Process Algebra: Specification and Verification in Bisimulation Semantics

J.A. Bergstra
University of Amsterdam, Department of Computer Science
P.O. Box 19268, 1000 GG Amsterdam, The Netherlands
and
State University of Utrecht, Department of Philosophy
P.O. Box 80010, 3508 TA Utrecht, The Netherlands

J.W. Klop
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

This paper addresses itself primarily to readers who have not had much exposure to algebraic approaches to concurrency, or as we will call it, process algebra. We will describe an algebraic framework called $ACP_{\tau}^{\#}$ (Algebra of Communicating Processes with abstraction and additional features), which is suitable for both specification and verification of communicating processes. Except in two instances we give no proofs; but there are many references to the places where these can be found. One instance where we do give a proof is the verification of the Alternating Bit Protocol. Here the point is that an algebraic proof can be given. The formal system $ACP_{\tau}^{\#}$ is, at least theoretically, very close to a universal system for process specification: every finitely branching computable process, can be finitely specified. In practice one needs additional operators for specifications; some of these are briefly discussed in a final section.

Our presentation will concentrate on process algebra as it has been developed since 1982 at the Centre for Mathematics and Computer Science, 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. Specifically, we will restrict ourselves to that side of the spectrum of process semantics which was initiated by MILNER [30] and which is

1. This research was partially sponsored by ESPRIT project nr. 432, Meteor.

called 'bisimulation semantics'. Thus, the important aspect of process algebra in which a unification and classification is sought for various algebraical approaches to process semantics ('comparative concurrency semantics') is not represented here. From the point of view of process specification and verification this restriction is justified: at present the specification and verification facilities are, at least in the setting of ACP, most highly developed in bisimulation semantics, in any case more than in the ACP treatment of e.g. failure semantics.

ACKNOWLEDGEMENT. We thank J. Heering and J.C.M. Baeten for suggesting many improvements.

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
$X = (zero + up.Y).X$
$Y = down + up.Y.Y$

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-.... A question of mathematical interest only is: can COUNTER be defined in a single equation, without auxiliary processes? The negative answer is an immediate consequence of the following fact:

THEOREM 1. *Let a system* $\{X_i = T(X_1, \ldots, X_n) | \ i = 1, \ldots, n\}$ *of guarded fixed point equations over BPA be given. Suppose the solutions* X_i *are all perpetual. Then they are regular.*

Two concepts in this statement need explanation: a fixed point equation, like $X = (zero + up.Y).X$ 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 $X = (zero + up.Y).X$ 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 $X = X + a.X$ is not guarded. Furthermore, a process is *regular* if it has only finitely many 'states'; clearly, COUNTER is not regular since it has just as many states as there are natural numbers. Let us mention one other property of processes which have a finite recursive specification (by means of guarded recursion equations) in BPA: such processes are *uniformly finitely branching*. A process is finitely branching if in each of its states it can take steps (and thereby transform itself) to only finitely many subprocesses; for instance, the process defined by $X = (a + b + c)X$ has in each state branching degree 3. 'Uniformly' means that there is uniform bound on the branching degrees throughout the process.

In fact, a more careful treatment is necessary to define concepts like 'branching degree' rigorously. For, clearly, the branching degree of $a + a$ ought to be the same as that of the process 'a', since $a + a = a$. And the process $X = aX$ will be the same as the process $X = aaX$; in turn these will be identified with the process $X = aX + aaX$. In the sequel we will discuss the semantic criterion by means of which these processes are identified ('bisimilarity'). MILNER [31] has found a simple axiom system (extending BPA) which is able to deal with recursion and which is complete for regular processes with respect to 'bisimilarity'.

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
$\delta+x=x$
$\delta.x=\delta$

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 without 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 succintly defined by the axioms:

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

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 ($\|$). We conclude this section about PA by mentioning the following fact:

THEOREM 2. *Every process which is recursively defined in PA and has an infinite trace, has an eventually periodic trace.*

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 as:

(1) projective limits ([14,15]);
(2) complete metrical spaces, as in the work of DE BAKKER and ZUCKER [6,7];
(3) quotients of graph domains (a graph domain is a set of process graphs or transition diagrams), as in MILNER [30];

(4) the 'explicit' models of HOARE [25];
(5) ultraproducts of finite models (KRANAKIS [28]).

In Section 13 we will discuss a model as in (3). As to (5), these models are only of theoretical interest: models thus obtained contain 'weird' processes such as $x=\sqrt{a^{\omega}}$, a process satisfying $x^2=a^{\omega}=a.a.a...$ while $x\neq x^2$.

Here, we look at (2). First, define the projection operators $\pi_n (n\geq 1)$, cutting off a process at level n:

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

TABLE 6

E.g., for X defining BAG:

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

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. Call this model the *standard model.* In fact, the guardedness condition is exactly what is needed to associate a contracting operator on the complete metrical space with a guarded recursion equation. (E.g. to the recursion equation $X=aX$ the contracting function $f(x)=ax$ is associated; indeed $d(f(x),f(y))\leq d(x,y)/2$.) The contraction theorem of Banach then proves the existence of a unique fixed point. This model construction has been employed in various settings by DE BAKKER and ZUCKER [6,7], who posed the question whether *unguarded* fixed point equations, such as $X=aX+X$ or $Y=(aY\|Y)+b$, always have a solution in the standard model as well. This turns out to be the case:

THEOREM 3 ([10]). *Let q be an arbitrary process in the standard model, and let $X=s(X)$ be a recursion equation in the signature of PA. Then the sequence q, $s(q)$, $s(s(q))$, $s(s(s(q)))$,... converges to a solution $q^*=s(q^*)$.*

In general, the fixed points $q^*=s(q^*)$ are not unique. The proof in [10] is combinatorial in nature; it is not at all clear whether this convergence result can be obtained by the 'usual' convergence proof methods, such as invoking

Banach's fixed point theorem or (in a complete partial order setting) the Knaster-Tarski fixed point theorem. In KRANAKIS [29] the present theorem is extended to the case where $s(X)$ may contain parameters.

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 $A=\{a,b,c,\ldots,\delta\}$ is the action alphabet, let us adopt a binary communication function $|:A\times A\rightarrow A$ satisfying

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

TABLE 7

(Here a,b vary over A, including δ.) 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 axiomatisation: left-merge ($\lfloor\!\lfloor$) as before and $|$ (communication merge or 'bar'), which is an extension of the communication function to all processes, not only the atoms. The axioms for $\|$ and its auxiliary operators are:

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

TABLE 8

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 $a \in H$
$\partial_H(x+y)=\partial_H(x)+\partial_H(y)$
$\partial_H(xy)=\partial_H(x).\partial_H(y)$

TABLE 9

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 $\partial_H(x_1 \| \ldots \| x_n)$. 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. A useful theorem to break down such 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. (In fact we have to require associativity of '|' first - see Table 10.)

THEOREM 4 (Expansion Theorem).

$$x_1 \| \ldots \| x_k = \Sigma_i x_i \mathbin{\|\!_} X_k^i + \Sigma_{i \neq j}(x_i|x_j) \mathbin{\|\!_} X_k^{i,j}$$

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 $x_i, x_j (k \geq 3)$. 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 holds:

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

TABLE 10

(As in Section 4 we can construct the 'standard' model for ACP; in this model the above axioms are valid. We will return to the existence and construction of models later.)

What about the defining power of ACP? The following is an example of a

process p, 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 $H=\{c,d\}$.

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

Then $p=ba(ba^2)^2(ba^3)^2(ba^4)^2$.... Indeed, using the axioms in ACP and putting $p_n=\partial_H(dc^nY\|Z)$ for $n\geq 1$, one proves that $p_n=ba^nba^{n+1}p_{n+1}$ (see [11]). By Theorem 2 in Section 3, p is not definable in PA, since the one infinite trace of p is not eventually periodic.

