# Algebra of Communicating Processes

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

A survey of process algebra is presented including the following features: merging processes without communication, merging processes with communication, data flow networks, regular processes, recursively defined processes, abstraction mechanisms both in absence and presence of communication. Throughout the paper emphasis is on equational specifications and graph theoretic models.

## Introduction

It is widely recognized that Milner's CCS constitutes a fundamental contribution to the theory of concurrency. Milner's aim is to establish laws for concurrent processes in the form of algebraic identities. We view process algebra, as developed in [3],...,[10], as a rephrasing of the basic issues of CCS. For a motivation of CCS as a theory of concurrency we refer to MILNER [14], [15]. We will not assume that the reader knows CCS. The differences with CCS in aims and techniques can be summarized as follows:

(1) We use these operators and constants:

| | |
|---|---|
| $+$ | alternative composition (sum) |
| $\cdot$ | sequential composition (product) |
| $\parallel$ | parallel composition (merge) |
| $\lfloor\!\lfloor$ | left merge |
| $\mid$ | communication merge |
| $\partial_H$ | encapsulation |
| $\tau_I$ | abstraction |
| $\delta$ | deadlock (failure) |
| $\tau$ | silent (internal) action |

TABLE 1

We will briefly discuss how these operators relate to CCS. The operators $+$, $\parallel$, and $\tau$ have exactly the same meaning; multiplication $\cdot$ is more general than the prefix multiplication of CCS; $\mathbin{\|\!\_}$ and $|$ are new; $\delta$ is similar to NIL in sums (but not in products). $\partial_H$ and $\tau_I$ are new operators. (However these are formally renaming operators in the sense of CCS.)

(2) This choice of operators allows a finite initial algebra specification of the behaviour of finite processes. Seen from CCS, $\mathbin{\|\!\_}$ and $|$ are hidden operators involved in this specification. We feel however that $\mathbin{\|\!\_}$ and $|$ are perfectly meaningful from an intuitive point of view.
Our presentation culminates in a system of equations $ACP_\tau$, and passes through several smaller specifications ($PA$, $PA_\tau, ACP$) involving only a subset of the operators.

(3) $ACP$ chooses from the onset the axiomatic approach. Thus, where CCS starts with a model of processes and derives identities in that model as theorems, $ACP$ reverses this procedure: a set of axioms is given first and its models are investigated next. In the course of our investigations we have met some twenty interesting process algebras (interesting as opposed to pathological; the axiomatic approach allows also some less useful models) and since there are so many it seems sensible to organize them as models of some axiomatic theory.

(4) We claim that $ACP$ is more amenable to a mathematical analysis than CCS (in its original form). As an example we would like to point out the simple formulation of the Expansion Theorem (2.2), and the specification of a Stack in subsection 3.5.
The core of this presentation is the system $ACP$. Infinite models for $ACP$ are constructed as projective limits of finite models, and as graph models modulo bisimulation. The projective limit models have been derived from the topological construction in DE BAKKER and ZUCKER [1], [2]. The work on process algebra originated from a problem in [2] (page 87) which was solved in [3] thereby essentially using the algebraic properties of $\mathbin{\|\!\_}$. (See 1.9 below.)

ACKNOWLEDGEMENTS

The material in the first three of the four sections of this paper was presented in the *Workshop on Concurrency and Abstract Data Types* (Mook, October 1983) organized by W.P. DE ROEVER. We thank him for giving us the opportunity to organize the present material in a set of lecture notes which was the basis for this paper.

Furthermore, we thank J.W. DE BAKKER for his continuous support and J.-J.CH. MEYER, J.V. TUCKER, J. TIURYN, E. BRINKSMA, C.J. KOOMEN, H. JONKERS and H. OBBINK for many discussions on the subject of this paper.

Most of the following material has been covered in more detail in our reports [3],...,[10]. Section 4 contains new results, centering around $ACP_\tau$, an axiom system for communicating processes with internal steps. Almost all proofs are omitted - these can be found in the above mentioned reports (except for most of Section 4).

The structure of this paper is as follows:

## 1. Process algebra: *PA*

In this section we will introduce the axiom systems *PA* for process algebra without communication (treated in Section 2) and without internal steps (treated in Section 4). The co-operation between processes described by *PA* is that of *interleaving*. As semantics for *PA* several 'process algebras' will be introduced of which the simplest one is the initial algebra of *PA*.

### *1.1. The axiom system PA*

The axiom system *PA* consists of the following list of axioms:

| | |
|---|---|
| $x+y=y+x$ | 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 |
| $x\|y=x\mathbin{\|\!\!\lfloor}y+y\mathbin{\|\!\!\lfloor}x$ | M1 |
| $a\mathbin{\|\!\!\lfloor}x=a\cdot x$ | M2 |
| $ax\mathbin{\|\!\!\lfloor}y=a(x\|y)$ | M3 |
| $(x+y)\mathbin{\|\!\!\lfloor}z=x\mathbin{\|\!\!\lfloor}z+y\mathbin{\|\!\!\lfloor}z$ | M4 |

Table 2

*1.1.1. The signature of PA.* The signature of *PA* consists of the following ingredients:

(i) $a,b,c, \ldots \in A$, the set of *axiomatic actions* (also called 'steps' or 'events'). $A$ is also referred to as the *alphabet*. Throughout this paper, we will assume that $A$ is *finite*. (This is done to safeguard the algebraic nature of our considerations — e.g. infinite sums of processes are not considered here.) In the axioms of *PA*, '$a$' varies over $A$.

(ii) $x,y,z, \ldots$ are *variables*, ranging over the domains of processes (process algebras) which will be constructed below.

(iii) binary *operators*. These are:

| | |
|---|---|
| $+$ | alternative composition, or sum |
| $\cdot$ | sequential composition, or product |
| $\|$ | parallel composition, or merge |
| $\mathbin{\|\!\!\lfloor}$ | left-merge. |

The 'main' operators are $+, \cdot, \|$. Left-merge $\lfloor\!\lfloor$ is an auxiliary operator.

*1.1.2. Process expressions.* Process expressions or process terms are built from the $a \in A$ by means of $+, \cdot, |, \lfloor\!\lfloor$. Examples of process expressions are:

$$(a+b), \quad ((((a \cdot a) \lfloor\!\lfloor b) + (c \cdot d)) \cdot e).$$

The following notational conventions will be employed: $xy$ stands for $x \cdot y$; outermost brackets are omitted; the operator $\cdot$
has the greatest binding power; $x^n$ stands for $xx \ldots x$ ($n$ times); $\|$ and $\lfloor\!\lfloor$ bind stronger than $+$. So the two process expressions above may be written as

$$a + b, \quad (a^2 \lfloor\!\lfloor b + cd)e.$$

*1.2. Semantics of PA*

A *process algebra* is a domain of processes satisfying the axioms of *PA*. The three most important process algebras for *PA* are:

(1) $A_\omega$, *the initial algebra of PA,*

(2) $\mathbf{A}^\infty$, *the graph model of PA,*

(3) $A^\infty$, *the standard model of PA*

It will turn out that these algebras properly extend each other (modulo isomorphism): $A_\omega \subsetneq \mathbf{A}^\infty \subsetneq A^\infty$.

