An Introduction to Process Algebra

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

J.W. Klop
Department of Software Technology, Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
Department of Mathematics and Computer Science, Free University
P.O. Box 7161, 1007 MC Amsterdam, The Netherlands

This article serves as an introduction to the basis of the theory, that will be used in the rest of this book. To be more precise, we will discuss the axiomatic theory ACP_τ (Algebra of Communicating Processes with abstraction), with additional features added, which is suitable for both specification and verification of communicating processes. As such, it can be used as background material for the other articles in the book, where all basic axioms are gathered. But we address ourselves not exclusively to readers with previous exposure to algebraic approaches to concurrency (or, as we will call it, process algebra). Also newcomers to this type of theory could find enough here, to get started. For a more thorough treatment of the theory, we refer to [1], which will be revised, translated and published in this CWI Monograph series. There, most proofs can also be found; we refer also to the original papers where the theory was developed. This article is an abbreviated version of reference [11].

Our presentation will concentrate on process algebra as it has been developed since 1982 at the Centre for Mathematics and Computer Science, Amsterdam (see [7]), since 1985 in cooperation with the University of Amsterdam and the University of Utrecht. This means that we make no attempt to give a survey of related approaches though there will be references to some of the main ones.

This paper is not intended to give a survey of the whole area of activities in process algebra.

We acknowledge the help of Jos Baeten in the preparation of this paper.

Partial support received from the European Community under ESPRIT project no. 432, An Integrated Formal Approach to Industrial Software Development (METEOR).

1. The basic constructors

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. (We will often suppress the dot and write $(a+b)c$.) 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 $x,y, \ldots$ satisfy the following axioms:

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

Thus far we used '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, then, is the term model of BPA (Basic Process Algebra), 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. This process algebra contains only finite processes; things get more lively if we admit recursion enabling us to define infinite processes. Even at this stage one can define, recursively, interesting processes:

COUNTER
$X=(zero+up\cdot Y)\cdot X$
$Y=down+up\cdot Y\cdot Y$

Table 2

where 'zero' is the action that asserts that the counter has value 0, and 'up' and 'down' are the actions of incrementing resp. decrementing the counter by one unit. The process COUNTER is now represented by X; Y is an auxiliary process. COUNTER is a 'perpetual' process, that is, all its execution traces are infinite. Such a trace is e.g. zero-zero-up-down-zero-up-up-up-....

Equations as in Table 2 are also called fixed point equations. An important property of such equations is whether or not they are guarded. A fixed point equation is *guarded* if every occurrence of a recursion variable in the right hand side is preceded ('guarded') by an occurrence of an action. For instance, the occurrence of X in the RHS of $X=(zero+up\cdot Y)\cdot X$ is guarded since, when this X is accessed, one has to pass either the guard zero or the guard up. A non-example: the equation $X=X+a\cdot X$ is not guarded.

Before proceeding to the next section, let us assure the reader that the omission of the other distributive law, $z(x+y)=zx+zy$, is intentional. The reason will become clear after the introduction of 'deadlock'.

2. Deadlock

A vital element in the present set-up of process algebra is the process δ, signifying 'deadlock'. The process ab performs its two steps and then stops, silently and happily; but the process $ab\delta$ deadlocks (with a crunching sound, one may imagine) after the a- and b-action: it wants to do a proper action but it cannot. So δ is the acknowledgement of stagnation. With this in mind, the axioms to which δ is subject, should be clear:

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

Table 3

(In fact, it can be argued that 'deadlock' is not the most appropriate name for the process constant δ. In the sequel we will encounter a process which can more rightfully claim this name: $\tau\delta$, where τ is the silent step. We will stick to the present terminology, however.)

The axiom system of BPA (Table 1) together with the present axioms for δ is called BPA_δ. Now suppose that the distributive law $z(x+y)=zx+zy$ is added to BPA_δ. Then: $ab=a(b+\delta)=ab+a\delta$. This means that a process with deadlock possibility is equal to one without; and that conflicts with our intention to model also deadlock behaviour of processes.

3. Interleaving or free merge

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. Using an auxiliary operator $\lfloor\!\lfloor$ (with the interpretation that $x\lfloor\!\lfloor y$ 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
$x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$
$a \lfloor\!\lfloor x = ax$
$ax \lfloor\!\lfloor y = a(x \| y)$
$(x+y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$

TABLE 4

One can show that an equivalent axiomatization of $\|$ without an auxiliary operator like $\lfloor\!\lfloor$, would require infinitely many axioms.

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

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 two fixed point equations $X = aYX$, $Y = b + aYY$. (This is a simplified version of the counter in Table 2, without the action zero.) To see that both ways of defining X yield the same process, one may 'unwind' according to the given equations:

$$X = a(b\|X) = a(b \lfloor\!\lfloor X + X \lfloor\!\lfloor b) = a(bX + a(b\|X) \lfloor\!\lfloor b)$$
$$= a(bX + a((b\|X)\|b)) = a(bX + a...),$$

while on the other hand

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

so at least up to level 2 the processes are equal. In fact they can be proved equal up to each finite level. Later on, we will introduce an infinitary proof rule enabling us to infer that, therefore, the processes are equal.

So, is the defining power (or expressibility) of PA greater than that of BPA? Indeed it is, as is shown by the following process:

BAG
$X = in(0)(out(0)\|X) + in(1)(out(1)\|X)$

TABLE 5

This equation describes the process behaviour of a 'bag' or 'multiset' that may contain finitely many instances of data 0, 1. The actions $in(0)$, $out(0)$ are: putting a 0 in the bag resp. getting a 0 from the bag, and likewise for 1. This process does not have a finite specification in BPA, that is, a finite specification without merge ($\|$).