We will often adopt the following special format for the communication function, called *read-write communication.* Let a finite set D of *data* d and a set $\{1,...,p\}$ of *ports* be given. Then the alphabet consists of *read* actions $ri(d)$ and *write* actions $wi(d)$, for $i=1,...,p$ and $d\in D$. The interpretation is: read datum d at port i, resp. write datum d at port i. Furthermore, the alphabet contains actions $ci(d)$ for $i=1,...,p$ 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 hand-shaking axiom: all communications are binary.

In order to illustrate the defining power of ACP, we will now give an infinite specification of the process behaviour of a queue with input port 1 and output port 2. Here D is a finite set of data (finite since otherwise the sums in the specification below would be infinite, and we do not consider infinite expressions), D^* is the set of finite sequences σ of elements from D; the empty sequence is λ. The sequence $\sigma*\sigma'$ is the concatenation of sequences σ,σ'.

QUEUE
$Q=Q_\lambda=\Sigma_{d\in D}r1(d).Q_d$
$Q_{\sigma*d}=s2(d).Q_\sigma+\Sigma_{e\in D}r1(e).Q_{e*\sigma*d}$ (for all $d\in D$ and $\sigma\in D^*$)

TABLE 11

Note that this infinite specification uses only the signature of BPA. We have the following remarkable fact:

THEOREM 5. *Using read-write communication, the process Queue cannot be specified in ACP by finitely many recursion equations.*

For the lengthy proof see [2,19]. It should be mentioned that the process Queue can be finitely specified in ACP if the read-write restriction is dropped and n-ary communications are allowed; in the next section it is shown how this can be done. In the sequel we will present some other finite specifications of Queue using features to be introduced later.

6. Renaming

A useful 'add-on' feature is formed by the renaming operators ρ_f, where $f:A\rightarrow A$ is a function keeping δ fixed. A renaming ρ_f replaces each action 'a' in a process by $f(a)$. In fact, the encapsulation operators ∂_H are renaming operators; f maps $H\subseteq A$ to δ and fixes $A-H$ pointwise. The following axioms, where 'id' is the identity function, are obvious:

RENAMING
$\rho_f(a)=f(a)$
$\rho_f(x+y)=\rho_f(x)+\rho_f(y)$
$\rho_f(xy)=\rho_f(x).\rho_f(y)$
$\rho_{id}(x)=x$
$(\rho_f\circ\rho_g)(x)=\rho_{f\circ g}(x)$

TABLE 12

Again the defining power is enhanced by adding this feature. While Queue as in the previous section could not yet be finitely specified, it can now.

The actions are the $r1(d)$, $s2(d)$ as before; there are moreover 'auxiliary' actions $r3(d)$, $s3(d)$, $c3(d)$ for each datum d. Communication is given by $r3(d)|s3(d)=c3(d)$ and there are no other communications. If we let $\rho_{c3\rightarrow s2}$ be the renaming $c3(d)\rightarrow s2(d)$ and $\rho_{s2\rightarrow s3}:s2(d)\rightarrow s3(d)$, then for $H=\{s3(d),r3(d)|d\in D\}$ the following two guarded recursion equations give a finite specification of Queue:

QUEUE, FINITE SPECIFICATION
$Q=\Sigma_{d\in D}r1(d)(\rho_{c3\rightarrow s2}\circ\partial_H)(\rho_{s2\rightarrow s3}(Q)\Vert s2(d).Z)$
$Z=\Sigma_{d\in D}r3(d).Z$

TABLE 13

(This little gem was inspired by a similar specification in HOARE [24]. The present formulation is from BAETEN and BERGSTRA [2].) The explanation that this is really Queue is as follows. We intend that Q processes data d in a queue-like manner, by performing 'input' actions $r1(d)$ and 'output' actions $s2(d)$. So $\rho_{s2\rightarrow s3}(Q)$ processes data in queue-like manner by performing input actions $r1(d)$, output actions $s3(d)$. First consider the parallel system

$Q' = \partial_H(\rho_{s2\to s3}(Q) \| Z)$: since Z universally accepts $s3(d)$ and transforms these into $c3(d)$, this is just the queue with input $r1(d)$, output $c3(d)$. Now the process $Q^* = \partial_H(\rho_{s2\to s3}(Q) \| s2(d).Z)$ appearing in the recursion equation, is just like Q' but with the obligation to perform output action $s2(d)$ *before all output actions* $c3(d)$; this obligation is enforced since $s2(d)$ must be passed before $\rho_{s2\to s3}(Q)$ and Z can communicate and thereby create the output actions $c3(d)$. So $\rho_{c3\to s2}(Q^*) = Q_d$, the queue loaded with d, in the earlier notation used for the infinite specification of Queue (Table 11). But then $Q = \Sigma_{d\in D} r1(d).Q_d$ and this is exactly what we want.

In fact, the renamings used in this specification can be removed in favour of a more complicated communication format, as follows. Replace in the specification above $\rho_{s2\to s3}(Q)$ by $\partial_{s2}(Q \| V)$ where $V = \Sigma_d s2^*(d).V$ and $S2 = \{s2(d), s2^*(d) | d \in D\}$ with communications $s2(d)|s2^*(d) = s3(d)$ for all d. To remove the other renaming operator, put $P = \partial_H(\partial_{S2}(Q \| V) \| s2(d).Z)$, and replace $\rho_{c3\to s2}(P)$ by $\partial_{C3}(P \| W)$ where $W = \Sigma_d c3^*(d).W$ and $c3(d)|c3^*(d) = s2(d)$ for all d. However, though the renamings are removed in this way, the communication is no longer of the read-write format, or even in the hand shaking format, since we have ternary nontrivial communications $s2(d) = c3(d)|c3^*(d) = r3(d)|s3(d)|c3^*(d)$. As we already stated in the last theorem, this is unavoidable.

7. 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.

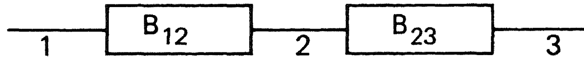


FIGURE 1

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 2, 3. However, some (rather involved) calculations learn that $\mathbf{B}_{13} = \Sigma_{d\in D} r1(d).((c2(d)s3(d)) \| \mathbf{B}_{13})$; 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 special renaming 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 $a \in I$
$\tau_I(x+y)=\tau_I(x)+\tau_I(y)$
$\tau_I(xy)=\tau_I(x).\tau_I(y)$

TABLE 14

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 [31] has devised some simple axioms which 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 15

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:

$$\tau_I(\mathbf{B}_{13}) = \tau_I(\Sigma r1(d)(c2(d).s3(d) \| \mathbf{B}_{13})) = {}^{(*)}\Sigma r1(d)(\tau.s3(d) \| \tau_I(\mathbf{B}_{13}))$$

$$= \Sigma(r\,1(d).\tau.s\,3(d)) \mathbin{\lfloor\!\lfloor} \tau_I(\mathbf{B}_{13}) = \Sigma(r\,1(d).s\,3(d)) \mathbin{\lfloor\!\lfloor} \tau_I(\mathbf{B}_{13})$$
$$= \Sigma r\,1(d)(s\,3(d) \| \tau_I(\mathbf{B}_{13}))$$

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

$$B_{13} = \Sigma r\,1(d)(s\,3(d) \| B_{13}).$$

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 now. (Specifically, in (*) an appeal to the 'alphabet calculus' below is needed and (**) requires the principle RSP, also below.)