*1.2.1. The initial algebra $A_\omega$.* The elements of $A_\omega$ are the *process expressions modulo the equivalence given by PA.* So, in $A_\omega$, '$a+b$' and '$b+a$' and '$a+b+a$' are the same. Likewise, the process expressions $((aa \lfloor\!\lfloor b+cd)e$ and $a(abe+bae)+cde$ denote the same element in $A_\omega$, since using *PA* one computes

$$(aa \lfloor\!\lfloor b+cd)e = (aa \lfloor\!\lfloor b)e+cde = a(a \| b)e+cde =$$

$$a(a \lfloor\!\lfloor b+b \lfloor\!\lfloor a)e+cde = a(ab+ba)e+cde =$$

$$a(abe+bae)+cde.$$

Note that this derivation has eliminated the $\|$, $\lfloor\!\lfloor$ operators in the original process term. We have the following general fact:

THEOREM 1.1.

(i) *Using the axioms of PA as rewrite rules from left to right, every process expression can be rewritten to an expression without $\|$ or $\lfloor\!\lfloor$.*

(ii) *If $PA \vdash t_1 = t_2$ and $t_1, t_2$ do not contain $\|$, $\lfloor\!\lfloor$, then $A1\text{-}5 \vdash t_1 = t_2$.*

This entails that elements of the initial algebra $A_\omega$ can be thought of as process expressions built from atoms via $+$ and$\cdot$only, modulo A1-5. Using this fact we arrive at a convenient representation of elements of $A_\omega$:

PROPOSITION 1.1. *Modulo PA-equivalence, $A_\omega$ is inductively generated as follows:*

$$x_i \in A_\omega,\quad a_i \in A\ (i=1,\dots,n),\quad b_j \in A(j=1,\dots,m) \Rightarrow \sum_{j=1}^{m} b_j + \sum_{i=1}^{n} a_i x_i \in A_\omega.$$

EXAMPLE 1.1.

$$\begin{aligned}
&bab\|ab = bab \mathbin{\lfloor\!\lfloor} ab + ab \mathbin{\lfloor\!\lfloor} bab = b(ab\|ab) + a(b\|bab) = \\
&b(ab \mathbin{\lfloor\!\lfloor} ab + ab \mathbin{\lfloor\!\lfloor} ab) + a(b \mathbin{\lfloor\!\lfloor} bab + bab \mathbin{\lfloor\!\lfloor} b) = \\
&b(ab \mathbin{\lfloor\!\lfloor} ab) + a(bbab + b(ab\|b)) = \\
&b(a(b\|ab)) + a(bbab + (b(ab \mathbin{\lfloor\!\lfloor} b + b \mathbin{\lfloor\!\lfloor} ab)) = \\
&b(a(bab+abb)) + a(bbab + b(abb+bab)).
\end{aligned}$$

Expressions like the last one, without $\|$ and $\lfloor\!\lfloor$, can conveniently be 'pictured' as finite trees:

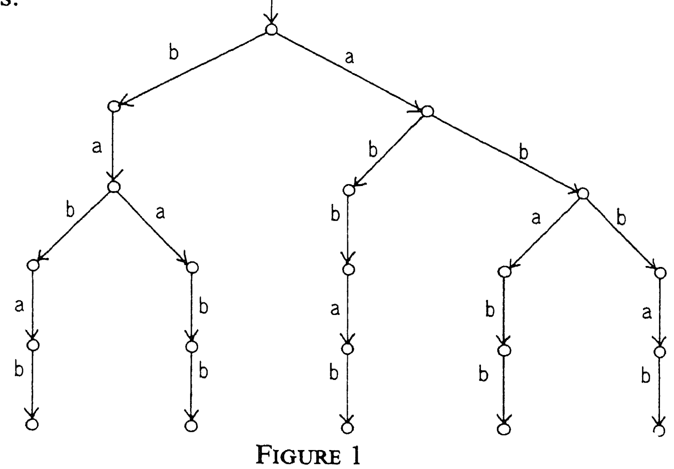


FIGURE 1

Let us note here (in advance to the definition of $\|$ for process graphs later on in this section) that the tree above, resulting from the interleaving of *bab* and *ab*, can be obtained quickly by 'unraveling' the cartesian product graph

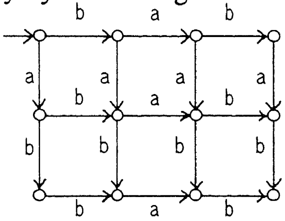


FIGURE 2

Vice versa, the above tree yields this product graph by identifying some nodes with identical subtrees. We will return to such process trees and graphs later.

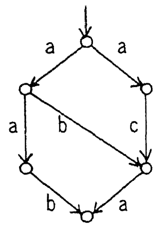
Note that *PA* does not contain the distributive law $x(y+z) = xy+xz$.

Indeed, for pairwise different atoms $a,b,c$ the processes $a(b+c)$ and $ab+ac$ are different in $A_\omega$. (Cf. also Example 2.2.)

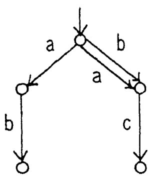
We have now constructed our first process algebra as semantics of $PA$, the initial algebra $A_\omega$, whose elements can also be thought of as *finitely branching and finitely deep process trees.* The fact that the processes in $A_\omega$ are only finitely deep, means that we cannot find solutions $p$ in $A_\omega$ for recursive definitions like $p = ap$; for, $p$ would be $aaaa$ ... or $a^\omega$.

Therefore we will now construct process algebras which do have infinite elements, and in which solutions of recursion equations can be found.

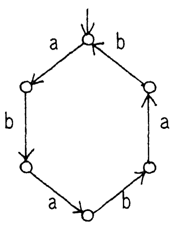
*1.2.2. The process graph model* $\mathbb{A}^\infty$. A *process graph* (also called: *transition diagram*) over a set of atoms $A$ is a *rooted, directed multigraph* whose edges are labeled by elements of $A$. Process graphs may be infinite and may contain cycles. *Process trees* are special cases: they are acyclic process graphs where no subgraph is shared (and containing no multiple edges); in other words, where no two edges have the same end-point. Some examples will clarify these concepts.



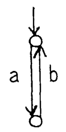
(i)



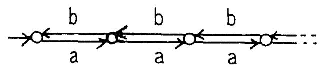
(ii)



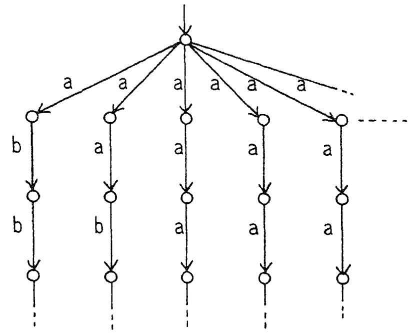
(iii)



(iv)



(v)



(vi)

FIGURE 3

Here (i), (ii) are finite acyclic process graphs, but not trees; (iii), (iv) are finite process graphs containing cycles; (v) is an infinite process graph containing cycles and (vi) is an infinite process tree.