If we want to define a bag over a general finite data set D (instead of just over $\{0,1\}$) we use a sum notation as an abbreviation, so

$$X = \sum_{d \in D} in(d) \cdot (out(d)\|X).$$

4. FIXED POINTS

We have already alluded to the existence of infinite processes; this raises the question how one can actually construct process algebras (for BPA or PA) containing infinite processes in addition to finite ones. Such models can be obtained by means of:

(1) projective limits ([8,10]);
(2) complete metrical spaces, as in the work of De Bakker and Zucker [5,6];
(3) quotients of graph domains (a graph domain is a set of process graphs or transition diagrams), as in Milner [18], Baeten, Bergstra and Klop [4]; or Van Glabbeek [14];
(4) the 'explicit' models of Hoare [16];
(5) ultraproducts of finite models (Kranakis [17]).

In Section 12 we will discuss a model as in (3).

5. COMMUNICATION

So far, the parallel composition or merge ($\|$) did not involve communication in the process $x\|y$: x and y are 'freely' merged. 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, 'hand shaking' is the paradigm for the type of communication which we will introduce now. If $A=\{a,b,c,\ldots,\}$ is the action alphabet, let us adopt a partial binary function γ on A, that is required to be commutative and associative. If $\gamma(a,b)$ is defined, a and b communicate, and $\gamma(a,b)$ is the result of the communication; if $\gamma(a,b)$ is not defined, a and b do not communicate. We can extend γ to a total function $|$ on $A\cup\{\delta\}$, by putting $a|b=\delta$ whenever $\gamma(a,b)$ is not defined (so also when one of a,b equals δ). The result is a binary communication function $|$ on $A\cup\{\delta\}$ satisfying

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

TABLE 6

(Here a,b vary over $A\cup\{\delta\}$.) We can now specify *merge with communication*; we use the same notation $\|$ as for the free merge, since in fact free merge is an instance of merge with communication (by choosing the communication function trivial, i.e. $a|b=\delta$ for all a,b). There are now two auxiliary operators, allowing a finite axiomatization: left-merge ($\lfloor\!\_$) as before and $|$ (communication merge or 'bar'), which is an extension of the communication function to all processes, not only the constants. The axioms for $\|$ and its auxiliary operators are:

MERGE WITH COMMUNICATION
$x \Vert y = x \mathbin{\llcorner} y + y \mathbin{\llcorner} x + x \mid y$
$a \mathbin{\llcorner} x = ax$
$ax \mathbin{\llcorner} y = a(x \Vert y)$
$(x+y) \mathbin{\llcorner} z = x \mathbin{\llcorner} z + y \mathbin{\llcorner} z$
$ax \mid b = (a \mid b)x$
$a \mid bx = (a \mid b)x$
$ax \mid by = (a \mid b)(x \Vert y)$
$(x+y) \mid z = x \mid z + y \mid z$
$x \mid (y+z) = x \mid y + x \mid z$

TABLE 7

We also need the so-called *encapsulation* operators $\partial_H(H \subseteq A)$ 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 8

The axioms for BPA, DEADLOCK together with the present ones 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 $\partial_H(x_1 \Vert \cdots \Vert x_n)$. Prefixing the encapsulation operator says that the system $x_1, \ldots, x_n$ is to be perceived as a separate unit w.r.t. the communication actions mentioned in H; no communications between actions in H with an environment are expected or intended.

We will often adopt the following special format for the communication function, called *read/write (receive/send) 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 *send* actions $si(d)$, for $i = 1, \ldots, p$ and $d \in D$. The interpretation is: read datum d at port i, resp. send datum d at port i. 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: $si(d) \mid ri(d) = ci(d)$. Instead of $si(d)$ we will also see the notation $wi(d)$ (write d along i).

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. Section 3) with action alphabets $\{r1(d),s2(d)|d\in D\}$ resp. $\{r2(d),s3(d)|d\in D\}$. That is, B_{12} is a bag-like channel reading data d at port 1, sending them at 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 2; that is, we adopt communications $s2(d)|r2(d)=c2(d)$ where $c2(d)$ is the transaction of d at 2.

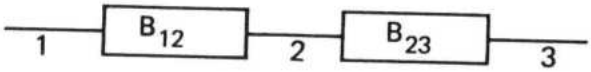

Figure 1

The composite system $\mathbf{B}_{13}=\partial_H(B_{12}\|B_{23})$ where $H=\{s2(d),\ r2(d)|d\in D\}$ should, intuitively, be again a bag between locations 1, 3. However, some (rather involved) calculations learn that $\mathbf{B}_{13}=\Sigma_{d\in D}r1(d)\cdot((c2(d)s3(d))\|\mathbf{B}_{13})$; so $\mathbf{B}_{13}$ is a 'transparant' bag: the passage of d through 2 is visible as the transaction event $c2(d)$.

How can we *abstract* from such internal details, 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 is called 'pre-abstraction' in [2]). This renaming operator is 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 9

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 (nor desirable) to remove *all* τ's in an expression if one is interested in a faithful description of deadlock behaviour of processes. 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 safely be adopted, but the other one is wrong. Fortunately, Milner [19] has devised some simple axioms which can be used to give a complete description of the properties of the silent step (complete w.r.t. a certain semantical notion of process equivalence called bisimulation, which does respect deadlock behaviour; this notion is discussed in the sequel), as follows.

