Theoretical Computer Science 82 (1991) 285-302
Elsevier

Recursive process definitions with the state operator

J.C.M. Baeten*

Department of Software Technology, Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, Netherlands

J.A. Bergstra*

Programming Research Group, University of Amsterdam, P.O. Box 41882, 1009 DB Amsterdam, Netherlands, and Department of Philosophy, State University of Utrecht, Heidelberglaan 2, 3584 CS Utrecht, Netherlands

Communicated by A. Meyer
Received May 1988
Revised May 1989

Abstract

Baeten, J.C.M., and J.A. Bergstra, Recursive process definitions with the state operator, Theoretical Computer Science 82 (1991) 285-302.

We investigate the defining power of finite recursive specifications over the theory with + (alternative composition) 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.

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. [5]), that can be classified as an algebraic and axiomatic approach to concurrency (in the vein of CCS, see [8] or CSP, see [7]).

BPA has two binary operators: + is alternative composition (non-deterministic choice, as in CCS), and · is sequential composition (as ; in CSP), and consists of

* Both authors are partially sponsored by ESPRIT contract 432, An Integrated Formal Approach to Industrial Software Development (METEOR). The first author is also partially sponsored by RACE contract 1046, Specification and Programming Environment for Communication Software (SPECS).

0304-3975/91/$03.50 © 1991—Elsevier Science Publishers B.V.

just five simple axioms. 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 [4]. There, it was found that a wider class of processes can be defined than the class of regular processes (essentially, the class of context-free languages). We obtain the theory PA by the addition of the parallel operator $\|$ (merge). It was found in [4] that this increases the defining power of recursive specifications even further.

The state operator λ was introduced in [1]. 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 programming or specification languages into process algebra (see [11] or [10]). 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 [1]). 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 $BPA_\delta + \lambda$.

Thus, we obtain a hierarchy of process classes. The results we obtain are pictured in Fig. 1. Each arrow denotes a strict inclusion relation. Between the three classes at the right, some non-inclusion results are obtained. We identify the classes in Fig. 1:

- BPA_δ lin.: processes definable by a linear specification over BPA_δ;
- $\lambda(BPA_\delta$ lin.): processes obtained by application of λ to processes in BPA_δ lin.;
- BPA_δ rec.: processes definable by a recursive specification over BPA_δ;
- $\lambda(BPA_\delta$ rec.): processes obtained by application of λ to processes in BPA_δ rec.;

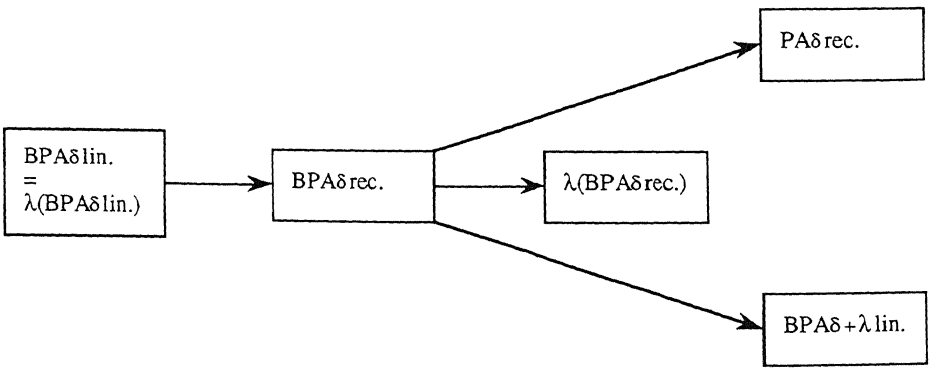

Fig. 1.

- $BPA_\delta + \lambda$ lin.: processes definable by a linear specification over $BPA_\delta + \lambda$;
- PA_δ rec.: processes definable by a recursive specification over PA_δ.

2. Preliminaries

2.1. Basic process algebra

The axiom system BPA consists of the axioms in Table 1. The signature of BPA consists of a set $A = \{a, b, c, \ldots\}$ of constants, called *atomic actions*, and the operators $+$ (alternative composition) and $\cdot$ (sequential composition). Often the dot $\cdot$ and parentheses will be suppressed. $\cdot$ 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.

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 $a(cd+bd) = acd+abd$. In general, we do not equate the processes $x(y+z)$ and $xy+xz$. We do this, because the moment of choice in these processes is different. This is important for instance in the analysis of deadlock behaviour (see below). As a consequence, we have a branching time semantics as in CCS.

