mathbb{N}.$$

Using the previous result, we find that $\lambda_1(C_1)$ and P are both solutions of the same infinitary guarded recursive specification, and so must be equal.

This finishes the proof of the claim, and also the proof of the theorem.

Now we turn to the proof that P cannot be defined over BPA_δ. We first need some preliminary facts.

3.6 DEFINITION.

A guarded recursive specification is in *restricted Greibach Normal Form* (restricted GNF) if each equation is of the form $X = \delta$ or $X = a_1\cdot\alpha_1 + ... + a_k\cdot\alpha_k$,

where $k \geq 1$, each $a_i \in A - \{\delta\}$, and each α_i is either

i. nothing (so the term consists just of an atomic action);

ii. a recursion variable;

iii. the product of two recursion variables.

3.7 LEMMA.

Each guarded recursive specification over BPA_δ is equivalent to one in restricted GNF.

PROOF: See BAETEN, BERGSTRA & KLOP [BBK].

3.8 NOTE.

The BPA_δ-specifications above are all in restricted GNF. Note that as a consequence of lemma 3.7, each subprocess of a process given by a recursive specification, can be represented by a finite product of recursion variables. Using the axioms of the state operator, λ_s applied to an equation $X = a_1\cdot\alpha_1 + ... + a_k\cdot\alpha_k$ in restricted GNF yields

$$\lambda_s(X) = action(a_1,s)\cdot\lambda_s(\alpha_1) + ... + action(a_k,s)\cdot\lambda s(\alpha_k),$$

again the same format, and each subprocess has the form $\lambda_s(X_1\cdot X_2\cdot...\cdot X_n)$.

3.9 THEOREM.

P cannot be defined over BPA_δ.

PROOF: Suppose, for a contradiction, that the guarded recursive specification E over BPA_δ defines process P. By 3.7, we may suppose that E is in restricted GNF. We may also suppose that superfluous equations are removed (an equation is *superfluous* if its recursion variable cannot be accessed by executing a number of actions, starting from the root variable). From the definition of the counter it is apparant, that never infinitely many b-actions can be executed consecutively. Thus, starting from any recursion variable, only finitely many consecutive b-actions are possible. Let m be the maximum number of b-actions, any recursion variable can perform. We also derive from the definition of the counter that in any situation, an unlimited number of a-actions is possible.

Now, starting from the root variable of E, perform 3m a-actions. Then we have a process

$$X_1 \cdot X_2 \cdot \ldots \cdot X_n,$$

a finite product of recursion variables. Since no d-action has taken place yet, X_1 must be able to do a d-action. On the other hand, the whole process must be able to perform 3m b-actions. Of these, X_1 can perform at most m. Thus, after X_1 has performed its maximum number of b-actions, it must terminate, so that X_2 can start on the next series of b-steps. But since after the b-actions of X_1 no d-action has taken place yet, X_2 must be able to do a d-action.

Now go back to X_1. After it has done the d-action, it is replaced in the product by at most 2 recursion variables. Together, they can perform at most 2m b-steps, so they must terminate, after doing their maximum number of b-steps. But next, X_2 can perform a second d-step, and we have reached a contradiction, for P may only do *one* d-step.

This finishes the proof of the theorem, and so we have proved that the state operator extends the defining power of BPA_δ.

4. OTHER RESULTS.

First, we turn to regular processes. A *regular process* (or a *finite automaton*) is a process that has only finitely many subprocesses. A well-known result is that the regular processes are exactly the processes that are the solution of a finite *linear* recursive specification. A linear equation is an equation of the form $X = \delta$ or

$$X = a_1 \cdot X_1 + \ldots + a_k \cdot X_k + b_1 + \ldots + b_m$$

where $k+m \geq 1$, and each $a_i, b_j \in A - \{\delta\}$. Notice that this only differs from restricted GNF, in that we do not allow products of two recursion variables. But we know already that the defining power differs considerably: the counter is not a regular process, so cannot be defined by a linear specification, but it has a specification in restricted GNF, see 3.1.

Now we solve the second problem in 2.8.

