ACP_τ

A Universal Axiom System for Process Specification

J.A. Bergstra
University of Amsterdam, Department of Computer Science
P.O. Box 19268, 1000 GG Amsterdam
State University of Utrecht, Department of Philosophy
P.O. Box 8810, 3508 TA Utrecht

J.W. Klop
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam
Free University, Department of Mathematics and Computer Science
De Boelelaan 1081, 1081 HV Amsterdam

0. INTRODUCTION

Following R. Milner's development of his widely known Calculus of Communicating Systems, there have been in the last decade several approaches to *process algebra,* i.e. the algebraic treatment of communicating processes. In this paper we give a short and informal presentation of some developments in process algebra which started five years ago at the Centre for Mathematics and Computer Science, and since two years in cooperation with the University of Amsterdam and the State University of Utrecht[1]. Most of the present paper can be found in the more complete survey [6], where the subjects of specification and verification of processes are treated in so-called bisimulation semantics. Here, we adopt a further restriction by concentrating on the specification issue.

We start with a very simple axiom system for processes called Basic Process Algebra, in which no communication facilities are present. This system is interesting not only because it is a nucleus for all process axiom systems that are devised and analyzed in the 'Algebra of Communicating Processes', but also because it provides a link with the classical and successful theory of formal languages, in particular where regular languages and context-free languages are concerned. In Section 2 we explain this link.

Next, we introduce more and more operators, leading first to the axiom system ACP (Algebra of Communicating Processes) where communication between processes is possible, and finally to ACP_τ (Algebra of Communicating Processes with abstraction). Examples are given showing that the successive extensions yield more and more specification power; and a culmination point

[1] Research partially supported by ESPRIT project 432, Meteor.

is the Finite Specification Theorem for ACP_τ, stating that every finitely branching, effectively presented process can be specified in ACP_τ by a finite system of recursion equations. Of course, an algebraic system for processes is only really interesting and useful if also sufficient facilities for process *verification* are present. These require an extension with some infinitary proof rules which will not be discussed here (for these, see the full version of this paper [6]). We refer also to the same paper for a more extensive list of references than the one below.

1. Basic Process Algebra

The kernel of all axiom systems for processes that we will consider, is Basic Process Algebra. Not only is for that reason an analysis of BPA and its models worth-while, but also because it presents a new angle on some old questions in the theory of formal languages, in particular about context-free languages and deterministic push-down automata. First let us explain what is meant by 'processes'.

The processes that we will consider are capable of performing atomic steps or *actions* $a,b,c, \ldots ,$ with the idealization that these actions are events without positive duration in time; it takes only one moment to execute an action. The actions are combined into composite processes by the operations $+$ and $\cdot$, with the interpretation that $(a+b)\cdot c$ is the process that first chooses between executing a or b and, second, performs the action c after which it is finished. At this stage it does not matter how the choice is made. These operations, *alternative composition* and *sequential composition* (or just sum and product), are the basic constructors of processes. Since time has a direction, multiplication is not commutative; but addition is, and in fact it is stipulated that the options (summands) possible at some stage of the process form a *set.* Formally, we will require that processes $x,y,z, \ldots$ satisfy the following axioms (where the product sign is suppressed):

BPA
$x+y=y+x$
$(x+y)+z=x+(y+z)$
$x+x=x$
$(x+y)z=xz+yz$
$(xy)z=x(yz)$

Table 1

In the Introduction we used the term 'process algebra' in the generic sense of denoting the area of algebraic approaches to concurrency, but we will also adopt the following technical meaning for it: any model of these axioms will be a *process algebra.* The simplest process algebra is the *term model* of BPA, whose elements are BPA-expressions (built from the atoms $a,b,c, \ldots$ by means of the basic constructors) modulo the equality generated by the axioms. The term model itself (let us call it $\mathbb{T}$) is not very exciting: it contains only finite processes. In order to specify also infinite processes, we introduce *recursion variables* $X, Y, Z, \ldots$. Using these, one can specify the process $aaaaaa \cdots$ (performing infinitely many consecutive a-steps) by the recursion equation $X = aX$; indeed, by 'unwinding' we have $X = aX = aaX = aaaX = \cdots$. In general, we will admit simultaneous recursion, i.e. systems of recursion equations. A non-trivial example is the following specification of the process behaviour of a Stack with data 0,1:

STACK
$X = 0{\downarrow}.YX + 1{\downarrow}.ZX$
$Y = 0{\uparrow} + 0{\downarrow}.YY + 1{\downarrow}.ZY$
$Z = 1{\uparrow} + 0{\downarrow}.YZ + 1{\downarrow}.ZZ$

TABLE 2

Here $0{\downarrow}$ and $0{\uparrow}$ are the actions 'push 0' and 'pop 0', respectively; likewise for 1. Now Stack is specified by the first recursion variable, X. Indeed, according to the first equation the process X is capable of performing either the action $0{\downarrow}$, after which the process is transformed into YX, or $1{\downarrow}$, after which the process is transformed into ZX. In the first case we have, using the second equation, $YX = (0{\uparrow} + 0{\downarrow}.YY + 1{\downarrow}.ZY)X = 0{\uparrow} \cdot X + 0{\downarrow}.YYX + 1{\downarrow}.ZYX$. This means that the process YX has three options; after performing the first one ($0{\uparrow}$) it behaves like the original X. Continuing in this manner we find a transition diagram or *process graph* as in Figure 1.

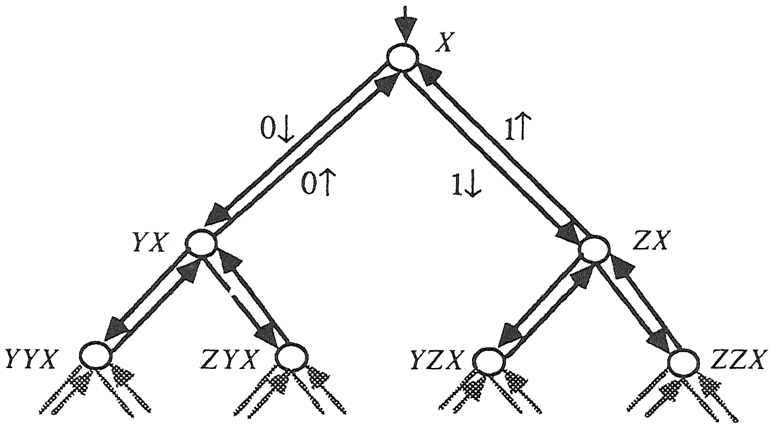

FIGURE 1. Stack

It is not hard to imagine how such a process graph (a rooted, directed, connected, labeled graph) can be associated with a system of recursion equations; we will not give a formal definition here. Actually, one can use such process graphs and build various models *(graph models)* for BPA from them; this will be discussed now.