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 any more, the absence of an alternative to proceed. The constant $\delta \in A$ has two special axioms, displayed in Table 2. We

Table 1
BPA

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

Table 2
Deadlock

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

denote the theory BPA+δ, with axioms A1-A7, by BPA$_\delta$. Using δ, we can describe two different ways of termination: in the process $a\delta+b$, we have unsuccessful termination (deadlock) after performing a, and successful termination after performing b. Only in the case of successful termination can we continue with the next process in a context of sequential composition. Note that this approach is different from the situation in CCS or CSP, where only one kind of termination is possible. Nevertheless, our results will also hold in the setting of CCS or (T)CSP, since the key use of δ is to define a notion of *restriction*, a notion that is also present in CCS and CSP.

We see, that with the rejected law $x(y+z)=xy+xz$ we can derive $ab=a(b+\delta)=ab+a\delta$, and this equates a process with deadlock possibility to one without such a possibility, a clearly undesirable situation.

Now we consider recursive specifications over BPA$_\delta$. We give some definitions.

2.4. *Definitions*

(1) A *system of recursion equations* or *recursive specification* (over BPA$_\delta$) is a finite set of equations

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

where the $s_i(\vec{X})$ are process expressions in the signature of BPA$_\delta$, 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

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

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 $p_1,\ldots$ 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 with respect to 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 ($j=0,\ldots,n$) 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 with respect to A4 and A5, it is said to be guarded if it is so after bringing the right-hand side into normal form.

(5) A recursive specification is called *linear* if all its equations are of the form $X = \delta$ or

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

where $k + m \geq 1$, and each $a_i, b_j \in A - \{\delta\}$. Obviously, linear specifications are always guarded.

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.

We mention a few algebras where the laws of BPA_δ hold and guarded recursive specifications have unique solutions:

- the projective limit model, see e.g. [5];
- the graph model, see e.g. [3];
- the action relation model, see e.g. [6] (the operational semantics that forms the basis of this model is presented below).

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.

A *subprocess* of process x is a process that can be reached by executing a number of steps from x. 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.

2.5. Trace consistency

We will also need a way to tell when two process expressions 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 this theory in [6], but appearing earlier in many places, see e.g. [9]). We will use this operational semantics in a rather informal way: when we say that process p can do an a-step to process q, we mean $p \xrightarrow{a} q$.

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 Table 3. The last lines give rules for recursion: the idea is that if we know that an action relation holds for the

Table 3
Action rules for BPA_δ + recursion

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

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 [6].

2.7. State operator

Now we add the state operator to the signature of BPA_δ. This operator was introduced and used in [1]. 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*).

The state operator has axioms SO1–SO3, displayed in Table 4. Here $s \in S$, $a \in A$ and x, y are arbitrary processes.

The state operator is a *renaming operator*, since the *action* function allows us to rename atoms. Notice that atomic actions may only be renamed into an atomic action, not into a general process. While such general renaming is consistent with BPA_δ, it is not consistent with several extensions of the theory, such as the theory PA to be discussed later. By renaming into δ, we can block the execution of an action; thus, the state operator also includes the notion of encapsulation or restriction.

Table 4
State operator

$\lambda_s(a) = action(a, s)$	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

By means of the state operators, we can express constructs like the guarded command, the *if... then... else...* construct or case distinction. Further, the operator can handle processes with data variables, and can be used to (mechanically) translate a given computer program into process algebra. We claim that the state operator can be very useful in the design of a programming language that is based on process algebra. For examples and more motivation, see [1, 11, 10].

We give the action rules for the state operator in Table 5.

We note that in [1] a *generalized* state operator is also defined (the result of executing an action is a sum of actions, possibly followed by different states). We remark that the results in this paper could also have been obtained using the generalized state operator.

2.8. Summary of results

Now we can state the central question of this paper as follows: does the state operator add to the defining power of BPA_δ? In Section 3, we will answer this question positively. More specifically, there exists a guarded recursive specification over BPA_δ with root variable X, a finite state space S with *action*, *effect* functions, and an $s \in S$, such that $\lambda_s(X)$ is not the solution of a guarded recursive specification over BPA_δ.