8. PROOF RULES FOR RECURSIVE SPECIFICATIONS

We have now presented a survey of ACP_τ; we refer to [12] for an analysis of this proof system as well as a proof that (when the hand shaking axiom is adopted) the Expansion Theorem carries over from ACP to ACP_τ unchanged. 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 $X = a.\tau_{\{a\}}(X)$ possesses infinitely many solutions, even though it looks very guarded. (The solutions are: $a.q$ where q satisfies $\tau_{\{a\}}(q) = q$.) That the presence of abstraction operators in recursive specifications causes trouble, was first noticed by HOARE [24,25].

As to (iii), we still have to define projections π_n in the presence of the τ-

action. The extra clauses are:

PROJECTION, CONTINUED
$\pi_n(\tau)=\tau$
$\pi_n(\tau x)=\tau.\pi_n(x)$

TABLE 16

So, τ-steps do not add to the depth; this is enforced by the τ-laws (since, e.g., $a\tau b=ab$ and $\tau a=\tau a+a$). Remarkably, there are infinitely many different terms t_n (that is, different in the term model of ACP_τ), built from τ and a single atom a, such that t_n has depth 1, i.e. $t=\pi_1(t)$. The t_n are inductively defined as follows:
$t_0=a,\ t_1=\tau a,\ t_2=\tau,\ t_3=\tau(a+\tau),\ t_4=a+\tau a,\ t_{4k+i}=\tau.t_{4k+i-1}$ for $i=1,3$, $t_{4k+i}=t_{4k+i-3}+t_{4k+i-5}$ for $i=0,2$.

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 13. 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 $a.a.....a$ (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 13) 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 6. $AIP^- \Rightarrow RSP$.

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. The following curious specification of Queue is obtained in this manner. We want to specify Q_{12}, the queue from port 1 to 2, using an auxiliary port 3 and concatenating auxiliary queues Q_{13}, Q_{32}; then we abstract from the internal transaction at port 3. Write, in an ad hoc notation, $Q_{12}=Q_{13}*Q_{32}$. Now Q_{13} can be similarly split up: $Q_{13}=Q_{12}*Q_{32}$. This gives rise to six similar equations: $Q_{ab}=Q_{ac}*Q_{cb}$ where $\{a,b,c\}=\{1,2,3\}$. (See Figure 2.)

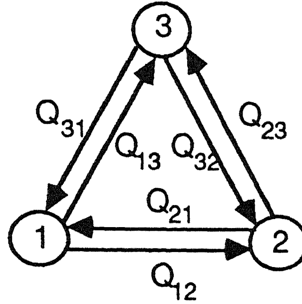


FIGURE 2

These six queues, which are merely renamings of each other, can now be specified in terms of each other as in the following table. One can prove that these recursion equations, though not abstraction-free, indeed have a unique solution.

QUEUE, FINITE SPECIFICATION WITH ABSTRACTION
$Q_{ab} = \Sigma_{d \in D} ra(d).\tau_c \circ \partial_c \, (Q_{ac} \| sb(d).Q_{cb})$ for $\{a,b,c\} = \{1,2,3\}$

TABLE 17

Here the usual read-write notation is used: $ri(d)$ means read d at i, $si(d)$: send d at i, communications are $ri(d)|si(d) = ci(d)$; further $\tau_i = \tau_{\{ci(d)|d \in D\}}$ and $\partial_i = \partial_{\{ri(d), si(d)|d \in D\}}$. This example shows that even with the restriction to read-write communication, ACP_τ is stronger than ACP.

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 $\alpha(x) \cap H = \varnothing \Rightarrow \partial_H(x) = x$. 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
$\alpha(\delta)=\varnothing$
$\alpha(\tau)=\varnothing$
$\alpha(a)=\{a\}$
$\alpha(\tau x)=\alpha(x)$
$\alpha(ax)=\{a\}\cup\alpha(x)$
$\alpha(x+y)=\alpha(x)\cup\alpha(y)$
$\alpha(x)=\cup_{n\geq 1}\alpha(\pi_n(x))$
$\alpha(\tau_I(x))=\alpha(x)-I$

TABLE 18

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
$\alpha(x)\vert(\alpha(y)\cap H)\subseteq H\Rightarrow\partial_H(x\Vert y)=\partial_H(x\Vert\partial_H(y))$
$\alpha(x)\vert(\alpha(y)\cap I)=\varnothing\Rightarrow\tau_I(x\Vert y)=\tau_I(x\Vert\tau_I(y))$
$\alpha(x)\cap H=\varnothing\Rightarrow\partial_H(x)=x$
$\alpha(x)\cap I=\varnothing\Rightarrow\tau_I(x)=x$

TABLE 19

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 $x\Vert y$.

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 $X=flip.(tail.X+head)$. 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 $I=\{flip, tail\}$. 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, $\tau_I(X)=\tau.head$. (This vivid example is from VAANDRAGER [33].)

Koomen's Fair Abstraction Rule (KFAR) is an algebraic rule enabling us to arrive at such a conclusion formally. (For an extensive analysis of this rule see [5].) The simplest form is

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

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

$$\frac{x_1 = i_1 x_2 + y_1, x_2 = i_2 x_1 + y_2 \quad (i_1, i_2 \in I)}{\tau_I(x_1) = \tau.\tau_I(y_1 + y_2)} \text{ KFAR}_2$$

and it is not hard to guess what the general formulation ($KFAR_n$, $n \geq 1$) will be (see Table 22 in Section 11). In fact, as observed by VAANDRAGER in [33], $KFAR_n$ can already be derived from $KFAR_1$ (at least in the framework of $ACP_\tau^\#$, to be discussed below).

KFAR is of great help in protocol verifications. An example is given in Section 14, where KFAR is 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 $\tau^\omega (= \tau.\tau.\tau.....)$ *livelock*. Formally, this is the process $\tau_{\{i\}}(x)$ where x is uniquely defined by the recursion equation $X = i.X$. Noting that $x = i.x = i.x + \delta$ 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 [17]).

11. $ACP_\tau^\#$, A FRAMEWORK FOR PROCESS SPECIFICATION AND VERIFICATION

We have now arrived at a framework which will be called $ACP_\tau^\#$, and which contains all the axioms and proof rules introduced so far. In Table 20 the list of all components of $ACP_\tau^\#$ is given; Table 21 contains the equational system ACP_τ and Table 22 contains the extra features leading first to, as we will call it, ACP_τ^+ and furthermore containing the proof principles which were just introduced, leading to $ACP_\tau^\#$. Note that for *specification* purposes one only needs ACP_τ or ACP_τ^+; for *verification* one will need $ACP_\tau^\#$ (an extensive example is given in Section 12). 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 for $ACP_\tau^\#$ has been provided yet; this will be done in Section 13. 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 by the verification of a simple protocol in the next section; at that point the semantics of $ACP_\tau^\#$ (in the form of some model) has, on purpose, not yet been provided. 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.

$ACP_{\tau}^{\#}$	
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
RENAMING	RN
PROJECTION	PR1-4
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^{-}
KOOMEN'S FAIR ABSTRACTION RULE	KFAR

TABLE 20

The system up to the first double bar is ACP; up to the second double bar we have ACP_{τ}, and up to the third double bar, ACP_{τ}^{+}.