2. Graph models for BPA

Let $\mathbb{G}$ be the set of all at most countably branching process graphs $g,h, \ldots$ over the action alphabet $A=\{a,b,c, \ldots\}$. (I.e. a node in such a graph may have at most countably many one-step successors.) On $\mathbb{G}$ we define operations $+$ and $\cdot$ as follows: $g \cdot h$ is the result of appending (the root of) h at each termination node of g, and $g+h$ is the result of identifying the roots of g and h. (To be more precise, we first have to unwind g and h a little bit so as to make their roots 'acyclic', otherwise the sum would not have the intended interpretation of making an irreversible choice.) Letting $\mathbf{a}$ be the graph consisting of a single arrow with label a, we now have a structure $\mathcal{G}=\mathbb{G}(+,\cdot,\mathbf{a},\mathbf{b},\mathbf{c}, \ldots)$ which corresponds to the signature of BPA. But it is not a model of BPA. For instance the law $x+x=x$ does not hold in $\mathcal{G}$, since $\mathbf{a}+\mathbf{a}$ is not the same as $\mathbf{a}$; the former is a graph with two arrows and the latter has one arrow.

Here we need the fundamental notion of D. Park (see [13]), called *bisimulation equivalence* or *bisimilarity.* Two graphs g and h are bisimilar if there is a matching between their nodes (i.e. a binary relation with domain the set of nodes of g, and codomain the set of nodes of h) such that (1) the roots are matched; (2) if nodes s,t in g,h respectively are matched and an a-step is possible from s to some s' then in h an a-step is possible from t to some t' *such that s' and t' again are matched* ; (3) likewise with the roles of g,h reversed. A matching satisfying (1-3) is a bisimulation. An example is given in Figure 2, where (part of) the matching is explicitly displayed; another example is given in Figure 3 where the matching is between each pair of nodes on the same horizontal level.

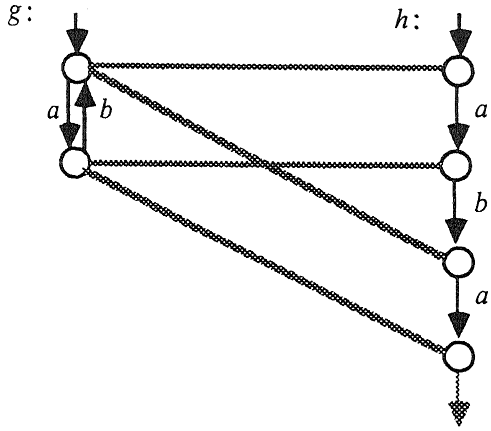

Figure 2

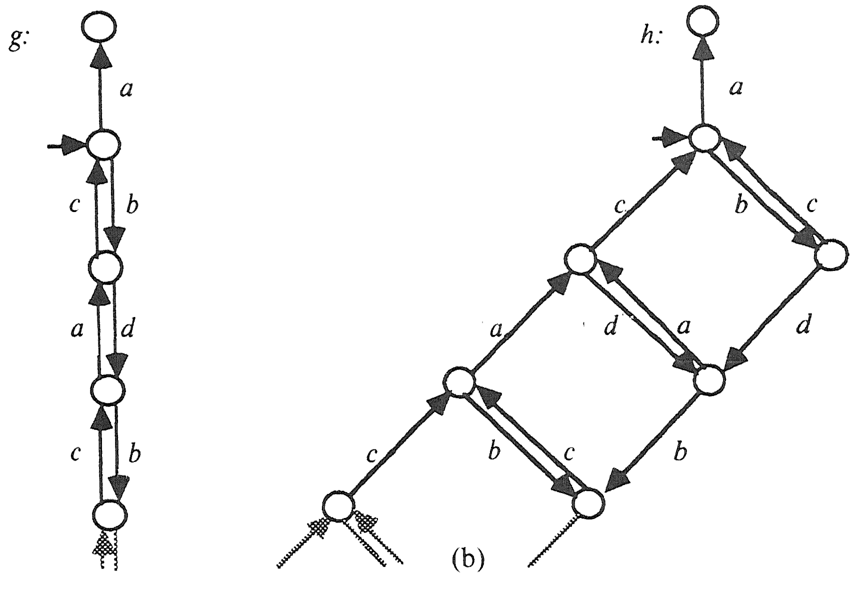

FIGURE 3

We use the notation $g \underline{\leftrightarrow} h$ to express that g and h are bisimilar. Now one proves that $\underline{\leftrightarrow}$ is not only an equivalence on $\mathbb{G}$, but even a congruence on $\mathcal{G}$. Thus the quotient $\mathbb{G} = \mathcal{G} / \underline{\leftrightarrow}$ is well-defined, and it is a model of BPA. ($\mathbb{G}$ has constants $a = \mathbf{a} / \underline{\leftrightarrow}$ etc., and operations $+, \cdot$ defined by $g + h = (\mathbf{g} + \mathbf{h}) / \underline{\leftrightarrow}$ for $g = \mathbf{g} / \underline{\leftrightarrow}$ and $h = \mathbf{h} / \underline{\leftrightarrow}$; likewise for $\cdot$. (For typographical reasons we will not distinguish between the syntactic $+, \cdot$ and the semantic $+, \cdot$ in our notation.)

Even more, $\mathbb{G}$ is a very nice model of BPA: all systems of recursion equations in the syntax of BPA have a solution in $\mathbb{G}$, and systems of *guarded* recursion equations like in Table 1 have moreover a *unique* solution. 'Guarded' means that in the right-hand sides of the recursion equations no recursion variable can be accessed without passing an atomic action. (E.g. $X = a + X$ is not a guarded equation; it has many solutions: $a + b$, $a + c$, ...)

Some submodels (all satisfying the axioms of BPA) of $\mathbb{G}$ are of interest: $\mathbb{G}_{fb}$, built from *finitely branching* process graphs; $\mathbb{R}$, built from finite (but possibly cycle-containing) graphs; and $\mathbb{F}$, built from finite and acyclic graphs. Also $\mathbb{G}_{fb}$ has the property of providing unique solutions for systems of guarded recursion equations. Without the condition of guardedness, there need not be solutions. E.g. the equation

$$X = Xa + a$$

cannot be solved in $\mathbb{G}_{fb}$. In the model $\mathbb{R}$ of regular processes one can always find unique solutions for guarded recursion equations provided they are *linear*, that is, the expressions (terms) in the equations may only be built by sum and a restricted form of product called *prefix multiplication* $a \cdot s$ ('a' an atom, s a general expression) which excludes products of recursion variables as in Table 1. For a complete proof system for regular processes, see [11].