To construct our second process algebra $\mathbf{A}^{\infty}$, we will restrict ourselves to *finitely branching* process graphs. (This also puts a bound on the cardinality of the edges and nodes of such graphs.)

Having this large collection of finitely branching process graphs available, we note that there are 'too many' of them — some process graphs should be identified. E.g. the five graphs in figure 4 all seem to denote the same process: in each node ('state of the process') there are in all five cases infinitely many $a$-steps possible.

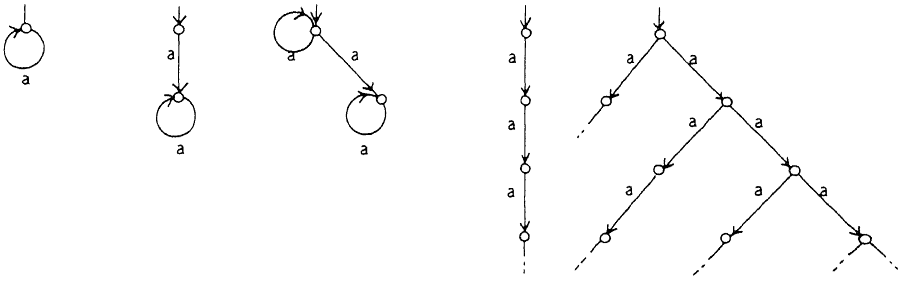


FIGURE 4

An elegant notion, introduced in PARK [16], called *bisimulation*, does indeed identify these graphs.

*1.2.2.1. Bisimulation of process graphs.* Bisimulation of process graphs is defined as follows.

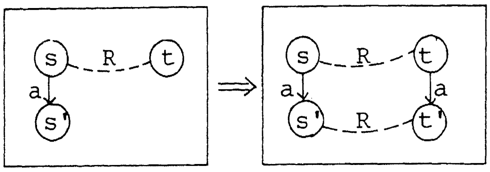
Let $g_1,g_2$ be process graphs with node sets Nodes $(g_1)$, Nodes $(g_2)$. Let $s_0,t_0$ be the roots of $g_1,g_2$ respectively. Then $g_1,g_2$ *are bisimilar,* in symbols:

$$g_1 \underline{\leftrightarrow}\, g_2$$

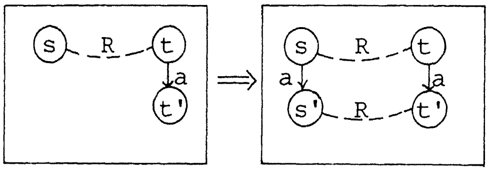
if there is a relation $R \subseteq$ Nodes $(g_1) \times$ Nodes $(g_2)$ such that

(i) $s_0Rt_0$ (the roots are related)

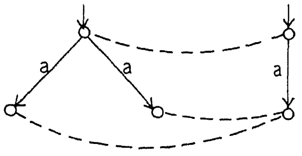
(ii) if $s \xrightarrow{a} s'$ is an edge of $g_1$ and $sRt$, there must be an edge $t \xrightarrow{a} t'$ of $g_2$ such that $s'Rt'$. In a diagram:



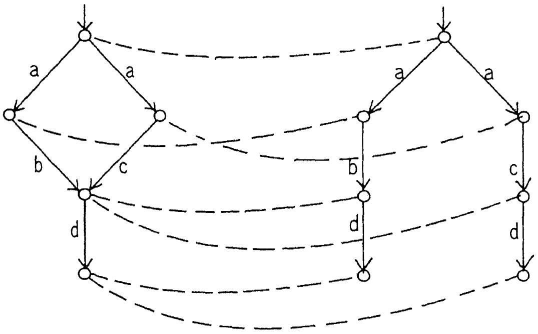
(iii) vice versa (with the role of $g_1,g_2$ interchanged):



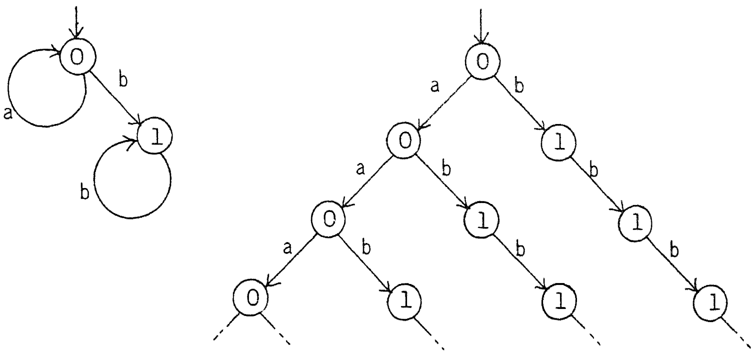
EXAMPLES.



(i)



(ii)



(iii)

(In figure 5 (iii) the bisimulation is given by the numbering of the nodes.)

(iv) A non-example:

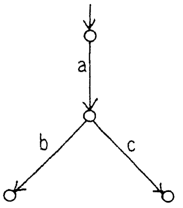
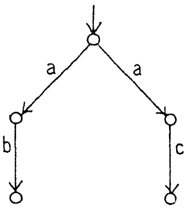
 and  are *not* bisimilar.

FIGURE 5

(Cf. our earlier remark that $A_\omega \not\models a(b+c) = ab+ac$.) Note that *unfolding* (or unwinding) a process graph respects bisimilarity. The same holds for *sharing* (identifying nodes with identical subgraphs).

We call the process graph with one node and no edges, the *trivial* graph. A node lying on a cycle is a *cyclic* node.

Now the second process algebra for *PA*, called the process graph algebra $\mathbf{A}^\infty$, is defined as follows.

The elements of $\mathbf{A}^\infty$ are the *finitely branching, nontrivial process graphs with acyclic roots modulo bisimulation.*

*The operations* $+, \cdot, \|, ⫝$ *on* $\mathbb{A}^\infty$ *are defined thus:*

(i) The *sum* $g_1+g_2$ is obtained by identifying the roots of $g_1, g_2$. E.g.:

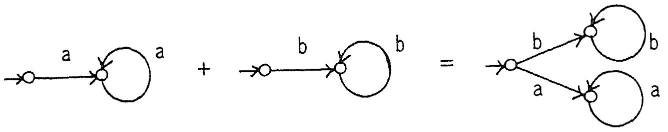


FIGURE 6

This example indicates why the roots have to be acyclic: otherwise

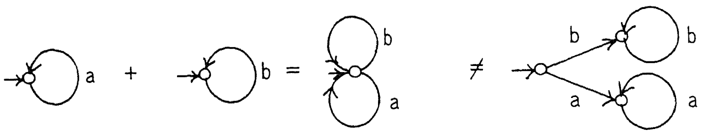


FIGURE 7

(ii) The *product* $g_1 \cdot g_2$ is obtained by appending $g_2$ to all end nodes of $g_1$.