4.1 THEOREM.

Let E be a linear recursive specification over BPA_δ with root variable X_1. Let a finite state space S with functions action, effect be given, and let $s_0 \in S$. Then $\lambda_{s_0}(X_1)$ is again the solution of a linear recursive specification over BPA_δ.

PROOF: Let E have variables $X_1, \ldots, X_n$. We will define a new linear recursive specification F with variables $Y_{i,s}$, for $i = 1, \ldots, n$ and $s \in S$. Now, let i,s be given. Let E have equation

$$X_i = a_1 \cdot X_{j_1} + \ldots + a_k \cdot X_{j_k} + b_1 + \ldots + b_m.$$

Then, F will have equation

$$Y_{i,s} = action(a_1,s) \cdot Y_{j_1,effect(a_1,s)} + \ldots + action(a_k,s) \cdot Y_{j_k,effect(a_k,s)} + action(b_1,s) + \ldots + action(b_m,s).$$

We see that after removing summands that are equal to δ, F becomes a linear recursive specification. It is obvious that the $\lambda_s(X_i)$ satisfy specification F, and thus $\lambda_s(X_i) = Y_{i,s}$, in particular $\lambda_{s_0}(X_1) = Y_{1,s_0}$. This finishes the proof.

Thus, the state operator applied to the solution of a linear specification gives a process, that again can be given by a linear specification. The situation changes drastically if we allow the state operator in the recursion, i.e. consider linear specifications over $BPA_\delta + \lambda$. First, we have the following theorem.

4.2 THEOREM.
Let the process X be definable over BPA_δ. Then X is also definable by a linear specification over $BPA_\delta + \lambda$.

PROOF: Let a recursive specification E over BPA_δ be given. We may suppose E is in restricted GNF. We have to define a linear specification over $BPA_\delta + \lambda$ that has the same solution. Let $E = \{X_i = s_i : i = 0,...,n\}$. As we saw in 3.6, each summand in each s_i has one of the following three forms:
1. a single atomic action, a;
2. the product of an atomic action and a recursion variable, $a \cdot X_j$;
3. the product of an atomic action and two recursion variables, $a \cdot X_j \cdot X_k$.
Now we introduce new atoms:
1. an atom $\langle a,i \rangle$ if atomic action a occurs in s_i singly (a summand of type 1);
2. an atom $\langle a,i,j \rangle$ if atomic action a occurs in s_i in the product $a \cdot X_j$ (type 2);
3. an atom $\langle a,i,j,k \rangle$ if atomic action a occurs in s_i in the product $a \cdot X_j \cdot X_k$ (type 3).
Now we define the state operator. The state space is $\{0,...,n\}$, and the action, effect functions are trivial except in the following cases:
i. $action(\langle a,i \rangle,m) = action(\langle a,i,j \rangle,m) = action(\langle a,i,j,k \rangle,m) = \delta$ if $i \neq m$;
ii. $action(\langle a,i \rangle,i) = action(\langle a,i,j \rangle,i) = action(\langle a,i,j,k \rangle,i) = a$;
iii. $effect(\langle a,i,j \rangle,i) = j$, $effect(\langle a,i,j,k \rangle,i) = k$.
Then we consider the following linear recursive equation:

$$X = \Sigma_{type\ 1} \langle a,i \rangle + \Sigma_{type\ 2} \langle a,i,j \rangle \cdot X + \Sigma_{type\ 3} \langle a,i,j,k \rangle \cdot \lambda_j(X).$$

CLAIM: $\lambda_0(X) = X_0$.

PROOF: The proof is easier to follow if we take a specific example. So take E to be:

$$X_0 = a \cdot X_0 + b + c \cdot X_1 \cdot X_0$$
$$X_1 = b \cdot X_0 \cdot X_1 + b.$$

Then the linear equation becomes:

$$X = \langle b,0 \rangle + \langle b,1 \rangle + \langle a,0,0 \rangle \cdot X + \langle c,0,1,0 \rangle \cdot \lambda_1(X) + \langle b,1,0,1 \rangle \cdot \lambda_0(X).$$

