1991

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

Process algebra with signals and conditions

Computer Science/Department of Software Technology Report CS-R9103 January

Process Algebra with Signals and Conditions

J.C.M. Baeten
*Department of Software Technology, CWI,
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands*
*Programming Research Group, University of Amsterdam,
P.O. Box 41882, 1009 DB Amsterdam, The Netherlands*

J.A. Bergstra
*Programming Research Group, University of Amsterdam,
P.O. Box 41882, 1009 DB Amsterdam, The Netherlands*
*Department of Philosophy, Utrecht University,
Heidelberglaan 2, 3584 CS Utrecht, The Netherlands*

Several new operators are introduced on top of the algebra of communicating processes (ACP) from [BK 84] in order to incorporate stable signals in process algebra. Semantically this involves assigning labels to nodes of process graphs in addition to the actions that serve as labels of edges. The labels of nodes are called signals. In combination with the operators of BPA, two signal insertion operators allow to describe each finite tree labeled with actions and signals. In the context of parallel processes there is a new feature connected with signals: the signal observation mechanism. This mechanism is organised on basis of a signal observation function in very much the same way as the communication mechanism of ACP is organised around a communication function. Also, we discuss conditionals on signals and processes, and make much use of them in our examples.

1980 Mathematics Subject Classification (1985 revision): 68Q10, 68Q55, 68Q45, 68Q40.
1987 CR Categories: F.1.2, F.3.2, D.1.3, D.3.1, D.4.1.
Key words & Phrases: process algebra, ACP, signals, observation, condition, guard, guarded command.
Note: This research was sponsored in part by ESPRIT under contract 432, METEOR as well as under contract 3006, CONCUR. This paper is a revised and extended version of [BE 88a].

1. INTRODUCTION.

This paper is concerned primarily with concrete process algebra in the sense of [BB 88]. Concrete process algebra is that part of process algebra that does not involve Milner's silent action τ [MI 80, MI 89] or the empty action ε due to Vrancken [VR 86]. This paper is a revised and substantially extended version of [BE 88a]. An extended abstract of [BE 88a] was published as [BE 88b].

Process algebra is a well-established part of computer science ([AB 84], [BW 90], [HE 88], [HO 85], [MI 80, 89]). We discuss process algebra in the setting of ACP (see [BK 84], [BW 90]). The new feature that we hope to contribute to process algebra with this paper is the presence of explicit signals of a persistent nature. In addition we discuss the use of conditional expressions in process algebra. The original set-up of process algebra inherited from Milner's CCS views processes semantically as trees of actions. These actions are best thought of as atomic actions because otherwise the intuition behind the axioms becomes rather obscure. A mechanism of particular importance, that has not yet been analyzed in the setting of ACP, is the presence of visible aspects of the state of a process. Usually in process algebra the state of a process can only be understood (or observed) via the actions that can be performed from that state. In the set-up that will be presented here some aspects of the system state are visible not so much through the actions that will follow but

Report CS-R9103
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

much more directly as signals that persist in time for some extended duration. Typically, one might think of some light signal on a control panel. Several signals may be visible simultaneously and one is easily led to a boolean algebra of signals based on an empty signal and a composition operator on signals. The main step is to introduce two operators called signal insertion operators. In terms of a graph model of processes these signal insertion operators are able to place a signal as a label to a node in the graph. Given finite sets A and AS, using the root signal insertion operator and the terminal signal insertion operator it is possible to describe every finite tree with edges labeled by atomic actions from A and nodes labeled by compositions of atomic signals from AS. Without problem the operators can be used to give guarded recursive definitions of infinite processes with signals.

There is a need for motivation for this work. It seems that every extension of the operator set of process algebra makes the subject less focussed and comprehensible in the eyes of mathematically oriented observers. Our point of view, however, is that whenever it is difficult to model process phenomena in terms of existing operators of process algebra there is a reason to investigate extensions of the formalism. An essential restriction is that existing axioms remain valid in a each new setting. In other words one may extend the set of operators and axioms of ACP but may not modify it grossly. This does not imply that we expect that all phenomena can eventually be modeled in process algebra. On the contrary it is quite likely that process algebra will come to real limits sooner or later. The reason for this restriction is that it should be quite clear when a departure of process algebra is needed in order to find an adequate modeling of some sequential or concurrent mechanism. Respecting these restrictions, it is excluded that process algebra becomes a moving target altogether which claims universality but in fact shows lack of commitment to its basic principles. Now these basic principles are exactly the operators and axioms of BPA, PA and ACP (see e.g. [BK 84]) or the exposition below) but not the semantic models that have been defined for these axiom systems. Therefore we feel that the approach of this paper is methodologically adequate: all axioms of ACP are preserved, more operators and axioms (and in fact processes) are added that nevertheless all satisfy the axioms of ACP. Moreover the recursive definitions of the new operators all have the spirit of term rewriting that underlies the design of ACP.

The term rewriting analysis of ACP with signals has been carried out in BROUWER [BR 90]. He uses this analysis to obtain an *implementation* of ACP with signals. Moreover he has designed syntax that allows to incorporate signals in the process specification language PSF of MAUW & VELTINK [MV 90].

Of course it remains to be seen that the approach to the problem of incorporating stable and nonatomic signals in the framework of process algebra as presented in this paper is a useful one. As weak points we see the following ones:

(i) There is very little structure on the signals in terms of data structuring. Of course such a structure can be found by simply requiring that the signals come from some sort in an abstract data type, but that is certainly not a connection between process structure and data structure.

(ii) In cases where a signal is present before as well as after an action this model presents in an unavoidable way an atomic interruption of the signal during the execution of the action. This interruption is not always adequate in view of the intuitions behind the mechanism.

(iii) It is not clear that both the root signal insertion operation and the terminal signal insertion operator are needed. Having two primitive operators for the simple concept of putting a label on a node seems quite overdone at first sight. Nevertheless the presence of both operators gives rise to a flexible and efficient algebra. It turns out that in the presence of an empty step terminal signal insertion is not needed.

Besides signals, this paper introduces a conditional expression and relates it with the guarded command of [BBMV 89]. These constructs are unproblematic and could (should) have been introduced in the ACP setting much earlier indeed. In designing the syntax for the extra operators we will make use of algebraic abstract syntax. Thus as much as possible the syntax will be no more than an algebraic signature and diagrams will be added to illustrate these signatures.

2. ADDING SIGNALS TO BASIC PROCESS ALGEBRA.

2.1 BPA AND ACS

In order to make the paper somewhat self contained all axioms of process algebra from [BK 84] are repeated. The reader is supposed to have some familiarity with these axioms, however, as there is no explanation of the intuitions behind the original operators of process algebra. Let A be a finite set. The elements of A will be called atomic actions. Every atomic action is an element of P, the sort of processes. There are also two binary operators on P, viz. + (alternative composition) and · (sequential composition). The core system BPA (Basic Process Algebra) over this signature has the following axioms ($x,y,z \in P$).

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)\cdot z = x\cdot z + y\cdot z$	A4
$(x\cdot y)\cdot z = x\cdot(y\cdot z)$	A5

TABLE 1. BPA.

In the sequel a rather large number of new operators and axioms will be presented. The first step is to introduce the algebra ACS (algebra of composed signals). This involves the introduction of a sort AS of atomic signals which is in fact a parameter for the design of the algebras and which should be defined specifically for each application.

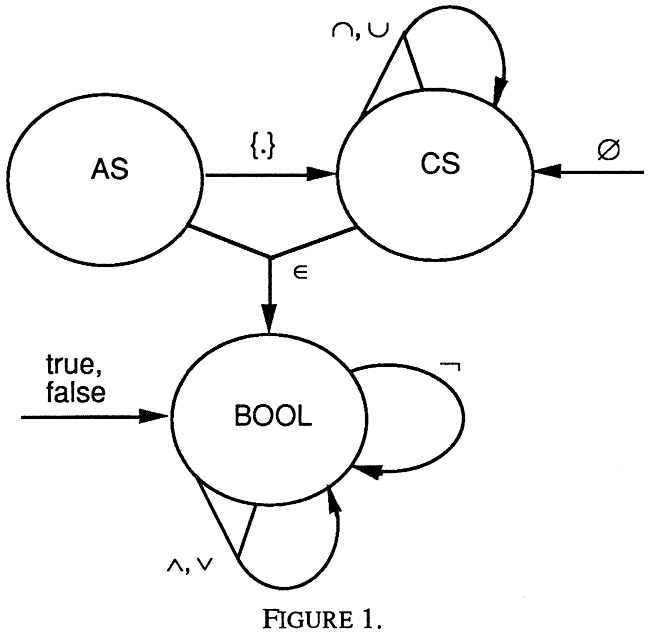

FIGURE 1.

The interpretation of AS will be a finite set and for each element of this finite set a constant is added in AS. Then there is sort CS of composed signals. CS is just the power set of AS and it is equipped with the usual operators on sets. The axioms of ACS are as follows ($a,b \in AS$, $u,v,w \in CS$):

$u \cup v = v \cup u$	CS1
$(u \cup v) \cup w = u \cup (v \cup w)$	CS2
$u \cup u = u$	CS3
$u \cup \varnothing = u$	CS4
$u \cap v = v \cap u$	CS5
$u \cap (v \cap w) = (u \cap v) \cap w$	CS6
$u \cap u = u$	CS7
$u \cap \varnothing = \varnothing$	CS8
$u \cup (v \cap w) = (u \cup v) \cap (u \cup w)$	CS9
$u \cap (v \cup w) = (u \cap v) \cup (u \cap w)$	CS10
$a \in \varnothing = \text{false}$	CS11
$a \in \{a\} = \text{true}$	CS12
$a \in \{b\} = \text{false} \quad \text{if } a \neq b$	CS13
$a \in u \cup w = (a \in u) \vee (a \in w)$	CS14
$a \in u \cap w = (a \in u) \wedge (a \in w)$	CS15

TABLE 2. ACS.

2.2 CONDITIONALS.

A useful feature in the coming examples is the addition of *conditionals* or *guards* to our language. It should be noted that this extension is independent of the presence of signals. We use letters ϕ,ψ to range over conditions COND. We first consider the ternary operator $.\triangleleft.\triangleright.: P \times COND \times P \to P$. The expression $x \triangleleft \phi \triangleright y$ should be read as **if** ϕ **then** x **else** y. We take this notation from [HHJ+ 87]. We think the notation with triangles is preferable over the one with key words, because the former notation does not have scoping problems and is more mathematical. We will also add this operator to our signal algebra, i.e. we will have a ternary operator $.\triangleleft.\triangleright.: CS \times COND \times CS$.

The advantage of adding the conditional expressions is easily understood in the use of parametrized process specifications. Let for instance, for $n \in \mathbb{N}$, $P(n)$ be the process $a^n \cdot b$. One obtains a uniform specification for the $P(n)$ as follows ($n \geq 0$):

$$P(n) = b \triangleleft n=0 \triangleright a \cdot P(pred(n))$$

(here pred is the predecessor function: $pred(0) = 0$ and $pred(n+1) = n$).

Besides this operator, we will use a variant of it that is a binary operator, the *guarded command*, that was introduced in [BBMV 89]. We have the notation $.:\to.: COND \times P \to P$. The expression $\phi :\to x$ is read as **if** ϕ **then** x. We have a similar operator on signals. There is the basic identity:

$$\phi :\to x = x \triangleleft \phi \triangleright \delta.$$

We see that the existence of this operator presupposes the presence of the δ constant. This is why the ternary operator is more basic and therefore preferable to the binary operator. Nevertheless, the binary operator will turn out to be very useful in examples.

In the presence of the empty process ε (see [VR 86] or [BW 90]) we can even use a *unary* operator $\{.\}$: $COND \to P$ (called *guard*, see [GP 90]) with the definition:

$$\{\phi\} = \varepsilon \triangleleft \phi \triangleright \delta.$$

Notice that this implies the following identity:

$$\phi :\to x = \{\phi\} \cdot x.$$

In [GP 90] a substantial amount of theory is developed for this notation for guards. In this paper, we will concentrate on the conditional and guarded command notation. As the guard operator presupposes the existence of the ε constant, it leads us out of concrete process algebra, and thus out of the scope of this paper.

Throughout this text we will discuss conditionals in two settings:
i. conditional expressions and guarded commands with conditions ranging over the set BOOL = {true, false}. Notice that in this case, conditionals may be removed from all closed expressions. It follows that no additional theory is required and that no consistency problem is raised whatsoever. The signature is shown in figure 2.

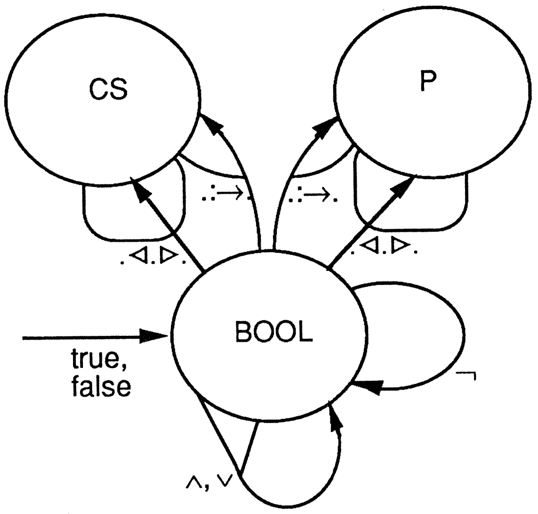

FIGURE 2.

ii. conditional expressions and guarded commands in the more general case that the condition is taken from the free Boolean algebra $\mathbb{B}_n$ with generators $\theta_1,...,\theta_n$. In this case we will usually add a distributive law for each new operator. Only in the case of the state operator some complications arise that can be solved in a natural way however. The signature is shown in figure 3.

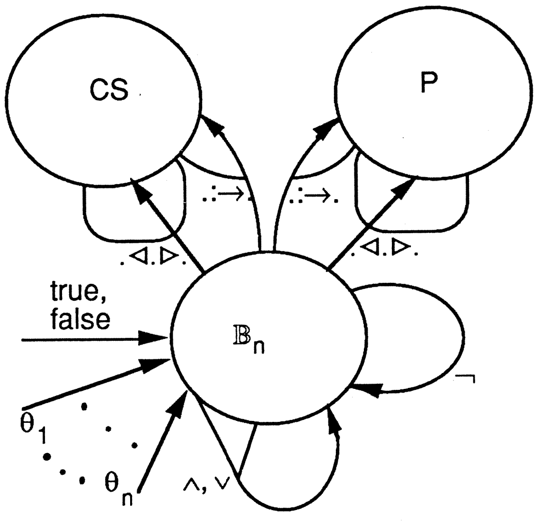

FIGURE 3.

We first consider case (i). The following axioms are obvious. $x,y \in P$, $u,v \in CS$.

$x \triangleleft true \triangleright y = x$	CO1
$x \triangleleft false \triangleright y = y$	CO2
$u \triangleleft true \triangleright v = u$	CO3
$u \triangleleft false \triangleright v = v$	CO4
$true :\rightarrow x = x$	GC1
$false :\rightarrow x = \delta$	GC2
$true :\rightarrow u = u$	GC3
$false :\rightarrow u = \varnothing$	GC4

TABLE 3. Conditionals and guarded command.

As we will see, the two operators are interdefinable. In the sequel, we will most often give the extra laws for the ternary operator, as this operator embodied a more classical construction, and has the advantages outlined above. Sometimes this is not possible, however. We will also use $:\rightarrow$ in case the law to be formulated would look incomprehensible for $.\triangleleft.\triangleright$. The laws for the other operator are usually derivable using the first four axioms of the following table. This table gives extra axioms needed in case (ii), when we have conditions ranging over an arbitrary finite Boolean algebra.