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

TABLE 10

To return to our example of the transparant bag $\mathbf{B}_{13}$, after abstraction of the set of transactions $I = \{c2(d) \mid d \in D\}$ the result is indeed an 'ordinary' bag:

$$\begin{aligned}\tau_I(\mathbf{B}_{13}) &= \tau_I(\Sigma r1(d)(c2(d)\cdot s3(d) \| \mathbf{B}_{13})) \overset{(*)}{=} \Sigma r1(d)(\tau\cdot s3(d) \| \tau_I(\mathbf{B}_{13})) \\ &= \Sigma(r1(d)\cdot\tau\cdot s3(d)) \between \tau_I(\mathbf{B}_{13}) = \Sigma(r1(d)\cdot s3(d)) \between \tau_I(\mathbf{B}_{13}) \\ &= \Sigma r1(d)(s3(d) \| \tau_I(\mathbf{B}_{13}))\end{aligned}$$

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

$$B_{13} = \Sigma r1(d)(s3(d) \| B_{13}).$$

Here we were able to eliminate all silent actions, but this will not always be the case. In fact, this computation is not as straightforward as was maybe suggested: to justify the equations marked with (*) and (**) we need more powerful principles, which we will discuss in the sequel. (Specifically, in (*) an appeal to the 'alphabet calculus' of Section 9 is needed and (**) requires the principle RSP, see Section 8 below.)

7. PROJECTION AND AUXILIARY AXIOMS

First, we define the projection operators $\pi_n (n \geqslant 1)$, cutting off a process at level n:

PROJECTION	
$\pi_n(a) = a$	$\pi_n(x+y) = \pi_n(x) + \pi_n(y)$
$\pi_1(ax) = a$	$\pi_n(\tau) = \tau$
$\pi_{n+1}(ax) = a\pi_n(x)$	$\pi_n(\tau x) = \tau\cdot\pi_n(x)$

TABLE 11

E.g., for X defining BAG as in Table 5:

$$\pi_2(X) = in(0)(out(0) + in(0) + in(1)) + in(1)(out(1) + in(0) + in(1)).$$

We have that τ-steps do not add to the depth; this is enforced by the τ-laws (since, e.g. $a\tau b = ab$ and $\tau a = \tau a + a$).

By means of these projections a distance between processes x, y can be defined: $d(x,y) = 2^{-n}$ where n is the least natural number such that $\pi_n(x) \neq \pi_n(y)$, and $d(x,y) = 0$ if there is no such n. If the term model of BPA (or PA) as in Section 1 is equipped with this distance function, the result is an ultrametrical space. By metrical completion we obtain a model of BPA (resp. PA) in which all systems of guarded recursion equations have a unique solution. This model construction has been employed in various settings by De Bakker and Zucker [5,6].

In the articles of Vaandrager in this volume a slightly different definition of the projection operators is used, which lead to the same theorems below, but which have the advantage that they also can be defined for $n = 0$, and are definable in our theory ACP_τ (see Section 11). We present the new axioms below.

PROJECTION, Second version
$\pi_0(ax) = \delta$
$\pi_{n+1}(ax) = a\pi_n(x)$
$\pi_n(x+y) = \pi_n(x) + \pi_n(y)$
$\pi_n(\tau) = \tau$
$\pi_n(\tau x) = \tau \cdot \pi_n(x)$

TABLE 12

In ACP_τ, systems are described as the parallel composition of their components, and so a system of communicating processes $x_1, \ldots, x_n$ is represented by the expression $\partial_H(x_1 \| \cdots \| x_n)$. When we want to focus on the external actions of such a system, we apply an abstraction operator, that abstracts from all communications between actions from H. A useful theorem to break down these expressions is the *Expansion Theorem* which holds under the assumption of the *handshaking axiom* $x|y|z = \delta$. This axiom says that all communications are binary.

THEOREM (EXPANSION THEOREM).

$$x_1 \| \cdots \| x_k = \sum_i x_i \mathbin{\llcorner} X_k^i + \sum_{i \neq j} (x_i | x_j) \mathbin{\llcorner} X_k^{ij}.$$

Here X_k^i denotes the merge of $x_1, \ldots, x_k$ except x_i, and X_k^{ij} denotes the same merge except $x_i, x_j (k \geq 3)$. In order to prove the Expansion Theorem, one first proves by simultaneous induction on term complexity that for all closed ACP_τ-terms (i.e. terms without free variables) the following holds:

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

TABLE 13

8. PROOF RULES FOR RECURSIVE SPECIFICATIONS

We have now presented a survey of ACP_τ; we refer to [9] for an analysis of this proof system. Note that ACP_τ (displayed in full in Section 11) is entirely equational. Without further proof rules it is not possible to deal (in an algebraical way) with infinite processes, obtained by recursive specifications, such as BAG; in the derivation above we tacitly used such proof rules which will be made explicit now.