(iii) The *merge* $g_1 \| g_2$ is the cartesian product graph as in the example:

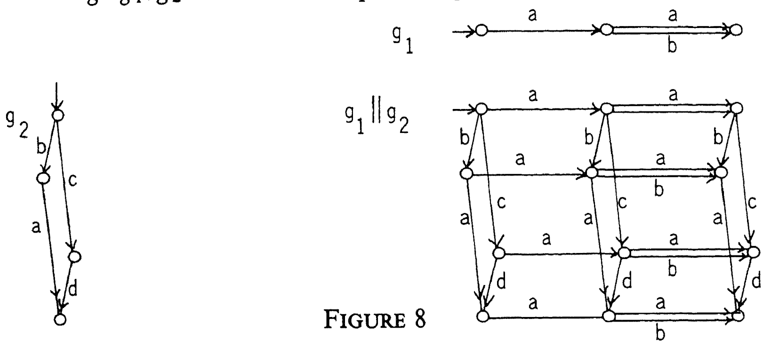


FIGURE 8

(iv) The *left-merge* $g_1 ⫝ g_2$ is obtained as a subgraph or $g_1 \| g_2$ as in the example:

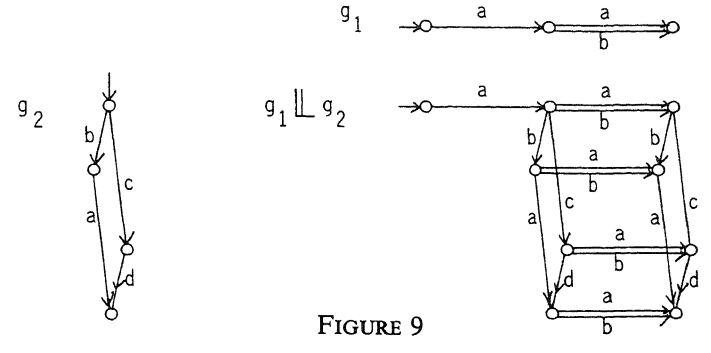


FIGURE 9

It is now easy to prove the following theorem.

THEOREM 1.2.
(i) $\mathbb{A}^\infty$ *is a process algebra (a model of PA).*
(ii) *The finite acyclic elements in* $\mathbb{A}^\infty$ *constitute a subalgebra which is (isomorphic to)* $A_\omega$.

*1.2.2.2. Approximations of processes in* $\mathbb{A}^\infty$. There is a clear sense in which a (possibly infinite) process tree $t$ can be approximated by finite process trees $(t)_n$ $(n \geqslant 1)$: $(t)_n$ is $t$ where everything below level $n$ is cut-off. (I.e. the branches of $(t)_n$ have at most $n$ steps.) This notion of projection induces one in $\mathbb{A}^\infty$ in a straightforward manner: writing $[\![g]\!]$ for the bisimulation equivalence class of the process graph $g$ (so $[\![g]\!] \in \mathbb{A}^\infty$, if $g$ is nontrivial etc.), we define

$$([\![g]\!])_n = [\![(tree(g))_n]\!]$$

where $tree(g)$ is a tree obtained by unwinding $g$. To establish the precise definition of $tree(g)$ and the well-definedness of the projection operation $(\ )_n$: $\mathbb{A}^\infty \rightarrow \mathbb{A}^\infty$ is a matter of routine. From now on, we write simply $g$ instead of $[\![g]\!]$ when dealing with elements of $\mathbb{A}^\infty$. There is the following interesting fact:

THEOREM 1.3. *Let* $g,h \in \mathbb{A}^\infty$. *Then* $g = h \Leftrightarrow \forall n \ (g)_n = (h)_n$.

So equality between finitely branching graphs (modulo bisimulation) is entirely determined by their finite approximations — i.o.w. a finitely branching graph modulo bisimulation is determined by its finite approximations.

The implication $\Rightarrow$ in this theorem is trivial; the proof of the reverse implication consists of an application of König's Lemma made possible by the condition that elements in $\mathbb{A}^\infty$ are (equivalence classes of) *finitely branching* graphs. (This is used as follows: construct the tree of all bisimulations between $(g)_n$ and $(h)_n$, for all $n \geqslant 1$. That is, on the $n$-th level are the bisimulations between $(g)_n$ and $(h)_n$. Ordering in the tree is: extension of bisimulations in the set-theoretic sense. Since this tree is finitely branching and infinite, it has an infinite branch which yields a bisimulation for the pair $g,h$.) That this condition is essential for the proposition in the theorem, follows from a consideration of these two process graphs which have the same finite approximations:

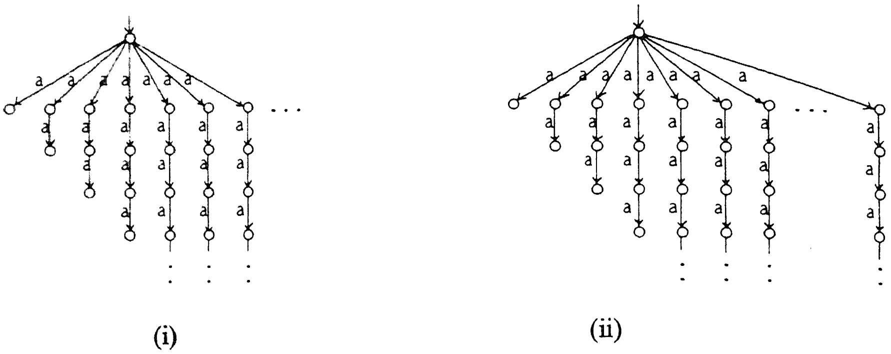


FIGURE 10

Although the elements of $\mathbf{A}^{\infty}$ are attractive objects, they are notoriously lacking in algebraic nature. On the basis of our intuitive understanding of the graph model $\mathbf{A}^{\infty}$, we will now construct a process algebra for *PA* which is algebraic in nature and which will be called the standard model $\mathbf{A}^{\infty}$ for *PA*.

*1.2.3. The standard model $A^{\infty}$ for PA.* Bearing in mind that an element $g \in \mathbf{A}^{\infty}$ gives rise to a sequence $((g)_1,(g)_2, \ldots)$ of approximations which by the previous theorem (1.3) determines $g$ and for which we obviously have: $(g)_n = ((g)_{n+1})_n$, we now define without any reference to graphs: A *projective sequence* is a sequence $(p_1,p_2,p_3, \ldots, p_n, \ldots)$ of elements of $A_{\omega}$ such that $p_n = (p_{n+1})_n$. Here the projections $(\ )_n\colon A_{\omega} \to A_{\omega}$ $(n \geqslant 1)$ are defined by

$$(a)_n = a$$

$$(ax)_1 = a, \quad (ax)_{n+1} = a(x)_n$$

$$(x+y)_n = (x)_n + (y)_n.$$

Furthermore we define: *the elements of $A^{\infty}$ are the projective sequences.* The operations $+, \cdot, \|, \mathbin{|\!\!\lfloor}$ on $A^{\infty}$ are defined coordinate-wise, thus:

$$(p_1,p_2, \ldots, p_n, \ldots) \,\square\, (q_1,q_2, \ldots, q_n, \ldots, ) =$$

$$((p_1 \,\square\, q_1)_1, (p_2 \,\square\, q_2)_2, \ldots, (p_n \,\square\, q_n)_n, \ldots )$$

where $\square \in \{+, \cdot, \|, \mathbin{|\!\!\lfloor}\}$. Note the outermost subscripts in the RHS, necessary to ensure that the result from applying the operation $\square$ is again a projective sequence. (The simple proof employs the fact that $(p \,\square\, q)_n = ((p)_n \,\square\, (q)_n)_n$.)