$\phi :\rightarrow x = x \triangleleft \phi \triangleright \delta$	CG1
$x \triangleleft \phi \triangleright y = \phi :\rightarrow x + (\neg\phi) :\rightarrow y$	CG2
$\phi :\rightarrow u = u \triangleleft \phi \triangleright \varnothing$	CG3
$u \triangleleft \phi \triangleright v = \phi :\rightarrow u \cup (\neg\phi) :\rightarrow v$	CG4
$\phi :\rightarrow \varnothing = \varnothing$	GC5
$\phi :\rightarrow (u \cup v) = (\phi :\rightarrow u) \cup (\phi :\rightarrow v)$	GC6
$(\phi \vee \psi) :\rightarrow u = (\phi :\rightarrow u) \cup (\psi :\rightarrow u)$	GC7
$\phi :\rightarrow (\psi :\rightarrow u) = (\phi \wedge \psi) :\rightarrow u$	GC8
$\phi :\rightarrow \delta = \delta$	GC9
$\phi :\rightarrow (x + y) = (\phi :\rightarrow x) + (\phi :\rightarrow y)$	GC10
$(\phi \vee \psi) :\rightarrow x = (\phi :\rightarrow x) + (\psi :\rightarrow x)$	GC11
$\phi :\rightarrow (\psi :\rightarrow x) = (\phi \wedge \psi) :\rightarrow x$	GC12
$(x \cdot z) \triangleleft \phi \triangleright (y \cdot z) = (x \triangleleft \phi \triangleright y) \cdot z$	CO5

TABLE 4. Conditionals over a Boolean algebra.

As an example, we show two calculations:

i. $x \triangleleft (\phi \vee \psi) \triangleright y = (\phi \vee \psi) :\rightarrow x + \neg(\phi \vee \psi) :\rightarrow v = (\phi \vee (\neg\phi \wedge \psi)) :\rightarrow x + (\neg\phi \wedge \neg\psi) :\rightarrow y =$
$= (\phi :\rightarrow x) + (\neg\phi :\rightarrow (\psi :\rightarrow x)) + (\neg\phi :\rightarrow (\neg\psi :\rightarrow y)) = (\phi :\rightarrow x) + (\neg\phi :\rightarrow (x \triangleleft \psi \triangleright y)) = x \triangleleft \phi \triangleright (x \triangleleft \psi \triangleright y).$

ii. $x \triangleleft \neg\phi \triangleright y = \neg\phi :\rightarrow x + \neg(\neg\phi) :\rightarrow y = \phi :\rightarrow y + (\neg\phi) :\rightarrow x = y \triangleleft \phi \triangleright x.$

LEMMA. Consider the algebra CS with operators $\cap, \cup, \{.\}, \in$ and $:\rightarrow$ over the Boolean algebra $\mathbb{B}_n$ with generators $\theta_1, \ldots, \theta_n$. Let u be a closed term over this algebra. Then u can be written in the form

$$u = \bigcup_{i=1}^{2^n} (\phi_i :\rightarrow u_i),$$

where for $1 \leq i \leq 2^n$, $u_i \in$ CS and the ϕ_i range over all 'complete' conjunctions of literals. A *literal* is a term of the form θ_i or $\neg\theta_i$.

THEOREM. Consider the algebra BPA with operators $+,\cdot$ and $:\rightarrow$ over the Boolean algebra $\mathbb{B}_n$ with generators $\theta_1,\ldots,\theta_n$. Let t be a closed term over this algebra. Then t can be written in the form

$$t = \sum_{i \in I} (\phi_i :\rightarrow a_i) \cdot t_i + \sum_{i \in J} (\phi_i :\rightarrow b_i),$$

where $I,J \subseteq \{1,\ldots,2^n\}$, $a_i, b_i \in A$, t_i again closed terms and the ϕ_i are as above. Here, we use the convention that a sum over an empty set is considered equal to δ.

A consequence of this theorem is that one may view BPA with guarded commands as a variation of BPA with atomic actions $\phi_i :\rightarrow a_i$.

2.3 ROOT AND TERMINAL SIGNAL INSERTION OPERATORS.

The next operators to be introduced are the signal insertion operators. [.,.] is the root signal insertion operator and $\langle .,. \rangle$ is the terminal signal insertion operator. The intuition behind these operators is that both assign labels (signals) to the states of processes. Root signal insertion places a signal at the root node of a process. Terminal signal insertion places one and the same signal at each terminal node of a process. If one is interested solely in processes that show signals exclusively in nonterminal states one may as well forget about the terminal signal insertion operator. Leaving out all axioms involving terminal insertion from the coming sections one will obtain an appropriate description of root signal insertion.

Whereas in process algebra one usually confines oneself to labeling the transitions and perhaps to some labeling of the nodes that is directly related to the mechanism of transition labeling, here it is intended to have labelings of states of processes with the same status as the labelings of the state transitions by means of atomic actions. With some effort it turns out that an algebraic specification of the resulting notion of processes can be given that indeed constitutes a conservative enrichment of ACP (at least regarding identities between finite closed process expressions). The following 10 equations are added to BPA thus obtaining BPAS (BPA with signals). Remarks on models for BPAS are given in section 7.

$[u, x] \cdot y = [u, x \cdot y]$	RS1
$[u, x] + y = [u, x + y]$	RS2
$[u, [v, x]] = [u \cup v, x]$	RS3
$[\varnothing, x] = x$	RS4

TABLE 5. Root signal insertion.

The first axiom expresses the fact that the root of a sequential product is the root of its first component. Axiom RS2 can be given in a more symmetric form as follows:

$$[u, x] + [v, y] = [u \cup v, x + y].$$

This equation depends on the fact that the roots of two processes in an alternative composition are identified. Therefore signals must be combined. The third axiom expresses the fact that there is no sequential order in the presentation of signals. Of course one might imagine that a sequential ordering on signals is introduced, but we think that the introduction of such a sequential ordering is far from obvious (it also leads to problems concerning the associativity of the parallel composition operator). The combination of the signals is taking 'both' of them whereas $x + y$ has to choose between x and y. The equations below regard terminal signal insertion.

$\langle x \cdot y, u\rangle = x \cdot \langle y, u\rangle$	TS1
$\langle x + y, u\rangle = \langle x, u\rangle + \langle y, u\rangle$	TS2
$\langle\langle x, u\rangle, v\rangle = \langle x, u \cup v\rangle$	TS3
$\langle x, \varnothing\rangle = x$	TS4
$\langle x, u\rangle \cdot y = x \cdot [u, y]$	TRS1
$\langle [u, x], v\rangle = [u, \langle x, v\rangle]$	TRS2

TABLE 6. Remaining axioms of BPAS.

The deadlock constant can be added without problems to the above axioms.

$x + \delta = x$	A6
$\delta \cdot x = \delta$	A7
$\langle \delta, u\rangle = \delta$	TS6

TABLE 7. Signals and deadlock.

An interesting identity that follows with the introduction of δ is the following:

$$[u, x] = [u, \delta] + x.$$

This equation is indeed very useful for writing efficient process specifications mainly because it allows to a large extent to work with process algebra expressions that are not cluttered with signal insertions.

If we add conditionals over an arbitrary finite Boolean algebra, we need the following extra axioms.

$[u, x] \lhd \phi \rhd [v, y] = [u \lhd \phi \rhd v, x \lhd \phi \rhd y]$
$\phi :\to \langle x, u\rangle = \langle \phi :\to x, u\rangle$

TABLE 8. Signal insertion and guarded command.

We formulate the second law of table 8 in terms of the binary operator, since the obvious law for the ternary operator, viz. $\langle x, u\rangle \lhd \phi \rhd \langle y, v\rangle = \langle x \lhd \phi \rhd y, u \lhd \phi \rhd v\rangle$, is *not* valid, since the validity of ϕ may change during the execution of the process. A variant for the ternary operator that does work, is the following:

$$\langle x, u\rangle \lhd \phi \rhd \langle y, v\rangle = \langle x \lhd \phi \rhd \delta, u\rangle + \langle \delta \lhd \phi \rhd y, v\rangle$$

As an example, we give a calculation:

$[u,a] \lhd \phi \rhd \langle b, v\rangle = [u,a] \lhd \phi \rhd [\varnothing, \langle b, v\rangle] = [u \lhd \phi \rhd \varnothing, a \lhd \phi \rhd \langle b, v\rangle] = [\phi :\to u, (\phi :\to a) + (\neg\phi :\to \langle b, v\rangle)] =$
$= [\phi :\to u, (\phi :\to a) + \langle \neg\phi :\to b, v\rangle)]$.

2.4 SIGNAL FILTERING AND GLOBAL SIGNAL INSERTION.

The signal filtering operation $\cap$ allows to reduce the number of visible signals by filtering them all with a fixed signal. The advantage of the use of signal filtering is to obtain a much simpler signal structure, which can then be specified quite precisely. So if it is hard to find a correct expression for X it may be workable to determine $u \cap X$ for some appropriate signal u.

$u \cap a = a$	F1
$u \cap (x + y) = (u \cap x) + (u \cap y)$	F2
$u \cap (x \cdot y) = (u \cap x) \cdot (u \cap y)$	F3
$u \cap [v, x] = [u \cap v, u \cap x]$	F4
$u \cap \langle x, v\rangle = \langle u \cap x, u \cap v\rangle$	F5

TABLE 9. Signal filtering.

Similarly, it is useful to extend $\cup$ over arbitrary processes. The appropriate name for this operation is global signal insertion.

$u \cup a = [u, \langle a, u\rangle]$	GSI1
$u \cup (x + y) = (u \cup x) + (u \cup y)$	GSI2
$u \cup (x \cdot y) = (u \cup x) \cdot (u \cup y)$	GSI3
$u \cup [v, x] = [u \cup v, u \cup x]$	GSI4
$u \cup \langle x, v\rangle = \langle u \cup x, u \cup v\rangle$	GSI5

TABLE 10. Global signal insertion.

The extra axioms needed for conditionals are equally straightforward.

$u \cap (x \lhd \phi \rhd y) = (u \cap x) \lhd \phi \rhd (u \cap y)$
$u \cup (x \lhd \phi \rhd y) = (u \cup x) \lhd \phi \rhd (u \cup y)$

TABLE 11. Signal filtering, insertion and conditionals.

We give an overview of the signature elements introduced in sections 2.3 and 2.4 in figure 4.

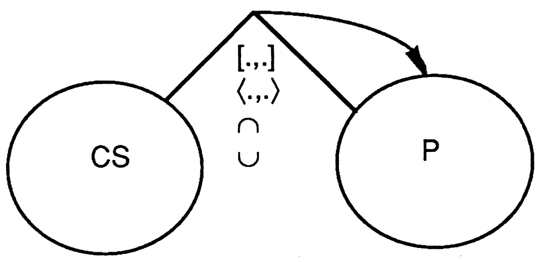

FIGURE 4.

An important application of signals is in the specification of reactive systems. The virtue of the signal mechanism is that it allows to express an asymmetry between action and reaction in the case of reactive systems. The action is an atomic act performed on the initiative of the external environment of a reactive system S. The effect of this action is to change the state of the system and to leave it in a stable resulting state from which it may show one or more signals. The environment can decide itself when to observe these signals, if at all. In the next section we will present a series of examples of process definitions involving signals. Most of these systems may be viewed as (parts of) reactive systems.

Processes with signals constitute a novel concept, at least as seen from the point of view of process algebra. Of course there is nothing new in the intuitions, the novelty lies entirely in the aspect that these matters are wrapped in an algebraic framework. Moreover it turns out that processes with signals allow a natural version of bisimulation semantics. We will exploit the possibilities that are opened by the introduction of signals in processes within the context of process algebra. Processes with signals constitute first and for all a meaningful semantic category. Only secondary is the algebraic treatment of these processes in a specific setting of operators. Nevertheless we hope to have found an algebraic form for the subject that is sufficiently flexible and powerful to be both of technical and conceptual relevance.

2.5 ROOT SIGNAL OPERATOR.

It is useful to extend the system with operators S and P. The operator S determines the root signal of a process. If $S(x) = \varnothing$ we say that x has a *trivial root signal*, otherwise x has a non-trivial root signal. Processes that were studied until now in the context of process algebra always have a trivial root signal. The operator P removes the root signal from its argument, thus obtaining a process with a trivial root signal.

$S(a) = \varnothing$	S1
$S(x + y) = S(x) \cup S(y)$	S2
$S(x \cdot y) = S(x)$	S3
$S([u, x]) = u \cup S(x)$	S4
$P(a) = a$	P1
$P(x + y) = P(x) + P(y)$	P2
$P(x \cdot y) = P(x) \cdot y$	P3
$P([u, x]) = P(x)$	P4
$P(\langle x, u\rangle) = \langle P(x), u\rangle$	P5

TABLE 12. Root signal operator, root signal deletion operator.

Notice that the equation $S(\langle x, u\rangle) = S(x)$ is derivable from the axioms in table 12:

$$S(\langle x, u\rangle) = S(\langle x, u\rangle \cdot y) = S(x \cdot [u, y]) = S(x).$$

Also $x = [S(x), P(x)]$ will now be derivable for finite closed process expressions. As a rewrite rule it is useless, however, because it will immediately introduce an infinite loop.

The extra axioms needed for conditionals are straightforward.

$S(x \triangleleft \phi \triangleright y) = S(x) \triangleleft \phi \triangleright S(y)$
$P(x \triangleleft \phi \triangleright y) = P(x) \triangleleft \phi \triangleright P(y)$

TABLE 13. Root signals and conditionals.

We can extend the normal form theorem of section 2.2 to BPAS as follows: if t is a term over BPAS, write t in the form $[S(t), P(t)]$. Then $S(t)$ can be written as shown in the lemma in section 2.2, and we can extend the theorem to cover also processes with signals, provided no signals occur in the root, and we also allow terms of the form $\sum_{i \in K} (\phi_i :\to \langle b_i, u_i\rangle)$.

2.6 EXAMPLES.

(i) A traffic light that changes color from green via yellow to red and back to green. As names for the traffic lights one may simply use the natural numbers. For the traffic light with number n, the signals are then green(n) , yellow(n) and red(n), the only action is change(n).

$$TL(n) = \quad [\{green(n)\}, change(n)] \cdot [\{yellow(n)\}, change(n)] \cdot [\{red(n)\}, change(n)] \cdot TL(n).$$

There are various alternatives to this definition which are interesting to display explicitly. For instance one may introduce a data type COLORS = {green, yellow, red} with a function next that permutes the colors as follows: green $\to$ yellow $\to$ red $\to$ green. Then the process TL can be parametrised by a color and one obtains one of the following four recursion equations. In this case we remove the numbers of the traffic lights for brevity.

$$TL(x) = [\{x\}, change] \cdot TL(next(x))$$
$$TL(x) = [\{x\}, change \cdot TL(next(x))]$$

TL(x) = [{x}, δ] + change·TL(next(x))
TL(x) = ([{x}, δ] + change)·TL(next(x))

(ii) An electric lamp with power switch. In the equations the names for signals and actions speak for themselves. We will assume that the lamp is indexed by a name n from

NAMES = {hall1, hall2, hall3, kitchen1, kitchen2, living1, living2, living3, living4, staircase, garage, cellar}.

The lamp is parametrised by two additional parameters: L(n, x, y), where

$x \in$ SWITCH = {on, off}, $y \in$ STATUS = {defect, functioning}.

The function sw permutes the elements of SWITCH and the actions switch(n) correspond to switching on or off lamp n. For each $n \in$ NAMES there are two signals: light(n) and dark(n). Together these signals constitute the atomic signals. It must be noticed that the combined signal {light(n), dark(n)} will not occur in any process describing a physically meaningful setting.

A recursion equation for the behavior of the lamp with name n is then as follows. Notice that we use a conditional statement on signals, and a guarded command on processes, as discussed in 2.2.

L(n, x, y) = [{light(n)◁(x=on & y=functioning)▷dark(n)}, δ] +
switch(n)·L(n,sw(x), y) +
(y=functioning) :→ defect(n)·L(n, x, defect) +
get_new_lamp(n)·L(n, x, functioning)

We return to this example in sections 3.2 and 3.5.