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		
$a\mid b=b\mid a$	C1		
$(a\mid b)\mid c=a\mid(b\mid c)$	C2		
$\delta\mid a=\delta$	C3		
$x\parallel y=x\mathbin{\lfloor\!\lfloor} y+y\mathbin{\lfloor\!\lfloor} x+x\mid y$	CM1		
$a\mathbin{\lfloor\!\lfloor} x=ax$	CM2	$\tau\mathbin{\lfloor\!\lfloor} x=\tau x$	TM1
$ax\mathbin{\lfloor\!\lfloor} y=a(x\parallel y)$	CM3	$\tau x\mathbin{\lfloor\!\lfloor} y=\tau(x\parallel y)$	TM2
$(x+y)\mathbin{\lfloor\!\lfloor} z=x\mathbin{\lfloor\!\lfloor} z+y\mathbin{\lfloor\!\lfloor} z$	CM4	$\tau\mid x=\delta$	TC1
$ax\mid b=(a\mid b)x$	CM5	$x\mid\tau=\delta$	TC2
$a\mid bx=(a\mid b)x$	CM6	$\tau x\mid y=x\mid y$	TC3
$ax\mid by=(a\mid b)(x\parallel y)$	CM7	$x\mid\tau y=x\mid y$	TC4
$(x+y)\mid z=x\mid z+y\mid z$	CM8		
$x\mid(y+z)=x\mid y+x\mid z$	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 21

TABLE 22

REMAINING AXIOMS AND RULES FOR $ACP_\tau^\#$			
$\rho_f(a)=f(a)$	RN1	$\pi_1(ax)=a$	PR1
$\rho_f(x+y)=\rho_f(x)+\rho_f(y)$	RN2	$\pi_{n+1}(ax)=a.\pi_n(x)$	PR2
$\rho_f(xy)=\rho_f(x).\rho_f(y)$	RN3	$\pi_n(a)=a$	PR3
$\rho_{id}(x)=x$	RN4	$\pi_n(x+y)=\pi_n(x)+\pi_n(y)$	PR4
$(\rho_f\circ\rho_g)(x)=\rho_{f\circ g}(x)$	RN5	$\pi_n(\tau)=\tau$	PR5
$\rho_f(\tau)=\tau$	RN6	$\pi_n(\tau x)=\tau.\pi_n(x)$	PR6
$x\|y\|z=\delta$			HA
$x\|y=y\|x$			SC1
$x\|\|y=y\|\|x$			SC2
$x\|(y\|z)=(x\|y)\|z$			SC3
$(x \mathbin{⌊\!⌊} y)\mathbin{⌊\!⌊} z=x\mathbin{⌊\!⌊}(y\|\|z)$			SC4
$(x\|ay)\mathbin{⌊\!⌊} z=x\|(ay\mathbin{⌊\!⌊} z)$			SC5
$x\|\|(y\|\|z)=(x\|\|y)\|\|z$			SC6

$$x_1\|\ldots\|x_n=\sum_{1\leq i\leq n} x_i ⌊\!⌊ \Big(\mathop{\|}_{\substack{1\leq k\leq n\\ k\neq i}} x_k\Big) + \sum_{1\leq i<j\leq n}(x_i|x_j) ⌊\!⌊ \Big(\mathop{\|}_{\substack{1\leq k\leq n\\ k\neq i,k\neq j}} x_k\Big)\quad (n\geq 3) \qquad \text{ET}$$

$\alpha(\delta)=\varnothing$	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
$\alpha(x)=\cup_{n\geq 1}\alpha(\pi_n(x))$	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

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 abstraction free specification is determined by its finite projections*

$$\frac{\forall n\in\mathbb{Z}_k\ \ x_n=i_n.x_{n+1}+y_n\quad(i_n\in I)}{\tau_I(x_n)=\tau.\tau_I(\Sigma_{m\in\mathbb{Z}_k}y_m)}\ \ KFAR_k$$

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 projection axioms PR1-6 turn out to be definable from the other operators). This would however not enhance clarity. Also note that one of the standard concurrency axioms, SC5, is different (namely more restrictive) than the corresponding one for the situation without τ in Table 9 (the second axiom).

So $ACP_\tau^{\#}$ is 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. A finite specification (of which an expression is a particular case) is trivially computable; an infinite specification $\{E_n | n \geq 0\}$ where E_n is the recursion equation $X_n = T(X_1, \ldots, X_{f(n)})$ is computable if after some coding, in which E_n is coded as a natural number e_n, the sequence $\{e_n | n \geq 0\}$ is computable. Here an important question arises: *is every computable specification provably equal to a finite specification ?* At present we are unable to answer this question; but we can state that the answer is affirmative *relative to certain models* of $ACP_\tau^{\#}$. Before we elaborate this, a verification of a simple protocol is demonstrated.

12. An algebraic verification of the Alternating Bit Protocol

In this section we will demonstrate a verification of a simple communication protocol, the Alternating Bit Protocol, in the framework of $ACP_\tau^{\#}$. (In fact, not all of $ACP_\tau^{\#}$ is needed.) This verification is from [13]; the present streamlined treatment was kindly made available to us by F.W. Vaandrager (CWI Amsterdam).

Let D be a finite set of data. Elements of D are to be transmitted by the ABP from port 1 to port 2. The ABP can be visualized as follows:

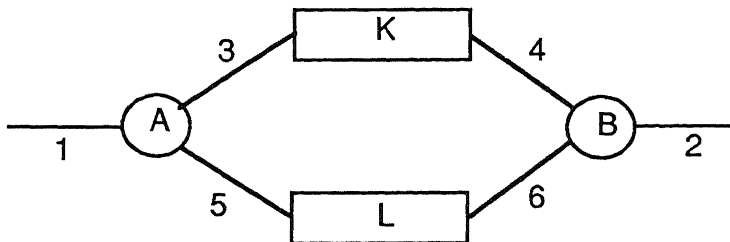


Figure 3

There are four components:

A: Reads a Message (RM) at 1. Thereafter it Sends a Frame (SF), consisting of the message and a control bit, into channel K until a correct

Acknowledgement has been Received (RA) via channel L. The equations for A are as follows. We will always use the notations: datum $d \in D$, bit $b \in \{0,1\}$, frame $f \in D \times \{0,1\}$ (so a frame f is of the form db).

$$\begin{array}{|l|}\hline A = RM^0 \\ RM^b = \Sigma_d\, r1(d).SF^{db} \\ SF^{db} = s3(db).RA^{db} \\ RA^{db} = (r5(1-b) + r5(e)).SF^{db} + r5(b).RM^{1-b} \\ \hline\end{array}$$

K: data transmission channel K communicates elements of $D \times \{0,1\}$, and may communicate these correctly or communicate an error value e. K is supposed to be fair in the sense that it will not produce an infinite consecutive sequence of error outputs.

$$\begin{array}{|l|}\hline K = \Sigma_f\, r3(f).K^f \\ K^f = (\tau.s4(e) + \tau.s4(f)).K \\ \hline\end{array}$$

The τ's in the second equation express that the choice whether or not a frame f is to be communicated correctly, cannot be influenced by one of the other components.
B: Receives a Frame (RF) via channel K. If the control bit of the frame is OK, then the Message is Sent (SM) at 2. B Sends back Acknowledgement (SA) via L.

$$\begin{array}{|l|}\hline B = RF^0 \\ RF^b = (\Sigma_d\, r4(d(1-b)) + r4(e)).SA^{1-b} + \Sigma_d\, r4(db).SM^{db} \\ SA^b = s6(b).RF^{1-b} \\ SM^{db} = s2(d).SA^b \\ \hline\end{array}$$

L: the task of acknowledgement transmission channel L is to communicate boolean values from B to A. The channel L may yield error outputs but is also supposed to be fair.

$$\begin{array}{|l|}\hline L = \Sigma_b\, r6(b).L^b \\ L^b = (\tau.s5(e) + \tau.s5(b)).L \\ \hline\end{array}$$

Define $\mathbf{D} = D \cup (D \times \{0,1\}) \cup \{0,1\} \cup \{e\}$. $\mathbf{D}$ is the set of 'generalized' data (i.e. plain data, frames, bits, error) that occur as parameter of atomic actions. We use the notation: $g \in \mathbf{D}$. For $t \in \{1,2,\ldots,6\}$ there are send, read, and communication actions:

$$A = \{st(g), rt(g), ct(g) \mid g \in \mathbf{D},\ t \in \{1,2,\ldots,6\}\}.$$

We define communication by $st(g) | rt(g) = ct(g)$ for $g \in \mathbf{D}$, $t \in \{1,2,\ldots,6\}$ and all other communications give δ. Define the following two subsets of A:

$$H = \{st(g), rt(g) \mid t \in \{3,4,5,6\},\ g \in \mathbf{D}\}$$

$$I = \{ct(g) \mid t \in \{3,4,5,6\},\ g \in \mathbf{D}\}.$$

Now the ABP is described by ABP $= \tau_I \circ \partial_H(A \| K \| B \| L)$. The fact that this is a

correct protocol is asserted by

THEOREM 7. $ACP_\tau^\# \vdash ABP = \Sigma_d\, r1(d).s2(d).ABP$.