EXAMPLE

$$\{X = aX + bY, Y = cX + dY\}$$

is a linear system;

$$\{X = aXX + bY, Y = cX + dYXY\}$$

is not.

The model $\mathbb{R}$ contains the *finite-state* processes; hence the notation $\mathbb{R}$ for 'regular' as in formal language theory. Finally, $\mathbb{F}$ contains only finite processes and is in fact isomorphic to the term model $\mathbb{T}$.

Some systems of recursion equations should be taken as equivalent. Clearly, $X = aX$ and $X = aaX$ specify the same process in $\mathbb{G}$. Less clearly, the two systems

$$E_1 = \{X = a + bYX,\ Y = c + dXY\}$$

$$E_2 = \{X = a + bU,\ U = cX + dZX,\ Y = c + dZ,\ Z = aY + bUY\}$$

are equivalent in this sense: E_1 specifies the process graph in Figure 3a above, and E_2 specifies the graph in Figure 3b. Moreover, as we already saw, these two graphs are bisimilar. So E_1 and E_2 denote the same process in $\mathbb{G}$. So the question arises: *Is equivalence of recursion equations over* BPA, *relative to the graph model* $\mathbb{G}$, *decidable?* At the moment this question is wide open. There is an interesting connection here with *context-free languages*, as follows.

A guarded system of recursion equations over BPA corresponds in an obvious way (for details see [2]) to a context-free grammar (CFG) in Greibach Normal Form, and vice versa. Hence each context-free language (CFL) can be obtained as the set of finite traces of a process in $\mathbb{G}$ denoted by a system of guarded recursion equations. (A finite trace is the word obtained by following a path from the root to a termination node.) In fact, to generate a CFL it is sufficient to look at certain restricted systems of recursion equations called 'normed'. A system is normed if in every state (of the corresponding process) there is a possibility to terminate. E.g. $X = aX$ is not normed, but $X = b + aX$ is. There is a simple syntactical check to determine whether a system is normed or not. Clearly, the property 'normed' also pertains to process graphs. In [2] it is proved that the equivalence problem stated above is solvable for such normed systems. This is rather surprising in view of the well-known fact that the equality problem for CFLs is unsolvable. The point is that the process semantics in $\mathbb{G}$ of a CFG bears much more information than the trace set semantics, which is an abstraction from the process semantics.

The link with deterministic context-free languages resides in the following observation from [2]:

THEOREM 2.1. *Let $g,h \in \mathrm{G}$ be two normed and deterministic process graphs. Then $g \leftrightarrow h$ iff g and h have the same sets of finite traces.*

Here a graph is 'deterministic' if two arrows leaving the same node always have different label. The CFL (i.e. the set of finite traces) determined by a normed and deterministic graph, corresponding to a system of guarded recursion equations in BPA, is known as a *simple* CFL; the simple CFLs form a proper subclass of the deterministic CFLs.

Summarizing, we can state that BPA and its graph model obtained via the concept of bisimulation provide a new angle on some problems in the theory of formal languages, concerned with context-free languages. Here we think especially of *deterministic* context-free languages (DCFLs), obtained by deterministic push-down automata, with the well-known open problem whether the equality problem for DCFLs is solvable. Thus, even in the absence of the many operators for parallellism, abstraction etc. which are still to be introduced below, we have in BPA and its models an interesting theory with potential implications for the DCFL problem.

3. DEADLOCK

After the excursion to semantics in the preceding section we return to the development of more syntax for processes. A vital element in the present setup of process algebra is the process δ, signifying 'deadlock'. The process ab performs its two steps and then terminates, succesfully; but the process $ab\delta$ *deadlocks* after the a- and b-action: it wants to do a proper (i.e. non-δ) action but it cannot. So δ is the acknowledgement of stagnation. With this in mind, the axioms to which δ is subject, may be clear:

DEADLOCK
$\delta + x = x$
$\delta \cdot x = \delta$

TABLE 3

The axiom system of BPA (Table 1) together with the present axioms for δ is called BPA$_\delta$. We are now in a position to motivate the absence in BPA of the 'other' distributive law: $z(x+y)=zx+zy$. For, suppose it would be added. Then $ab=a(b+\delta)=ab+a\delta$. This means that a process with deadlock possibility is equal to one without, conflicting with our intention to model also deadlock behaviour of processes.

The essential role of the new process δ will only be fully appreciated after the introduction of communication, below.

4. The merge operator

If x,y are processes, their 'parallel composition' $x \| y$ is the process that first chooses whether to do a step in x or in y, and proceeds as the parallel composition of the remainders of x,y. In other words, the steps of x,y are interleaved or merged. Using an auxiliary operator $\mathbin{|\!\!\lfloor}$ (with the interpretation that $x \mathbin{|\!\!\lfloor} y$ is like $x \| y$ but with the commitment of choosing the initial step from x) the operation $\|$ can be succinctly defined by the axioms:

FREE MERGE
$x \| y = x \mathbin{|\!\!\lfloor} y + y \mathbin{|\!\!\lfloor} x$
$ax \mathbin{|\!\!\lfloor} y = a(x \| y)$
$a \mathbin{|\!\!\lfloor} y = ay$
$(x+y) \mathbin{|\!\!\lfloor} z = x \mathbin{|\!\!\lfloor} z + y \mathbin{|\!\!\lfloor} z$

Table 4

The system of nine axioms consisting of BPA and the four axioms for merge will be called PA. Moreover, if the axioms for δ are added, the result will be PA_δ. The operators $\|$ and $\mathbin{|\!\!\lfloor}$ will also be called *merge* and *left-merge* respectively.

The merge operator corresponds to what in the theory of formal languages is called *shuffle.* The shuffle of the words *ab* and *cd* is the set of words $\{abcd,\ acbd,\ cabd,\ acdb,\ cadb,\ cdab\}$. Merging the processes *ab* and *cd* yields the process

$$\begin{aligned} ab \| cd &= ab \mathbin{|\!\!\lfloor} cd + cd \mathbin{|\!\!\lfloor} ab = a(b \| cd) + c(d \| ab) \\ &= a(b \mathbin{|\!\!\lfloor} cd + cd \mathbin{|\!\!\lfloor} b) + c(d \mathbin{|\!\!\lfloor} ab + ab \mathbin{|\!\!\lfloor} d) \\ &= a(bcd + c(d \| b)) + c(dab + a(b \| d)) \\ &= a(bcd + c(db + bd)) + c(dab + a(bd + db)), \end{aligned}$$

a process having as trace set the shuffle above.