Thus, the state operator applied to a recursively definable process does not necessarily yield a recursively definable process. However, if we limit ourselves to *linear* specifications, we get a different result. We will show in Section 4: if E is a *linear* recursive specification over BPA_δ with root variable X, if S is a finite state space with *action*, *effect* given, and if $s \in S$, 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. If, however, we allow the state operator inside the recursion, we get a very different picture. We will show that *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 that 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.

We can summarize our results in the following picture. Each arrow denotes a strict inclusion relation. Alongside the arrows, we give the sections where this result is obtained. Moreover, we have some non-inclusion results: in Lemma 4.9 and Section 5.1, we show that no inclusion relation exists between PA_δrec. and $BPA_\delta + \lambda$. In Theorem 4.5, we show that $BPA_\delta + \lambda$ is not included in $\lambda(BPA_\delta\text{rec.})$.

Table 5
Action rules for the state operator

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

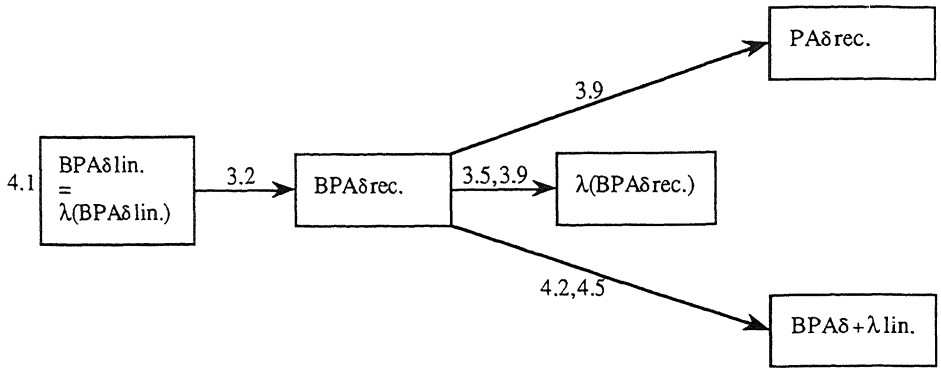

Fig. 2.

3. Answers and proofs

Definition 3.1. 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. [4]) 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 sub-processes, 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.

Lemma 3.2. *Not every guarded recursive specification over* BPA *gives a regular process.*

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 axiomatization of merge, we use an auxiliary operator $\lfloor\!\lfloor$ (*left-merge*). Now, $x \lfloor\!\lfloor y$ 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. [5].

The theory PA has operators $+$, $\cdot$, $\|$, $\lfloor\!\lfloor$ and adds axioms M1–M4 of Table 6 to the axioms A1–A5 of BPA. The theory PA_δ adds constant δ and axioms A6–A7 to this.

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

Table 6
PA

$x \parallel y = x \mathbin{\mathbb{L}} y + y \mathbin{\mathbb{L}} x$	M1
$a \mathbin{\mathbb{L}} x = a \cdot x$	M2
$ax \mathbin{\mathbb{L}} y = a(x \parallel y)$	M3
$(x+y) \mathbin{\mathbb{L}} z = x \mathbin{\mathbb{L}} z + y \mathbin{\mathbb{L}} z$	M4

Table 7
Action rules for PA

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

Definition 3.3. Now let C be the process defined in Definition 3.1, and let $d \in A$ be different from a, b, δ. Define the process P by

$$P = C \parallel d.$$

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 in $\lambda(\mathrm{BPA}_\delta$ rec.).

Theorem 3.4. *P can be defined in $\lambda(\mathrm{BPA}_\delta$ rec.).*

Proof. Consider the following guarded recursive specification over BPA:

$$C' = a \cdot D' \cdot C' + d \cdot C',$$

$$D' = b + a \cdot D' \cdot D' + d \cdot D'.$$

This specification always adds a d-possibility to the one in Definition 3.1, and the solution can be seen to be $C \parallel d^\omega$, 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. $\mathit{action}(a, s) = a$ & $\mathit{effect}(a, s) = s$) except in two cases:

(1) $\mathit{action}(d, 0) = \delta$;

(2) $\mathit{effect}(d, 1) = 0$.

Claim 3.5. $P = \lambda_1(C')$.