(Actually, we need only the part of $ACP_\tau^\#$ consisting of $ACP_\tau + SC + RDP + RSP + CA + KFAR$ - see Tables 21, 22.)

PROOF. Let $I' = \{ct(g) | t \in \{3,4,5\}, f \in \mathbf{D}\}$. We will use $[x]$ as a notation for $\tau_{I'} \circ \partial_H(x)$. Consider the following system of recursion equations:

(0) $X = X_1^0$
(1) $X_1^b = \Sigma_d\, r1(d).X_2^{db}$
(2) $X_2^{db} = \tau.X_3^{db} + \tau.X_4^{db}$
(3) $X_3^{db} = c6(1-b).X_2^{db}$
(4) $X_4^{db} = s2(d).X_5^{db}$
(5) $X_5^{db} = c6(b).X_6^{db}$
(6) $X_6^{db} = \tau.X_5^{db} + \tau.X_1^{1-b}$

We claim that $ACP_\tau^\# \vdash X = [A \| K \| B \| L]$. We prove this by showing that $[A \| K \| B \| L]$ satisfies the same recursion equations (0)-(6) as X does. In the computations below, the bold-face part denotes the part of the expression currently being 'rewritten'.

$$[A \| K \| B \| L] = [RM^0 \| K \| RF^0 \| L] \tag{0}$$

$$\begin{aligned} [\mathbf{RM^b} \| K \| RF^b \| L] &= \Sigma_d\, r1(d).[\mathbf{SF^{db}} \| \mathbf{K} \| RF^b \| L] \\ &= \Sigma_d\, r1(d).\tau.[RA^{db} \| K^{db} \| RF^b \| L] \\ &= \Sigma_{d \in D}\, r1(d).[RA^{db} \| K^{db} \| RF^b \| L] \end{aligned} \tag{1}$$

$$\begin{aligned} [RA^{db} \| \mathbf{K^{db}} \| RF^b \| L] &= \tau.[RA^{db} \| \mathbf{s4(e).K} \| \mathbf{RF^b} \| L] \\ + \tau.[RA^{db} \| \mathbf{s4(db).K} \| \mathbf{RF^b} \| L] &= \tau.[RA^{db} \| K \| SA^{1-b} \| L] \\ &\quad + \tau.[RA^{db} \| K \| SM^{db} \| L] \end{aligned} \tag{2}$$

$$\begin{aligned} [RA^{db} \| K \| \mathbf{SA^{1-b}} \| \mathbf{L}] &= c6(1-b).[RA^{db} \| K \| RF^b \| \mathbf{L^{1-b}}] \\ &= c6(1-b).(\tau.[\mathbf{RA^{db}} \| K \| RF^b \| \mathbf{s5(e).L}] \\ &\quad + \tau.[\mathbf{RA^{db}} \| K \| RF^b \| \mathbf{s5(1-b).L}]) \\ &= c6(1-b).\tau.[\mathbf{SF^{db}} \| \mathbf{K} \| RF^b \| L] \\ &= c6(1-b).\tau.\tau.[RA^{db} \| K^{db} \| RF^b \| L] \\ &= c6(1-b).[RA^{db} \| K^{db} \| RF^b \| L]. \end{aligned} \tag{3}$$

$$[RA^{db} \| K \| \mathbf{SM^{db}} \| L] = s2(d).[RA^{db} \| K \| SA^b \| L]. \tag{4}$$

$$[RA^{db} \| K \| \mathbf{SA^b} \| \mathbf{L}] = c6(b).[RA^{db} \| K \| RF^{1-b} \| L^b]. \tag{5}$$

$$[RA^{db} \| K \| RF^{1-b} \| \mathbf{L^b}] = \tau.[\mathbf{RA^{db}} \| K \| RF^{1-b} \| \mathbf{s5(e).L}] \tag{6}$$

$$+\ \tau.[\mathbf{RA^{db}}\|K\|RF^{1-b}\|\mathbf{s5(b).L}]$$
$$=\tau.[SF^{db}\|K\|RF^{1-b}\|L]$$
$$+\ \tau.[RM^{1-b}\|K\|RF^{1-b}\|L].$$

$$[\mathbf{SF^{db}}\|\mathbf{K}\|RF^{1-b}\|L]=\tau.[RA^{db}\|\mathbf{K^{db}}\|RF^{1-b}\|L] \tag{7}$$
$$=\tau.(\tau[RA^{db}\|\mathbf{s4(e).K}\|\mathbf{RF^{1-b}}\|L]$$
$$+\ \tau.[RA^{db}\|\mathbf{s4(db).K}\|\mathbf{RF^{1-b}}\|L])$$
$$=\tau.[RA^{db}\|K\|SA^{b}\|L].$$

Now substitute (7) in (6) and apply RSP + RDP. Using the conditional axioms (see Table 22, Section 11) we have ABP $=\tau_I(X)=\ \tau_I(X_1^0)$. Further, an application of $KFAR_2$ gives $\tau_I(X_2^{db})=\ \tau.\tau_I(X_4^{db})$ and $\tau_I(X_5^{db})=\ \tau.\tau_I(X_1^{1-b})$. Hence,

$$\tau_I(X_1^b)=\Sigma_d\ r1(d).\tau_I(X_2^{db})=\Sigma_d\ r1(d).\tau_I(X_4^{db})$$
$$=\Sigma_d\ r1(d).s2(d).\tau_I(X_5^{db})=\Sigma_d\ r1(d).s2(d).\tau_I(X_1^{1-b})$$

and thus

$$\tau_I(X_1^0)=\Sigma_d\ r1(d).s2(d).\Sigma_{d'}r1(d').s2(d').\tau_I(X_1^0)$$
$$\tau_I(X_1^1)=\Sigma_d\ r1(d).s2(d).\Sigma_{d'}r1(d').s2(d').\tau_I(X_1^1).$$

Applying RDP + RSP gives $\tau_I(X_1^0)=\ \tau_I(X_1^1)$ and therefore $\tau_I(X_1^0)=\Sigma_d r1(d).s2(d).\tau_I(X_1^0)$, which finishes the proof of the theorem. □

More complicated communication protocols have been verified in $ACP_\tau^\#$ recently by VAANDRAGER [33]: a Positive Acknowledgement with Retransmission protocol and a One Bit Sliding Window protocol. There the notion of *redundancy in a context* is used as a tool which facilitates the verifications. A related method, using a modular approach, is employed in KOYMANS and MULDER [26], where a version of the Alternating Bit Protocol called the Concurrent Alternating Bit Protocol is verified in $ACP_\tau^\#$. (In fact, also in the verifications in [26], [33] one only needs the part of $ACP_\tau^\#$ mentioned after Theorem 7.)