(iii) A further example is a single lamp with double switch, and the status of the lamp can be controlled from both switches. In these equations the intended meaning of the actions is as follows:

ui =	put switch i in upward position (i=1,2)
di =	put switch i in downward position (i=1,2)

$L_{0,0}$ = [{dark}, u1]·$L_{1,0}$ + [{dark}, u2]·$L_{0,1}$
$L_{1,0}$ = [{light}, d1]·$L_{0,0}$ + [{light}, u2]·$L_{1,1}$
$L_{0,1}$ = [{light}, u1]·$L_{1,1}$ + [{light}, d2]·$L_{0,0}$
$L_{1,1}$ = [{dark}, d1]·$L_{0,1}$ + [{dark}, d2]·$L_{1,0}$.

(iv) A system of two lifts seen from an intermediate floor, say floor 2, has two signals: the signal sur (signal upwards request) indicates that somebody wants (or wanted) to travel in the upward direction, the signal sdr (signal downwards request) indicates that a downward travel has been requested. The signals disappear as soon as a lift has departed in the requested direction. Both lifts move all the way from bottom floor to top floor and back all the time, be it that a request is needed to trigger their action. The actions that can be performed or observed by users at floor 2 are as follows:

up_request (asking for an upward travel),
down_request (asking for a downward travel),
arrive_j_b, (lift j arrives from below),
arrive_j_a (lift j arrives from above),

leave_j_up (lift j leaves in upward direction),

leave_j_down (lift j leaves in downward direction).

Only the leave actions will influence the signals. In the following specification, the movement of the lifts is taken into account, be it that the internal logic of the lifts is not modeled. For instance, it can happen that lift 1 leaves in upwards direction twice without having left for the opposite direction in between, whereas all the time the signal sdr was present and lift 2 has not departed in downward direction either. This course of events is not allowed by most lift systems. Let $u \subseteq \{sdr, sur\}$.

$LL_u = [u, \delta] +$
$\quad \langle up\_request, u \cup \{sur\} \rangle \cdot LL_{u \cup sur\}} +$
$\quad \langle down\_request, u \cup \{sdr\} \rangle \cdot LL_{u \cup \{sdr\}} +$
$\quad \langle (leave\_1\_up + leave\_2\_up), u - \{sur\} \rangle \cdot LL_{u - \{sur\}} +$
$\quad \langle (leave\_1\_down + leave\_2\_down), u - \{sdr\} \rangle \cdot LL_{u - \{sdr\}} +$
$\quad \langle (arrive\_1\_b + arrive\_1\_a + arrive\_2\_b + arrive\_2\_a), u \rangle \cdot L_u.$

(v) This example again discusses a lift connecting three floors but now as seen from the inside. The number of lifts involved in the system is of no importance for this specification. In this case the signals are the various sets of requests for travel to floors 0, 1 and 2. These signals for requests are denoted with sr(0), sr(1) and sr(2). Notice that the requests come either from inside the lift or from outside, but in both cases all requests are signalled inside the lift, e.g. by lighted buttons. We see there are 8 possible signals, as there are 8 subsets of the collection of atomic signals.

The actions are:

- req(j) to request transport to floor j for j = 0,1,2;
- move(0,1), move(1,0), move(1,2), move(2,1), move_past(1) denoting the movements of the lift;
- halt(j) denotes the halting of the lift at floor j;
- i_open(j), i_close(j) are the actions of opening and closing the (inner) lift door
  (these actions are controlled by the lift system);
- o_open(j) and o_close(j) are the actions of opening and closing the outer lift door
  (presumably controlled by the users).

There is no time-out mechanism that decides whether or not the outer lift door is going to be opened at all after a halt has been made. The lift has 32 states while moving and 120 states while resting at a floor. The 32 states between floors are organised as follows:

LM(0/1,up, V): the lift is moving between 0 and 1 traveling in an upward direction at the point where it will either continue its travel further without halting at 1 or decide to halt at 1 (be it because of an internal or an external request), moreover V is the signal (i.e. the set of atomic signals) that is visible.

Similarly there are states LM(0/1, down, V), LM(1/2, up, V) and LM(1/2, down,V). Starting from a halted lift, not all moving states can be reached, as the following specification is such, that the lift will only move in a certain direction if there is a request to go in that direction. Thus, only 20 travelling states will actually occur.

The 120 resting states are organised as follows:

for each $j \in \{0, 1, 2\}$,

$d \in$ {closed1, inner_door_open1, outer_door_open, inner_door_open2, closed2},

$V \subseteq \{sr(0), sr(1), sr(2)\}$,

the object (data structure) LR(j, d, V) is a state. Again, not all 120 states will actually occur, as the signal for a given floor is not present when the lift is on that floor and the door is open. Thus, we see that only 84 states can actually occur.
The meaning of these states is as follows: for instance L(2, inner_door_open2, {sr(0), sr(2)}) indicates that the lift rests at floor 2, with its inner door open but its outer again closed and the request signals for floors 0 and 2 on.

Then one needs an equation for each of the 104 (or 152) states of the lift. We will present the whole specification by means of two equations, making heavy use of the guarded command from section 2.2. This model is much simpler than the real situation in most cases. For instance, after halting, the lift does not know anymore in which direction it was travelling.
Let $i \in \{1, 2\}$, $j \in \{0, 1, 2\}$, $b \in$ {up, down}, $V \subseteq$ {sr(0), sr(1), sr(2)}, $d \in$ {closed1, inner_door_open1, outer_door_open, inner_door_open2, closed2}. Then

```
L(i/i+1, b, V) = [V, δ] +
        req(0)·L(i/i+1, b, V∪{sr(0)}) +
        req(1)·L(i/i+1, b, V∪{sr(1)}) +
        req(2)·L(i/i+1, b, V∪{sr(2)}) +
        (sr(i)∈V & b=down) :→ halt(i)·LR(i, closed1, V) +
        (sr(i+1)∈V & b=up) :→ halt(i+1)·LR(i+1, closed1, V) +
        (i=0 & b=up & sr(1)∉V) :→ move_past(1)·L(1/2, up, V) +
        (i=1 & b=down & sr(1)∉V) :→ move_past(1)·L(0/1, down, V).

LR(j, d, V) = [V, δ] +
        req(0)·(LR(j, d, V∪{sr(0)}) ◁ j≠0 ▷
                (d=closed1 :→ LR(j, d, V∪{sr(0}) +
                d∈{inner_door_open1, outer_door_open}) :→ LR(j, d, V) +
                d=inner_door_open2 :→ LR(j, inner_door_open1, V) +
                d=closed2 :→ (LR(0, closed1, {sr(0}) ◁ V=∅ ▷ LR(j, d, V∪{sr(0})))) +
        req(1)·(LR(j, d, V∪{sr(1)}) ◁ j≠1 ▷
                (d=closed1 :→ LR(j, d, V∪{sr(1}) +
                d∈{inner_door_open1, outer_door_open}) :→ LR(j, d, V) +
                d=inner_door_open2 :→ LR(j, inner_door_open1, V) +
                d=closed2 :→ (LR(1, closed1, {sr(1}) ◁ V=∅ ▷ LR(j, d, V∪{sr(1})))) +
        req(2)·(LR(j, d, V∪{sr(2)}) ◁ j≠2 ▷
                (d=closed1 :→ LR(j, d, V∪{sr(2}) +
                d∈{inner_door_open1, outer_door_open}) :→ LR(j, d, V) +
                d=inner_door_open2 :→ LR(j, inner_door_open1, V) +
                d=closed2 :→ (LR(2, closed1, {sr(2}) ◁ V=∅ ▷ LR(j, d, V∪{sr(2})))) +
        (d=closed1) :→ i_open(j)·LR(j, inner_door_open1, V−{sr(j)}) +
        (d=inner_door_open1) :→ (o_open(j)·LR(j, outer_door_open, V) +
                                 + i_close(j)·LR(j, closed2, V)) +
        (d=outer_door_open) :→ o_close(j)·LR(j, inner_door_open2, V) +
        (d=inner_door_open2) :→ i_close(j)·LR(j, closed2, V) +
```

(d=closed2 & j=0 & (sr(1)$\in$V or sr(2)$\in$V)) :$\rightarrow$ move(0,1)·L(0/1, up, V) +
(d=closed2 & j=1 & sr(0)$\in$V) :$\rightarrow$ move(1,0)·L(0/1, down, V) +
(d=closed2 & j=1 & sr(2)$\in$V) :$\rightarrow$ move(1,2)·L(1/2, up, V) +
(d=closed2 & j=2 & (sr(0)$\in$V or sr(1)$\in$V)) :$\rightarrow$ move(2,1)·L(1/2, down, V).

(vi) An alarm clock. In this example one needs a rather more substantial underlying data type than in the previous examples. We will describe the data type in an informal way but given the data type the description of the clock will be quite precise. Of course there are many formalisms that allow to present a precise description of the data type as well. The atomic signals in this example are:

- The digital time indications measuring time in seconds, collected in the set TIMES. There is a successor function next on TIMES which increases time with one second, counting modulo 24 hours.
- The alarm signal: alarm.

The actions are the following:

switch_off_alarm	(takes no time),
tick	(a tick of the clock occurs every second),
set_alarm(t)	(this action takes 2 seconds).

The process has a state vector consisting of three attributes:
(a) an element time of TIME, the current time,
(b) a boolean alarm_set that indicates whether or not the alarm has been set
(c) an element a_time of TIME that indicates when (if at all) the alarm must start. The alarm will then be on for two consecutive minutes unless it is switched off before.

Now the state of the clock is given by a record (frame, tuple):

CLOCK(time, alarm_set, a_time).

All actions result in a modification of the (values of the attributes) of the record, and the signals depend directly on the record. In particular, the time signal is just the current value of the first attribute of CLOCK and the signal alarm is on exactly if alarm_set = true and time $\in$ [a_time, a_time + 00.02.00 (modulo 24.00.00)].

Recursion equations for CLOCK are then as follows:
CLOCK = CLOCK(00.00.00, false, 00.00.00)
CLOCK(t, b, t') = [t, δ] +
tick·CLOCK(next(t), b, t') +
$\sum_{r\in TIME}$ set_alarm(r)·CLOCK(next(next(t)), true, r) +
switch_off_alarm·CLOCK(t, false, t') +
(b=true & t' $\leq$ t $\leq$ t' + 00.02.00 (modulo 24.00.00)) :$\rightarrow$ [{alarm}, δ].

In the last line, we used the guarded command of 2.2.

3. SIGNALS AND PARALLEL COMPOSITION.

3.1 FREE MERGE.

We can extend the system BPAS of section 2 to PAS$^-$, including the free merge (parallel composition without communication) and the left merge with the usual axioms and adding axioms that handle the interaction between the signal insertion operators and the left merge. The superscript $^-$ indicates that there is no feature present that allows the observation of signals, addition of that feature is the subject of section 4. (Of course the extension to ACP will require a modification of the merge expansion axiom by adding an additional term for the communication merge, this will be described in section 5.)

$x \parallel y = x \mathbin{\|\!\!\lfloor} y + y \mathbin{\|\!\!\lfloor} x$	M1
$a \mathbin{\|\!\!\lfloor} x = a{\cdot}x$	M2
$a{\cdot}x \mathbin{\|\!\!\lfloor} y = a{\cdot}(x \parallel y)$	M3
$(x + y) \mathbin{\|\!\!\lfloor} z = x \mathbin{\|\!\!\lfloor} z + y \mathbin{\|\!\!\lfloor} z$	M4
$[u, x] \mathbin{\|\!\!\lfloor} y = [u, x \mathbin{\|\!\!\lfloor} y]$	MSI1
$\langle a\ , u\rangle \mathbin{\|\!\!\lfloor} x = a{\cdot}(u \cup x)$	MSI2

TABLE 14. PAS$^-$.

3.2 EXAMPLES.

(i) Simple examples for the application of the free merge can be produced on the basis of the previous series of examples. The simultaneous behavior of traffic lights (see example 2.6.i) with names in {1,...,4} is found by simply merging their process descriptions:

TL(1-4) = TL(1) $\parallel$ [{red(2)}, change(2)]·TL(2) $\parallel$ TL(3) $\parallel$ [{red(4)}, change(4)]·TL(4).

In an intersection where lights 1 and 3 are in the east-west direction, and lights 2 and 4 are in the north-south direction, this system starts out correctly, but as there is no communication between the different traffic lights, the coordination will soon disappear. Some communication mechanism is needed to describe this system correctly. We will return to this example in section 3.5.

(ii) This example refers to the previous example 2.6.ii that introduced a collection of lamps in a private house. The simultaneous behavior of the collection of lamps in the living room is appropriately described by

L(living) = L(living1) $\parallel$ L(living2) $\parallel$ L(living(3) $\parallel$ L(living4).

For the kitchen one obtains: L(kitchen) = L(kitchen1) $\parallel$ L(kitchen2).

For the hall we have L(hall) = L(hall1) $\parallel$ L(hall2). Composing these one obtains:

L(living/kitchen/hall) = L(kitchen) $\parallel$ L(living) $\parallel$ L(hall).

(iii) Examples of derived identities in PAS$^-$.

$[u, \delta] \parallel [v, \delta] = [u \cup v, \delta]$,

$[u, a] \parallel [v, b] = [u \cup v, a]{\cdot}[v, b] + [u \cup v, b]{\cdot}[u, a]$,

$[u, \langle a, u'\rangle] \parallel [v, \langle b, v'\rangle] = [u \cup v, \delta] + a{\cdot}[v \mathbin{\&} u', \langle b, u' \mathbin{\&} v'\rangle] + b{\cdot}[u \cup v', \langle a, u' \mathbin{\&} v'\rangle]$.

3.3 SIGNALS AND SYNCHRONOUS COMMUNICATION.

The axiom system ACPS$^-$ describes the addition of signals to processes with synchronous communication as modeled by ACP. The superscript $^-$ indicates that there is no communication by means of observation of signals. Addition of that feature will involve the addition of two more summands to the merge expansion axiom. The remaining axioms of ACPS$^-$ can now be added without leading to any inconsistency, but it is necessary to add some equations that describe the interaction of left merge and communication merge with the node labels. Notice that the axiom CM1 replaces the axiom M1, apart from this the family of axiom systems increases in a monotonic way.

$a \mid b = b \mid a$	C1
$(a \mid b) \mid c = a \mid (b \mid c)$	C2
$a \mid \delta = \delta$	C3
$x \parallel y = x \mathbin{⌊\!⌊} y + y \mathbin{⌊\!⌊} x + x \mid y$	CM1
$a \mathbin{⌊\!⌊} x = a \cdot x$	CM2
$(a \cdot x) \mathbin{⌊\!⌊} y = a \cdot (x \parallel y)$	CM3
$(x + y) \mathbin{⌊\!⌊} z = (x \mathbin{⌊\!⌊} z) + (y \mathbin{⌊\!⌊} z)$	CM4
$[u,x] \mathbin{⌊\!⌊} y = [u, x \mathbin{⌊\!⌊} y]$	MSI1
$\langle a, u\rangle \mathbin{⌊\!⌊} y = a \cdot (u \cup x)$	MSI2
$a \mid (b \cdot x) = (a \mid b) \cdot x$	CM5
$(a \cdot x) \mid b = (a \mid b) \cdot x$	CM6
$(a \cdot x) \mid (b \cdot y) = (a \mid b) \cdot (x \parallel y)$	CM7
$(x + y) \mid z = (x \mid z) + (y \mid z)$	CM8
$x \mid (y + z) = (x \mid y) + (x \mid z)$	CM9
$[u, x] \mid y = [u, x \mid y]$	MSI3
$x \mid [u, y] = [u, x \mid y]$	MSI4
$\langle a, u\rangle \mid b = \langle a \mid b, u\rangle$	MSI5
$a \mid \langle b, u\rangle = \langle a \mid b, u\rangle$	MSI6
$\langle a, u\rangle \mid \langle b, v\rangle = \langle a \mid b, u \cup v\rangle$	MSI7
$\langle a, u\rangle \mid (b \cdot x) = (a \mid b) \cdot (u \cup x)$	MSI8
$(a \cdot x) \mid \langle b, u\rangle = (a \mid b) \cdot (u \cup x)$	MSI9
$\partial_H(a) = a \quad \text{if } a \notin H$	D1
$\partial_H(a) = \delta \quad \text{if } a \in H$	D2
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D3
$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$	D4
$\partial_H([u,x]) = [u, \partial_H(x)]$	DSI1
$\partial_H(\langle x, u\rangle) = \langle \partial_H(x), u\rangle$	DSI2

TABLE 15. ACPS$^-$.