An example of a process recursively defined in PA, is $X = a(b \| X)$. It turns out that this process can already be defined in BPA, by the system of recursion equations

$$\{X = aYX,\ Y = b + aYY\}.$$

To see that both ways of defining X yield the same process, one may 'unwind' according to the given equations:

$$\begin{aligned} X &= a(b \| X) = a(b \mathbin{|\!\!\lfloor} X + X \mathbin{|\!\!\lfloor} b) = a(bX + a(b \| X) \mathbin{|\!\!\lfloor} b) \\ &= a(bX + a((b \| X) \| b)) \\ &= a(bX + a \ldots), \end{aligned}$$

while on the other hand

$$X = aYX = a(b + aYY)X = a(bX + aYYX) = a(bX + a\ \ldots).$$

So at least up to level 2 the processes are equal. By further unwinding they can be proved equal up to each finite level.

Yet there are processes definable in PA but not in BPA. An example (from [4]) of such a process is given by the recursion equation

$$X = 0{\downarrow}\cdot(0{\uparrow} \| X) + 1{\downarrow}\cdot(1{\uparrow} \| X)$$

describing the process behaviour of a Bag (or multiset), in which arbitrarily many instances of the data 0,1 can be inserted (the actions $0{\downarrow}, 1{\downarrow}$ respectively) or retrieved ($0{\uparrow}, 1{\uparrow}$), with the restriction that no more 0's and 1's can taken from the Bag than were put in first. The difference with a Stack or a Queue is that all order between incoming and outgoing 0's and 1's is lost. The process graph corresponding to the process Bag is as in Figure 4.

We conclude this section on PA by mentioning the following fact (see [4]), which is useful for establishing non-definability results:

THEOREM 4.1. *Every process which is recursively defined in PA and has an infinite trace, has an eventually periodic trace.*

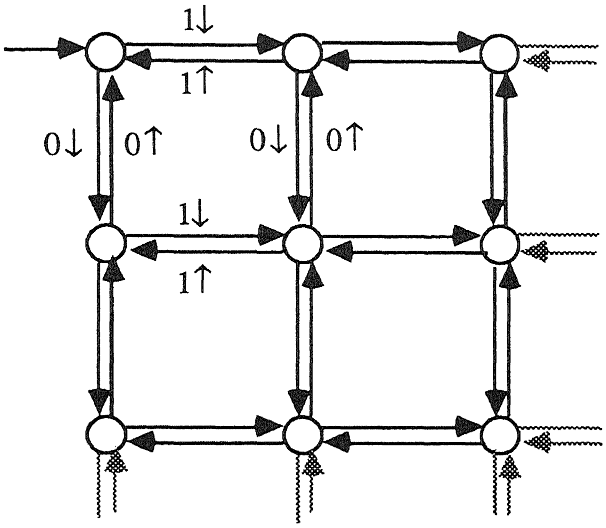

FIGURE 4. Bag

5. Communication

So far, the parallel composition or merge ($\Vert$) did not involve communication in the process $x \Vert y$: one could say that x and y are 'freely' merged or interleaved. However, some actions in one process may need an action in another process for an actual execution, like the act of shaking hands requires simultaneous acts of two persons. In fact, 'handshaking' is the paradigm for the type of communication which we will introduce now. If $A = \{a,b,c, \ldots, \delta\}$ is the action alphabet, let us adopt a binary communication function $\mid : A \times A \rightarrow A$ satisfying the axioms in Table 5.

COMMUNICATION FUNCTION
$a \mid b = b \mid a$
$(a \mid b) \mid c = a \mid (b \mid c)$
$\delta \mid a = \delta$

TABLE 5

Here a,b vary over A, including δ. We can now specify *merge with communication* ; we use the same notation $\Vert$ as for the 'free' merge in Section 4 since in fact 'free' merge is an instance of merge with communication by choosing the communication function trivial, i.e. $a \mid b = \delta$ for all $a,b \in A$. There are now two auxiliary operators, allowing a finite axiomatisation: left-merge ($\lfloor\!\!\lfloor$) as before and $\mid$ *(communication merge* or simply 'bar'), which is an extension of the communication function in Table 5 to all processes, not only the atoms. The axioms for $\Vert$ and its auxiliary operators are given in Table 6.

MERGE WITH COMMUNICATION
$x \Vert y = x \lfloor\!\!\lfloor y + y \lfloor\!\!\lfloor x + x \mid y$
$ax \lfloor\!\!\lfloor y = a(x \Vert y)$
$a \lfloor\!\!\lfloor y = ay$
$(x+y) \lfloor\!\!\lfloor z = x \lfloor\!\!\lfloor z + y \lfloor\!\!\lfloor z$
$ax \mid b = (a \mid b)x$
$a \mid bx = (a \mid b)x$
$ax \mid by = (a \mid b)(x \Vert y)$
$(x+y) \mid z = x \mid z + y \mid z$
$x \mid (y+z) = x \mid y + x \mid z$

TABLE 6

We also need the so-called *encapsulation* operators ∂_H (for every $H \subseteq A$) for removing unsuccessful attempts at communication:

ENCAPSULATION
$\partial_H(a)=a$ if $a \notin H$
$\partial_H(a)=\delta$ if $a \in H$
$\partial_H(x+y)=\partial_H(x)+\partial_H(y)$
$\partial_H(xy)=\partial_H(x)\cdot\partial_H(y)$

TABLE 7

These axioms express that ∂_H 'kills' all atoms mentioned in H, by replacing them with δ. The axioms for BPA, DEADLOCK together with the present ones in Tables 5-7 constitute the axiom system ACP (Algebra of Communicating Processes). Typically, a system of communicating processes $x_1, \ldots, x_n$ is now represented in ACP by the expression $\partial_H(x_1 \| \cdots \| x_n)$. Prefixing the encapsulation operator says that the system $x_1, \ldots, x_n$ is to be perceived as a separate unit with respect to the communication actions mentioned in H; no communications between actions in H with an environment are expected or intended.

A useful theorem to break down such expressions is the *Expansion Theorem* (first formulated by Milner, for the case of CCS; see [12]) which holds under the assumption of the *handshaking axiom* $x|y|z=\delta$. This axiom says that all communications are binary. (In fact we have to require associativity of '$\|$' first - see Table 8.)