(i) RDP, the Recursive Definition Principle: *Every guarded and abstraction-free recursive specification has a solution.*

(ii) RSP, the Recursive Specification Principle: *Every guarded and abstraction-free recursive specification has at most one solution.*

(iii) AIP, the Approximation Induction Principle: *A process is determined by its finite projections.*

In a more formal notation, AIP can be rendered as the infinitary rule

$$\frac{\forall n \quad \pi_n(x) = \pi_n(y)}{x = y}.$$

As to (i), the restriction to guarded specifications is not very important (for the definition of 'guarded' see Section 1); in the process algebras that we have encountered and that satisfy RDP, also the same principle without the guardedness condition is true. More delicate is the situation in principle (ii): first, *τ-steps may not act as guards*: e.g. the recursion equation $X = \tau X + a$ has infinitely many solutions, namely $\tau(a+q)$ is a solution for arbitrary q; and second, the *recursion equations must not contain occurrences of abstraction operators τ_I*. That is, they are 'abstraction-free' (but there may be occurrences of τ in the equations). The latter restriction is in view of the fact that, surprisingly, the recursion equation $X = a \cdot \tau_{\{a\}}(X)$ possesses infinitely many solutions, even though it looks very guarded. (The solutions are: $a \cdot q$ where q satisfies $\tau_{\{a\}}(q) = q$.) That the presence of abstraction operators in recursive specifications causes trouble, was first noticed by Hoare [15,16].

The unrestricted form of AIP as in (iii) will turn out to be too strong in some circumstances; it does not hold in one of the main models of ACP_τ, namely the graph model which is introduced in Section 12. Therefore we also introduce the following weaker form.

(iv) AIP$^-$ (Weak Approximation Induction Principle): *Every process which has an abstraction-free guarded specification is determined by its finite projections.*

Roughly, a process which can be specified without abstraction operators is one in which there are no infinite τ-traces (and which is definable). E.g. the process X_0 defined by the infinite specification $\{X_0 = bX_1, X_{n+1} = bX_{n+2} + a^n\}$, where a^n is $a \cdot a \cdot \cdots \cdot a$ (n times), contains an infinite trace of b-actions; after abstraction w.r.t. b, the resulting process, $Y = \tau_{\{b\}}(X_0)$, has an infinite trace of τ-steps; and (at least in the main model of ACP_τ of Section 12) this Y is not definable without abstraction operators.

Even the Weak Approximation Induction Principle is rather strong. In fact a short argument shows the following:

THEOREM. AIP$^- \Rightarrow$RSP.

As a rule, we will be very careful in admitting abstraction operators in recursive specifications. Yet there are processes which can be elegantly specified by using abstraction inside recursion.

9. ALPHABET CALCULUS

In computations with infinite processes one often needs information about the *alphabet* $\alpha(x)$ of a process x. E.g. if x is the process uniquely defined by the recursion equation $X = aX$, we have $\alpha(x) = \{a\}$. An example of the use of this alphabet information is given by the implication $\alpha(x) \cap H = \emptyset \Rightarrow \partial_H(x) = x$. For finite closed process expressions this fact can be proved with induction to the structure, but for infinite processes we have to require such a property axiomatically. In fact, the example will be one of the 'conditional axioms' below (conditional, in contrast with the purely equational axioms we have introduced thus far). First we have to define the alphabet:

ALPHABET
$\alpha(\delta) = \emptyset$
$\alpha(\tau) = \emptyset$
$\alpha(a) = \{a\}$ (if $a \neq \delta$)
$\alpha(\tau x) = \alpha(x)$
$\alpha(ax) = \{a\} \cup \alpha(x)$ (if $a \neq \delta$)
$\alpha(x + y) = \alpha(x) \cup \alpha(y)$
$\alpha(x) = \cup_{n \geq 1} \alpha(\pi_n(x))$
$\alpha(\tau_I(x)) = \alpha(x) - I$

TABLE 14

To appreciate the non-triviality of the concept $\alpha(x)$, let us mention that a finite specification can be given of a process for which the alphabet is uncomputable (see [3] for an example).

Now the following conditional axioms will be adopted:

CONDITIONAL AXIOMS
$\alpha(x)\|(\alpha(y)\cap H)\subseteq H \Rightarrow \partial_H(x\|y)=\partial_H(x\|\partial_H(y))$
$\alpha(x)\|(\alpha(y)\cap I)=\emptyset \Rightarrow \tau_I(x\|y)=\tau_I(x\|\tau_I(y))$
$\alpha(x)\cap H=\emptyset \Rightarrow \partial_H(x)=x$
$\alpha(x)\cap I=\emptyset \Rightarrow \tau_I(x)=x$
$H=H_1\cup H_2 \Rightarrow \partial_H(x)=\partial_{H_1}\circ\partial_{H_2}(x)$
$I=I_1\cup I_2 \Rightarrow \tau_I(x)=\tau_{I_1}\circ\tau_{I_2}(x)$
$H\cap I=\emptyset \Rightarrow \tau_I\circ\partial_H(x)=\partial_H\circ\tau_I(x)$

TABLE 15

Using these axioms, one can derive for instance the following fact: if communication is of the read-write format and I is disjoint from the set of transactions (communication results) as well as disjoint from the set of communication actions, then the abstraction τ_I distributes over merges $x\|y$.

10. KOOMEN'S FAIR ABSTRACTION RULE

Suppose the following statistical experiment is performed: somebody flips a coin, repeatedly, until head comes up. This process is described by the recursion equation $X=\mathit{flip}\cdot(\mathit{tail}\cdot X+\mathit{head})$. Suppose further that the experiment takes place in a closed room, and all information to be obtained about the process in the room is that we can hear the experimenter shout joyfully: 'Head!'. That is, we observe the process $\tau_I(X)$ where $I=\{\mathit{flip},\mathit{tail}\}$. Now, if the coin is 'fair', it is to be expected that sooner or later (i.e., after a τ-step) the action 'head' will be perceived. Hence, intuitively, $\tau_I(X)=\tau\cdot\mathit{head}$. (This vivid example is from Vaandrager [21].)