Now we show that for each sequence $b_1...b_n$ of 0's and 1's we have $X_{b_1} \cdot X_{b_2} \cdot ... \cdot X_{b_n} = \lambda_{b_n} \circ \lambda_{b_{n-1}} \circ ... \circ \lambda_{b_1}(X)$, by showing they satisfy the same infinitary recursive specification. We give the equations for the processes $\lambda_{b_n} \circ \lambda_{b_{n-1}} \circ ... \circ \lambda_{b_1}(X)$. We use the abbreviation $\lambda_{b_n ... b_1}(X)$ for $\lambda_{b_n} \circ \lambda_{b_{n-1}} \circ ... \circ \lambda_{b_1}(X)$. Let σ be any sequence of 0's and 1's. Then:

$$\lambda_\sigma \circ \lambda_0(X) = \lambda_\sigma(b + \delta + a \cdot \lambda_0(X) + c \cdot \lambda_0 \circ \lambda_1(X) + \delta \cdot \lambda_0 \circ \lambda_0(X)) = b + a \cdot \lambda_{\sigma 0}(X) + c \cdot \lambda_{\sigma 01}(X), \text{ and}$$
$$\lambda_\sigma \circ \lambda_1(X) = \lambda_\sigma(\delta + b + \delta \cdot \lambda_1(X) + \delta \cdot \lambda_1 \circ \lambda_1(X) + b \cdot \lambda_1 \circ \lambda_0(X)) = b + b \cdot \lambda_{\sigma 10}(X).$$

This finishes the proof of the claim, and also the proof of the theorem.

Next, we will give an example of a process, that is not definable over BPA_δ, but is definable by a linear specification over $BPA_\delta + \lambda$. In fact, we will show more than that it is not definable over BPA_δ: we will show that there is no recursive specification over BPA_δ such that a state operator applied to its solution yields this process.

4.3 DEFINITION.

Let us define another copy of a counter, with different names:

$$G = e \cdot H \cdot G$$

$$H = f + e \cdot H \cdot H \cdot G.$$

($a,b,e,f \in A - \{\delta\}$ are all distinct). Then define

$$B = C \| G.$$

As shown in [BK1], B can be considered as a bag (not order-preserving channel) over two elements, with a,e the input actions, and b,f the output actions. An alternative specification for the bag, in one equation, is the following:

$$B = a \cdot (b \| B) + e \cdot (f \| B).$$

It was shown in [BK1], that B cannot be defined over BPA. We strengthen this result in the following theorem.

4.4 THEOREM.

There is no recursive specification over BPA_δ with root variable X, and a finite state space S with functions action, effect, and $s \in S$, such that $\lambda_s(X) = B$.

PROOF: Suppose not, so there is a guarded recursive specification E over BPA_δ with root variable X, and there is a finite state space S with element s and functions action, effect such that $\lambda_s(X) = B$. We may suppose that E is in restricted GNF and has no superfluous equations. We see that for each $s \in S$ and each recursion variable Y, $\lambda_s(Y)$ can perform only finitely many b-actions and finitely many f-actions. Let m be the maximum number of b or f-steps any $\lambda_s(Y)$ can do. Let k be the cardinality of S.

Now, starting from $\lambda_s(X)$, perform $m(k+2)$ a-actions and $m(k+2)$ e-actions. Then, we have a subprocess of the form

$$\lambda_t(X_1 \cdot \ldots \cdot X_k \cdot X_{k+1} \cdot X_{k+2} \cdot \ldots \cdot X_n)$$

for certain $t \in S$ and recursion variables X_i ($i = 1,\ldots,n$). Note that this product must contain at least k+2 factors, since this process can do $m(k+2)$ b-actions and $m(k+2)$ f-actions, and each variable can account for at most m. Now we will "eat up" the variables $X_1,\ldots,X_{k+1}$ in k+1 different ways.

In the first way, we keep on doing b-actions. After at most m of them, X_1 will terminate. We continue with b-actions, until X_{k+1} terminates. Then, we have a process $\lambda_{s_1}(X_{k+2} \cdot \ldots \cdot X_n)$.