THEOREM 5.1 (EXPANSION THEOREM).

$$x_1 \| \cdots \| x_k = \sum_i x_i \mathbin{|\!\lfloor} X_k^i + \sum_{i \neq j} (x_i | x_j) \mathbin{|\!\lfloor} X_k^{i,j}$$

Here X_k^i denotes the merge of $x_1, \ldots, x_k$ except x_i, and $X_k^{i,j}$ denotes the same merge except x_i, x_j $(k \geqslant 3)$. For instance, for $k=3$:

$$x \| y \| z = x \mathbin{|\!\lfloor} (y \| z) + y \mathbin{|\!\lfloor} (x \| x) + z \mathbin{|\!\lfloor} (x \| y) + (y|z) \mathbin{|\!\lfloor} x + (z|x) \mathbin{|\!\lfloor} y + (x|y) \mathbin{|\!\lfloor} z.$$

In order to prove the Expansion Theorem, one first proves by simultaneous induction on term complexity that for all closed ACP-terms (i.e. ACP-terms without free variables) the following *axioms of standard concurrency* hold:

AXIOMS OF STANDARD CONCURRENCY
$(x \mathbin{\|\!\!\lfloor} y)\mathbin{\|\!\!\lfloor} z = x \mathbin{\|\!\!\lfloor}(y \| z)$
$(x \mid y)\mathbin{\|\!\!\lfloor} z = x \mid (y \mathbin{\|\!\!\lfloor} z)$
$x \mid y = y \mid x$
$x \| y = y \| x$
$x \mid (y \mid z) = (x \mid y) \mid z$
$x \| (y \| z) = (x \| y) \| z$

TABLE 8

As in Section 2 one can construct graph models $\mathbb{G},\mathbb{G}_{fb},\mathbb{R},\mathbb{F}$ for ACP; in these models the axioms in Table 8 are valid. We will discuss the construction of these models in Section 7. (It is however also possible to construct 'non-standard' models of ACP in which these axioms do not hold. We will not be interested in such pathological models.)

The defining power of ACP is strictly greater than that of PA. The following is an example (from [4]) of a process U, recursively defined in ACP, but not definable in PA: let the alphabet be $\{a,b,c,d,\delta\}$ and let the communication function be given by $c|c=a$, $d|d=b$, and all other communications equal to δ. Let $H=\{c,d\}$. Now we recursively define the process U as in Table 9:

$U=\partial_H(dcY \| Z)$
$X=cXc+d$
$Y=dXY$
$Z=dXcZ$

TABLE 9

Then, we claim, $U=ba(ba^2)^2(ba^3)^2(ba^4)^2 \cdots$. Indeed, using the axioms in ACP and putting

$$U_n=\partial_H(dc^n Y \| Z)$$

for $n \geqslant 1$, a straightforward computation shows that

$$U_n = ba^n ba^{n+1} U_{n+1}.$$

By Theorem 4.1, U is not definable in PA, since the one infinite trace of U is not eventually periodic.

We will often adopt a special format for the communication function, called *read-write communication*. Let a finite set D of *data* d and a set $\{1, \ldots, p\}$ of *ports* be given. Then the alphabet consists of *read* actions $ri(d)$ and *write* actions $wi(d)$, for $i=1,\ldots,p$ and $d \in D$. The interpretation is: read datum d at port i, write datum d at port i respectively. Furthermore, the alphabet contains actions $ci(d)$ for $i=1,\ldots,p$ and $d \in D$, with interpretation: *communicate* *d at i*. These actions will be called *transactions*. The only non-trivial communications (i.e. not resulting in δ) are: $wi(d)|\ ri(d) = ci(d)$. Instead of $wi(d)$ we will also use the notation $si(d)$ (send d along i). Note that read-write communication satisfies the handshaking axiom: all communications are binary.

EXAMPLE 5.1.
Using the present read-write communication format we can write the recursion equation for a Bag B_{12} (cf. Section 4) which reads data $d \in D$ at port 1 and writes them at port 2 as follows:

$$B_{12} = \sum_{d \in D} r1(d)(w2(d) \| B_{12}).$$

6. ABSTRACTION

A fundamental issue in the design and specification of hierarchical (or modularized) systems of communicating processes is *abstraction*. Without having an abstraction mechanism enabling us to abstract from the inner workings of modules to be composed to larger systems, specification of all but very small systems would be virtually impossible. We will now extend the axiom system ACP, obtained thus far, with such an abstraction mechanism.

Consider two Bags B_{12}, B_{23} (cf. Example 5.1) with action alphabets $\{r1(d), s2(d) | d \in D\}$ and $\{r2(d), s3(d) | d \in D\}$, respectively. That is, B_{12} is a bag-like channel reading data d at port 1, sending them to port 2; B_{23} reads data at 2 and sends them to 3. (That the channels are bags means that, unlike the case of a queue, the order of incoming data is lost in the transmission.) Suppose the bags are connected at port 2; so we adopt communications $s2(d)|r2(d) = c2(d)$ where $c2(d)$ is the transaction of d at 2.

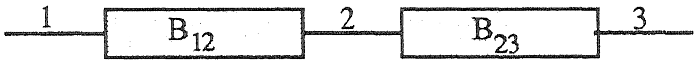

FIGURE 5. Transparent Bag $\mathbb{B}_{13}$

The composite system $\mathbb{B}_{13} = \partial_H(B_{12} \| B_{23})$ where $H = \{s2(d), r2(d) | d \in D\}$, should, intuitively, be again a Bag between ports 1,3. However, from some (rather involved) calculations we learn that

$$\mathbb{B}_{13} = \sum_{d \in D} r1(d) \cdot (c2(d) \cdot s3(d)) \| \mathbb{B}_{13}).$$

So $\mathbb{B}_{13}$ is a 'transparent' Bag: the passage of d through 2 is visible as the transaction event $c2(d)$. (Note that this terminology conflicts with the usual one in the area of computer networks, where a network is called transparent if the internal structure is *not* visible.)

How can we *abstract* from such internal events, if we are only interested in the external behaviour at 1,3? The first step to obtain such an abstraction is to remove the distinctive identity of the actions to be abstracted, that is, to rename them all into one designated action which we call, after Milner, τ: the *silent* action. This renaming is realised by the *abstraction operator* τ_I, parameterized by a set of actions $I \subseteq A$ and subject to the following axioms:

ABSTRACTION
$\tau_I(\tau)=\tau$
$\tau_I(a)=a$ if $a \notin I$
$\tau_I(a)=\tau$ if $a \in I$
$\tau_I(x+y)=\tau_I(x)+\tau_I(y)$
$\tau_I(xy)=\tau_I(x)\cdot\tau_I(y)$

TABLE 10

The second step is to attempt to devise axioms for the silent step τ by means of which τ can be removed from expressions, as e.g. in the equation $a\tau b=ab$. However, it is not possible to remove *all* τ*'s* in an expression if one is interested in a faithful description of deadlock behaviour of processes (at least in bisimulation semantics, the framework adopted in this paper). For, consider the process (expression) $a+\tau\delta$; this process can deadlock, namely if it chooses to perform the silent action. Now, if one would propose naively the equations $\tau x = x\tau = x$, then $a+\tau\delta = a+\delta=a$, and the latter process has no deadlock possibility. It turns out that one of the proposed equations, $x\tau = x$, can be safely adopted, but the other one is wrong. Fortunately, R. Milner has devised some simple axioms which give a complete description of the properties of the silent step (complete with respect to a certain semantical notion of process equivalence called $r\tau\delta$-bisimulation, which does respect deadlock behaviour; this notion is discussed below), as follows.