EXAMPLE 1.2.

(i) The atomic action $'a'$ is represented by $(a,a,a,\ \ldots)$.

(ii) $(a,\ a+a^2,\ a+a^2+a^3,\ \ldots,\ \Sigma_{i=1}^{n}a^i,\ \ldots)\in A^\infty$. We will refer to this element as $\Sigma_{i=1}^{\infty}a^i$. (Note however that except for this ad hoc notation we will not use infinite sums.)

(iii) Call $a^\omega=(a,a^2,a^3,\ \ldots)$. Then $a^\omega\cdot b^\omega=((a\cdot b)_1,$ $(a^2\cdot b^2)_2,\ \ldots)=a^\omega$.

(iv) $a^\omega\|b^\omega=((a\|b)_1,\ (a^2\|b^2)_2,\ \ldots)=(a+b,\ a(a+b)+b(a+b),\ \ldots)$ $=(a+b,\ (a+b)^2,\ \ldots)=(a+b)^\omega$.

(v) $[(a^\omega\|b^\omega)+a^\omega]\|b^\omega=a^\omega\|b^\omega$.

Again it is straightforward to verify that $A^\infty$ is a model of $PA$.

A natural question is how $A^\infty$ and $\mathbf{A}^\infty$ compare. The answer is that $A^\infty$ is an extension of $\mathbf{A}^\infty$: it contains all the processes in $\mathbf{A}^\infty$ (modulo an isomorphism) but also some processes which are not finitely branching, like $\Sigma_{i=1}^{\infty}a^i$ above. Strictly speaking, we have not yet defined when an element of $A^\infty$, a projective sequence, is finitely branching.

This can be done by assigning to a $p\in A^\infty$ a process graph $G(p)$, as follows. First we define what a 'subprocess' of $p\in A^\infty$ is.

*The collection of subprocesses of p* is given by

(i) $p\in Sub(p)$,

(ii) $ax\in Sub(p)\Rightarrow x\in Sub(p)$,

(iii) $ax+y\in Sub(p)\Rightarrow x\in Sub(p)$.

From the subprocesses of $p$ (which may be thought of as the nonterminal states of the process) we can assemble a process graph $G(p)$. This process graph will be called *the canonical process graph* $G(p)$ *for* $p$. It is defined as follows:

(i) the set of nodes of $G(p)$ is $Sub(p)\cup\{\bigcirc\}$,

(ii) the root of $G(p)$ is $p$,

(iii) the edges of $G(p)$ are given by:

  (1) $a\in Sub(p)\Rightarrow a\xrightarrow{a}\bigcirc$ is an edge,

  (2) $ax\in Sub(p)\Rightarrow ax\xrightarrow{a}x$ is an edge,

  (3) $a+y\in Sub(p)\Rightarrow a+y\xrightarrow{a}\bigcirc$ is an edge,

  (4) $ax+y\in Sub(p)\Rightarrow ax+y\xrightarrow{a}x$ is an edge.

(If $p$ has only infinite branches, the termination node $\bigcirc$ can be discarded.) So now the statement that $\Sigma_{i=1}^{\infty}a^i$ $(=(a,a+a^2,\ \ldots))$ is infinitely branching makes sense: it is meant that its canonical process graph is so. In fact, the canonical process graph of $p=\Sigma_{i=1}^{\infty}a^i$ is

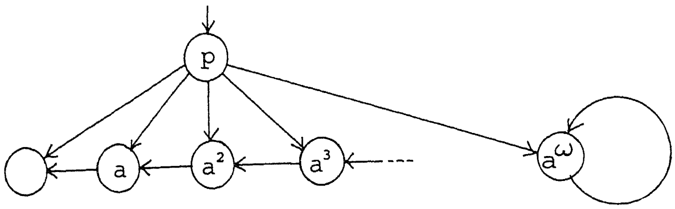


FIGURE 11

which is bisimilar to the process graph in figure 10 (ii).
(Note that $G(p)$ contains the infinite branch $a^{\omega}$; for: $p=p+a^{\omega}=p+a\cdot a^{\omega}$, hence $a^{\omega}\in Sub(p)$.)

We conclude this section about $PA$ with a number of remarks which give some additional information about $PA$ and its models (but which are not strictly necessary for an understanding of the following sections).

*1.3. The cardinality of* $\mathbf{A}^{\infty}$ *and* $A^{\infty}$

The cardinality of $\mathbf{A}^{\infty}$ and $A^{\infty}$ is $2^{\aleph_0}$ for all finite $A$ (as supposed throughout the paper). In contrast, one may consider the following. If there is no condition imposed on the branching degree of process graphs, and $\mathbf{A}^{\infty}$ is constructed as before, then even for a singleton alphabet $A$ the process domain $\mathbf{A}^{\infty}$ would be a proper class in the sense of axiomatic set theory. This shows that in order to obtain a set-sized domain of process graphs modulo bisimulation one has to specify some cardinal as upper bound on the branching degree in advance.

*1.4. The finite process algebras* $A_n$

Some interesting *finite* process algebras (models of $PA$) which were not introduced above, can be obtained as follows. Define $A_n=\{(p)_n|p\in A_{\omega}\}$ and define as operations $\Box_n$ on the finite set $A_n$:

$$x\,\Box_n y = (x\Box y)_n$$

where the $\Box\in\{+,\cdot,\|,\lfloor\!\lfloor\}$ are the operations from $A_{\omega}$. Then: $A_n(+_n,\cdot_n,\|_n,\lfloor\!\lfloor_n)\models PA$. Now $A^{\infty}$ can be defined simply as the *projective limit of the algebras* $A_n$ $(n\geqslant 1)$.

*1.5. Commutativity and associativity of merge*

From the axioms of $PA$, the commutativity of merge follows immediately:

$$x\|y = x\lfloor\!\lfloor y+y\lfloor\!\lfloor x=y\lfloor\!\lfloor x+x\lfloor\!\lfloor y=y\|x.$$

The associativity

$$x\|(y\|z) = (x\|y)\|z$$

does *not* follow from $PA$. (Indeed one can construct a process algebra with nonassociative merge operator.) However, in the process algebras introduced above $(A_{\omega},A_n,\mathbf{A}^{\infty},A^{\infty})$ the associativity does hold. A proof, by induction on the structure of the elements, can be given simultaneously with a proof of the useful identity

$$(x |\!\!\lfloor y) |\!\!\lfloor z \;=\; x |\!\!\lfloor (y \| z).$$

*1.6. Adding a zero process to PA*

One can argue about the desirability of an element 0 in process algebras, with the properties

$$x + 0 \;=\; x$$
$$0x \;=\; x0 = x.$$

Naive addition of such axioms to *PA* yields an 'inconsistency', though. For consider:

$$ab \;=\; (a + 0)b = ab + 0b = ab + b$$

contrary to our intention to distinguish $ab$ from $ab + b$.