Koomen's Fair Abstraction Rule (KFAR) is an algebraic rule enabling us to arrive at such a conclusion formally. The rule was introduced in this form in Bergstra and Klop [12]. (For an extensive analysis of the rule see [4].) The simplest form is

$$\frac{x=ix+y \quad (i\in I)}{\tau_I(x)=\tau\cdot\tau_I(y)} \ \mathrm{KFAR}_1.$$

So, KFAR_1 expresses the fact that the 'τ-loop' (originating from the i-loop) in $\tau_I(x)$ will not be taken infinitely often. In case this 'τ-loop' is of length 2, the same conclusion is expressed in the rule

$$\frac{x_1=i_1x_2+y_1,\ x_2=i_2x_1+y_2 \quad (i_1,i_2\in I)}{\tau_I(x_1)=\tau\cdot\tau_I(y_1+y_2)} \ \mathrm{KFAR}_2$$

and it is not hard to guess what the general formulation (KFAR_n, $n\geqslant 1$) will be. In fact, we will need an even more general formulation, CFAR (the Cluster Fair Abstraction Rule). This principle was introduced by Vaandrager [21]. There, he showed that CFAR can already be derived from KFAR_1 (at least in

the framework to be discussed below).

Suppose E is a recursive specification (a system of fixed point equations) over variables V, and suppose I is the set of atomic actions to be abstracted from. We call a subset C of V a *cluster of I in E* if for all X in C the equation for X in E has the form

$$X = \sum_{k=1}^{m} i_k \cdot X_k + \sum_{l=1}^{n} Y_l,$$

where $m \geqslant 1$, $n \geqslant 0$, $i_1,\ldots,i_m \in I \cup \{\tau\}$, $X_1,\ldots,X_m \in C$, $Y_1,\ldots,Y_n \in V - C$. The variables in C are called *cluster variables*. For variables $X, Y \in V$ we write $X \rightsquigarrow Y$ if Y occurs in the right hand side of the equation of X. Then, the *exits* of the cluster are those variables outside C, that can be reached from C, i.e.

$$exits(C) = \{Y \in V - C : X \rightsquigarrow Y \text{ for some } X \in C\}.$$

Let $\rightsquigarrow^*$ be the transitive and reflexive closure of $\rightsquigarrow$. We call a cluster C of I in E *conservative* if every exit can be reached from every cluster variable, i.e. for all $X \in C$ and all $Y \in exits(C)$ we have $X \rightsquigarrow^* Y$. Now we can formulate the rule CFAR as follows.

DEFINITION. The *Cluster Fair Abstraction Rule* is the following statement: let E be a guarded recursive specification; let $I \subseteq A$ be such that $|I| \geqslant 2$; let C be a finite conservative cluster of I in E; and let $X \in C$. Then:

$$\tau_I(X) = \tau \cdot \sum_{Y \in exits(C)} \tau_I(Y).$$

We see that CFAR can only be applied when we are dealing with a conservative cluster. In practice, most specifications will not contain conservative clusters. If, in such a situation, we state that a certain result is obtained by the use of CFAR, we mean that there is a specification which is equivalent to the one we are dealing with (using RSP), which contains a conservative cluster, and that the result follows when we apply CFAR to this second specification.

KFAR and CFAR are of great help in protocol verifications. As an example, KFAR can be used to abstract from a cycle of internal steps which is due to a defective communication channel; the underlying fairness assumption is that this channel is not defective forever, but will function properly after an undetermined period of time. (Just as in the coin flipping experiment the wrong option, tail, is not chosen infinitely often.)

An interesting peculiarity of the present framework is the following. Call the process τ^ω $(= \tau \cdot \tau \cdot \tau \cdots)$ *livelock*. Formally, this is the process $\tau_{\{i\}}(x)$ where x is uniquely defined by the recursion equation $X = i \cdot X$. Noting that $x = i \cdot x = i \cdot x + \delta$ and applying KFAR_1 we obtain $\tau^\omega = \tau_{\{i\}}(x) = \tau\delta$. In words: *livelock = deadlock*. There are other semantical frameworks for processes, also in the scope of process algebra but not in the scope of this paper, where this equality does not hold (see [13]).

11. A FRAMEWORK FOR PROCESS SPECIFICATION AND VERIFICATION

We have now arrived at a framework which contains all the axioms and proof rules introduced so far. In Table 16 the list of all components of this system is given; Table 17 contains the equational system ACP_τ and Table 18 contains the extra features and furthermore the proof principles which were introduced. Note that for *specification* purposes one only needs ACP_τ; for *verification* one will need the whole system. Also, it is important to notice that this framework resides entirely on the level of syntax and formal specifications and verification using that syntax - even though some proof rules are infinitary. No semantics has been provided yet; this will be done in Section 12. The idea is that 'users' can stay in the realm of this formal system and execute algebraical manipulations, without the need for an excursion into the semantics. That this can be done is demonstrated throughout this book. This does not mean that the semantics is unimportant; it does mean that the user needs only be concerned with formula manipulation. The underlying semantics is of great interest for the theory, if only to guarantee the consistency of the formal system; but applications should not be burdened with it, in our intention.