In the second way, we do b-actions until X_k terminates. Then, we do f-actions until X_{k+1} terminates. Again, we have a process $\lambda_{s_2}(X_{k+2} \cdot \ldots \cdot X_n)$. In general, for $i = 1,\ldots,k+1$, we do b-actions until X_{k+2-i} terminates. Then, we continue doing f-actions until X_{k+1} terminates. Then, we have a process $\lambda_{s_i}(X_{k+2} \cdot \ldots \cdot X_n)$.

We have found $s_1,\ldots,s_{k+1} \in S$ but since S contains only k elements, at least two of these must be equal, say $s_i = s_j$ with $i<j$. But then we have a contradiction, for $\lambda_{s_i}(X_{k+2} \cdot \ldots \cdot X_n) = \lambda_{s_j}(X_{k+2} \cdot \ldots \cdot X_n)$, and $\lambda_{s_i}(X_{k+2} \cdot \ldots \cdot X_n)$ can perform less consecutive b-actions and more consecutive f-actions than $\lambda_{s_j}(X_{k+2} \cdot \ldots \cdot X_n)$.

This finishes the proof.

4.5 THEOREM.
B is definable by a linear recursive specification over $BPA_\delta + \lambda$.

PROOF: We need two new atoms, b^* and f^*. The state space is $S = \{0,1,B,F\}$, where 0 is the starting state, and 1 is the state where the job is finished, in this state the state operator becomes inert. We list the non-trivial cases of functions action, effect:
i. $action(b^*,0) = action(f^*,0) = \delta$;
ii. $action(b^*,B) = b$, $effect(b^*,B) = 1$;
ii. $action(f^*,F) = f$, $effect(f^*,F) = 1$.
Then we consider the following linear recursive equation:

$$X = a\cdot\lambda_B(X) + e\cdot\lambda_F(X) + b^*\cdot X + f^*\cdot X.$$

CLAIM: $\lambda_0(X) = B$.

PROOF: Let $B_{n,m}$ be the subprocess of B where counter C stands at n (i.e. there is a trace beginning with n b's, but no trace beginning with n+1 b's) and counter G stands at m. Let $\lambda_{Bn,Fm}$ be any sequence of λ-operators, in which λ_B occurs exactly n times, λ_F occurs exactly m times, and which further consists of a number of occurrences of λ_1. We will show that $B_{n,m} = \lambda_0\circ\lambda_{Bn,Fm}(X)$, by showing they satisfy the same infinitary recursive specification. We will calculate this specification for the $\lambda_0\circ\lambda_{Bn,Fm}(X)$:
<u>Case 1:</u> n=0, m=0. (Since the λ_1 are inert, we might as well leave them out.)

$$\lambda_0(X) = a\cdot\lambda_0\circ\lambda_B(X) + e\cdot\lambda_0\circ\lambda_F(X) + \delta\cdot\lambda_0(X) + \delta\cdot\lambda_0(X) = a\cdot\lambda_0\circ\lambda_B(X) + e\cdot\lambda_0\circ\lambda_F(X).$$

<u>Case 2:</u> n=0, m>0.

$$\lambda_0\circ\lambda_{Fm}(X) = a\cdot\lambda_0\circ\lambda_{B1,Fm}(X) + e\cdot\lambda_0\circ\lambda_{Fm+1}(X) + \delta\cdot\lambda_0\circ\lambda_{Fm}(X) + f\cdot\lambda_0\circ\lambda_{Fm-1}\circ\lambda_1(X) =$$
$$= a\cdot\lambda_0\circ\lambda_{B1,Fm}(X) + e\cdot\lambda_0\circ\lambda_{Fm+1}(X) + f\cdot\lambda_0\circ\lambda_{Fm-1}(X).$$

<u>Case 3:</u> n>0, m=0. Just like case 2.
<u>Case 4:</u> n>0, m>0.

$$\lambda_0\circ\lambda_{Bn,Fm}(X) = a\cdot\lambda_0\circ\lambda_{Bn+1,Fm}(X) + e\cdot\lambda_0\circ\lambda_{Bn,Fm+1}(X) + b\cdot\lambda_0\circ\lambda_{Bn-1,Fm}\circ\lambda_1(X) + f\cdot\lambda_0\circ\lambda_{Bn,Fm-1}\circ\lambda_1(X) =$$
$$= a\cdot\lambda_0\circ\lambda_{Bn+1,Fm}(X) + e\cdot\lambda_0\circ\lambda_{Bn,Fm+1}(X) + b\cdot\lambda_0\circ\lambda_{Bn-1,Fm}(X) + f\cdot\lambda_0\circ\lambda_{Bn,Fm-1}(X).$$

Since the processes $B_{n,m}$ satisfy the same infinitary specification, we have proved the claim, and thereby the theorem.