3.4 CONDITIONALS.

We can extend the results in chapter 2 about conditionals to a setting with parallel composition and communication if we add the following axioms.

$(x \lhd \phi \rhd y) \mathbin{⌊\!⌊} z = (x \mathbin{⌊\!⌊} z) \lhd \phi \rhd (y \mathbin{⌊\!⌊} z)$
$(x \lhd \phi \rhd y) \mid z = (x \mid z) \lhd \phi \rhd (y \mid \mathbin{⌊\!⌊} z)$
$x \mid (y \lhd \phi \rhd z) = (x \mid y) \lhd \phi \rhd (x \mid z)$
$\partial_H(x \lhd \phi \rhd y) = \partial_H(x) \lhd \phi \rhd \partial_H(y)$

TABLE 16. Conditionals and parallel composition.

3.5 EXAMPLES OF SYSTEM SPECIFICATIONS IN ACPS$^-$.

(i) Consider the traffic light system of example 3.2.i. Within ACPS$^-$ it is possible to introduce a control unit that performs the switching of the lights. New actions are needed to deal with switching. In particular we will need names for the switch actions as performed by the control unit. The actions set_to(color, number) have an evident meaning. Because the traffic light knows to which color it is set the actions change(n) are still useful. Having the color as an additional parameter is not strictly needed for programming the control unit either but it serves to get a more readable specification. As new communications we will introduce:

change(n) | set_to(c,n) = c(n).

All other communications are δ. The encapsulation set H simply contains all actions that can engage in a nontrivial communication (the change and the set_to actions). Here it is understood that the traffic lights are numbered clockwise. The crossing that will be considered in more detail in this discussion is the one in the direction 2 to 4. We will assume that this crossing is for pedestrians. A picture of the situation is as follows:

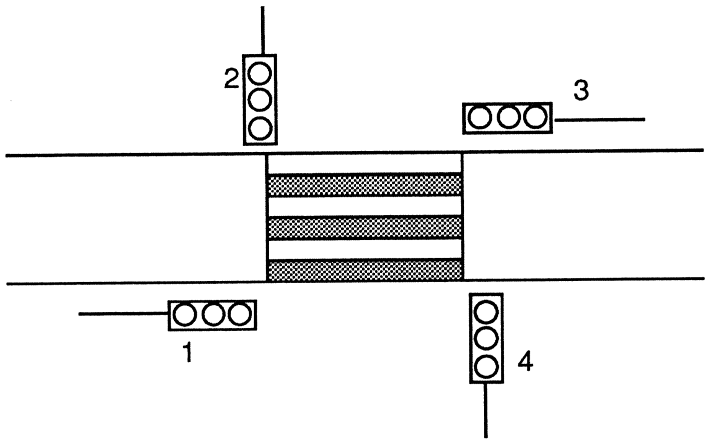

FIGURE 5.

Pedestrians who want to cross from north to south (direction 2 to 4) will wait until the light at 2 is green. A specification for the control unit might be as follows.

```
control =
    (set_to(yellow, 1) || set_to(yellow, 3))·waityellow(1, 3)·
    (set_to(red, 1) || set_to(red, 3))·waitred(1, 3)·
    (set_to(green, 2) || set_to(green, 4))·waitgreen(2, 4)·
    (set_to(yellow, 2) || set_to(yellow, 4))·waityellow(2, 4)·
    (set_to(red, 2) || set_to(red, 4))·waitred(2, 4)·
    (set_to(green, 1) || set_to(green, 3))·waitgreen(1, 3)

loop = control·loop
```

The entire system is obtained by means of an encapsulated merge of the traffic lights and the control unit:

SYSTEM = ∂_H(loop || TL(1-4)).

An interesting complication arises if one allows the control unit to receive requests for one of the directions. For instance one may assume that an action read(request(2→4)) is needed to trigger the transition to a state in which the 2 and 4 are green. The program may then look as follows:

$\text{control2} = (\text{read}(\text{request}(2{\to}4)) + \text{read}(\text{request}(4{\to}2))\cdot\text{control}$
$\text{loop2} = \text{control2}\cdot\text{loop2}$

Of course this leads to
$\text{SYSTEM2} = \partial_H(\text{loop2} \parallel \text{TL}(1\text{-}4))$.

The third step is to add agents on the scene that want to use the traffic lights. For instance one may consider a pedestrian who wants to use the crossing in the north-south direction. If the pedestrian wants to cross from 2 to 4 (s)he will have to inspect the light at 2, and wait until it is green.
Here is a process describing this person:

$\text{ped}(i) = \text{arrive\_at\_2}(i)\cdot\text{wait}(i)\cdot\text{cross}(2{\to}4,\ i)$
$\text{wait}(i) = \text{obs}(\{\text{green}(2)\}) + \text{put}(\text{request}(2{\to}4))\cdot\text{wait}(i)$

The action obs({green(2)}) has no special meaning in this specification, be it that in an implementation one would like this action to be implemented in correspondence with its intuitive meaning, namely the observation of the green signal at light 2. In section 5 an observation mechanism will be introduced. Here we assume that put(request(2→4)) and read(request (2→4)) communicate, resulting in an action request(2→4).

H is then extended by the actions read(request(2→4)), yielding H*. The partial order ≤ says that request(2→4) has higher priority than the action put(request(2→4). This order is used as a parameter for the priority operator below. For information on the priority operator see [BBK 86]. For completeness sake we list the axiom system, that makes use of an auxiliary operator ◁ (*unless*). Unfortunately, this binary operator shares a symbol with the ternary conditional operator of 2.2. Appropriate use of brackets is needed to avoid ambiguities.

$\theta(a) = a$	TH1
$\theta(x\cdot y) = \theta(x)\cdot\theta(y)$	TH2
$\theta(x + y) = \theta(x)\lhd y + \theta(y)\lhd x$	TH3
$a\lhd b = a \qquad \text{if } \neg(a < b)$	P1
$a\lhd b = \delta \qquad \text{if } a < b$	P2
$x\lhd(y\cdot z) = x\lhd y$	P3
$x\lhd(y + z) = (x\lhd y)\lhd z$	P4
$(x\cdot y)\lhd z = (x\lhd z)\cdot y$	P5
$(x + y)\lhd z = x\lhd z + y\lhd z$	P6
$\theta([u, x]) = [u, \theta(x)]$	THS1
$\theta(\langle x, u\rangle) = \langle\theta(x), u\rangle$	THS2
$x\lhd[u, y] = x\lhd y$	PS1
$x\lhd\langle y, u\rangle = x\lhd y$	PS2
$[u, x]\lhd y = [u, x\lhd y]$	PS3
$\langle x, u\rangle\lhd y = \langle x\lhd y, u\rangle$	PS4
$\theta(x\lhd\phi\rhd y) = \theta(x)\lhd\phi\rhd\theta(y)$	THC
$x\lhd(y\lhd\phi\rhd z) = (x\lhd y)\lhd\phi\rhd(x\lhd z)$	PC1
$(x\lhd\phi\rhd y)\lhd z = (x\lhd z)\lhd\phi\rhd(y\lhd z)$	PC2

TABLE 17. Priority operator.

This use of the priority operator corresponds precisely to the put mechanism for synchronous but unreliable message passing (see [BW90]). The aim is to allow arbitrarily many put(request(2→4)) actions and to let a communication take place just if the put is a vital one in the sense that it conveys new information to the control unit. A system with three pedestrians about to cross from 2 to 4 is then as follows:

$$\text{composite\_system} = \theta_{\leq}(\partial_{H^*}(\text{ped}(1) \parallel \text{ped}(2) \parallel \text{ped}(3) \parallel \text{SYSTEM2})).$$

As was said before, the weakness of this combination is of course that the pedestrian takes no caution to prevent crossing when the light is red. Notice that the action observe_green _at_2 (i) will not be blocked when a green signal at 2 is absent. In the setting of ACPS$^-$ that cannot be improved. This example is taken one step further in section 5, however. There, a feature is introduced that allows a proper interaction between the presence of the green signal at 2 and the proper execution of an action observe_green_at_ 2(i).

(ii) Consider once more the CLOCK of example 2.6.v. A plausible user of the clock can execute the following process user = uset_alarm(07.28.00). We assume that uset_alarm(t) can communicate with the CLOCK action set_alarm(t) to alarm(t). Let H = {uset_alarm(t), set_alarm(t) | t ∈ TIMES } then

$\partial_H(\text{user} \parallel \text{CLOCK})$

describes the proper cooperation between user and CLOCK.

(iii) Consider the example of a merge of light behaviours from example 3.2.ii. In this example a person is added to the scene. The person is supposed to enter the house at night and under the assumption that all lights are off. The actions that (s)he can perform are the actions pswitch(n) for n a name in {hall1, hall2, kitchen1, kitchen2, living1, living2, living3, living4}. The communication function works as follows: pswitch(n) | switch(n) = t.

Here t denotes some action that plays the role of an internal step of the system. The encapsulation set H contains all switch and pswitch actions.The person can for instance execute the following process:

```
person = enter_front_door · pswitch(hall1) · enter_living ·
            (pswitch(living1) || pswitch(living2)) · pswitch(living3) ·
    enter_kitchen · pswitch(kitchen1) · leave_kitchen ·
pswitch(hall1) · enter_living
```

The combined actions of system and person are then represented by the process expression

$$\text{person\_in\_house} = \partial_H(\text{person} \parallel \text{L(living/kitchen/hall)}).$$

Let w be the signal light(hall1) & light(hall2) & light(living1) & light(living2) & light(living3) & light(living4) & light(kitchen). Then the following identity can be shown:

```
w ∩ ∂{get_new_lamp(n), defect(n) | n ∈ NAMES}(person_in_house) =
    enter_front_door ·
    t ·
    ({light(hall1)} ∪
            (enter_living ·
        ((⟨t, {light(living1)}⟩ + ⟨t, {light(living2)}⟩) ·
        ⟨t, {light(living1), light(living2)}⟩ ·
        t·
        ({light(living1), light(living2), light(living3)} ∪
    enter_kitchen ·
            t·
            ({light(kitchen)} ∪
                leave_kitchen
            )
        )
    )·
    t·
    ({light(kitchen), light(living1), light(living2), light(living3)} ∪
        enter_living ·
        δ
    ).
```

4. PROCESSES WITH FREE MERGE AND OBSERVATION.

4.1 THE SIGNAL OBSERVATION FUNCTION.

In this section an observation mechanism is added to the features available in process algebra. In order to simplify the discussion, the communication mechanism is first left out in order to be added later on, in section 5. It is assumed that some actions a are able to read the signals of processes that are put in parallel with the process executing a.

We start out with the observation function as a function ρ: $A_\delta \times AS \to A_\delta$. An action $a \in A$ such that $obs(a, p) = b$ for some $b \in A$, $p \in AS$ is called an *observation action.* The signal observation function satisfies the following axioms:

$\rho(\delta, p) = \delta$	OBS1
$\rho(\rho(a, p), q)) = \rho(\rho(a, q), p)$	OBS2
$\rho(\rho(a, p), p) = \rho(a, p)$	OBS3

TABLE 18. Observation function for atomic signals.

Then, we extend the observation function to a function ρ: $A_\delta \times CS \to A_\delta$ by means of the following additional axioms:

$\rho(a, \varnothing) = a$	OBS4
$\rho(a, \{p\} \cup v) = \rho(\rho(a, p), v)$	OBS5

TABLE 19. Observation function for composite signals.

From the observation function we will manufacture an additional operator on processes denoted with /. This *signal observation operator* describes the inspection of a signal by a process. The inspection of a signal will check whether or not some given atomic signal is contained in it. First, we give axioms for the signal observation operator on atomic actions, i.e. we consider $/: A_\delta \times CS \to P$. Notice that a/u is always δ, an atomic action or a sum of atomic actions.

$\delta/u = \delta$	O1
$a/\varnothing = a$	O2
$a/v = \sum_{u \subseteq v} \rho(a, u)$	O3

TABLE 20. Signal observation on atomic actions.

Notice that axioms O2 and O3 imply that a/v always contains a as a summand, and in case we have a trivial observation function ($\rho(a, p) = \delta$ for all $a \in A$), we get $a/v = a$ for all $v \in CS$.

Next we extend signal observation to a operator $/: P \times CS \to P$. The intuitive meaning of x/u is a process that behaves like x be it that the first action of x is performed as an observation on the signal u.

$(x + y)/u = x/u + y/u$	O4
$(x \cdot y)/u = (x/u) \cdot y$	O5
$[u, x]/v = [u, x/v]$	O6
$\langle x, u \rangle/v = \langle x/v, u \rangle$	O7
$(x \lhd \phi \rhd y)/u = (x/u) \lhd \phi \rhd (y/u)$	O8
$x/(u \lhd \phi \rhd v) = (x/u) \lhd \phi \rhd (x/v)$	O9

TABLE 21. Signal observation on processes.

Note again that $x/v = x$ for all processes and signals in case of a trivial observation function.

4.2. PAS: SIGNAL OBSERVATION AND FREE MERGE.

The merge expansion equation M1 of 3.1 has to be modified, in order to obtain an equation for a merge operator that takes signal observation into account as well. The following version of this axiom was suggested in BROUWER [BR 90]. This completes the description of the axiom system PAS.

$x \parallel y = (x/S(y)) \mathbin{⫡} y + (y/S(x)) \mathbin{⫡} x$	OM

TABLE 22. Free merge with observation.

Notice that this equation reduces to the equation M1 in case of a trivial observation function.

In order to support the intuition for the observation mechanism we will now describe several examples of observation functions.

4.3. EXAMPLES OF SIGNAL OBSERVATION FUNCTIONS.

In all examples below we specify the observation function $\rho: A_\delta \times AS \to A_\delta$. Thus, p,q denote elements of AS.

i. The observation actions all have the form $obs(p)$ with p an element of AS. These actions represent the intention to observe a signal p. There is a special atomic action yes which is the result (confirmation) of a successful observation. The action yes is not an observation action. The signal observation function then works as follows.

$\rho(\mathrm{obs}(p), p) = \mathrm{yes}$
$\rho(\mathrm{obs}(p), q) = \delta$ if $p \neq q$
$\rho(a, p) = \delta$ for all actions a not of the form $\mathrm{obs}(p)$.

In connection with this observation function one will use a form of encapsulation which shields off all observations except yes. In this way seen from outside the encapsulation context, only succesful observations will be visible.

In this format it suffices to indicate which non-trivial observations exist and to omit the complementary definition of the observations that lead to δ.

Notice that an unsuccessful observation cannot be affirmed by an atomic action: if we put $\rho(\mathrm{obs}(p), q) = \mathrm{no}$ in case $p \neq q$, we get $\mathrm{obs}(p) / \{p,q\} = \mathrm{yes} + \mathrm{no}$, which is certainly counterintuitive.

ii. The second example takes the same sets of signals and observations as model (i) except for the addition of a label p to the act of confirmation of successful observation, which thus becomes yes(p). The definition is as follows.

$\rho(\mathrm{obs}(p), p) = \mathrm{yes}(p)$
$\rho(\mathrm{obs}(p), q) = \delta$ if $p \neq q$

All other observations lead to δ as in the previous example. This kind of observation allows one to keep track of the observed signals after encapsulation. The advantage being that no abstraction through renaming is involved and that a potential source of non-determinism is thereby removed. Notice that non-determinism in specifications is generally due to the level of abstraction rather than to the intrinsic non-determinism. Almost every system becomes non-deterministic when viewed at a sufficiently high level of abstraction.

iii. This example takes an element u of CS as the parameter of the observation actions. The observation test(u) succeeds if some atomic signal p in u is present. In this case the observation function can be defined as follows.

$\rho(\mathrm{test}(u), p) = \mathrm{yes}$ if $p \in u$
$\rho(\mathrm{test}(u), q) = \delta$ if $p \notin u$.

iv. We consider the generalization of the format in (i) above to actions obs(u) with u an element of CS. These actions represent the intention to observe all atomic signals in the composite signal u. The action obs(∅) will occur when all signals in u have been observed.