A PROCESS SPECIFICATION AND VERIFICATION FRAMEWORK	
Basic Process Algebra	A1-5
Deadlock	A6,7
Communication Function	C1-3
Merge with Communication	CM1-9
Encapsulation	D1-4
Silent Step	T1-3
Silent Step: Auxiliary Axioms	TM1,2; TC1-4
Abstraction	DT; TI1-5
Projection	PR1-6
Hand Shaking	HA
Standard Concurrency	SC
Expansion Theorem	ET
Alphabet Calculus	CA
Recursive Definition Principle	RDP
Recursive Specification Principle	RSP
Weak Approximation Induction Principle	AIP^-
Cluster Fair Abstraction Rule	CFAR

TABLE 16

ACP_τ			
$x+y=y+x$	A1	$x\tau=x$	T1
$(x+y)+z=x+(y+z)$	A2	$\tau x=\tau x+x$	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		
$x\|\|y=x\|\_y+y\|\_x+x\|y$	CM1		
$a\|\_x=ax$	CM2	$\tau\|\_x=\tau x$	TM1
$ax\|\_y=a(x\|\|y)$	CM3	$\tau x\|\_y=\tau(x\|\|y)$	TM2
$(x+y)\|\_z=x\|\_z+y\|\_z$	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	$\tau_I(xy)=\tau_I(x)\cdot\tau_I(y)$	TI5

TABLE 17

REMAINING AXIOMS AND RULES	
$\pi_1(ax)=a$	PR1
$\pi_{n+1}(ax)=a\cdot\pi_n(x)$	PR2
$\pi_n(a)=a$	PR3
$\pi_n(x+y)=\pi_n(x)+\pi_n(y)$	PR4
$\pi_n(\tau)=\tau$	PR5
$\pi_n(\tau x)=\tau\cdot\pi_n(x)$	PR6
$x\|y\|z=\delta$	HA
$x\|y=y\|x$	SC1
$x\|\|y=y\|\|x$	SC2
$x\|(y\|z)=(x\|y)\|z$	SC3
$(x\lfloor y)\lfloor z=x\lfloor(y\|z)$	SC4
$(x\|ay)\lfloor z=x\|(ay\lfloor z)$	SC5
$x\|\|(y\|\|z)=(x\|\|y)\|\|z$	SC6

$$x_1\|\cdots\|x_n=\sum_{1\leq i\leq n}x_i\lfloor\Big(\|_{\substack{1\leq k\leq n\\ k\neq i}}x_k\Big)+\sum_{1\leq i<j\leq n}(x_i|x_j)\lfloor\Big(\|_{\substack{1\leq k\leq n\\ k\neq i,k\neq j}}x_k\Big)\quad(n\geqslant 3)\quad\text{ET}$$

$\alpha(\delta)=\varnothing$	AB1
$\alpha(\tau)=\varnothing$	AB2
$\alpha(a)=\{a\}$ (if $a\neq\delta$)	AB3
$\alpha(\tau x)=\alpha(x)$	AB4
$\alpha(ax)=\{a\}\cup\alpha(x)$ (if $a\neq\delta$)	AB5
$\alpha(x+y)=\alpha(x)\cup\alpha(y)$	AB6
$\alpha(x)=\cup_{n\geqslant 1}\alpha(\pi_n(x))$	AB7
$\alpha(\tau_I(x))=\alpha(x)-I$	AB8
$\alpha(x)\|(\alpha(y)\cap H)\subseteq H\Rightarrow\partial_H(x\|\|y)=\partial_H(x\|\|\partial_H(y))$	CA1
$\alpha(x)\|(\alpha(y)\cap I)=\varnothing\Rightarrow\tau_I(x\|\|y)=\tau_I(x\|\|\tau_I(y))$	CA2
$\alpha(x)\cap H=\varnothing\Rightarrow\partial_H(x)=x$	CA3
$\alpha(x)\cap I=\varnothing\Rightarrow\tau_I(x)=x$	CA4
$H=H_1\cup H_2\Rightarrow\partial_H(x)=\partial_{H_1}\circ\partial_{H_2}(x)$	CA5
$I=I_1\cup I_2\Rightarrow\tau_I(x)=\tau_{I_1}\circ\tau_{I_2}(x)$	CA6
$H\cap I=\varnothing\Rightarrow\tau_I\circ\partial_H(x)=\partial_H\circ\tau_I(x)$	CA7

RDP	*Every guarded and abstraction-free specification has a solution*
RSP	*Every guarded and abstraction-free specification has at most one solution*
AIP$^-$	*Every process which has an guarded abstraction-free specification is determined by its finite projections*

CFAR *If E is a guarded recursive specification, and C a finite conservative cluster of I in E, then for each $X\in C$:*

$$\tau_I(X)=\tau\sum_{Y\in exits(C)}\tau_I(Y).$$

TABLE 18

It should be noted that there is redundancy in this presentation; as we already stated, AIP^- implies RSP and there are other instances where we can save some axioms or rules (for instance, the axioms CM2,5,6 turn out to be derivable from the other axioms). This would however not enhance clarity.

So we have here a medium for formal process specifications and verifications; let us note that we also admit infinite specifications. As the system is meant to have practical applications, we will only encounter *computable* specifications.

12. The graph model for ACP_τ

We will give a quick introduction to what we consider to be the 'main' model of ACP_τ. The basic building material consists of the domain of *countably branching, labeled, rooted, connected, directed multigraphs.* Such a graph, also called a process graph, consists of a possibly infinite set of nodes s with one distinguished node s_0, the root. The edges, also called transitions or steps, between the nodes are labeled with an element from the action alphabet; also δ and τ may be edge labels. We use the notation $s\rightarrow_a t$ for an a-transition from node s to node t; likewise $s\rightarrow_\tau t$ is a τ-transition and $s\rightarrow_\delta t$ is a δ-step. That the graph is connected means that every node must be accessible by finitely many steps from the root node.