Proof. First we establish that $\lambda_0(C') = C$:

$$\lambda_0(C') = a \cdot \lambda_0(D' \cdot C') + \delta \cdot \lambda_0(C') = a \cdot \lambda_0(D' \cdot C'),$$

$$\lambda_0(D'^{n+1} \cdot C') = b \cdot \lambda_0(D'^n \cdot C') + a \cdot \lambda_0(D'^{n+2} \cdot C') + \delta \cdot \lambda_0(D'^{n+1} \cdot C')$$

$$= b \cdot \lambda_0(D'^n \cdot C') + a \cdot \lambda_0(D'^{n+2} \cdot C'), \quad \text{for each } n \in \mathbb{N}.$$

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

Then we establish the claim

$$\lambda_1(C') = a \cdot \lambda_1(D' \cdot C') + d \cdot \lambda_0(C'),$$

$$\lambda_1(D'^{n+1} \cdot C') = b \cdot \lambda_1(D'^n \cdot C') + a \cdot \lambda_1(D'^{n+2} \cdot C') + d \cdot \lambda_0(D'^{n+1} \cdot C'),$$

for each $n \in \mathbb{N}$.

On the other hand, we find

$$P = C \parallel d = C \mathbin{\mathbb{L}} d + d \mathbin{\mathbb{L}} C = (a \cdot D \cdot C) \mathbin{\mathbb{L}} d + d \cdot C$$
$$= a \cdot (D \cdot C \parallel d) + d \cdot C,$$

and

$$D^{n+1} \cdot C \parallel d = (b \cdot D^n \cdot C + a \cdot D^{n+2} \cdot C) \mathbin{\mathbb{L}} d + d \mathbin{\mathbb{L}} D^{n+1} \cdot C$$
$$= b \cdot (D^n \cdot C \parallel d) + a \cdot (D^{n+2} \cdot C \parallel d) + d \cdot D^{n+1} \cdot C,$$

for each $n \in \mathbb{N}$.

Using the previous result, we find that $\lambda_1(C')$ 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.

Definition 3.6. A guarded recursive specification is in *restricted Greibach Normal Form* (restricted GNF) if each equation is of the form $X = \delta$ or $X = s_1 + \cdots + s_k$, where $k \geqslant 1$ and each s_i has one of the following forms:

(i) $s_i \equiv a_i$ (for some $a_i \in A - \{\delta\}$);
(ii) $s_i \equiv a_i \cdot X_i$ (for some $a_i \in A - \{\delta\}$ and some recursion variable X_i);
(iii) $s_i \equiv a_i \cdot X_i' \cdot X_i''$ (for some $a_i \in A - \{\delta\}$ and recursion variables X_i', X_i'').

Lemma 3.7. *Each guarded recursive specification over* BPA_δ *is equivalent to one in restricted* GNF.

Proof. See [3]. □

Remark 3.8. 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

$$X = a_1 + \cdots + a_k + b_1 \cdot X_1 + \cdots + b_m \cdot X_m + c_1 \cdot X_1' \cdot X_1'' + \cdots + c_n \cdot X_n' \cdot X_n''$$

in restricted GNF yields

$$\lambda_s(X) = action(a_1, s) + \cdots + action(b_1, s) \cdot \lambda_{effect(b_1, s)}(X_1) + \cdots + action(c_1, s) \cdot \lambda_{effect(c_1, s)}(X_1' \cdot X_1'') + \cdots,$$

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

Theorem 3.9. *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 Lemma 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 apparent that infinitely many b-actions can never 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. Further results

First, we turn to regular processes. Regular processes are definable by linear specifications. Notice that linear specifications only differ 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 (for it

has infinitely many subprocesses), so cannot be defined by a linear specification, but it has a specification in restricted GNF (see Definition 3.1).

Theorem 4.1. *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

$$X_i = a_1 \cdot X_{j1} + \cdots + a_k \cdot X_{jk} + b_1 + \cdots + b_m.$$

Then, F will have equation

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

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 $\mathrm{BPA}_\delta + \lambda$. First, we have the following theorem.

Theorem 4.2. *Let the process X be definable over BPA_δ (not necessarily a regular process). 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 $\mathrm{BPA}_\delta + \lambda$ that has the same solution. Let $E = \{X_i = s_i : i = 0, \ldots, n\}$. As we saw in Definition 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 $\{0, \ldots, n\}$, 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:

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

Claim 4.3. $\lambda_0(X) = X_0$.