$\rho(\mathrm{obs}(u), p) = \mathrm{obs}(u-\{p\})$

v. The last example generalizes (ii) as in the previous format. Action obs(u,v) expresses that signals in u still have to be observed, and signals in v have been observed. We start with an action obs(u,∅) and complete success is then given by obs(∅,u). We call this observation function the *standard observation function*. The definition is as follows.

$\rho(\text{obs}(u,v), p) = \text{obs}(u-\{p\}, v\cup(u\cap\{p\}))$.

5. ACPS: SYNCHRONOUS COMMUNICATION AND OBSERVATION.

5.1. AXIOMS.

Now we will combine the above theories ACPS$^-$ and PAS to obtain ACPS (ACP with signals and signal observation) by changing the axiom of the merge. The purpose of the following equations is to ensure that all operators except +, ·, [.,.] and ⟨.,.⟩ can be eliminated from finite process expressions by means of left to right term rewriting. Notice that the original case of ACP can be viewed similarly with the understanding that all operators except + and · can be eliminated in that case.

Besides taking the axioms for ACPS$^-$ and PAS together while taking the sum of their merge expansion axioms, three axioms are needed that ensure that observation and communication will not interfere. The purpose of these axioms is to guarantee that merge will be associative. It is possible that more general schemes exist, but in any case having an observation that communicates at the same time seems to be a rather fancy feature. In table 23, $a,b \in A$, $p \in AS$.

$(\exists p\in AS\ \rho(a,p)\neq\delta) \Rightarrow a \mid b = \delta$	OBS6
$\rho(a,p) \mid b = \delta$	OBS7
$\rho(a \mid b, p) = \delta$	OBS8

TABLE 23. Observation and communication.

The new expansion axiom for merge with communication is then as follows:

$x \parallel y = (x/S(y)) ⌊\!⌊ y + (y/S(x)) ⌊\!⌊ x + x \mid y$	OCM

TABLE 24. Merge with observation and communication.

In case of a trivial observation function, this expression reduces to axiom CM1 of 3.3 and the interaction between processes works exactly as in ACP. ACP is therefore the special case of ACPS if the signal observation function vanishes everywhere. Similarly PA is the special case of ACP if the communication function vanishes everywhere. Obviously there is a situation where the communication function vanishes everywhere but the signal observation function may assume non-trivial values. In such cases one may omit the summand with the communication merge and we get PAS.

5.2. EXAMPLES.

(i) Let us reconsider the alarm clock of example 3.5.ii. We are now able to allow the user of the clock to observe the alarm and to switch off the clock subsequently. Let the user be described by the following process:

user1 = uset_alarm(07.00.00)·obs({alarm}, ∅)·uswitch_off_alarm

Here we assume that uswitch_off_alarm communicates with switch_off_alarm to off_alarm. The standard signal observation from section 4.3.v is used.

Finally the encapsulation set H contains all actions uset_alarm(t) as well as the action obs({alarm}, ∅). Then the cooperation of user and clock is given by:

$\partial_H(\mathrm{user1} \parallel \mathrm{CLOCK})$.

The specification allows but does not require the user to observe the alarm. If it is essential that the user hears the alarm a priority operator is needed that gives a higher priority to obs(∅, {alarm}) than to tick. A second step introduces a slightly more complex behavior for the user:

$$\begin{aligned}\mathrm{user2} &= \mathrm{uset\_alarm}(07.00.00)\cdot \mathrm{sleep}\cdot \mathrm{user2loop}\\ \mathrm{user2loop} &= \mathrm{obs}(\{\mathrm{alarm}\}, \emptyset)\cdot \mathrm{uswitch\_off\_alarm} + \mathrm{wakeup}\cdot \mathrm{sleep}\cdot \mathrm{user2loop}\end{aligned}$$

A third step is to make the behavior of the user again more complex, for instance if the user wakes up before the alarm has gone off (s)he may decide to sleep again without switching off the alarm provided $t < 06.45.00$. Here we need an observation for the time signal. We again use the standard signal observation function.
A new version user3 of the user then turns into:

$$\begin{aligned}\mathrm{user3} &= \mathrm{uset\_alarm}(07.00.00)\cdot \mathrm{sleep}\cdot \mathrm{user3loop}\\ \mathrm{user3loop} &= \mathrm{obs}(\{\mathrm{alarm}\}, \emptyset)\cdot \mathrm{uswitch\_off\_alarm} +\\ &\quad \mathrm{wakeup}\cdot\Big(\sum_{r\in\{00.00.00,..,06.45.00\}} \mathrm{obs}(\{r\},\emptyset)\cdot \mathrm{sleep}\cdot \mathrm{user3loop} \quad +\\ &\quad \sum_{r\in\{06.45.01,..,07.02.00\}} \mathrm{obs}(\{r\},\emptyset)\cdot \mathrm{uswitch\_off\_alarm}\Big)\end{aligned}$$

Of course the observations obs({t}, ∅) must be added to the encapsulation set H. Then the cooperation between user and alarm clock is again given by

$\partial_H(\mathrm{user3} \parallel \mathrm{CLOCK})$.

Of course substantially more involved types of user behavior can be modelled using the primitives of ACPS.

(ii) Returning to the example of the traffic lights (3.5.i), using the signal observation function of 4.3.iv, one adds the actions obs({green(k)}) to H. By simply working in the setting of ACPS the composite_system of example (3.5.i) will now work properly.

(iii) In this example we will elaborate once more on the example of the person in a house with various light switches (see 3.5.iii). The point here is that the person may want to inspect whether or not a light that was switched on indeed functions correctly. If not (s)he will replace the bulb by a new one. The actions that must be introduced here are inspections of the signals of the various lamps. We use the standard observation function. Recall that the atomic signals are in this case of the form light(n) and dark(n).
The action pswitch(n) can now be replaced, whenever it is assumed to switch the lamp on, by a more involved non-atomic process as follows:

$\mathrm{pswitch}(n)\cdot(\mathrm{obs}(\{\mathrm{light}(n)\},\emptyset) + \mathrm{obs}(\{\mathrm{dark}(n)\},\emptyset)\cdot \mathrm{put\_new\_lamp}(n))$

Of course the communication function has to be extended. In particular one needs a communication for the new lamp actions, for instance:

$\mathrm{put\_new\_lamp}(n) \mid \mathrm{get\_new\_lamp}(n) = \mathrm{new\_lamp}(n)$

A modification of the person process from (3.5.iii) to incorporate this additional feature is as follows:

```
person* = enter_front_door · pswitch(hall1) ·
   (obs({light(hall1)},∅) + obs({dark(hall1)},∅)·put_new_lamp(hall1)) ·
enter_living ·
   (pswitch(living1) || pswitch(living2)) · pswitch(living3) ·
    (      (obs({light(living1)},∅) + obs({dark(living1)},∅)·put_new_lamp(living1)) ||
           (obs({light(living2)},∅) + obs({dark(living2)},∅)·put_new_lamp(living2)) ||
           (obs({light(living3)},∅) + obs({dark(living3)},∅)·put_new_lamp(living3)
    ) ·
   enter_kitchen · pswitch(kitchen1) ·
   (obs({light(kitchen)},∅) + obs({dark(kitchen)},∅)·put_new_lamp(kitchen)) ·
   leave_kitchen · pswitch(hall1) · enter_living
```

Composing this process person* with L(living/kitchen/hall) in the system ACPS will indeed produce a process that shows appropriate interaction of the person with its environment. Notice however that there are many different plausible behaviours of the person. There seems to be no universal generic behavior for person even in this very simple case. Obviously other factors that bear no relation to the switching of light influence the behavior of the person. Nevertheless it is a fair hypothesis to assume that the person behaves for some short time as a process. Thus locally (in time) person (or person*) is a process in the sense of process algebra, but globally a much more complex stucturing mechanism is needed. One can imagine, however, that it is possible to formally incorporate in process algebra a small knowledge base which structures the decision taking process of the person.

6. FURTHER EXTENSIONS OF ACPS.

Having the mechanisms of ACPS available it is natural to formulate many additional features on top of it which will increase expressive power without leading to semantical difficulties. In this section some of such mechanisms will be reviewed.

6.1. STATE OPERATORS THAT GENERATE SIGNALS.

Let us assume that a state operator in the sense of [BB 88] is given by a domain S and functions act: $A_\delta \times S \to A_\delta$ and eff: $A \times S \to S$. The expression $\lambda_s(x)$ with $s \in S$ denotes process x working on the state space S with the current state being $s \in S$.

We can assume that there is an additional function sig: $S \to CS$ which determines for each state the signal that is produced by that state. The absence of signals is modeled by taking $sig(s) = \varnothing$ of course. Now the six equations for the state operator are as shown below.

$\lambda_s(\delta) = [sig(s), \delta]$	LS1
$\lambda_s(a) = [sig(s), act(a, s)]$	LS2
$\lambda_s(a \cdot x) = [sig(s), act(a, s) \cdot \lambda_{eff(a, s)}(x)]$	LS3
$\lambda_s(x + y) = \lambda_s(x) + \lambda_s(y)$	LS4
$\lambda_s([u, x]) = [u, \lambda_s(x)]$	LS5
$\lambda_s(\langle x, u\rangle) = \langle\lambda_s(x), u\rangle$	LS6

TABLE 25. State operator generating signals.

Using a state operator that generates signals one can define signaling processes in such a way that the recursion equations need not contain any signal at all, thus considerably optimizing the notation. We will illustrate this in two examples.

EXAMPLE 1.

Let D be a finite alphabet of data, and let ST(D) be the collection of finite sequences over D. The empty sequence is denoted by $\emptyset$ and adding an element d to the list x results in push(d, x). The atomic signals are as follows:

top(d) for $d \in D$,
empty.

ST(D) will be the state space for a process that represents a stack over D. The signal function sig is de fined by $sig(\emptyset) = \{empty\}$, $sig(push(d, x)) = \{top(d)\}$. The atomic actions are:

push_int(d), push(d) for $d \in D$ (the suffix int denotes an *intended* action),
pop_int, pop.

The functions act and eff are given by:

act(push_int(d), x) = push(d) (the act function transforms an intended action into an actual action),
act(pop_int, x) = pop,
eff(push_int(d), x) = push(d, x) (the eff function gives the resulting contents of the stack),
eff(pop_int, $\emptyset$) = $\emptyset$,
eff(pop_int, push(d, x)) = x.

(For act only those cases are given where act will not lead to δ.). The behavior of a stack over D is the given by the following process definition.

$$stack(D) = \lambda_{\emptyset}(ST\_INT)$$
$$ST\_INT = (\sum_{d \in D} push\_int(d) + pop\_int)\cdot ST\_INT.$$

EXAMPLE 2.

In this example two buffers A and B with data from the finite set D are maintained in the state. Both buffers have length $k > 1$. The process to be defined allows to read data in both buffers in a concurrent mode. For both buffers A and B there are two atomic signals: open_A indicates that there is still room in A, closed_A indicates that A has been filled (likewise for B). When both buffers have been loaded the action comp compares the contents of the buffers. The comparison will send value true if the buffers were equal and false otherwise. Thereafter the buffers are made empty again and the process restarts. We will describe the system in a top-down fashion, first explaining the overall architecture and then completing the details. As a notational convention we have adopted moreover that intended actions (which have not yet been processed by the state operator but for which this is envisaged in the design) are marked with the suffix int.

$$SYSTEM = \lambda_{\langle\emptyset,\emptyset\rangle} \circ \partial_H (X)$$
$$X = (int\_A \parallel int\_B)\cdot X.$$

The state consists of a pair of buffers A and B. Initially both are empty. The signals produced by a state $\langle\alpha, \beta\rangle$ are as follows.

sig_A(⟨α, β⟩) =	{open_A}	if length(α) < k,
	{closed_A}	if length(α) = k.
sig_B(⟨α, β⟩) =	{open_B}	if length(β) < k,
	{closed_B}	if length(β) = k.

sig(x) = sig_A(x) ∪ sig_b(x).

The processes int_A and int_B are defined by:

$$\text{int\_A} = \sum_{d\in D} \text{read\_int\_A}(d)\cdot\text{int\_A} + \text{comp\_int\_A}$$

$$\text{int\_B} = \sum_{d\in D} \text{read\_int\_B}(d)\cdot\text{int\_B} + \text{comp\_int\_B}$$

Communication is defined as follows:

comp_int_A | comp_int_B = comp_int,

H = {comp_int_A, comp_int_B}

The next step is to explain the effect function for those actions that pass the encapsulation operator:

eff(read_int_A(d), ⟨α, β⟩) =	α * d	if length(α) < k
	α	otherwise
eff(read_int_B(d), ⟨α, β⟩) =	β * d	if length(β) < k
	β	otherwise
eff(comp_int, ⟨α, β⟩) =	∅	

Finally the action function must be specified:

act(read_int_A(d), ⟨α, β⟩) =	read_A(d)	if length(α) < k
	δ	otherwise
act(read_int_B(d), ⟨α, β⟩) =	read_B(d)	if length(β) < k
	δ	otherwise
act(comp_int, ⟨α, β⟩) =	write(true)	if lenght(α) = k and α = β
	write(false)	if length(α) = length(β) = k and α ≠ β,
	δ	otherwise.

The use of the state operator in this example is hard to avoid because of the parallel reading of data that must be used simultaneously later on. This issue is worked out in [VE 90].

6.2. STATE OPERATOR WITH CONDITIONS.

In case we have conditions over an arbitrary Boolean algebra $\mathbb{B}_n$ with n generators, we need a function eval: $\mathbb{B}_n \times S \to \mathbb{B}_n$ (S the set of states) satisfying the following axioms. With the help of this function we can give an axiom for the state operator on conditional expressions.

eval(s, true) = true	E1
eval(s, false) = false	E2
eval(s, $\phi \wedge \psi$) = eval(s, ϕ) $\wedge$ eval(s, ψ)	E3
eval(s, $\phi \vee \psi$) = eval(s, ϕ) $\vee$ eval(s, ψ)	E4
eval(s, $\neg\phi$) = $\neg$eval(s, ϕ)	E5
$\lambda_s(x \lhd \phi \rhd y) = \lambda_s(x) \lhd \text{eval}(\phi, s) \rhd \lambda_s(y)$	LC

TABLE 26. State operator and conditions.

6.3 INFINITE NUMBER OF ATOMIC SIGNALS.

This extension is a rather plausible one. Indeed there is no particular reason why the set of atomic signals must be finite as long as all composite signals that are used in a process are finite combinations of atomic signals. For instance let the natural numbers be written in unary form by means of 0 and a function S, act as atomic signals. Then process count(k) for $k \in \mathbb{N}$ can be defined as follows:

$$\text{count}(k) = [\{S^k(0)\}, t]\cdot(\text{count}(k+1) + \text{stop}).$$

Assume the presence of an observation action above(10) that allows the following observations (in the format of 4.3.i): $\text{obs}(\text{above}(10), S^{k+10}(0)) = \text{yes}$.

Let the only non-trivial communication be $\text{stop} \mid \text{yes} = \text{end}$, then with $H = \{\text{above}(10), \text{stop}\}$, the process $\partial_H(\text{count}(0) \parallel \text{above}(10))$ denotes a process that will count until some value above 9 and then terminate. Under the assumption that there are only finitely many atomic actions, the use of these signals taken from an arbitrary data type is only limited as far as process interaction within process algebra is concerned. Nevertheless, one can imagine that an external observer can indeed distinguish all signals so that the complex signals are useful to model unintended external behavior.

6.4 AN ALTERNATIVE NOTATION FOR ROOT SIGNAL INSERTION.

An alternative to the use of the binary root signal insertion operator [.,.] is to introduce the unary deadlock signal mapping $\delta(u)$ which is defined by $\delta(u) = [u, \delta]$.
In fact root signal insertion can be expressed using this operation:

$$[u, x] = x + \delta(u).$$

Because the deadlock signal mapping and root signal insertion are expressible with respect to one another it is possible to provide an axiomatisation of the algebra of communicating processes with signals using $\delta(.)$ rather than [.,.]. In terms of the complexity of the signature that leads to a substantial simplification, because a unary operator is a simpler concept than a binary one. The main reason, however, not to use the deadlock signal operator instead of root signal insertion is that it would lead to a set of axioms which is considerably harder to read than the given equations.