However, with the added proviso in axiom A4:

$$(x + y)z \;=\; xz + yz \quad \text{if} \quad x, y \neq 0$$

(and adding $0 |\!\!\lfloor x = 0$, $x |\!\!\lfloor 0 = x$) this inconsistency is removed and we have a conservative extension of PA.

Yet we will not pursue this option, since we have no need for 0. One reason is found in the next remark, another reason is the wish to adhere to an equational format for process algebra as long as possible.

*1.7. The (non)existence of a suitable partial order on process algebras*

It would be most convenient to have a cpo structure for process algebras such as $A_\omega$, $A^\infty$. One could think of adding an element 0 as in the previous remark, to function as the least element in a supposed partial order $\leqslant$ on $A_\omega$, $A^\infty$. Moreover, such a p.o. should be 'suitable' in the sense of respecting substitution (i.o.w. being monotone in the operations).

However, a partial order on $A_\omega$ or $A^\infty$ (extended with 0) with these properties:

$$\begin{cases} 0 \leqslant p \\ p \leqslant q \Rightarrow s(p) \leqslant s(q) \end{cases}$$

(where $s(\;)$ is some 'context'), does not exist, since it would yield the contradictory equation $aa = aa + a$:

$$aa \;=\; aa + 0 \leqslant aa + a = aa + a0 \leqslant aa + aa = aa.$$

Also there does not exist a p.o. on $A_\omega$, $A^\infty$ satisfying the properties

$$\begin{cases} x \leqslant x + y \\ x \leqslant y \Rightarrow s(x) \leqslant s(y). \end{cases}$$

For, this would result in the contradictory equation $a(b + c) = a(b + c) + ab$:

$$a(b+c)\leqslant a(b+c)+ab\leqslant a(b+c)+a(b+c) = a(b+c).$$

### *1.8. The auxiliary operator left-merge*

The theory of the initial algebra $A_\omega(+,\cdot,\|,\mathbin{|\!\!\lfloor})$, that is the set of true equations between closed terms, is finitely axiomatized by $PA$. Without $\mathbin{|\!\!\lfloor}$ however such a finite axiomatization of the theory of the reduct $A_\omega(+,\cdot,\|)$ does not seem possible. Of course the main advantage of $\mathbin{|\!\!\lfloor}$ is the ease in algebraical computation.

Another advantage of $\mathbin{|\!\!\lfloor}$ is the greater defining power it gives. E.g. the unique solution of the recursion equation

$$X = p\mathbin{|\!\!\lfloor} X$$

(a topic considered in detail in Section 3) can be seen as the 'ω-merge' of $p$, notation: $p^{\overset{\omega}{=}}$, which is intuitively

$$p\|p\|p\|....$$

i.e. the limit of the sequence $p$, $p\|p$, $p\|p\|p$, .... (see also the next remark). Without $\mathbin{|\!\!\lfloor}$, such a uniform definition of $p^{\overset{\omega}{=}}$ does not seem possible.

### *1.9. Solving equations in $A^\infty$*

In Section 3 recursion equations and systems of recursion equations will be considered under the condition that the equations are *guarded*. Here, we want to mention a theorem for the unguarded case:

THEOREM 1.4. *Let $E_X=\{X_i=T_i(X)|i=1,...,n\}$ be a system of equations for $X=X_1$, ..., $X_n$. Then $E_X$ has a solution $(p_1, ..., p_n)$ in each of the above introduced process algebras.*

In general this solution will not be unique. In the case that $n=1$ solutions can be obtained as follows:

THEOREM 1.5. *Let $X=T(X)$ be a recursion equation for $X$. Then a solution for $X$ can be obtained as the limit of the iteration sequence*

$$q,\ T(q),\ T(T(q)),\ ...,\ T^n(q),\ ...$$

*for arbitrary $q$.*

(Here $\lim_{k\to\infty}T^k(q)=p$ means: $\forall n\,\exists m\,(T^m(q))_n=(p)_n$.) At present however we do not see applications for the possibility of solving unguarded fixed point equations.

## 2. Process algebra with communication: ACP

We will now extend the axiom system *PA* of Section 1 with the facility of *communication* between processes. The communication will be modeled by *actions sharing*. In *PA* all atomic actions were on equal footing, and capable of being performed independently. In *ACP*, *Algebra of Communicating Processes,* we will introduce next to this kind of independent or autonomous actions, so-called *subatomic* actions which need one or more other subatomic actions as partners in order to be executed. (Cf. the subatomic actions $C!t$ and $C?x$ in Hoare's CSP (see [12]), whose simultaneous execution amounts to the assignment $x := t$.) The execution is then an 'ordinary' atomic action.

Using this model of shared actions, of which a particular case is *'handshaking'*, we will as an application model the process given by a *dataflow network.*

As a first illustration, consider the following processes $p=(abc)^{\omega}$ and $q=(efg)^{\omega}$.

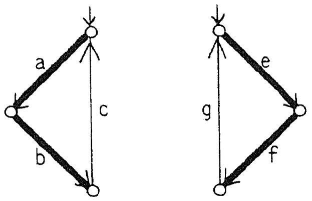


FIGURE 12

The heavy lines denote atomic actions, the steps $c$ and $g$ are subatomic actions and need each other to perform the action $h$, notation: $c|g=h$. (In Petri net notation, the process resulting from the co-operation of $p,q$ would be given by

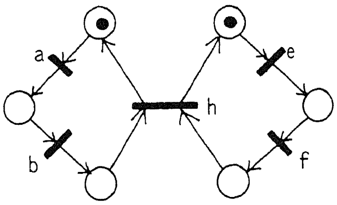


FIGURE 13

Now the process $r$ resulting from the co-operation of $p$ and $q$ would be:

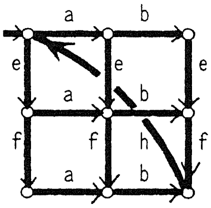


FIGURE 14

That is, $r=([a(e(bf+fb)+bef)+e(a(bf+fb)+fab))\cdot h)^{\omega}$.

The axiom system $ACP$ (Table 3) gives the means to compute the results of such communicating processes in an algebraic way. $ACP$ is an extension of $PA$ (Table 2), but not only in the sense that axioms are added; one axiom from $PA$ (viz. $M1$) is adapted: $x\|y$ is in $ACP$ a sum of *three* terms, namely $x \mathbin{\|\!\!\_} y$, $y \mathbin{\|\!\!\_} x$ and the new summand $x|y$. Here $x \mathbin{\|\!\!\_} y$ is, as in $PA$, 'like $x\|y$' but taking its first step from $x$; likewise $y \mathbin{\|\!\!\_} x$; and $x|y$ is like $x\|y$ but requires the first action to be the result of a communication (between a first subatomic 'step' of $x$ and a first subatomic step of $y$).