SILENT STEP
$x\tau=x$
$\tau x=\tau x+x$
$a(\tau x+y)=a(\tau x+y)+ax$

TABLE 11

To return to our example of the 'transparent' Bag $\mathbb{B}_{13}$, after abstraction of the set of transactions $I = \{c\,2(d) | d \in D\}$ the result is indeed an 'ordinary' Bag:

$$\tau_I(\mathbb{B}_{13})=\tau_I(\sum_{d\in D} r\,1(d)(c\,2(d)\cdot s\,3(d)\|\mathbb{B}_{13})) \qquad (\star)$$

$$=\sum_{d\in D} r\,1(d)(\tau\cdot s\,3(d)\|\tau_I(\mathbb{B}_{13}))=\sum_{d\in D}(r\,1(d)\cdot\tau\cdot s\,3(d))\mathbin{\lfloor\!\lfloor}\tau_I(\mathbb{B}_{13})$$

$$=\sum_{d\in D}(r\,1(d)\cdot s\,3(d))\mathbin{\lfloor\!\lfloor}\tau_I(\mathbb{B}_{13})=\sum_{d\in D} r\,1(d)(s\,3(d)\|\tau_I(\mathbb{B}_{13}))$$

from which it follows that $\tau_I(\mathbb{B}_{13}) = B_{13}$ (**), the Bag defined by

$$B_{13} = \sum_{d \in D} r1(d)(s3(d) \| B_{13}).$$

Here we were able to eliminate all silent actions, but this will not always be the case. For instance, 'chaining' two Stacks instead of Bags as in Figure 5 yields a process with 'essential' τ-steps. Likewise for a Bag followed by a Stack. (Here 'essential' means: non-removable in bisimulation semantics.) In fact, the computation above is not as straightforward as was suggested: to justify the equations marked with (*) and (**) we need additional proof principles. As to (**), this equation is justified by the *Recursive Specification Principle* (RSP) stating that a *guarded system of recursion equations in which no abstraction operator τ_I appears, has a unique solution.* We will not discuss the justification of equation (*) here. The justification of a principle like RSP is that it is valid in all 'sensible' models of our axioms; however note that for formal computations one has to postulate such a principle explicitly.

Combining all the axioms presented above in Tables 1,3,4,5,6,7,10,11 and a few axioms specifying the interaction between τ and communication merge $|$, we have arrived at the system ACP$_\tau$, *Algebra of Communicating Processes with abstraction* (see Table 12).

Actually, in spite of our restriction to specification of processes as stated in the Introduction, the last computation concerned a very simple process *verification,* showing that the combined system has the desired external behaviour of a Bag. Abstraction, realized in ACP$_\tau$ by the abstraction operator and the silent process τ, clearly is of crucial importance for process verification. But also for process specification abstraction is important. Let $f:\mathbb{N} \rightarrow \{a,b\}$ be a sequence of symbols a,b, and let p_f be the proces $f(0) \cdot f(1) \cdot f(2) \ldots$, that is, the unique solution of the infinite system of recursion equations $\{X_n = f(n) \cdot X_{n+1} | n \geqslant 0\}$. Now we have:

THEOREM 6.1. *There is a computable function f such that process p_f is not definable by a finite system of recursion equations in* ACP$_\tau$ *without abstraction operator.*

On the other hand, according to the Finite Specification Theorem 8.1, every process p_f with computable f *is* definable by a finite system of recursion equations in full ACP$_\tau$.

ACP_τ			
$x+y=y+x$	A1	$x\tau=x$	T1
$x+(y+z)=(x+y)+z$	A2	$\tau x+x=\tau x$	T2
$x+x=x$	A3	$a(\tau x+y)=a(\tau x+y)+ax$	T3
$(x+y)z=xz+yz$	A4		
$(xy)z=x(yz)$	A5		
$x+\delta=x$	A6		
$\delta x=\delta$	A7		
$a\|b=b\|a$	C1		
$(a\|b)\|c=a\|(b\|c)$	C2		
$\delta\|a=\delta$	C3		
$x\\|y=x \mathbin{\\|\\!\\_} y+y \mathbin{\\|\\!\\_} x+x\|y$	CM1		
$a \mathbin{\\|\\!\\_} x=ax$	CM2	$\tau \mathbin{\\|\\!\\_} x=\tau x$	TM1
$ax \mathbin{\\|\\!\\_} y=a(x\\|y)$	CM3	$\tau x \mathbin{\\|\\!\\_} y=\tau(x\\|y)$	TM2
$(x+y) \mathbin{\\|\\!\\_} z=x \mathbin{\\|\\!\\_} z+y \mathbin{\\|\\!\\_} z$	CM4	$\tau\|x=\delta$	TC1
$ax\|b=(a\|b)x$	CM5	$x\|\tau=\delta$	TC2
$a\|bx=(a\|b)x$	CM6	$\tau x\|y=x\|y$	TC3
$ax\|by=(a\|b)(x\\|y)$	CM7	$x\|\tau y=x\|y$	TC4
$(x+y)\|z=x\|z+y\|z$	CM8		
$x\|(y+z)=x\|y+x\|z$	CM9	$\partial_H(\tau)=\tau$	DT
		$\tau_I(\tau)=\tau$	TI1
$\partial_H(a)=a$ if $a\notin H$	D1	$\tau_I(a)=a$ if $a\notin I$	TI2
$\partial_H(a)=\delta$ if $a\in H$	D2	$\tau_I(a)=\tau$ if $a\in I$	TI3
$\partial_H(x+y)=\partial_H(x)+\partial_H(y)$	D3	$\tau_I(x+y)=\tau_I(x)+\tau_I(y)$	TI4
$\partial_H(xy)=\partial_H(x)\cdot\partial_H(y)$	D4	$\tau_I(xy)=\tau(x)\cdot\tau_I(y)$	TI5

TABLE 12

7. GRAPH MODELS FOR ACP_τ

We will now construct graph models for ACP_τ, in analogy with the construction of these models for BPA in Section 2. Again we start with a domain of at most countably branching process graphs **G**, the only difference being that arrows may now also bear label τ and δ. (By abuse of language we use the same notation G.) Next, we define on **G** in addition to $+,\cdot$ operations $\|, \mathbin{\|\!\_}, |, \tau_I, \partial_H$ corresponding to the syntactic operations $\|, \mathbin{\|\!\_}, |, \tau_I, \partial_H$. We will only discuss the definition of the first operation $\|$. Let **ab** and **cd** be two process graphs as in Figure 6, and suppose there are communications $a|d=f$ and $b|c=k$, all other communications being trivial (i.e. resulting in δ). Then **ab**$\|$**cd** is the process graph indicated in Figure 6, a cartesian product with diagonal edges for the successful communications.