7. BISIMULATION SEMANTICS FOR PROCESSES WITH SIGNALS.

7.1. PROCESS GRAPHS WITH NODE LABELS.

Many possible semantic worlds exist for ACP. It is impossible and unnecessary to review all models of importance here. The bisimulation model being the initial algebra semantics of ACP stands out as the most important model in our view, in spite of its ignorance of the concept of true concurrency and it not being fully abstract. We will indicate how a bisimulation semantics can be obtained for processes with signals. This new notion of bisimulation specialises to the original form in the case of processes with trivial signals only.

We assume that a finite set A of atoms is given as well as a finite set AS of signals. We assume A and AS to be disjoint. Outside A we have an additional constant δ. A process in the setting of this simple form of bisimulation semantics is a finitely branching, directed, acyclic and rooted graph, with nodes labeled by a signal in CS (a subset of AS) and edges labeled by actions in A. Terminal nodes in the graph may have an additional label δ besides the signal label from CS. This indicates that the branch terminates in deadlock. Absence of the label δ at a terminal node implies proper and correct termination.

The interpretation of the operators is as follows.

- δ corresponds to the single node graph with two labels: $\emptyset$ and δ.
- The atomic action a corresponds to a graph with two nodes and one connecting edge. Both nodes are labeled with $\emptyset$ and the edge is labeled with a.
- Given graphs g and h, their sum is obtained by identifying the root nodes and taking the union of the root signals. Only if both roots have label δ, the new root obtains a label δ as well.
- The product g·h of two graphs g and h is obtained from g by attaching a copy of h at each terminal node of g that has no label δ. The signal labels of the terminal nodes of g in g·h are recomputed by taking the union in each case with the root signal of Q.
- The graph [u, g] is obtained from g by combining its root signal with u, and the graph ⟨g, u⟩ is found from g by combining each signal label of a properly terminating node of g with u.
- S(g) is the root signal of g and P(g) results from g by replacing the root signal of g by $\emptyset$.
- The merge of process graphs g and h without communication corresponds to taking the cartesian product of the two graphs. The signal of a pair (p, q) of nodes is just the combination of the signals of the individual nodes. A transition $(p, q) \xrightarrow{a} (p', q')$ exists if either $p \xrightarrow{a} p'$ and $q = q'$ or $q \xrightarrow{a} q'$ and $p = p'$. In the case of communication the diagonal transitions $(p, q) \xrightarrow{c} (p', q')$ are added whenever there is a non-trivial communication c between actions a and b such that $p \xrightarrow{a} p'$ and $q \xrightarrow{b} q'$. The left merge is constructed by first defining the merge and then removing all initial transitions that correspond to first move of the second argument of the left merge. Similarly a communication merge is defined starting from the merge by removing al initial transitions that are not communications. Of course both in the case of the left merge and in the case of the merge there may result inaccessible parts of the graph after deletion of the relevant initial transitions. These superfluous parts of the process graphs of course have to be removed afterwards.

Finally the presence of signal observations must be taken into account. The merge of two processes in the case of ACPS starts with constructing their merge in the sense of ACPS^{-}. The domain of the new graph is then obtained but more edges have to be added in order to take the succesful observations into account. If $q = q'$, $p \xrightarrow{a} p'$ and $a / S(q) \xrightarrow{b} \surd$, or $p = p'$, $q \xrightarrow{a} q'$ and $a / S(p) \xrightarrow{b} \surd$, then a transition $(p, q) \xrightarrow{b} (p', q')$ is added.

Equally interesting and informative as these graph constructions is an operational semantics based on actions and labeled transitions between expressions over the free syntax of ACPS. Such an operational semantics is given in section 8.

7.2 BISIMULATION FOR PROCESS GRAPHS WITH NODE LABELS.

A *bisimulation* between graphs g and h is a bisimulation in the sense usual for process algebra with the additional restriction that if a bisimulation R relates a pair (s, t) of states of g and h the labels of the nodes s and t must coincide. In this way a model of BPAS is obtained. It can be shown that BPAS provides a complete axiomatisation for bisimulation congruence on processes with signals. The proof uses the second canonical form as described below and is postponed until after the discussion of these canonical forms. Without proof we state the following properties of bisimulation semantics on processes with signals.

(i) In the initial algebra of ACPS the merge operator is associative.

(ii) The initial algebra of ACPS is an expansion of the initial algebra of ACP, whence ACPS is a conservative enrichment of ACP (at least as far as identities between finite closed process expressions are concerned).

(iii) The initial algebra of ACPS is isomorphic to the bisimulation model with appropriate definitions of all operators other than + and ·.
(iv) ACPS constitutes (on finite closed process expressions) a terminating and confluent term rewriting system modulo its permutative identities for +.
(v) The notion of a regular process is defined just as in the case without signals. A process graph is regular if it bisimulates with a finite graph. All regular processes with signals can be specified by means of finite linear systems of equations. All operations on processes described in this paper transform regular processes into regular processes. The algebra of regular processes exists just as well for ACPS as it does for ACP.
(vi) ACPS allows a projective limit model in very much the same way as ACP does.

The equations of ACPS first and for all express an intuition. The virtue of these equations is to be consistent in bisimulation semantics, secondly and not of less importance to allow the transformation of each finite process expression in each of several canonical forms. These canonical forms are discussed below. Finally an aim is to show the axioms complete with respect to bisimulation semantics, at least in the case of finite processes. A third objective is to anticipate for guarded recursive definitions and to enforce that finite projections of processes defined with guarded recursion can be transformed to finite process expressions without recursion. These finite process expressions can then be transformed into one of the canonical forms.

7.3 Canonical forms for ACPS

The canonical forms for finite processes are less easily established in the presence of signals and observations. First of all one observes that each process graph can be unfolded into a bisimilar tree. The canonical forms are all derived from the tree representation. Consider the following classes of finite closed process expressions.

(i) CANP1.

- For u in CS, $[u, \delta]$ is in CANP1.
- If $n, m, k \geq 0$ with $n + m + k > 0$, and $a_1,\dots,a_n, b_1,\dots,b_m, c_1,\dots,c_k$ are atomic actions different from δ; if moreover $u, u_1,\dots, u_m$ are signals in CS and $X_1,\dots,X_n$ are in CANP1 then

$$[u, a_1{\cdot}X_1 + \dots + a_n{\cdot}X_n + \langle b_1, u_1\rangle + \dots + \langle b_m, u_m\rangle + \dots + c_1 + \dots + c_k]$$

is an expression in CANP1.

In CANP1 the number of occurrences of $\langle .,. \rangle$ is minimised. Moreover each node label is displayed at exactly one position in the expression. Closing brackets of $[.,.]$ are moved to the last possible position. This type of canonical form arises from the most obvious translation of processes as labeled trees into process expressions. These canonical forms are needed if one is to understand the equations for / above. CANP1 contains exactly the normal forms of expressions according to BPAS.

(ii) CANP2.

- For u in CS, $[u, \delta]$ is in CANP2.
- If $n, m, k \geq 0$ with $n + m + k > 0$, and $a_1,\dots,a_n, b_1,\dots,b_m, c_1,\dots,c_k$ are atomic actions different from δ; if moreover $u, u_1,\dots, u_m$ are signals in CS and $X_1,\dots,X_n$ are in CANP2 then

$$[u, a_1]{\cdot}X_1 + \dots + [u, a_n]{\cdot}X_n + [u, \langle b_1, u_1\rangle] + \dots + [u, \langle b_m, u_m\rangle] + \dots + [u, c_1] + \dots + [u, c_k]$$

is an expression in CANP2.

The expressions in CANP2 have the disadvantage that the root label of each subexpression is duplicated for every branch of the subexpression. The advantage of this kind of canonical form lies in the simple definition of finite projections and the corresponding option to consider infinite expressions as limits of finite expressions and to manufacture a projective limit model in the style of [BK 84].

(iii) CANP3.

- For u in CS, $[u, \delta]$ is in CANP3.
- If $n, m, k \geq 0$ with $n + m + k > 0$, and $a_1,...,a_n, b_1,...,b_m, c_1,...,c_k$ are atomic actions different from δ; if moreover $u, v_1,..., v_n, u_1,..., u_m$ are signals in CS and $X_1,...,X_n$ are in CANP3 such that for X_i the root signal is v_i then

$$[u, \langle a_1,v_1\rangle]\cdot X_1 +... + [u, \langle a_n,v_n\rangle]\cdot X_n + [u, \langle b_1,u_1\rangle] +... + [u, \langle b_m,u_m\rangle] +... + [u, c_1] +... + [u, c_k]$$

is an expression in CANP3.

The expressions in CANP3 have the disadvantage that the root label and terminal label of each subexpression is duplicated for every branch of the subexpression. The advantage is that this form is useful if one is to translate the signal observation mechanism into the more primitive communication mechanism.

(iv) CANP4.

Let CANP4* be the following class of expressions:

- δ is in CANP4*.
- If $n, m, k \geq 0$ with $n + m + k > 0$, and $a_1,...,a_n, b_1,...,b_m, c_1,...,c_k$ are atomic actions different from δ; if moreover $v_1,..., v_n, u_1,..., u_m$ are signals in CS and $X_1,...,X_n$ are in CANP4* then

$$\langle a_1, v_1\rangle\cdot X_1 + ... + \langle a_n, v_n\rangle\cdot X_n + \langle b_1, u_1\rangle + ... + \langle b_m, u_m\rangle + ... + c_1 + ... + c_k$$

is an expression in CANP4*.

Now CANP4 consists of all expressions of the form $[u, X]$ with X an expression in CANP4*. Clearly the point of CANP4 is to minimise the use of [.,.].

(v) CANP5.

If $n, m, k \geq 0$ and $a_1,...,a_n, b_1,...,b_m, c_1,...,c_k$ are atomic actions different from δ; if moreover $u, u_1,..., u_m$ are signals in CS and $X_1,...,X_n$ are in CANP5 then

$$[u, \delta] + a_1\cdot X_1 + ... + a_n\cdot X_n + \langle b_1, u_1\rangle + ... + \langle b_m, u_m\rangle + ... + c_1 + ... + c_k$$

is an expression in CANP5.

These normal forms depend entirely on the presence of δ. Clearly these forms are very simple and in particular this form helps in the design of recursion equations for parametrised processes with signals. Of course this canonical form cannot be used in absence of the deadlock constant.

7.4 COMPLETENESS OF THE AXIOMS FOR BISIMULATION IN THE CASE OF FINITE PROCESSES.

Using the third canonical form, a completeness proof for bisimulation congruence for processes with signals can be given. We will consider the simplest case without deadlock. We also have to assume that the set of atomic signals, AS, is finite. One introduces a new set of atomic actions, viz. for each atomic action a and signals u, v we have $[u, \langle a, v\rangle]$ as a new atomic action. Let B be the alphabet of these new actions. Let CANP3 be the collection of process expressions over B that is in the third canonical form.

Let P and Q be two bisimilar process expressions with signals. Using the equations of BPAS both expressions can be transformed into a CANP3 form, this results in P' and Q'. Now observe that two processes in third canonical form bisimulate in the new sense if and only if they bisimulate as processes over B in the usual sense. Therefore P' and Q' are bisimilar as processes over B. Because BPA is a complete axiomatisation for bisimulation semantics on finite processes P' and Q' can be proven equal by means of axioms of BPA a considering them as processes over B. It turns out however that the axioms of BPA are just as valid for P' and Q' seen as CANP3 expressions. Indeed the axioms of BPA when applied as rewriting rules to CANP3 expressions do not lead outside that class.

7.5 REMARK.
Models in the presence of conditonal expressions can be given similarly, be it that transitions will be labeled by pairs of actions and (enabling) conditions, and a notion of bisimulation must be adapted to that setting. As an example, we would have $\phi :\rightarrow (\neg\psi :\rightarrow a\cdot(b + c)) \xrightarrow{a,\phi\wedge\neg\psi} (b + c)$.

We will not do so at this point. Instead, we will provide a structural operational semantics involving signals in 8.2.

8. AN OPERATIONAL SEMANTICS FOR ACPS

8.1 RULES FOR ACP AND ACPS

An operational semantics of ACP is nothing new and can be retrieved from several sources, such as [VGL 87]. We prefer the approach in which only the operators of ACP that genuinely represent program constructions are provided with an operational semantics. The motivation for this style is the point of view that the auxiliary operators are there to facilitate the algebra but that the primary operators have an importance that far exceeds that of the algebra. Therefore the operational semantics of the syntax for process expressions should be accessible also to readers who have no knowledge of the auxiliary operators and their sometimes not quite straightforward role.

In the case of ACPS one assumes that the communication function is given as a partial function on the atomic actions (i.e. not for δ) and that the signal observation function is given as a total operator on atomic actions and atomic signals. Each of these operations must satisfy the required axioms.

In these circumstances the proper atomic actions and the operators +, ·, [.,.], ⟨.,.⟩, ||, ∂_H represent appropriate system description mechanisms, all others are auxiliary operators used to make the algebraic specification possible or concise. We need the notation $\surd(u)$ to indicate that a process terminates in a proper terminal state with node label u. As an abbreviation for $\surd(\varnothing)$ we use the simpler notation $\surd$. These expressions are itself not a process expression and will not occur in complicated ways as a subexpression of large process expression. In the rules below X and Y denote arbitrary process expressions.

$a \xrightarrow{a} \surd$	
$\dfrac{X \xrightarrow{a} X'}{X + Y \xrightarrow{a} X'}$	$\dfrac{X \xrightarrow{a} \surd(u)}{X + Y \xrightarrow{a} \surd(u)}$
$\dfrac{Y \xrightarrow{a} Y'}{X + Y \xrightarrow{a} Y'}$	$\dfrac{Y \xrightarrow{a} \surd(u)}{X + Y \xrightarrow{a} \surd(u)}$

TABLE 27. Rules for BPAS (first part).

$\dfrac{X \xrightarrow{a} X'}{X \cdot Y \xrightarrow{a} X' \cdot Y}$	$\dfrac{X \xrightarrow{a} \surd(u)}{X \cdot Y \xrightarrow{a} [u,Y]}$
$\dfrac{X \xrightarrow{a} X'}{\langle X,u \rangle \xrightarrow{a} \langle X',u \rangle}$	$\dfrac{X \xrightarrow{a} \surd(v)}{\langle X,u \rangle \xrightarrow{a} \surd(v \cup u)}$
$\dfrac{X \xrightarrow{a} X'}{[u,X] \xrightarrow{a} X'}$	$\dfrac{X \xrightarrow{a} \surd(v)}{[u,X] \xrightarrow{a} \surd(v)}$

TABLE 27. Rules for BPAS (second part).

$\dfrac{X \xrightarrow{a} X'}{X \| Y \xrightarrow{a} X' \| Y}$	$\dfrac{X \xrightarrow{a} \surd(u)}{X \| Y \xrightarrow{a} u \cup Y}$
$\dfrac{Y \xrightarrow{a} Y'}{X \| Y \xrightarrow{a} X \| Y'}$	$\dfrac{Y \xrightarrow{a} \surd(u)}{X \| Y \xrightarrow{a} u \cup X}$

TABLE 28. Additional rules for PAS$^-$.

$\dfrac{X \xrightarrow{a} X',\ Y \xrightarrow{b} Y',\ a \mid b = c}{X \| Y \xrightarrow{c} X' \| Y'}$	$\dfrac{X \xrightarrow{a} \surd(u),\ Y \xrightarrow{b} Y',\ a \mid b = c}{X \| Y \xrightarrow{c} u \cup Y'}$
$\dfrac{X \xrightarrow{a} X',\ Y \xrightarrow{b} \surd(u),\ a \mid b = c}{X \| Y \xrightarrow{c} u \cup X'}$	$\dfrac{X \xrightarrow{a} \surd(u),\ Y \xrightarrow{b} \surd(v),\ a \mid b = c}{X \| Y \xrightarrow{c} \surd(u \cup v)}$
$\dfrac{X \xrightarrow{a} X',\ a \notin H}{\partial_H(X) \xrightarrow{a} \partial_H(X')}$	$\dfrac{X \xrightarrow{a} \surd(u),\ a \notin H}{\partial_H(X) \xrightarrow{a} \surd(u)}$

TABLE 29. Additional rules for ACPS$^-$.