This new operator '$|$' is called *communication merge*; on the set $A$ of atoms and subatomic actions it is a binary function, the *communication function*, which is given a priori. It is commutative and associative. The precise choice of the communication function varies with the application of $ACP$ which one has in mind — just as the choice of the alphabet $A$. Thus $ACP$ is in fact parametrized by $A$ and by the communication function $|:A\times A\to A$.

The difference between what we called 'independent' atoms and 'subatomic actions' needs, fortunately, not to be made explicit in the axiom system. What is atomic and what subatomic follows by an inspection of the communication function '$|$'.

Besides a new operator '$|$', communication merge, there appear two new ingredients in the signature of $ACP$ as compared to that of $PA$.

The first is a constant $\delta$, which is a 'zero' for $+$ and moreover satisfies the axiom $\delta\, x=\delta$ (A7). The 'process' $\delta$ exhibits some (but not all) of the features of deadlock or rather failure. The main reason for introducing $\delta$ is algebraical: by means of $\delta$ the unsuccessful communications are eliminated. We will refer to the constant $\delta$ as *'deadlock'* (without claiming that $\delta$ models all of the deadlock phenomenon). An intuitive view of $\delta$ which 'explains' the axioms A6, 7 in Table 3 is: $\delta$ is the action in which the process acknowledges the fact that it cannot further execute actions. So, whenever the process has another option, it will not perform this acknowledgement of stagnation: $x+\delta=x$.

The second new ingredient is formed by the *encapsulation operators* $\partial_H$ where $H\subseteq A$. Putting $\partial_H$ in front of at process expression $p$, result $\partial_H(p)$, means that the subatomic actions mentioned in $H$ and occurring in $p$, cannot anymore communicate with an 'external' process — they have had their chance inside $p$.

Summarizing, we have the following signature for $ACP$:

| | |
|---|---|
| $x+y$ | alternative composition (sum) |
| $x\cdot y$ | sequential composition (product) |
| $x\|y$ | parallel composition (merge) |
| $x \mathbin{\|\!\!\_} y$ | left merge |
| $x\mid y$ | communication merge |
| $\mid :A\times A\to A$ | communication function |
| $\partial_H(x)$ | encapsulation |
| $\delta$ | deadlock |

Note that *ACP* is an extension of *PA* in the following sense: let the communication function be trivial, i.e. $a \mid b = \delta$ for all $a, b \in A$. Then the models $A_\omega$, $\mathbb{A}^\infty$, $A^\infty$ for *PA* (with signature $+, \cdot, \parallel, \mathbin{\lfloor\!\!\lfloor}$) are just *reducts* in the modeltheoretic sense of the models $A_\omega$, $\mathbb{A}^\infty$, $A^\infty$ for *ACP* which we will construct below and which have signature $+, \cdot, \parallel, \mathbin{\lfloor\!\!\lfloor}, \mid, \partial_H, \delta$.

*ACP*

| | |
|---|---|
| $x + y = y + x$ | 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 |
| $x + \delta = x$ | A6 |
| $\delta \cdot x = \delta$ | A7 |
| | |
| $a \mid b = b \mid a$ | C1 |
| $(a \mid b) \mid c = a \mid (b \mid c)$ | C2 |
| $\delta \mid a = \delta$ | C3 |
| | |
| $x \parallel y = x \mathbin{\lfloor\!\!\lfloor} y + y \mathbin{\lfloor\!\!\lfloor} x + x \mid y$ | CM1 |
| $a \mathbin{\lfloor\!\!\lfloor} x = a \cdot x$ | CM2 |
| $(ax) \mathbin{\lfloor\!\!\lfloor} y = a(x \parallel y)$ | CM3 |
| $(x + y) \mathbin{\lfloor\!\!\lfloor} z = x \mathbin{\lfloor\!\!\lfloor} z + y \mathbin{\lfloor\!\!\lfloor} z$ | CM4 |
| $(ax) \mid b = (a \mid b) \cdot x$ | CM5 |
| $a \mid (bx) = (a \mid b) \cdot x$ | CM6 |
| $(ax) \mid (by) = (a \mid b) \cdot (x \parallel y)$ | CM7 |
| $(x + y) \mid z = x \mid z + y \mid z$ | CM8 |
| $x \mid (y + z) = x \mid y + x \mid z$ | CM9 |
| | |
| $\partial_H(a) = a$ if $a \notin H$ | D1 |
| $\partial_H(a) = \delta$ 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 |

TABLE 3

### 2.1. *Process algebras for ACP*

The development of models for *ACP* is analogous to that for *PA*, so we will be much shorter now in its description. Again we introduce:

(1) $A_\omega$, *the initial algebra of ACP,*

(2) $\mathbb{A}^\infty$, *the process graph model of ACP,*

(3) $A^\infty$, *the standard model of ACP.*

Here some confusion may arise as to which signature, that of *PA* or that of *ACP*, is meant when speaking about $A_\omega$, $\mathbb{A}^\infty$, $A^\infty$. When this confusion is not solved by the context, we will mention the intended signature explicitly, as in $A_\omega(+,\cdot,\|,\mathbin{\|\!\_})$ vs. $A_\omega(+,\cdot,\|,\mathbin{\|\!\_},|,\partial_H,\delta)$.

*2.1.1. The initial algebra $A_\omega$ of ACP.* Before building $A_\omega$, we have fixed the alphabet $A$, a communication function $|:A\times A\to A$, and a subset $H\subseteq A$ (hence an encapsulation operator $\partial_H$).

Now $A_\omega$ contains as elements: the process expressions (in the signature of *ACP*) modulo the equality given by *ACP*. By the following theorem.

THEOREM 2.1 (NORMAL FORM). *For each closed term $t$ there is a closed term $t'$ not containing $\|,\mathbin{\|\!\_},|,\partial_H$ such that $ACP\vdash t=t'$.*

We may think of elements of $A_\omega$ as built from $A$, $+,\cdot$ only (just as in the case of *PA*), or as the finite process trees encountered in Section 1.