Corresponding to the operations $+,\cdot,\|,\lfloor\!\lfloor,\lfloor,\partial_H,\tau_I,\pi_n,\alpha$ in our theory we define operations in this domain of process graphs. Precise definitions can be found in [1,4]; we will sketch some of them here. The sum $g+h$ of two process graphs g,h is obtained by glueing together the roots of g and h (see Figure 2(i)); there is one caveat: if a root is cyclic (i.e. lying on a cycle of transitions leading back to the root), then the initial part of the graph has to be 'unwound' first so as to make the root acyclic (see Figure 2(ii)). The product $g\cdot h$ is obtained by appending copies of h to each terminal node of g; alternatively, one may first identify all terminal nodes of g and then append one copy of h to the unique terminal node if it exists (see Figure 2 (iii)). The merge $g\|h$ is obtained as a cartesian product of both graphs, with 'diagonal' edges for communications (see Figure 2(v) for an example without communication, and Figure 2(vi) for an example with communication action $a|b$). Definitions of the auxiliary operators are somewhat more complicated and not discussed here. The encapsulation and abstraction operators are simply renamings, that replace the edge labels in H resp. in I by δ resp. τ. Definitions of the projection operators π_n and α should be clear from the axioms by which they are specified. As to the projection operators, it should be emphasized that τ-steps are 'transparent': they do not increase the depth.

OPERATIONS ON PROCESS GRAPHS

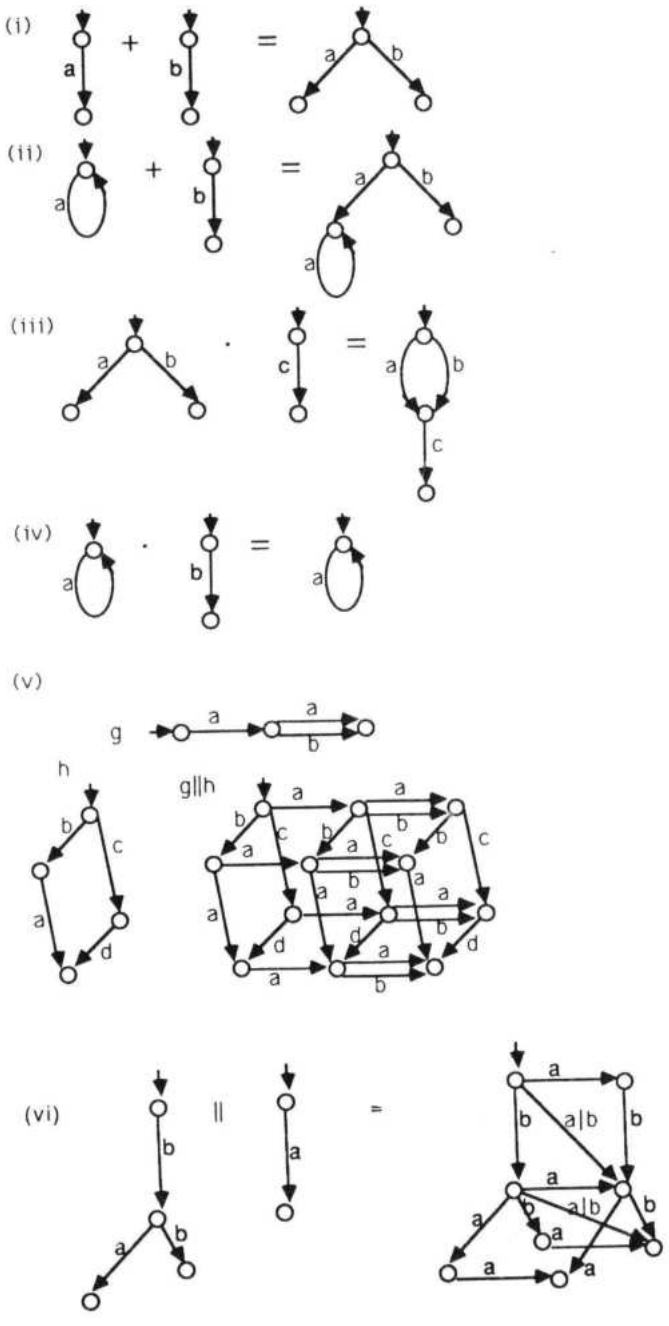

FIGURE 2

This domain of process graphs equipped with the operations just introduced, is not yet a model of ACP_τ: for instance the axiom $x+x=x$ does not hold. In order to obtain a model, we define an equivalence on the process graphs which is moreover a congruence w.r.t. the operations. This equivalence is called *bisimulation congruence* or *bisimilarity*. (The original notion is due to Park [20]; it was anticipated by Milner's observational equivalence, see [18].) In order to define this notion, let us first introduce the notation $s \Rightarrow_a t$ for nodes s, t of graph g, indicating that from node s to node t there is a finite path consisting of zero or more τ-steps and one a-step followed by zero or more τ-steps. Let us say that in this situation there is a 'generalized a-step' from s to t. Likewise with 'a' replaced by 'τ'. Next, let a *coloring* of process graph g be a surjective mapping from a set of 'colors' C to the node set of g, such that the color assigned to the root of g is different from all other colors, and furthermore, such that all end nodes are assigned the same color which is different from other colors. Now two process graphs g, h are bisimilar if there are colorings of g, h such that (1) the roots of g, h have the same color and (2) whenever *somewhere* in the two graphs a generalized a-step is possible from a node with color c to a node with color c', then *every* c-colored node admits a generalized a-step to a c'-colored node (be it in g or in h). We use the notation $g \leftrightarrows h$ to indicate that g, h are bisimilar. One can prove that $\leftrightarrows$ is a congruence and, if $\mathbf{G}$ is the original domain of countably branching process graphs:

THEOREM ([4]). *$\mathbf{G}/\leftrightarrows$ is a model of all axioms in Tables 17 and 18.*

Remarkably, this graph model does not satisfy the unrestricted Approximation Induction Principle. A counterexample is given (in a self-explaining notation) by the two graphs $g=\Sigma_{n\geqslant 1}a^n$ and $h=\Sigma_{n\geqslant 1}a^n+a^\omega$; while g and h have the same finite projections $\pi^n(g)=\pi^n(h)=a+a^2+a^3+\cdots+a^n$, they are not bisimilar due to the presence of the infinite trace of a-steps in h. It might be thought that it would be helpful to restrict the domain of process graphs to finitely branching graphs, in order to obtain a model which satisfies AIP, but there are two reasons why this is not the case: (1) the finitely branching graph domain would not be closed under the operations, in particular the communication merge ($|$); (2) a similar counterexample can be obtained by considering the finitely branching graphs $g'=\tau_{\{t\}}(g'')$ where g'' is the graph defined by $\{X_n=a^n+tX_{n+1} | n\geqslant 1\}$ and $h'=g'+a^\omega$.

REFERENCES

1. J.C.M. BAETEN (1986). *Procesalgebra,* Kluwer Programmatuurkunde, Deventer (in Dutch).
2. J.C.M. BAETEN, J.A. BERGSTRA (1988). Global renaming operators in concrete process algebra. *Information and Computation 78(3),* 205-245.
3. J.C.M. BAETEN, J.A. BERGSTRA, J.W. KLOP (1987). Conditional axioms and α/β calculus in process algebra. M. WIRSING (ed.). *Proc. IFIP Conf. on Formal Description of Programming Concepts - III,* Ebberup 1986, North-Holland, Amsterdam, 53-75.

4. J.C.M. Baeten, J.A. Bergstra, J.W. Klop (1987). On the consistency of Koomen's Fair Abstraction Rule. *Theoretical Computer Science 51 (1/2),* 129-176.
5. J.W. de Bakker, J.I. Zucker (1982). Denotational semantics of concurrency. *Proc. 14th ACM Symp. Theory of Comp.,* 153-158.
6. J.W. de Bakker, J.I. Zucker (1982). Processes and the denotational semantics of concurrency. *Information and Control 54 (1/2),* 70-120.
7. J.A. Bergstra, J.W. Klop (1982). *Fixed Point Semantics in Process Algebras,* MC Report IW 206, Centre for Mathematics and Computer Science, Amsterdam.
8. J.A. Bergstra, J.W. Klop (1984). Process algebra for synchronous communication. *Information and Control 60 (1/3),* 109-137.
9. J.A. Bergstra, J.W. Klop (1985). Algebra of communicating processes with abstraction. *Theoretical Computer Science 37 (1),* 77-121.
10. J.A. Bergstra, J.W. Klop (1986). Algebra of communicating processes. J.W. de Bakker, M. Hazewinkel, J.K. Lenstra (eds.). *Mathematics and Computer Science,* CWI Monograph 1, North-Holland, Amsterdam, 89-138.
11. 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.). *Mathematics and Computer Science II,* CWI Monograph 4, North-Holland, Amsterdam, 61-94.
12. J.A. Bergstra, J.W. Klop (1986) Verification of an Alternating Bit Protocol by means of process algebra. W. Bibel, K.P. Jantke (eds.). *Math. Methods of Spec. and Synthesis of Software Systems '85, Math. Research 31,* Akademie-Verlag Berlin, 9-23. Also appeared as CWI Report CS-R8404, Centre for Mathematics and Computer Science, Amsterdam, 1984.
13. J.A. Bergstra, J.W. Klop, E.-R. Olderog (1987). Failures without chaos: a new process semantics for fair abstraction. M. Wirsing (ed.). *Proc. IFIP Conf. on Formal Description of Programming Concepts - III,* Ebberup 1986, North-Holland, Amsterdam, 77-103.
14. R.J. van Glabbeek (1987). Bounded nondeterminism and the approximation induction principle in process algebra. F.J. Brandenburg, G. Vidal-Naquet, M. Wirsing (eds.). *Proc. STACS 87,* LNCS 247, Springer-Verlag, 336-347.
15. C.A.R. Hoare (1984). *Notes on Communicating Sequential Processes,* International Summer School in Marktoberdorf: Control Flow and Data Flow, Munich.
16. C.A.R. Hoare (1985). *Communicating Sequential Processes,* Prentice Hall.
17. E. Kranakis (1986). *Approximating the Projective Model,* CWI Report CS-R8607, Centre for Mathematics and Computer Science, Amsterdam.

To appear in: *Proc. of Conf. on Math. Logic and Applications,* Druzhba, Plenum Publ. Corp., New York, 273-282.

18. R. MILNER (1980). *A Calculus of Communicating Systems,* LNCS 92, Springer-Verlag.
19. R. MILNER (1984). A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences 28 (3),* 439-466.
20. D.M.R. PARK (1981). Concurrency and automata on infinite sequences. *Proc. 5th GI Conference,* LNCS 104, Springer-Verlag.
21. 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.