Finally, we give an example of a PA-definable process, that is not definable over $BPA_\delta + \lambda$. This proves the last claim in 2.8: not every process is definable over $BPA_\delta + \lambda$.

4.6 DEFINITION.
We call a process p *uniformly finitely branching* if there is some natural number n such that for every subprocess q of p (i.e. every process reachable from p by use of action relations) there are at most n processes q' such that $q \xrightarrow{a} q'$ (for some atom a). (In other words: the *branching degree* of the process is uniformly bounded.)

4.7 LEMMA.
Every $BPA_\delta + \lambda$-definable process is uniformly finitely branching.

PROOF: In [BK1], it is proved that every BPA-definable process is uniformly finitely branching. The proof is easy: every subprocess of a process defined by a recursive specification over BPA (or BPA_δ, for that matter) is given by a product of recursion variables, and every step possible from this process is determined by the first variable in the product. But they in turn are determined by the equation for this variable, in which only a finite sum occurs. The uniform bound is the maximum number of summands in any equation of the specification.

Then, this result extends to $BPA_\delta + \lambda$, if we realize that applying the state operator to a term can only *decrease* the branching degree (by renaming into δ), but can never increase it.

Then, if we combine lemma 4.7 with the following result of BERGSTRA & KLOP [BK1], we have finished the proof of the last claim in 2.8:

the solution of the PA-equation $X = a + b\cdot(X\cdot c \| X\cdot d)$ is not uniformly finitely branching.

5. CONCLUSIONS.

We have shown that the defining power of the state operator, a natural addition to the operators of basic process algebra, is considerable. Applying the state operator to a BPA-process sometimes gives a process that is not BPA-definable. On the other hand, applying the state operator to a regular process gives again a regular process. If we allow the state operator inside the recursion, even more processes become definable, for instance the bag, although there still remain PA-processes that are not definable.

5.1 The following remarkable result, that strengthens theorem 4.4, was communicated to us by VAANDRAGER [V2]. It concerns the process *queue*. A (FIFO) queue Q (over two elements) is given by the following infinitary recursive specification, with variables Q_σ, with σ a sequence of b's and f's. (Again, a and e are two different input actions, with corresponding output actions b,f.)

$$Q_\varepsilon = a\cdot Q_b + e\cdot Q_f$$

$$Q_{b\sigma} = a\cdot Q_{b\sigma b} + e\cdot Q_{b\sigma f} + b\cdot Q_\sigma \qquad \text{for any sequence } \sigma$$

$$Q_{f\sigma} = a\cdot Q_{f\sigma b} + e\cdot Q_{f\sigma f} + f\cdot Q_\sigma \qquad \text{for any sequence } \sigma.$$

Now it was shown in BAETEN & BERGSTRA [BB], that Q cannot be defined over PA. VAANDRAGER [V2] shows that Q can be defined by a linear recursive specification over $BPA_\delta + \lambda$. He uses the following specification.

- out is a new atom;
- take S = {0,B,F,1} (1 again inert), with the functions trivial except for the following cases:

i. action(out,B) = b, effect(out,B) = 1; action(out,F) = f, effect(out,F) = 1;

ii. action(b,F) = f, effect(b,F) = F; action(f,B) = b, effect(f,B) = F;

iii. action(out,0) = δ.

Then the following equation yields a queue (proof omitted):

$$Q = \lambda_0(X)$$

$$X = a\cdot\lambda_B(X) + e\cdot\lambda_F(X) + out\cdot X.$$

5.2 Obviously, we can repeat all the questions in this paper with the theory PA in the place of BPA (or still other theories). Most of these questions we leave as open problems. The main question, does the state operator add to the defining power of PA, was answered in the positive in 5.1 above.