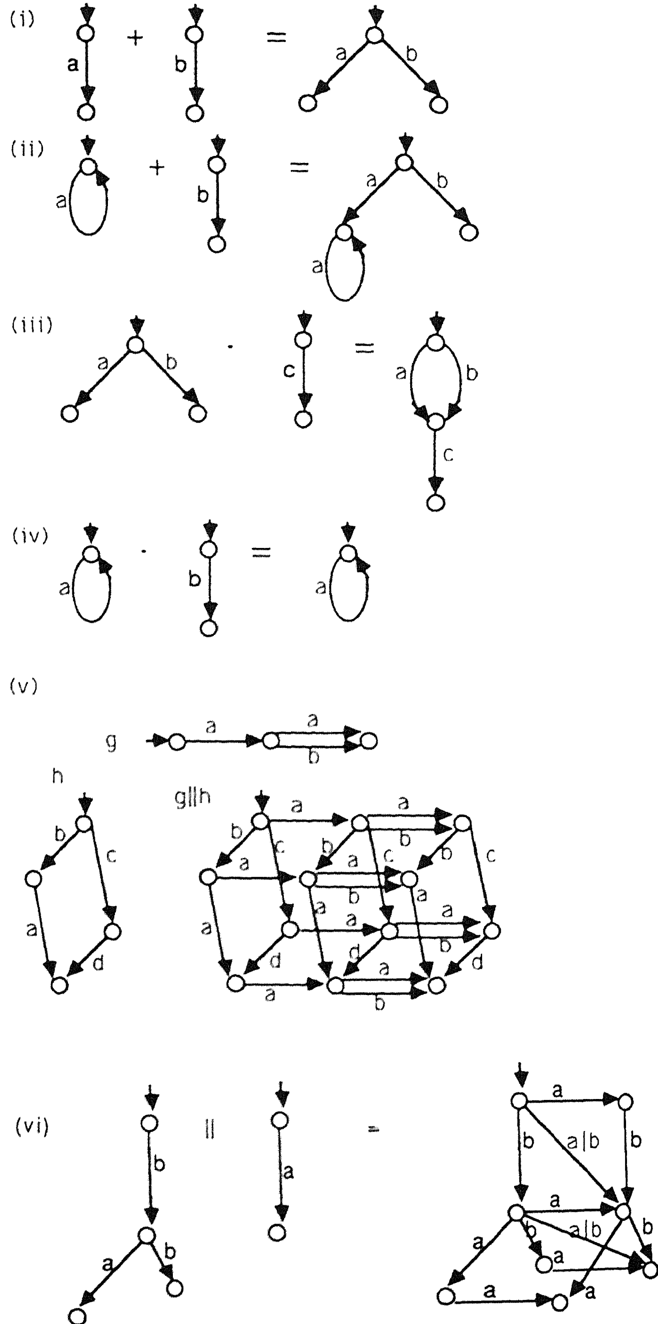
13. THE GRAPH MODEL FOR $ACP_\tau^\#$

We will give a quick introduction to what we consider to be the 'main' model of $ACP_\tau^\#$. 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 $+, \cdot, \|, \lfloor\!\lfloor, |, \partial_H, \tau_I, \pi_n, \alpha$ in $ACP_\tau^\#$ we define operations in this domain of process graphs. Precise definitions can be found in [1,5]; 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 4(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 4(ii)). The product $g.h$ 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 4 (iii)). The merge $g\|h$ is obtained as a cartesian product of both graphs, with 'diagonal' edges for communications (see Figure 4(v) for an example without communication, and Figure 4(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.

FIGURE 4



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 [32]; it was anticipated by Milner's observational equivalence, see [30].) 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 8 ([5]). *$\mathbf{G}/\leftrightarrows$ is a model of $ACP_\tau^\#$.*

Remarkably, this graph model (as we will call it henceforth) 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 $\pi^n(g)=\pi^n(h)=a+a^2+a^3+...+a^n$, 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 does satisfy 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 $h'=g'+a^\omega$.

14. THE EXPRESSIVE POWER OF ACP_τ

ACP_τ is a powerful specification mechanism; in a sense it is a universal specification mechanism: *every finitely branching, computable process can be finitely specified* in ACP_τ. We have to be more precise about the notion of 'computable process'. First, an intuitive explanation: suppose a finitely branching process graph 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': a finite recipe describes the graph, in

the form of a coding of the nodes in natural numbers and recursive functions giving in-degree, out-degree, edge-labels. This notion of a computable process graph is rather obvious, and we will not give details of the definition here (these can be found in [5]).

Now even if g is an infinite process graph, it can 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_{s0} 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 $E=\{X_s=T_s(\mathbf{X})|s\in NODES(g_t)\}$ 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 $E_n:X_{sn}=T_{sn}(\mathbf{X})$ is a computable sequence). Now the specification which uniquely determines g is simply: $\{Y=\tau_{\{t\}}(X_{s0})\}\cup E$. In fact all specifications below will have the form $\{X=\tau_I(X_0),X_n=T_n(\mathbf{X})|n\geqslant 0\}$ where the guarded expressions $T_n(\mathbf{X})(=T_n(X_{i1},\ldots,X_{in}))$ 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 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 9. *Let the finitely branching and computable process graph g determine $\tilde{g}$ in the graph model of ACP_τ. Then there is a finite specification with restricted abstraction E in ACP_τ such that $[\![E]\!]=\tilde{g}$.*

Here $[\![E]\!]$ is the semantics of E in the graph model. The proof in [5] is by constructing a Turing machine in ACP$_\tau$; the 'tape' is obtained by glueing together two stacks. A stack has a simple finite specification, already in BPA; see [15]. A stronger fact would be the assertion that every computable specification with restricted abstraction in ACP$_\tau$ is provably equivalent (in ACP$_\tau^{\#}$) to a finite specification with restricted abstraction. At present we do not know whether this is true.

It should be noted that abstraction plays an essential role in this finite specification theorem. If $f:\mathbb{N}\rightarrow\{a,b\}$ is a sequence of a,b, let p_f be the process $f(0).f(1).f(2)\ldots\ldots$ (more precisely: the unique solution of the specification $\{X_n=f(n).X_{n+1}|n\geqslant 0\}$). Now:

THEOREM 10. *There is a computable function f such that process p_f is not definable by a finite specification without abstraction operator.*

A fortiori, p_f is not finitely definable in ACP. The proof in [5] is via a simple diagonalization argument.

The finite specification theorem, which is relative to the graph model of ACP$_\tau^{\#}$, in fact generalizes to the class of 'extensional' models. In order to

define this concept we first define the notion of 'canonical process graph' of a process in an arbitrary process algebra.

Let $\mathfrak{A}$ be a process algebra (i.e. a model of the axiom system under consideration, in casu ACP_τ). Let $p,q \in \mathfrak{A}$. We define *transition relations* $\rightarrow_a$, for every atomic action a, and $\rightarrow_\tau$, as follows: $p \rightarrow_a q$ iff $p = a.q + r$ for some r. Moreover, if $p = a + r$ for some r, then $p \rightarrow_a o$ where o is an auxiliary element not in the domain of $\mathfrak{A}$. The same with τ instead of 'a'. Now the *canonical process graph* of p (notation: $G(p)$) is the labeled and directed graph with root: p and with nodes all elements accessible from p via the transitions $\rightarrow_a, \rightarrow_\tau$. The edges of the canonical process graph are given by the transitions. Note that every element in every process algebra thus has a canonical process graph. In analogy with the situation in set theory, we will call a process algebra *extensional* if whenever p,q have the same process graph, they are equal. (Cf. the 'observable' process spaces in HESSELINK [22].) In an extensional model an element is fully determined by its transition relations to other elements. The models that we have introduced are all extensional. A process is *finitely branching* when its canonical graph is. Now we can define that a process is *computable* when its canonical graph is. The finite specification theorem above generalizes to:

THEOREM 11. *Let p be a finitely branching, computable process in an extensional process algebra (a model of ACP_τ). Then p can, in ACP_τ, be specified by a finite specification with restricted abstraction.*

It should be possible to remove the assumption 'finitely branching' in favour of 'countably branching', but we will not attempt to do so here.