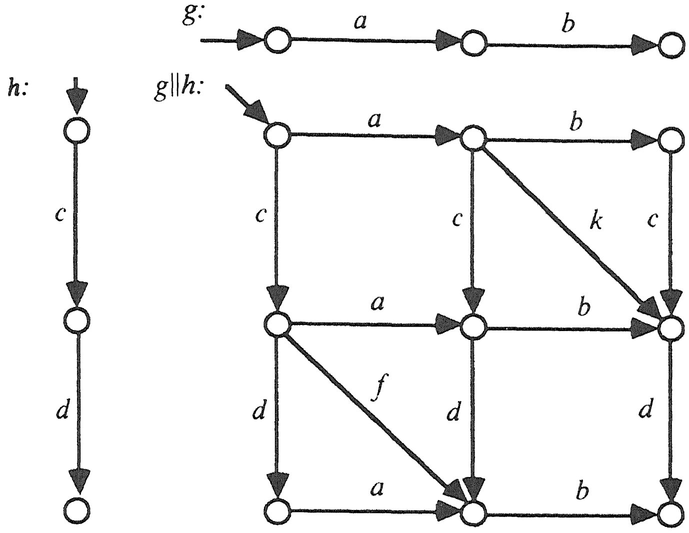

FIGURE 6

We now have a structure $\mathcal{G}=\mathbf{G}(+,\cdot,\|,\|\!\!\lfloor,|,\tau_I,\partial_H,\tau,\delta,\mathbf{a},\mathbf{b},\mathbf{c},\ldots)$, which is not yet a model of ACP$_\tau$ but becomes so after dividing out the congruence *$r\tau\delta$-bisimilarity* (notation: $\leftrightarrow_{r\tau\delta}$), a generalization of the 'ordinary' bisimilarity $\leftrightarrow$ of Section 2. Here we say that $g \leftrightarrow_{r\tau\delta} h$ if there is a relation between the nodes of g and the nodes of h such that (1) the roots are related; (2) a non-root node is only related to non-root nodes; (3) if nodes s,t in g,h respectively are related and there is in g an a-step from s to some s', then there is in g a path $\tau\tau\tau \cdots \tau a \tau\tau \cdots \tau$ (i.e. zero or more τ-steps followed by an a-step followed by zero or more τ-steps) from t to some t' such that s' and t' are again related; (4) as (3) with the roles of g,h interchanged. (See for an example of such a $r\tau\delta$-bisimulation Figure 7.) Again, this equivalence is a congruence on $\mathcal{G}$ and putting $\mathbb{G}=\mathcal{G}/\leftrightarrow_{r\tau\delta}$ we have a model for ACP$_\tau$, in which all systems of guarded recursion equations have a solution, and even a unique solution if abstraction operators are absent from the system.

As before in Section 2, $\mathbb{G}$ has submodels $\mathbb{R},\mathbb{F}$ (regular and finite processes, respectively). Remarkably, as observed in [1], there is no model $\mathbb{G}_{fb}$ based on all finitely branching graphs now. (For ACP such a model does exist.) The reason is that there is no structure $\mathcal{G}_{fb}$, since $\mathbf{G}_{fb}$ is not closed under the operations $\|,\|\!\!\lfloor,|,\tau_I$. The auxiliary operator $|$ is the culprit here.

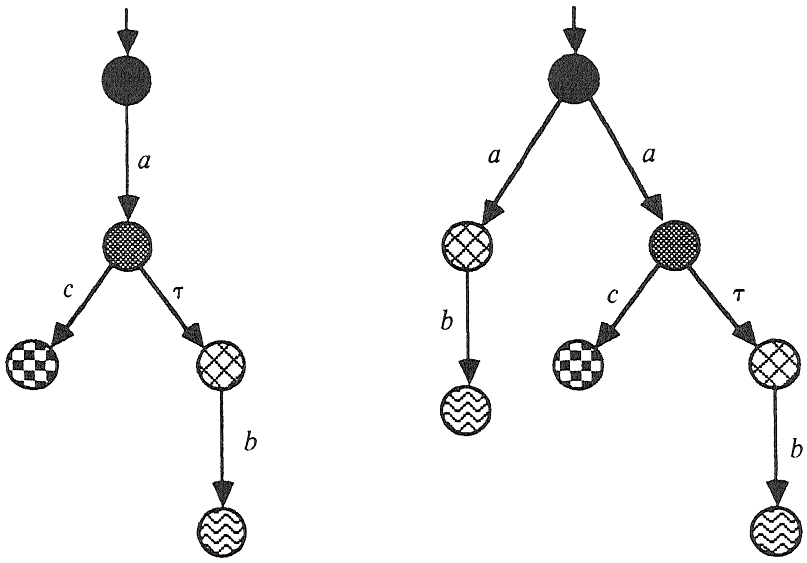

FIGURE 7. Example of $r\tau\delta$-bisimulation: nodes of the same colour are related

8. THE FINITE SPECIFICATION THEOREM

ACP$_\tau$ is a powerful specification mechanism; in a sense it is a universal specification mechanism: *every finitely branching, computable process in the graph model* $\mathbb{G}$ *can be finitely specified* in ACP$_\tau$. (We use the word 'specification' for 'system of recursion equations'.) We have to be more precise about the notion of 'computable process'. First, an intuitive explanation: suppose a finitely branching process graph $g \in G$ is 'actually' given; the labels may include τ, and there may be even infinite τ-traces. That g is 'actually' given means that the process graph g must be 'computable': g can be described by some coding of the nodes in natural numbers and recursive functions giving in-degree, out-degree, edge-labels, etc. This notion of a computable process graph is rather obvious, and we will not give details of the definition here.