$\dfrac{X \xrightarrow{a} X',\ \rho(a,u) = b \neq \delta,\ u \subseteq S(Y)}{X \| Y \xrightarrow{b} X' \| Y}$	$\dfrac{X \xrightarrow{a} \surd(v),\ \rho(a,u) = b \neq \delta,\ u \subseteq S(Y)}{X \| Y \xrightarrow{b} v \cup Y}$
$\dfrac{Y \xrightarrow{a} Y',\ \rho(a,u) = b \neq \delta,\ u \subseteq S(X)}{X \| Y \xrightarrow{b} X \| Y'}$	$\dfrac{Y \xrightarrow{a} \surd(v),\ \rho(a,u) = b \neq \delta,\ u \subseteq S(X)}{X \| Y \xrightarrow{b} v \cup X}$

TABLE 30. Additional rules for ACPS.

In the two next rules E is a (guarded) system of recursion equations that contains the equation $X = s$. $\langle X \mid E \rangle$ denotes a process that is a solution for this equation. $\langle s \mid E \rangle$ denotes a process as follows: s is a process expression with some free variables in E, for these process variables the unique solution of these variables in E is substituted.

$\dfrac{\langle s \mid E\rangle \xrightarrow{a} Y}{\langle X \mid E\rangle \xrightarrow{a} Y}$	$\dfrac{\langle s \mid E\rangle \xrightarrow{a} \surd(u)}{\langle X \mid E\rangle \xrightarrow{a} \surd(u)}$

TABLE 31. Rules for recursion.

8.2 CONDITIONALS AND STATE OPERATOR.

In the presence of conditionals over an arbitrary finite Boolean algebra $\mathbb{B}_n$, we will label the transitions with pairs a,ϕ, where $a \in A$ is an atomic action (so $a \neq \delta$) and ϕ a non-false expression over $\mathbb{B}_n$ ($\phi \neq$false). We obtain the following rules.

$a \xrightarrow{a,true} \surd$	
$\dfrac{X \xrightarrow{a,\phi} X'}{X+Y \xrightarrow{a,\phi} X'}$	$\dfrac{X \xrightarrow{a,\phi} \surd(u)}{X+Y \xrightarrow{a,\phi} \surd(u)}$
$\dfrac{Y \xrightarrow{a,\phi} Y'}{X+Y \xrightarrow{a,\phi} Y'}$	$\dfrac{Y \xrightarrow{a,\phi} \surd(u)}{X+Y \xrightarrow{a,\phi} \surd(u)}$
$\dfrac{X \xrightarrow{a,\phi} X'}{X \cdot Y \xrightarrow{a,\phi} X' \cdot Y}$	$\dfrac{X \xrightarrow{a,\phi} \surd(u)}{X \cdot Y \xrightarrow{a,\phi} [u,Y]}$
$\dfrac{X \xrightarrow{a,\phi} X'}{\langle X,u\rangle \xrightarrow{a,\phi} \langle X',u\rangle}$	$\dfrac{X \xrightarrow{a,\phi} \surd(v)}{\langle X,u\rangle \xrightarrow{a,\phi} \surd(v \cup u)}$
$\dfrac{X \xrightarrow{a,\phi} X'}{[u,X] \xrightarrow{a,\phi} [u,X'}$	$\dfrac{X \xrightarrow{a,\phi} \surd(v)}{[u,X] \xrightarrow{a,\phi} \surd(v)}$
$\dfrac{X \xrightarrow{a,\phi} X',\ \phi \wedge \psi \neq \text{false}}{X \lhd \psi \rhd Y \xrightarrow{a,\phi \wedge \psi} X'}$	$\dfrac{X \xrightarrow{a,\phi} \surd(u),\ \phi \wedge \psi \neq \text{false}}{X \lhd \psi \rhd Y \xrightarrow{a,\phi \wedge \psi} \surd(u)}$
$\dfrac{Y \xrightarrow{a,\phi} Y',\ \phi \wedge \neg\psi \neq \text{false}}{X \lhd \psi \rhd Y \xrightarrow{a,\phi \wedge \neg\psi} Y'}$	$\dfrac{Y \xrightarrow{a,\phi} \surd(u),\ \phi \wedge \neg\psi \neq \text{false}}{X \lhd \psi \rhd Y \xrightarrow{a,\phi \wedge \neg\psi} \surd(u)}$

TABLE 32. Rules for BPAS with conditionals.

Adding the rules for parallel composition is straightforward. It is interesting to look at the rules for the state operator. This works as shown in table 33.

With this extended operational semantics involving conditions on the arrows comes a new definition for bisimulation. Instead of just requiring matching actions, we also require matching conditions. The following defintion starts from the set of valuations of the generators of the boolean algebra, i.e. all mappings v: $\{\theta_1,\dots,\theta_n\} \rightarrow$ BOOL. Each such mapping naturally extends to a mapping v: $\mathbb{B}_n \rightarrow$ BOOL. Then we say that a relation $\approx$ on a transition system is a *bisimulation* when the following holds:

i. if $X \approx Y$ then $S(X) = S(Y)$

$$\frac{X \xrightarrow{a,\phi} X',\ act(a,s)=b\neq\delta,\ eval(s,\phi)=\psi\neq false}{\lambda_s(X) \xrightarrow{b,\psi} \lambda_{eff(a,s)}(X')}$$

$$\frac{X \xrightarrow{a,\phi} \surd(u),\ act(a,s)=b\neq\delta,\ eval(s,\phi)=\psi\neq false}{\lambda_s(X) \xrightarrow{b,\psi} \surd(u)}$$

TABLE 33. Rules for state operator.

ii. if $X \approx Y$ and $X \xrightarrow{a,\phi} X'$, then for all valuations v with $v(\phi)$ = true, there is a condition ψ with $v(\psi)$ = true and an expression Y' such that $Y \xrightarrow{a,\psi} Y'$ and $X' \approx Y'$

iii. if $X \approx Y$ and $Y \xrightarrow{a,\phi} Y'$, then for all valuations v with $v(\phi)$ = true, there is a condition ψ with $v(\psi)$ = true and an expression X' such that $X \xrightarrow{a,\psi} X'$ and $X' \approx Y'$

iv. if $X \approx Y$ and $X \xrightarrow{a,\phi} \surd(u)$ then for all valuations v with $v(\phi)$ = true, there is a condition ψ with $v(\psi)$ = true such that $Y \xrightarrow{a,\psi} \surd(u)$

v. if $X \approx Y$ and $Y \xrightarrow{a,\phi} \surd(u)$ then for all valuations v with $v(\phi)$ = true, there is a condition ψ with $v(\psi)$ = true such that $X \xrightarrow{a,\psi} \surd(u)$.

We call two expressions X,Y bisimilar if there is a bisimulation relating X and Y. We claim that transition systems modulo bisimulation form a model for our theory. It is an open question whether our axiomatization is complete for this model.

8.3 STANDARD SIGNAL OBSERVATION FUNCTION.

In ACPS it is quite often convenient to have a fixed signal observation function in mind. This was already mentioned in 4.3 and here, we give an operational semantics for some of the functions described there. First the case 4.3.i. Then, the rules in table 30 specialize to the following form.

$$\frac{X \xrightarrow{obs(p)} X',\ p\in S(Y)}{X \| Y \xrightarrow{yes} X' \| Y} \qquad \frac{X \xrightarrow{obs(p)} \surd(u),\ p\in S(Y)}{X \| Y \xrightarrow{yes} u \cup Y}$$

$$\frac{Y \xrightarrow{obs(p)} Y',\ p\in S(X)}{X \| Y \xrightarrow{yes} X \| Y'} \qquad \frac{Y \xrightarrow{obs(p)} \surd(u),\ p\in S(X)}{X \| Y \xrightarrow{yes} u \cup X}$$

TABLE 34. Signal observation function of 4.3.i.

As another example, we can consider the format 4.3.iv. This looks as follows.

$$\frac{X \xrightarrow{obs(u)} X',\ w\subseteq u\cap S(Y)}{X \| Y \xrightarrow{obs(u-w)} X' \| Y} \qquad \frac{X \xrightarrow{obs(u)} \surd(v),\ w\subseteq u\cap S(Y)}{X \| Y \xrightarrow{obs(u-w)} v \cup Y}$$

$$\frac{Y \xrightarrow{obs(u)} Y',\ w\subseteq u\cap S(X)}{X \| Y \xrightarrow{obs(u-w)} X \| Y'} \qquad \frac{Y \xrightarrow{obs(u)} \surd(v),\ w\subseteq u\cap S(X)}{X \| Y \xrightarrow{obs(u-w)} v \cup X}$$

TABLE 35. Signal observation function of 4.3.iv.

9. ABSTRACTION.

In this section, we will consider extensions of process algebra with signals to the case where internal, non-observable actions (so-called τ-actions) are present. We will consider two semantics of such an extension:
i. branching bisimulation equivalence of [GW 89], [BW 90];
ii. weak bisimulation equivalence (or observational equivalence) of Milner [MI 80, MI 89], [BK 85].
In both cases, the constant τ is the result of abstraction, as given by the abstraction operator τ_I, that renames all actions from I into τ. In the next table, $I \subseteq A$, $a \in A \cup \{\tau\}$ (so always $\tau_I(\delta) = \delta$).

$\tau_I(a) = a$	if $a \notin I$	TI1
$\tau_I(a) = \tau$	if $a \in I$	TI2
$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$		TI3
$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$		TI4
$\tau_I([u, x]) = [u, \tau_I(x)]$		TSI1
$\tau_I(\langle x, u\rangle) = \langle \tau_I(x), u\rangle$		TSI2
$\tau_I(x \lhd \phi \rhd y) = \tau_I(x) \lhd \phi \rhd \tau_I(y)$		TIC

TABLE 36. Abstraction operator.

9.1 BRANCHING BISIMULATION.

Graphs now have edges labeled by elements of $A \cup \{\tau\}$, and nodes labeled by an element of CS. Endnodes may have an additional label δ. If s and t are two nodes in a process graph g, then we put $s \Rightarrow t$ if there is a (possibly empty) path from s to t containing only τ-edges.

$R: g \leftrightarrow_b h$, relation R between nodes of g and nodes of h is a *branching bisimulation* between g and h iff
i. the roots of g and h are related by R
ii. if R(s,t) and the atomic signal p is present in s, then there is a node t' in h and a path $t \Rightarrow t'$ in h such that R(s,t') and signal p is present in t'
iii. if R(s,t) and the atomic signal p is present in t, then there is a node s' in g and a path $s \Rightarrow s'$ in g such that R(s',t) and signal p is present in s'
iv. if R(s,t) and $s \xrightarrow{a} s'$ is an edge in g, then either $a=\tau$ and R(s',t), or there are nodes t*, t' in h and a path $t \Rightarrow t^* \xrightarrow{a} t'$ in h such that R(s,t*) and R(s',t')
v. if R(s,t) and $t \xrightarrow{a} t'$ is an edge in h, then either $a=\tau$ and R(s,t'), or there are nodes s*, s' in g and a path $s \Rightarrow s^* \xrightarrow{a} s'$ in g such that R(s*,t) and R(s',t')
vi. if R(s,t) then s is an endnode in g iff t is an endnode in h.

We say graphs g and h are *branching bisimilar*, $g \leftrightarrow_b h$ iff there is an $R: g \leftrightarrow_b h$.
A branching bisimulation R is called *rooted* iff in addition
vii. if $root(g) \xrightarrow{a} s$ is an edge in g ($a \in A \cup \{\tau\}$), then there is a node t in h and an edge $root(h) \xrightarrow{a} t$ in h such that R(s,t)
viii. if $root(h) \xrightarrow{a} t$ is an edge in h ($a \in A \cup \{\tau\}$), then there is a node s in g and an edge $root(g) \xrightarrow{a} s$ in g such that R(s,t).
We say graphs g and h are rooted branching bisimilar , $g \leftrightarrow_{rb} h$ iff there is a rooted $R: g \leftrightarrow_b h$.

Now we claim that the set of process graphs modulo rooted branching bisimulation forms a model for BPAS. In this model, moreover the following τ-law is valid ($a \in A\cup\{\tau\}$). This law is taken from [BW 90]. See also [GW 89].

$a\cdot(\tau\cdot(x + y) + x) = a\cdot(x + y)$	BE

TABLE 37. Branching bisimulation with signals.

We leave as an open problem whether BPAS + BE constitutes a *complete* axiomatization of this model. In order to extend these results to theories with parallel composition, extra laws are not needed, since all laws that hold for atomic actions also hold for the constant τ. Thus, we can use the axiomatization of PAS$^-$ in table 12, ACPS$^-$ in table 14, PAS in tables 16-19 and ACPS in tables 20-21, with in all cases $a,b,c \in A\cup\{\delta,\tau\}$. All we need in addition is an extra requirement on the communication and observation function. In table 38, $a \in A$, $p \in AS$.

$\tau \mid a = \delta$	C4
$\rho(\tau, p) = \delta$	OBS4

TABLE 38. τ with communication and observation function.

9.2 WEAK BISIMULATION.

We give the definition of weak bisimulation in the following.

$R: g \leftrightarrow_\tau h$, relation R between nodes of g and nodes of h is a τ-bisimulation between g and h iff

i. the roots of g and h are related by R

ii. if $R(s,t)$ and the atomic signal p is present in s, then there is a node t' in h and a path $t \Rightarrow t'$ in h such that $R(s,t')$ and p is present in t'

iii. if $R(s,t)$ and the atomic signal p is present in t, then there is a node s' in g and a path $s \Rightarrow s'$ in g such that $R(s',t)$ and p is present in s'

iv. if $R(s,t)$ and $s \xrightarrow{a} s'$ is an edge in g ($a \neq \tau$), then there is a node t' in h and a path $t \Rightarrow \xrightarrow{a} \Rightarrow t'$ in h such that $R(s',t')$

v. if $R(s,t)$ and $t \xrightarrow{a} t'$ is an edge in h ($a \neq \tau$), then there is a node s' in g and a path $s \Rightarrow \xrightarrow{a} \Rightarrow s'$ in g such that $R(s',t')$

vi. if $R(s,t)$ and $s \xrightarrow{\tau} s'$ is an edge in g, then there is a node t' in h and a path $t \Rightarrow t'$ in h such that $R(s',t')$

vii. if $R(s,t)$ and $t \xrightarrow{\tau} t'$ is an edge in h, then there is a node s' in g and a path $s \Rightarrow s'$ in g such that $R(s',t')$

viii. if $R(s,t)$ then s is an endnode in g iff t is an endnode in h.

We say graphs g and h are τ-bisimilar, $g \leftrightarrow_\tau h$ iff there is an $R: g \leftrightarrow_\tau h$.

A τ-bisimulation R is called *rooted* iff in addition

ix. if $root(g) \xrightarrow{\tau} s$ is an edge in g, then there is a node t in h and a path $root(h) \xrightarrow{\tau} \Rightarrow t$ in h such that $R(s,t)$

x. if $root(h) \xrightarrow{\tau} t$ is an edge in h, then there is a node s in g and a path $root(g) \xrightarrow{\tau} \Rightarrow s$ in g such that $R(s,t)$.

We say graphs g and h are weakly bisimilar or rooted τ-bisimilar, $g \leftrightarrow_{r\tau} h$ iff there is a rooted $R: g \leftrightarrow_\tau h$.

Now we claim that the set of process graphs modulo rooted τ-bisimulation forms a model for BPAS. In this model, moreover the following three τ-laws are valid ($a \in A$). They are variants of the τ-laws of Milner [MI 80].