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

$$\lambda_\sigma \circ \lambda_0(X) = \lambda_\sigma(b + \delta + a \cdot \lambda_0(X) + c \cdot \lambda_0 \circ \lambda_1(X) + \delta \cdot \lambda_0 \circ \lambda_0(X))$$

$$= b + a \cdot \lambda_{\sigma 0}(X) + c \cdot \lambda_{\sigma 01}(X),$$

$$\lambda_\sigma \circ \lambda_1(X) = \lambda_\sigma(\delta + b + \delta \cdot \lambda_1(X) + \delta \cdot \lambda_1 \circ \lambda_1(X) + b \cdot \lambda_1 \circ \lambda_0(X)) = b + b \cdot \lambda_{\sigma 10}(X).$$

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 it is not in $\lambda(BPA_\delta$ rec.).

Definition 4.4. Let us define another copy of a counter, with different names:

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

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

$$B = C \parallel G.$$

As shown in [4], 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:

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

It was shown in [4], that B cannot be defined over BPA. We strengthen this result in the following theorem.

Theorem 4.5. *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. □

Theorem 4.6. *B is definable by a linear recursive specification over* $\mathrm{BPA}_\delta + \lambda$.

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 trivial. We list the non-trivial cases of functions *action, effect*:

(i) $action(b^*, 0) = action(f^*, 0) = \delta$;

(ii) $action(b^*, B) = b$, $effect(b^*, B) = 1$;
(iii) $action(f^*, F) = f$, $effect(f^*, F) = 1$.

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 4.7. $\lambda_0(X) = B$.

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. Thus, the $B_{n,m}$ have the following infinitary linear specification:

$$B_{0,0} = a \cdot B_{1,0} + e \cdot B_{0,1}$$

$$B_{0,m} = a \cdot B_{1,m} + e \cdot B_{0,m+1} + f \cdot B_{0,m-1} \qquad (m > 0)$$

$$B_{n,0} = a \cdot B_{n+1,0} + e \cdot B_{n,1} + b \cdot B_{n-1,0} \qquad (n > 0)$$

$$B_{n,m} = a \cdot B_{n+1,m} + e \cdot B_{n,m+1} + b \cdot B_{n-1,m} + + f \cdot B_{0,m-1} \qquad (n > 0, m > 0).$$

Next, let λ_{B^n,F^m} 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 $B_{n,m} = \lambda_0 \circ \lambda_{B^n,F^m}(X)$, by showing they satisfy the same infinitary recursive specification. We now calculate this specification for the $\lambda_0 \circ \lambda_{B^n,F^m}(X)$:

Case 1: $n = 0$, $m = 0$. (Since the λ_1 are trivial, we might as well leave them out.)

$$\begin{aligned}\lambda_0(X) &= a \cdot \lambda_0 \circ \lambda_B(X) + e \cdot \lambda_0 \circ \lambda_F(X) + \delta \cdot \lambda_0(X) + \delta \cdot \lambda_0(X) \\ &= a \cdot \lambda_0 \circ \lambda_B(X) + e \cdot \lambda_0 \circ \lambda_F(X).\end{aligned}$$

Case 2: $n = 0$, $m > 0$.

$$\begin{aligned}\lambda_0 \circ \lambda_{F^m}(X) &= a \cdot \lambda_0 \circ \lambda_{B^1,F^m}(X) + e \cdot \lambda_0 \circ \lambda_{F^{m+1}}(X) + \delta \cdot \lambda_0 \circ \lambda_{F^m}(X) \\ &\quad + f \cdot \lambda_0 \circ \lambda_{F^{m-1}} \circ \lambda_1(X) \\ &= a \cdot \lambda_0 \circ \lambda_{B^1,F^m}(X) + e \cdot \lambda_0 \circ \lambda_{F^{m+1}}(X) + f \cdot \lambda_0 \circ \lambda_{F^{m-1}}(X).\end{aligned}$$

Case 3: $n > 0$, $m = 0$. Just like case 2.

Case 4: $n > 0$, $m > 0$.

$$\begin{aligned}\lambda_0 \circ \lambda_{B^n,F^m}(X) &= a \cdot \lambda_0 \circ \lambda_{B^{n+1},F^m}(X) + e \cdot \lambda_0 \circ \lambda_{B^n,F^{m+1}}(X) \\ &\quad + b \cdot \lambda_0 \circ \lambda_{B^{n-1},F^m} \circ \lambda_1(X) + f \cdot \lambda_0 \circ \lambda_{B^n,F^{m-1}} \circ \lambda_1(X) \\ &= a \cdot \lambda_0 \circ \lambda_{B^{n+1},F^m}(X) + e \cdot \lambda_0 \circ \lambda_{B^n,F^{m+1}}(X) \\ &\quad + b \cdot \lambda_0 \circ \lambda_{B^{n-1},F^m}(X) + f \cdot \lambda_0 \circ \lambda_{B^n,F^{m-1}}(X).\end{aligned}$$

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 Section 2.8: not every process is definable over $BPA_\delta+\lambda$.