Now even if the computable graph g is an infinite process graph, it can trivially be specified by an infinite computable specification, as follows. First rename all τ-edges in g to t-edges, for a 'fresh' atom t. Call the resulting process graph: g_t. Next assign to each node s of g_t a recursion variable X_s and write down the recursion equation for X_s according to the outgoing edges of node s. Let X_{s_0} be the variable corresponding to the root s_0 of g_t. As g is computable, g_t is computable and the resulting 'direct' specification

$$E = \{X_s = T_s(\mathbf{X}) | s \in \mathrm{NODES}(g_t)\}$$

is evidently also computable (i.e.: the nodes can be numbered as s_n $(n \geqslant 0)$, and after coding the sequence e_n of codes of equations $E_n : X_{s_n} = T_{s_n}(\mathbf{X})$ is a computable sequence). Now the infinite specification which uniquely determines g, is simply: $\{Y = \tau_{\{t\}}(X_{s_0})\} \cup E$. In fact all specifications below will have the form $\{X = \tau_I(X_0),\ X_n = T_n(\mathrm{X}) | n \geqslant 0\}$ where the guarded expressions $T_n(X)\ (= T_n(X_{i_1}, \ldots, X_{i_n}))$ contain no abstraction operators τ_J. They may contain all other process operators. We will say that such specifications have *restricted abstraction.*

However, we want more than a computable infinite specification with restricted abstraction: to describe process graph g we would like to find a *finite* specification with restricted abstraction for g. Indeed this is possible:

THEOREM 8.1 (FINITE SPECIFICATION THEOREM). *Let the finitely branching and computable process graph g determine* $\mathbf{g}$ *in the graph model* $\mathbb{G}$ *of* ACP$_\tau$. *Then there is a finite specification with restricted abstraction E in* ACP$_\tau$ *such that* $[\![E]\!] = g$. *Here* $[\![E]\!]$ *is the solution of E in* $\mathbb{G}$.

The proof in [1] is by constructing a Turing machine in ACP$_\tau$; the 'tape' is obtained by glueing together two Stacks as defined in Table 2. There does not seem to be an essential difficulty in removing the condition 'finitely branching' in the theorem, in favour of 'at most countably branching'.

9. CONCLUDING REMARKS

Even though the Finite Specification Theorem declares the set of operators of ACP$_\tau$ to be sufficient for all specifications, in practice one will need more operators to make specifications not only theoretically but also practically possible. Therefore some additional operators have been defined and studied in the present branch of process algebra, notably an operator by means of which different priorities can be given to different atomic actions, and a state operator taking into account information from a suitable state space. Using priorities imposed on atomic actions enables us to model interrupts in a system of communicating processes; the state operator has turned out to be indispensable in the construction of process algebra semantics for some object-oriented programming languages. For these developments we refer to [6]. Lately, some thorough studies have been made about extending ACP$_\tau$ with some new constants: ϵ for the empty process and η for an alternative to the silent step τ ([16,3]). The typical equation here is $\tau = \eta + \epsilon$.

A substantial amount of effort has been invested in extending ACP$_\tau$ to a suitable framework also for process verification, which was barely discussed in the present paper. Process verifications have been realized now for several non-trivial protocols ([14,9]), and recently also for some systolic algorithms ([10,15]) for tasks like palindrome recognition, matrix-vector multiplication. Some positive experience was also obtained using process algebra for the specification and verification of a simple production control system for a configuration of workcells.

Finally we mention that bisimulation semantics, as adopted in the present paper, is by no means the only process semantics. It is possible to identify many processes which are different in bisimulation semantics while still retaining an adequate description of relevant aspects such as deadlock behaviour, leading for instance to readiness semantics or failure semantics, embodying different views on processes. For a study in this area we refer to [7]. For an investigation of models of ACP_τ based on Petri Nets, see [8].

REFERENCES

1. J.C.M. BAETEN, J.A. BERGSTRA, J.W. KLOP (1987). On the consistency of Koomen's Fair Abstraction Rule. *TCS 51 (1/2),* 129-176.
2. J.C.M. BAETEN, J.A. BERGSTRA, J.W. KLOP (1987). Decidability of bisimulation equivalence for processes generating context-free languages. J.W. DE BAKKER, A.J. NIJMAN, P.C. TRELEAVEN (eds.). *Proceedings of the PARLE Conference, Eindhoven 1987, Vol. II,* Springer LNCS 259, 94-113.
3. J.C.M. BAETEN, R.J. VAN GLABBEEK (1987). *Abstraction and Empty Process in Process Algebra,* CWI Report CS-R8721, Centre for Mathematics and Computer Science, Amsterdam.
4. J.A. BERGSTRA, J.W. KLOP (1984). The algebra of recursively defined processes and the algebra of regular processes. J. PAREDAENS (ed.). *Proc. 11th ICALP,* Antwerpen 1984, Springer LNCS 172, 82-95.
5. J.A. BERGSTRA, J.W. KLOP (1986). Algebra of communicating processes. J.W. DE BAKKER, M. HAZEWINKEL, J.K. LENSTRA (eds.). *CWI Monograph* I, *Proceedings of the CWI Symposium Mathematics and Computer Science,* North-Holland, Amsterdam, 89-138.
6. J.A. BERGSTRA, J.W. KLOP (1986). Process algebra: specification and verification in bisimulation semantics. M. HAZEWINKEL, J.K. LENSTRA, L.G.L.T. MEERTENS (eds.). *CWI Monograph 4, Proceedings of the CWI Symposium Mathematics and Computer Science* II, North-Holland, Amsterdam, 61-94.
7. J.A. BERGSTRA, J.W. KLOP, E.-R. OLDEROG (1987). Failures without chaos: a new process semantics for fair abstraction. M. WIRSHING (ed.). *Proceedings IFIP Conference on Formal Description of Programming Concepts, Gl. Avernaes 1986,* North-Holland, Amsterdam, 77-103.
8. R.J. VAN GLABBEEK, F.W. VAANDRAGER (1987). Petri net models for algebraic theories of concurrency. J.W. DE BAKKER, A.J. NIJMAN, P.C. TRELEAVEN (eds.). *Proc. PARLE Conference, Eindhoven 1987, Vol. II,* Springer LNCS 259, 224-242.
9. C.P.J. KOYMANS, J.C. MULDER (1986). *A Modular Approach to Protocol Verification using Process Algebra,* Logic Group Preprint Series Nr.6, Dept. of Philosophy, State University of Utrecht.
10. L. KOSSEN, W.P.WEIJLAND (1987). *Correctness Proofs for Systolic Algorithms: Palindromes and Sorting,* Report FVI 87-04, Computer Science Department, University of Amsterdam.
11. S. MAUW (1987). *A Constructive Version of the Approximation Induction Principle,* Report FVI 87-09, Computer Science Department, University of

Amsterdam.

12. R. MILNER (1980). *A Calculus of Communicating Systems*, Springer LNCS 92.
13. D. PARK (1981). Concurrency and automata on infinite sequences. *Proc. 5th GI Conference*, Springer LNCS 104.
14. F.W. VAANDRAGER (1986). *Verification of Two Communication Protocols by Means of Process Algebra*, CWI Report CS-R8608, Centre for Mathematics and Computer Science, Amsterdam.
15. W.P.WEIJLAND (1987). *A Systolic Algorithm for Matrix-Vector Multiplication*, Report FVI 87-08, Computer Science Department, University of Amsterdam.
16. J.L.M. VRANCKEN (1986). *The Algebra of Communicating Processes with Empty Process*, Report FVI 86-01, Computer Science Department, University of Amsterdam.