15. A FUNDAMENTAL INCOMPATIBILITY

As we have seen, the graph model of $ACP_\tau^{\#}$ (Section 13) does not satisfy the unrestricted Approximation Induction Principle which states that every process is uniquely determined by its finite projections. It is natural to search for a model in which this principle does hold. However, R.J. VAN GLABBEEK (CWI Amsterdam) recently noticed that such a model does not exist, if one wishes to adhere to the very natural assumption that composition of abstraction operators is commutative. As always, we refer here to models which are trace consistent. We will consider the following consequence of RN5 in Table 22: $\tau_{\{a\}} \circ \tau_{\{b\}} = \tau_{\{b\}} \circ \tau\{a\}$ which we will denote by CA (commutativity of abstraction). Now we have:

THEOREM 12 ([21]). $ACP_\tau + KFAR_1 + RDP + RSP + CA + AIP \vdash \tau = \tau + \tau\delta$.

By way of exception we include the interesting proof. Consider the specifications $E = \{X_n = aX_{n+1} + b^n | n \geq 0\}$, $F = \{Y = bY\}$ and $G = \{Z = aZ + \tau\}$. By RDP+RSP we have unique solutions for these specifications; they will be denoted by $\underline{X}_n (n \geq 0)$, $\underline{Y}, \underline{Z}$. By $KFAR_1$ we have immediately: $\tau_{\{b\}}(\underline{Y}) = \tau\delta$ and $\tau_{\{a\}}(\underline{Z}) = \tau.\tau = \tau.$ Further, $\tau_{\{b\}}(\underline{X}_n) =$

$a.\tau_{\{b\}}(\underline{X}_{n+1})+\tau^n = a.\tau_{\{b\}}(\underline{X}_{n+1})+\tau$, so the sequence $\{\tau_{\{b\}}(\underline{X}_n)|n\geqslant 0\}$ is a solution of the infinite specification $G'=\{Z_n = aZ_{n+1}+\tau|n\geqslant 0\}$. Clearly this last specification is also satisfied by the sequence $\{\underline{Z},\underline{Z},\ldots\}$. Hence, by RSP, $\tau_{\{b\}}(\underline{X}_n)=\underline{Z}$. It follows that $\tau_{\{b\}}(a\underline{X}_0)=a\underline{Z}$; whence

$$\tau_{\{a\}}\circ\tau_{\{b\}}(a\underline{X}_0) = \tau_{\{a\}}(a\underline{Z}) = \tau.\tau_{\{a\}}(\underline{Z}) = \tau.\tau = \tau. \tag{1}$$

Now, using the τ-law T2 and in particular its consequence $\tau(x+y)=\tau(x+y)+x$, one proves easily that for all k:

$$\tau_{\{a\}}(a\underline{X}_1) = \tau_{\{a\}}(a\underline{X}_1) + b^k.$$

(E.g. for $k=0$: $\tau_{\{a\}}(a\underline{X}_0)= \tau.\tau_{\{a\}}(\underline{X}_0)= \tau.\tau_{\{a\}}(a\underline{X}_1+b)= \tau(\tau_{\{a\}}(a\underline{X}_1)+b)= \tau(\tau_{\{a\}}(a\underline{X}_1)+b)+b = \tau_{\{a\}}(a\underline{X}_0)+b$.)

So $\pi_k(\tau_{\{a\}}(a\underline{X}_0))= \pi_k(\tau_{\{a\}}(a\underline{X}_0))+b^k= \pi_k(\tau_{\{a\}}(a\underline{X}_0)+\underline{Y})$, for all k. Therefore by AIP: $\tau_{\{a\}}(a\underline{X}_0)= \tau_{\{a\}}(a\underline{X}_0)+\underline{Y}$. Hence, using (1) and CA:

$$\begin{aligned}\tau_{\{b\}}\circ\tau_{\{a\}}(a\underline{X}_0) &= \tau_{\{b\}}\circ\tau_{\{a\}}(a\underline{X}_0) + \tau_{\{b\}}(\underline{Y}) = \tau_{\{a\}}\circ\tau_{\{b\}}(a\underline{X}_0) + \tau_{\{b\}}(\underline{Y}) \\ &= \tau + \tau\delta,\end{aligned} \tag{2}$$

and again by (1) and CA: $\tau=\tau+\tau\delta$. □

So, in every theory extending ACP_τ, the combination of features AIP, KFAR, CA, RDP+RSP is impossible. Among such theories are also theories where the equivalence on processes is much coarser, such as in Hoare's well-known failure model [25]; this semantics is not discussed in the present paper. VAN GLABBEEK [21] moreover notices that there is quite a subtle trade-off between these four features. In the graph model of $ACP_\tau^{\#}$ we have AIP^-, KFAR, CA, RDP+RSP. There is also a failure model satisfying AIP, $KFAR^-$, CA, RDP+RSP, where $KFAR^-$ is a restricted form of KFAR (see [17]). In fact it seems that models can be found by weakening any of the four features that make up the impossible combination.

16. ADDITIONAL FEATURES

As we have seen in Section 14, ACP_τ is a universal specification system for (finitely branching) computable processes. Yet this does not preclude the search for additional operators on processes, in order to make finite specifications of computable processes not only theoretically possible, but also practically feasible. The two main additional operators which have been defined and studied in process algebra are the *priority operator* and the *state operator*.

By means of the priority operator θ one can enforce that certain actions are privileged and have priority over others. Thus θ is parameterized by a partial order $>$ on the set of atomic actions; the constant δ (deadlock) will always be the least element in this partial ordering. As an example, let atomic actions a,b,c be ordered by: $a,b<c$. Then $\theta(a+b+c)=c$, $\theta(a+b)=a+b$, $\theta(ax+cy)=c\theta(y)$. Using an auxiliary operator $\lhd$ ('unless') we can axiomatize θ in finitely many equations:

PRIORITY OPERATOR
$a \lhd b = a \quad \text{if} \neg(a < b)$
$a \lhd b = \delta \quad \text{if} a < b$
$x \lhd yz = x \lhd y$
$x \lhd (y + z) = (x \lhd y) \lhd z$
$xy \lhd z = (x \lhd z)y$
$(x + y) \lhd z = x \lhd z + y \lhd z$
$\theta(a) = a$
$\theta(xy) = \theta(x) \cdot \theta(y)$
$\theta(x + y) = \theta(x) \lhd y + \theta(y) \lhd x$

TABLE 23

The priority operator θ (with its axioms) can be joined with ACP (see Section 11); the result is called ACP_θ. Note that we do not join θ and ACP_τ; at present the interaction between τ and θ is not clear. In [4] an elimination theorem is proved stating that every closed ACP_θ-term is (in ACP_θ) provably equal to a BPA_δ-term, that is a term without occurrences of other operators than $\cdot$ and $+$. Using θ, one can model *interrupts* (see [4]). Another application is given in [9]: there a *put* and *get* mechanism has been modeled using ACP_θ. Communication by means of put and get mechanism differs from the synchronous hand shaking mechanism: even if the 'receiving' process is not enabled to receive the message, the 'sending' process can perform a put action, and proceed with its execution. Likewise, a receiving process can perform a get action even when there is nothing to get, and continue in that case. Using the put mechanism, it is shown in [9] how a *broadcasting* mechanism for arbitrarily many receivers can be modeled.