Of course, the subject matter of this paper has many connections with formal language theory: all our results can be translated to that setting, and well-known examples in formal language theory can be translated to our setting. As an example, we can define a process with finite traces $a^n \cdot b^n \cdot c^n$ (for each $n \in \mathbb{N}$), that will not be BPA-definable (roughly, context-free means BPA-definable), but is definable over $BPA_\delta + \lambda$ ([V2]).

REFERENCES.

[BB] J.C.M. BAETEN & J.A. BERGSTRA, *Global renaming operators over concrete process algebra*, report P8709, Programming Research Group, University of Amsterdam 1987. To appear in Inf. & Comp.

[BBK] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP, *Decidability of bisimulation equivalence for processes generating context-free languages*, in: Proc. PARLE, Vol. II (Parallel Languages) (J.W. de Bakker, A.J. Nijman, P.C. Treleaven, eds.), Eindhoven, Springer LNCS 259, pp. 94-113.

[BK1] J.A. BERGSTRA & J.W. KLOP, *The algebra of recursively defined processes and the algebra of regular processes*, in: Proc. 11th ICALP, Antwerpen (J. Paredaens, ed.), Springer LNCS 172, pp. 82-94, 1984.

[BK2] J.A. BERGSTRA & J.W. KLOP, *Process algebra: specification and verification in bisimulation semantics*, Proc. CWI Symp. Math. & Comp. Sci. II (M. Hazewinkel, J.K. Lenstra & L.G.L.T. Meertens, eds.), CWI Monograph 4, pp. 61-94, North-Holland, Amsterdam 1987.

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

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

[M] R. MILNER, *A calculus of communicating systems*, Springer LNCS 92, 1980.

[P] G.D. PLOTKIN, *An operational semantics for CSP*, in: Proc. Conf. Formal Descr. of Progr. Concepts II, Garmisch (E. Bjørner, ed.), pp. 199-225, North-Holland 1982.

[V1] F.W. VAANDRAGER, *Process algebra semantics of Pool*, report CS-R8629, Centre for Mathematics and Computer Science, Amsterdam 1986.

[V2] F.W. VAANDRAGER, personal communication, January 1988.

P8701 **R.A. Groenveld**
Verification of a sliding window protocol by means of process algebra

P8702 **M.R. Dasselaar**
Development of an expert system, from theory to practice

P8703 **A.V. Hurkmans**
Een declaratieve en procedurele kennisrepresentatievorm voor kennissystemen, toegepast op NEXT

P8704 **F. Wiedijk**
Termherschrijfsystemen in Prolog

P8705 **J.L.M. Vrancken**
The algebraic specification of semicomputable datatypes

P8706 **J.C.M. Baeten, J.A. Bergstra & J.L.M. Vrancken**
Processen en procesexpressies

P8707 **J.H. Verhagen**
Expertsystemen bij de Nederlandse Spoorwegen

P8708 **S. Mauw**
Process algebra as a tool for the specification and verification of CIM-architectures

P8709 **J.C.M. Baeten & J.A. Bergstra**
Global renaming operators in concrete process algebra *(revised version)*

P8710 **J. Treur**
Volledigheid en definieerbaarheid in diagnostische redeneersystemen

P8711 **J. Treur**
Een logische analyse van diagnostische redeneerprocessen; redeneren met en over hypothesen

P8712 **W. Syski & J. Treur**
Reasoning about uncertainty represented by meta-reasoning; the endorsements approach

P8713 **W. Syski**
On a certain probabilistic approximation method for reasoning with uncertainty

P8714 **L.G. Bouma & H.R. Walters**
Implementing algebraic specifications

P8801 **F.R. Burggraaff & E. van der Meulen**
ASF Specification of a B-tree of order 1

P8802 **J.C.M. Baeten & J.A. Bergstra**
Recursive process definitions with the state operator

P8803 **J.C.M. Baeten & F.W. Vaandrager**
Specification and verification of a circuit in ACP