$u \subseteq S(x) \Rightarrow a\cdot[u, \tau]\cdot x = a\cdot x$	TO1
$u \subseteq S(x) \Rightarrow \tau\cdot x + x = [u, \tau\cdot x]$	TO2
$a\cdot(\tau\cdot x + y) = a\cdot(\tau\cdot x + y) + a\cdot x$	TO3

TABLE 39. Observational equivalence with signals ($a \neq \tau$).

We leave as an open problem whether BPAS + TO1-3 constitutes a *complete* axiomatization of this model.

Unfortunately, two of the earlier laws are not valid anymore, viz. the laws S3 and P3 of table 12 in 2.5. They have to be replaced by the following laws ($a \in A$).

$S(ax) = \varnothing$	S3*
$S(\tau x) = S(x)$	TS
$P(ax) = ax$	P3*
$P(\tau x) = \tau\cdot P(x)$	TP

TABLE 40. τ and root signals.

In order to extend these results to theories with parallel composition, we need extra laws, as not all laws for atomic actions will hold for τ. In order to obtain PAS$^-$ with internal action, we have the following obvious laws (see also [BK 85]).

$\tau \mathbin{\lfloor\!\lfloor} x = \tau\cdot x$	TM1
$\tau x \mathbin{\lfloor\!\lfloor} y = \tau\cdot(x \parallel y)$	TM2
$\langle\tau, u\rangle \mathbin{\lfloor\!\lfloor} x = \tau\cdot(u \cup x)$	TMO

TABLE 41. τ and free merge.

Adding communication and encapsulation necessitates the following laws (see also [BK 85]).

$\tau \mid x = \delta$	TC1
$x \mid \tau = \delta$	TC2
$\tau x \mid y = x \mid y$	TC3
$x \mid \tau y = x \mid y$	TC4
$\langle\tau, u\rangle \mid x = \delta$	TCO1
$x \mid \langle\tau, u\rangle = \delta$	TCO2
$\partial_H(\tau) = \tau$	D0

TABLE 42. τ and communication.

Finally, adding observations leads to the following additional axioms.

$\tau/u = \delta$	OT1
$\tau x/u = x/u$	OT2

TABLE 43. τ and observation.

10. EXAMPLES.

In this section we discuss a number of examples of the use of signals and observations.

10.1 QUEUE.

A specification of a (FIFO) queue can be given as follows.

We have a given finite data set D, and the following specification has variables indexed by sequences over D. ε is the empty sequence, and concatenation of a sequence and an element is denoted by $*$. Signals are empty, nonempty.

$$Q_\varepsilon = [\{\text{empty}\}, \sum_{d\in D} \text{enqueue}(d)\cdot Q_d]$$
$$Q_{\sigma*d} = [\{\text{nonempty}\}, \text{dequeue}(d)]\cdot Q_\sigma + \sum_{e\in D} \text{enqueue}(e)\cdot Q_{e*\sigma*d}\ .$$

10.2 BAG.

The bag is indexed by multi-sets. Signals are again empty, nonempty.

$$B_\varnothing = [\{\text{empty}\}, \sum_{d\in D} \text{inbag}(d)\cdot B_{\{d\}}]$$
$$V\neq\varnothing \Rightarrow B_V = [\{\text{nonempty}\}, \sum_{d\in D} \text{outbag}(d)\cdot B_{V-\{d\}}] + \sum_{d\in D} \text{inbag}(d)\cdot B_{V\cup\{d\}}.$$

10.3 STACK.

We use conventions as above. We give a number of alternatives. First a stack without signals.

$$S^1_\varepsilon = \sum_{d\in D} \text{push}(d)\cdot S^1_d$$
$$S^1_{\sigma*d} = \text{pop}\cdot S^1_\sigma + \text{top}(d)\cdot S^1_\sigma + \sum_{e\in D} \text{push}(e)\cdot S^1_{\sigma*d*e}.$$

Next, we add a signal showing the top of the stack.

$$S^2_\varepsilon = \sum_{d\in D} \text{push}(d)\cdot S^2_d$$
$$S^2_{\sigma*d} = \text{pop}\cdot S^2_\sigma + \text{top}\cdot[\{\text{show}(d)\}, S^2_{\sigma*d}] + \sum_{e\in D} \text{push}(e)\cdot S^2_{\sigma*d*e}.$$

In the third specification, we add signals empty, nonempty, and also allow actions top, pop in case the stack is empty. If this happens, an error signal is emitted, and no further action is possible.

$$S^3_\varepsilon = [\{\text{empty}\}, \sum_{d\in D} \text{push}(d)\cdot S^3_d] + \langle\text{top}, \{\text{error}\}\rangle + \langle\text{pop}, \{\text{error}\}\rangle$$
$$S^3_{\sigma*d} = \text{pop}\cdot S^3_\sigma + \text{top}\cdot[\{\text{show}(d)\}, S^3_{\sigma*d}] + [\{\text{nonempty}\}, \sum_{e\in D} \text{push}(e)\cdot S^3_{\sigma*d*e}].$$

The fourth specification has a state of underflow, when an empty stack is popped. A subsequent push leads out of the error situation.

$$S^4_\varepsilon = [\{\text{empty}\}, \sum_{d\in D} \text{push}(d)\cdot S^4_d] + \langle\text{top}, \{\text{error}\}\rangle + \text{pop}\cdot U^4$$

$$U^4 = [\{\text{underflow}\}, \sum_{d\in D} \text{push}(d)\cdot S^4_\varepsilon] + \langle\text{top}, \{\text{error}\}\rangle + \text{pop}\cdot U^4$$

$$S^4_{\sigma*d} = \text{pop}\cdot S^4_\sigma + \text{top}\cdot[\{\text{show}(d)\}, S^4_{\sigma*d}] + [\{\text{nonempty}\}, \sum_{e\in D} \text{push}(e)\cdot S^4_{\sigma*d*e}].$$

The fifth stack keeps functioning, when a pop or top is executed on an empty stack.

$$S^5_\varepsilon = [\{\text{empty}\}, \sum_{d\in D} \text{push}(d)\cdot S^5_d] + \text{top}\cdot[\{\text{show}(\perp)\}, S^5_\varepsilon] + \text{pop}\cdot[\{\text{error}\}, S^5_\varepsilon]$$

$$S^5_{\sigma*d} = \text{pop}\cdot S^5_\sigma + \text{top}\cdot[\{\text{show}(d)\}, S^5_{\sigma*d}] + [\{\text{nonempty}\}, \sum_{e\in D} \text{push}(e)\cdot S^5_{\sigma*d*e}].$$

In the sixth specification, a pop or top executed on an empty stack leads to an irrecoverable error state, but actions can still be executed.

$$S^6_\varepsilon = [\{\text{empty}\}, \sum_{d\in D} \text{push}(d)\cdot S^6_d] + \text{top}\cdot\text{ERROR} + \text{pop}\cdot\text{ERROR}$$

$$\text{ERROR} = [\{\text{error}\}, (\text{top} + \text{pop} + \sum_{d\in D}\text{push}(d))]\cdot\text{ERROR}$$

$$S^6_{\sigma*d} = \text{pop}\cdot S^6_\sigma + \text{top}\cdot[\{\text{show}(d)\}, S^6_{\sigma*d}] + [\{\text{nonempty}\}, \sum_{e\in D} \text{push}(e)\cdot S^6_{\sigma*d*e}].$$

10.4 COMMUNICATING BUFFERS.

In this example we will give an example where both observation and communication play a role. We consider a one element buffers, that always signals its contents on the output port. It receives a communication when the contents are read out. On the input port, it tries to read an item. When it has succeeded in doing this, it sends a communication acknowledging this. The buffer B^{ij} has input port i and output port j. The signal ${}^jd$ means that message d is offered at port j.

$$B^{ij} = [\{{}^j\perp\}, \delta] + \text{obs}({}^i\perp)\cdot B^{ij} + \sum_{d\in D} \text{obs}({}^id)\cdot[\{{}^jd\}, s_i(\text{ack})]\cdot B^{ij}_d$$

$$B^{ij}_d = [\{{}^jd\}, \delta] + r_j(\text{ack})\cdot B^{ij}.$$

Now we connect two buffers together. We have the communication $r_2(\text{ack}) \mid s_2(\text{ack}) = c_2(\text{ack})$, and we use the signal observation function of section 4.3.ii. We define

$$X = \partial_H(B^{12} \| B^{23}),$$

where the encapsulation set is $H = \{\text{obs}({}^2\perp), r_2(\text{ack}), s_2(\text{ack})\} \cup \{\text{obs}({}^2d) : d \in D\}$.
Some calculations result in the following recursive specification:

$$X = [\{{}^2\perp, {}^3\perp\}, \delta] + \text{obs}({}^1\perp)\cdot X + \sum_{d\in D} \text{obs}({}^1d)\cdot X^d_1$$

$$X^d_1 = [\{{}^2d, {}^3\perp\}, \delta] + s_1(\text{ack})\cdot X^d_2 + \text{yes}({}^2d)\cdot[\{{}^2d, {}^3d\}, s_1(\text{ack})]\cdot X^d_3$$

$X_2^d = [\{^2d, {}^3\bot\}, yes(^2d)]\cdot X_3^d$

$X_3^d = [\{^2d, {}^3d\}, c_2(ack)]\cdot X_4^d$

$X_4^d = [\{^2\bot, {}^3d\}, \delta] + r_3(ack)\cdot X + obs(^1\bot)\cdot X_4^d + \sum_{e \in D} obs(^1e)\cdot X_5^{de}$

$X_5^{de} = [\{^2e, {}^3d\}, \delta] + r_3(ack)\cdot X_1^e + s_1(ack)]\cdot[\{^2e, {}^3d\}, r_3(ack)]\cdot X_2^e.$

Let us now abstract from the set of actions $I = \{obs(^1\bot), s_1(ack), c_2(ack)\} \cup \{yes(^2d) : d \in D\}$. This leaves only the input actions $obs(^1d)$ and the output action $r_3(ack)$. We get the following specification for $Y = \tau_I(X)$:

$Y = \tau_I(X) = [\{^2\bot, {}^3\bot\}, \delta] + \sum_{d \in D} obs(^1d)\cdot Y_1^d$

$Y_1^d = \tau_I(X_1^d) = \tau_I(X_2^d) = [\{^2d, {}^3\bot\}, \tau]\cdot Y_3^d$

$Y_3^d = \tau_I(X_3^d) = [\{^2d, {}^3d\}, \tau]\cdot Y_4^d$

$Y_4^d = \tau_I(X_4^d) = [\{^2\bot, {}^3d\}, \delta] + r_3(ack)\cdot Y + \sum_{e \in D} obs(^1e)\cdot Y_5^{de}$

$Y_5^{de} = \tau_I(X_5^{de}) = [\{^2e, {}^3d\}, \delta] + r_3(ack)\cdot Y_1^e.$

Having hidden the actions at port 2, we can proceed by also hiding the signals at port 2. We do this by means of the signal filtering operator of 2.4. We put $u = \{^3d : d \in D\} \cup \{^3\bot\}$, and derive the following specification for $Z = u \cap Y$:

$Z = u \cap Y = u \cap \tau_I(X) = [\{^3\bot\}, \delta] + \sum_{d \in D} obs(^1d)\cdot Z_1^d$

$Z_1^d = u \cap Y_1^d = [\{^3\bot\}, \tau]\cdot Z_3^d$

$Z_3^d = u \cap Y_3^d = u \cap Y_4^d = [\{^3d\}, \delta] + r_3(ack)\cdot Z + \sum_{e \in D} obs(^1e)\cdot Z_5^{de}$

$Z_5^{de} = u \cap Y_5^{de} = [\{^3d\}, \delta] + r_3(ack)\cdot Z_1^e.$

ACKNOWLEDGEMENTS.
We thank Wouter Brouwer for many comments and improvements. Frits Vaandrager pointed out an error in a previous version of equation MSI2 (section 3.3) as well as a difficulty in a first version of the observation function. Henk Goeman, Sjouke Mauw, Hans Mulder and Gert Veltink also contributed by suggesting improvements. Our treatment of conditionals benefitted from correspondence with Tony Hoare.

REFERENCES.

[AB 84] G. AUSTRY & G. BOUDOL, *Algèbre de processus et synchronisation,* TCS 30, 1984, pp. 91-131.

[BB 88] J.C.M. BAETEN & J.A. BERGSTRA, *Global renaming operators in concrete process algebra,* Information & Computation 78 (3), 1988, pp. 205-245.

[BBK 86] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP, *Syntax and defining equations for an interrupt mechanism in process algebra,* Fundamenta Informaticae IX (2), 1986, pp. 127-168.

[BBMV 89] J.C.M. BAETEN, J.A. BERGSTRA, S. MAUW & G.J. VELTINK, *A process specification formalism based on static COLD,* report CS-R8930, CWI Amsterdam 1989. To appear in Proc. METEOR Workshop Methods based on Formal Specifications, Mierlo 1989 (L. Feijs & J.A. Bergstra, eds.), Springer LNCS.

[BW90] J.C.M. BAETEN & W.P. WEIJLAND, *Process algebra,* Cambridge Tracts in TCS 18, Cambridge University Press 1990.

[BE 88a] J.A. BERGSTRA, *Process algebra for synchronous communication and observation,* Report P8815, Programming Research Group, University of Amsterdam 1988. *Revised version,* report P8815b, 1989.

[BE 88b] J.A. BERGSTRA, *ACP with signals,* in: Algebraic and Logic Programming (J. Grabowski, P. Lescanne & W. Wechler, eds.), Springer LNCS 343, 1988, pp. 11-20.

[BK 84] J.A. BERGSTRA & J.W. KLOP, *Process algebra for synchronous communication,* Information and Control 60 (1/3), 1984, pp. 109-137.

[BK 85] J.A. BERGSTRA & J.W. KLOP, *Algebra of communicating processes with abstraction,* Theoretical Computer Science 37 (1), 1985, pp. 77-121.

[BR 90] W.S. BROUWER, *Stable signals and observation in a process specification formalism,* M.Sc. thesis, University of Amsterdam 1990.

[VGL 87] R.J. VAN GLABBEEK, *Bounded nondeterminism and the approximation induction principle in process algebra,* in: Proc. STACS 87 (F.J. Brandenburg, G. Vidal-Naquet & M. Wirsing, eds.), LNCS 247, Springer Verlag 1987, pp. 336-347.

[GW 89] R.J. VAN GLABBEEK & W.P. WEIJLAND, *Branching time and abstraction in bisimulation semantics. Extended abstract,* in: Information Processing 89, IFIP World Congress, San Francisco (G.X. Ritter, ed.), North-Holland, Amsterdam 1989, pp. 613-618.

[GP 90] J.F. GROOTE & A. PONSE, *Process algebra with guards,* report CS-R9069, CWI Amsterdam 1990.

[HE 88] M. HENNESSY, *Algebraic theory of processes,* MIT Press 1988.

[HO 85] C.A.R. HOARE, *Communicating sequential processes,* Prentice Hall 1985.

[HHJ+ 87] C.A.R. HOARE, I.J. HAYES, HE JIFENG, C.C. MORGAN, A.W. ROSCOE, J.W. SANDERS, I.H. SORENSEN, J.M. SPIVEY & B.A. SUFRIN, *Laws of programming,* Comm. of the ACM 30 (8), 1987, pp. 672-686.

[MV 90] S. MAUW & G.J. VELTINK, *A process specification formalism,* Fund. Inf. XII, 1990, pp. 85-139.

[MI 80] R. MILNER, *A calculus of communicating systems,* Springer LNCS 92, 1980.

[MI 89] R. MILNER, *Communication and concurrency,* Prentice Hall 1989.

[VA 90] F.W. VAANDRAGER, *Process algebra semantics for POOL,* in: Applications of Process Algebra (J.C.M. Baeten, ed.), Cambridge Tracts in TCS 17, Cambridge University Press 1990, pp. 173-236.

[VE 90] C. VERHOEF, *On the register operator,* report P9003, Programming Research Group, University of Amsterdam 1990.

[VR 86] J.L.M. VRANCKEN, *The algebra of communicating processes with empty process,* Report FVI 86-01, Programming Research Group, University of Amsterdam 1986.