Another very useful operator is the *state operator* λ_s, where s is some state from a state space S. The essential equation is $\lambda_s(ax) = a'.\lambda_{s'}(x)$. Here s' and a' are the state and action respectively resulting from executing a in state s. The state operator is useful in designing an algebraic semantics for programming languages; when dealing with object-oriented programming languages or specification languages, it is useful to provide the state operator with a name m of the object in question. Thus $\lambda_s^m(x)$ can intuitively be perceived as the process resulting from input x (the 'program') in m (the 'machine') in s (the state of m). Writing $a' = a(m,s)$ (the *action function*) and $s' = s(m,a)$ (the *effect function*) the state operator is axiomatized by:

STATE OPERATOR
$\lambda_s^m(\delta)=\delta$
$\lambda_s^m(a)=a(m,s)$
$\lambda_s^m(ax)=a(m,s)\cdot\lambda_{s(m,a)}^m(x)$
$\lambda_s^m(x+y)=\lambda_s^m(x)+\lambda_s^m(y)$

TABLE 24

In fact, this state operator is a generalization of the renaming operator in Section 6. In [1] asynchronous communication is modeled using the state operator: here a message from sender to receiver may have some delay.

Another mechanism which is of interest for specifications is *process creation.* In [8] axioms for a process creation operator have been given; for some examples of its use see also [1]. A typical example is the modeling of the *sieve of Eratosthenes.*

Finally, we mention the work of VRANCKEN [34] where the *empty process* ϵ has been axiomatized. The basic axioms for this process are $\epsilon x=x$ and $x\epsilon=x$. It should be pointed out that addition of such a process requires careful consideration in order to preserve the consistency of the whole axiomatization. Using this process several short-cuts in process specifications can be obtained.

REFERENCES

1. J.C.M. BAETEN (1986). *Procesalgebra,* course notes (in Dutch), Department of Computer Science, University of Amsterdam.
2. J.C.M. BAETEN, J.A. BERGSTRA (1985). *Global Renaming Operators in Concrete Process Algebra,* CWI Report CS-R8521, Amsterdam.
3. J.C.M. BAETEN, J.A. BERGSTRA, J.W. KLOP (1985). *Conditional Axioms and α/β Calculus in Process Algebra,* CWI Report CS-R8502, Amsterdam.
4. J.C.M. BAETEN, J.A. BERGSTRA, J.W. KLOP (1985). *Syntax and Defining Equations for an Interrupt Mechanism in Process Algebra,* CWI Report CS-R8503, Amsterdam.
5. J.C.M. BAETEN, J.A. BERGSTRA, J.W. KLOP (1985). *On the Consistency of Koomen's Fair Abstraction Rule,* CWI Report CS-R8511, Amsterdam.
6. J.W. DE BAKKER, J.I. ZUCKER (1982). Denotational semantics of concurrency. *Proc. 14th ACM Symp. Theory of Comp.,* 153-158.
7. J.W. DE BAKKER, J.I. ZUCKER (1982). Processes and the denotational semantics of concurrency. *Information and Control 54 (1/2),* 70-120.
8. J.A. BERGSTRA (1985). *A Process Creation Mechanism in Process Algebra,* Logic Group Preprint Series Nr. 2, Dept. of Philosophy, State University of Utrecht.
9. J.A. BERGSTRA (1985). *Put and Get, Primitives for Synchronous Unreliable Message Passing,* Logic Group Preprint Series Nr. 3, Dept. of Philosophy, State University of Utrecht.
10. J.A. BERGSTRA, J.W. KLOP (1982). *Fixed Point Semantics in Process*

Algebras, MC Report IW 206, Amsterdam.

11. J.A. BERGSTRA, J.W. KLOP (1984). The algebra of recursively defined processes and the algebra of regular processes. J. PAREDAENS (ed.). *Proceedings 11th ICALP, Antwerpen 1984, Springer LNCS 172,* 82-95.
12. J.A. BERGSTRA, J.W. KLOP (1985). Algebra of communicating processes with abstraction. *TCS 37 (1),* 77-121.
13. J.A. BERGSTRA, J.W. KLOP (1984). Verification of an alternating bit protocol by means of process algebra. W. BIBEL, K.P. JANTKE (EDS.). Proc. of the International Spring School, Wendisch.-Rietz (GDR), April 1985, *Math. Methods of Spec. and Synthesis of Software Systems '85,* Akademie-Verlag Berlin 1986.
14. J.A. BERGSTRA, J.W. KLOP (1984). Process algebra for synchronous communication. *Information and Control 60 (1/3),* 109-137.
15. 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.
16. J.A. BERGSTRA, J.W. KLOP, E.-R. OLDEROG (1985). *Readies and Failures in the Algebra of Communicating Processes,* CWI Report CS-R8523, Amsterdam.
17. J.A. BERGSTRA, J.W. KLOP, E.-R. OLDEROG (1986). *Failure Semantics with Fair Abstraction,* CWI Report CS-R8609, Amsterdam.
18. J.A. BERGSTRA, J.W. KLOP, J.V. TUCKER (1983). Algebraic tools for system construction. E. CLARKE, D. KOZEN (eds.). *Logics of Programs, Proceedings '83, Springer LNCS 164,* 34-45.
19. J.A. BERGSTRA, J. TIURYN (1983). *Process Algebra Semantics for Queues,* MC Report IW 241, Amsterdam.
20. J.A. BERGSTRA, J.V. TUCKER (1984). *Top Down Design and the Algebra of Communicating Processes,* Sci. of Comp. Progr. 5 (2), p. 171-199.
21. R.J. VAN GLABBEEK (1986). *Bounded Nondeterminism and the Approximation Induction Principle in Process Algebra,* to appear as CWI Report, Amsterdam.
22. W. HESSELINK (1986). *Morfismen van Procesruimten, Universele Domeinen, Formele Implementaties en Fairness,* manuscript, Univ. of Groningen.
23. C.A.R. HOARE (1978). Communicating sequential processes. *Comm. ACM 21,* 666-677.
24. C.A.R. HOARE (1984). *Notes on Communicating Sequential Processes,* International Summer School in Marktoberdorf: Control Flow and Data Flow, Munich.
25. C.A.R. HOARE (1985). *Communicating Sequential Processes,* Prentice Hall.
26. 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.
27. C.P.J. KOYMANS, J.L.M. VRANCKEN (1985). *Extending Process Algebra with the Empty Process* ϵ, Logic Group Preprint Series Nr. 1, Dept. of Philosophy, State University of Utrecht.
28. E. KRANAKIS (1986). *Approximating the Projective Model,* CWI Report

CS-R8607, Amsterdam.

29. E. KRANAKIS (1986). *Fixed Point Equations with Parameters in the Projective Model,* CWI Report CS-R8606, Amsterdam.
30. R. MILNER (1980). *A Calculus of Communicating Systems, Springer LNCS 92.*
31. R. MILNER (1984). A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences 28, 3,* 439-466.
32. D.M.R. PARK (1981). Concurrency and automata on infinite sequences. *Proc. 5th GI Conference, Springer LNCS 104.*
33. F.W. VAANDRAGER (1986). *Verification of Two Communication Protocols by Means of Process Algebra,* CWI Report CS-R8608, Amsterdam.
34. J.L.M. VRANCKEN (1986). *The Algebra of Communicating Processes with Empty Process,* Report FVI 86-01, Computer Science Department, University of Amsterdam.