Definition 4.8. We call a process p *boundedly branching* if there is some natural number n such that for every subprocess q of p, 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.)

Lemma 4.9. *Every* $BPA_\delta+\lambda$*-definable process is boundedly branching.*

Proof. In [4], it is proved that every BPA-definable process is boundedly branching. The proof is easy: every subprocess of a process defined by a recursive specification in restricted GNF 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 these steps 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 4.9 with the following result of Bergstra and Klop [4], we have finished the proof of the last claim in Section 2.8:

the solution of the PA-equation $X=a+b\cdot(X\cdot c\,\|\,X\cdot d)$ is not boundedly 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 again gives 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, which strengthens Theorem 4.5, was communicated to us by Vaandrager [12]. 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.)

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

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

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

Now it was shown in [1], that Q cannot be defined over PA. Vaandrager [12] shows that Q can be defined by a linear recursive specification over $\mathrm{BPA}_\delta + \lambda$. 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) $action(out, 0) = \delta$.

Then the following equation yields a queue:

$$Q = \lambda_0(X), \qquad X = a \cdot \lambda_B(X) + e \cdot \lambda_F(X) + out \cdot X.$$

The proof of this fact is along the same lines as the proof of Theorem 4.6; a state Q_σ will correspond to an expression $\lambda_0 \circ \lambda_{\sigma^*}(X)$, where λ_{σ^*} is any sequence of λ-operators, in which each λ_B corresponds to a b in σ, each λ_F corresponds to a f in σ (in the same order), and which further consists of a number of occurrences of λ_1.

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 positively in Section 5.1.

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 $\mathrm{BPA}_\delta + \lambda$ [12].

Acknowledgment

This article is a revision of [2]. We thank an anonymous referee for his/her valuable comments and suggestions for improvements.

References

[1] J.C.M. Baeten and J.A. Bergstra, Global renaming operators over concrete process algebra, *Inform. and Comput.* **78** (3) (1988) 205-245.

[2] J.C.M. Baeten and J.A. Bergstra, Recursive process definitions with the state operator, in: *Proc. CSN 88, CWI*, Amsterdam (1988) 279-294.

[3] J.C.M. Baeten, J.A. Bergstra and J.W. Klop, Decidability of bisimulation equivalence for processes generating context-free languages, in: J.W. de Bakker, A.J. Nijman and P.C. Treleaven, eds., *Proc. PARLE, Vol. II (Parallel Languages)*, Eindhoven, Lecture Notes in Computer Science **259** (Springer, Berlin, 1987) 94-113.

[4] J.A. Bergstra and J.W. Klop, The algebra of recursively defined processes and the algebra of regular processes, in: J. Paredaens, ed., *Proc. 11th ICALP*, Antwerpen, Lecture Notes in Computer Science **172** (Springer, Berlin, 1984) 82-94.

[5] J.A. Bergstra and J.W.Klop, Process algebra for synchronous communication, *Inform. and Control* **60**(1/3) (1984) 109-137.

[6] R.J. van Glabbeek, Bounded nondeterminism and the approximation induction principle in process algebra, in: F.J. Brandenburg, G. Vidal-Naquet and M. Wirsing, eds., *Proc. STACS*, Passau, Lecture Notes in Computer Science **247** (Springer, Berlin, 1987) 336-347.

[7] C.A.R. Hoare, *Communicating Sequential Processes* (Prentice Hall, Englewood Cliffs, NJ, 1985).

[8] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science **92** (Springer, Berlin, 1980).

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

[10] SPECS Consortium/PTT-RNL, Definition of MR, version 1, SPECS document D.WP5.2, 1989.

[11] F.W. Vaandrager, Process algebra semantics of POOL, report CS-R8629, Centre for Mathematics and Computer Science, Amsterdam, 1986; in: J.C.M. Baeten, ed., *Applications of Process Algebra* (Cambridge University Press, Cambridge, 1990) 173-236.

[12] F.W. Vaandrager, personal communication, January 1988.