EXAMPLE 2.1. Let $A=\{a,b,c,c^0,d,\delta\}$. Let $|:A\times A\to A$ be given by $c|c=c^0$, and all other communication equal $\delta$ (thus $a|b=c|c^0=d|a=\delta|a=\ldots=\delta|\delta=\delta$). Further, let $H=\{c\}$. Then:

$$\partial_H[(ab+ac)\|cd] =$$

$$\partial_{\{c\}}[ab\mathbin{\|\!\_}cd+ac\mathbin{\|\!\_}cd+cd\mathbin{\|\!\_}(ab+ac)+cd|ab+cd|ac] =$$

$$\partial_{\{c\}}[a(b\|cd)+a(c\|cd)+c(d\|(ab+ac))+(c|a)(d\|b)+(c|a)(d\|c)] =$$

$$\partial_{\{c\}}[a(bcd+c(d\|b)+(b|c)d)+a(ccd+c(d\|c)+(c|c)d)+$$

$$+c(d\|(ab+ac))+\delta(d\|b)+\delta(d\|c)]=$$

$$\partial_{\{c\}}[a(bcd+c(d\|b))+a(ccd+c(d\|c)+c^0d)+c(d\|(ab+ac))] =$$

$$ab\delta+ac^0d.$$

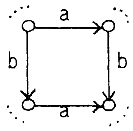
EXAMPLE 2.2. Consider the alphabet $\{a,b,b^0,c,c^0,\delta\}$ with the only proper communications $c|c=c^0$, $b|b=b^0$. Now $a(b+c)$ and $ab+ac$ behave differently in the context $C[\;]=\partial_{\{b,c\}}(\ldots\|c)$; namely:

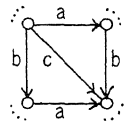
$$C[a(b+c)] = ac^0,$$

$$C[ab+ac] = a\delta+ac^0.$$

*2.1.2. The process graph algebra $\mathbb{A}^{\infty}$ for ACP.* The definition of $\mathbb{A}^{\infty}(+,\cdot,\|,|\!\lfloor,|,\partial_H,\delta)$ parallels that of $\mathbb{A}^{\infty}(+,\cdot,\|,|\!\lfloor)$ for $PA$, except for two additions.

Let $g,h$ be finitely branching process graphs with acyclic roots. Then the merge $g\|h$ is now the cartesian product graph enriched with 'diagonal' edges $\xrightarrow{a|b}$in the following situation:

if 

is a subgraph of the cartesian product graph, then the arrow $\bigcirc\xrightarrow{c}\bigcirc$ (where $c=a|b$) is inserted; result: 

The left merge $g|\!\lfloor h$ and the communication merge yield results which can now be guessed. An example will suffice:

EXAMPLE 2.3. Let $A=\{a,b,c,\delta\}$, $a|b=c$ and all other communications equal $\delta$. Then $ab\|bab$, $ab|\!\lfloor bab$, $bab|\!\lfloor ab$ and $ab|bab$ are the following graphs respectively:

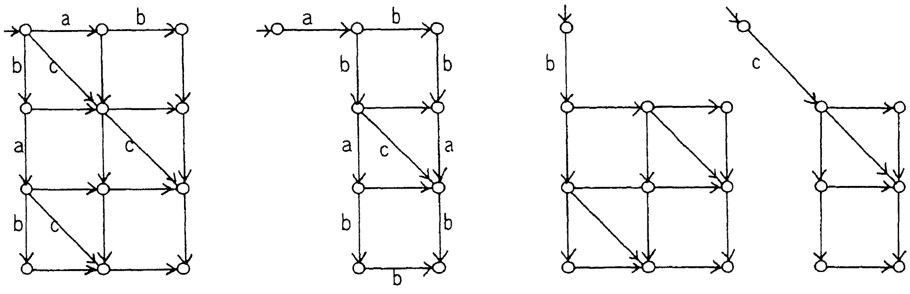


FIGURE 15

Note that we have omitted the diagonal edges labeled with $\delta$, resulting from trivial communications. This brings us to the second addition *bisimulation between process graphs containing $\delta$-steps.*

The old concept of bisimulation in Section 1 would not do now, since it would not satisfy the laws $x+\delta=x$ and $\delta x=\delta$. We will choose the following solution: first define the *$\delta$-normal form* of the process graph $g$ as the process graph $g'$ obtained by deleting all $\delta$-steps which have a 'brother' step and creating for the remaining $\delta$-steps if necessary separate end nodes. Afterwards disconnected pieces of the graph are removed.

Now $g$ and $h$ are bisimilar if their $\delta$-normal forms are bisimilar in the old sense.

Finally, the effect of $\partial_H$ on the graph $g$ is simply to replace all $a \in H$ which occur in $g$, by $\delta$.

The effect of these definitions is that $\mathbb{A}^{\infty}$ is a model of $ACP$. Using these graphs, we have an easy way to 'compute' the result of Example 2.1.:

$(ab+ac)\|cd =$

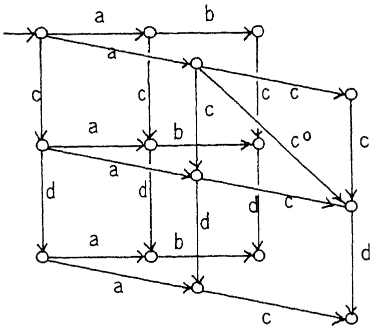


FIGURE 16

$\partial_{\{c\}}[(ab+ac)\|cd] = ab+ac^{\circ}d=$

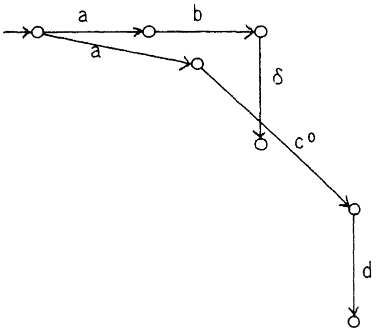


FIGURE 17

*2.1.3. The standard model $A^{\infty}$ for $ACP$.* The standard model $A^{\infty}$ for $ACP$ is constructed entirely analogous to the corresponding model for $PA$. An example of a computation in the standard model: Let $A = \{a,b,c,d,\delta\}$, $a/a = d$ the only proper communication. Now let

$$p = (a, ab, aba, abab, \ ...)$$

and

$$q = (a, ac, aca, acac, \ ...).$$

Then

$$\begin{aligned}\partial_{\{a\}}(p\|q) &= \partial_{\{a\}}((a\|a)_1, (ab\|ac)_2, (aba\|aca)_3, \ ...)\\ &= (\partial_{\{a\}}(a\|a)_1, \partial_{\{a\}}(ab\|ac)_2, \ ...)\\ &= (\partial_{\{a\}}(aa+d)_1, \partial_{\{a\}}( \ ... \ ), \ ...)\\ &= (d, d(b+c), d(bc+cb), d(bc+cb)d, \ ...)\end{aligned}$$

*2.2. Process algebras with standard concurrency and handshaking*

A useful intuition about communicating processes is to postulate that $\|$ is commutative and associative. This does not follow from the axioms of $ACP$; pathological process algebras with noncommutative and nonassociative $\|$ are possible. But in the process algebras $A_{\omega}$, $\mathbb{A}^{\infty}$ and $A^{\infty}$, $\|$ is indeed commutative and associative. In fact these algebras satisfy the following *axioms of standard concurrency:*

$$\boxed{\begin{aligned}
&(x \mathbin{\|\!\lfloor} y) \mathbin{\|\!\lfloor} z = x \mathbin{\|\!\lfloor} (y \| z)\\
&(x | y) \mathbin{\|\!\lfloor} z = x | (y \mathbin{\|\!\lfloor} z)\\
&x | y = y | x\\
&x \| y = y \| x\\
&x | (y | z) = (x | y) | z\\
&x \| (y \| z) = (x \| y) \| z
\end{aligned}}$$

TABLE 4

*(These axioms are not independent relative to $ACP$. E.g. commutativity and associativity of $\|$ are derivable from the other four plus $ACP$.)*

Moreover, matters are greatly simplified by adopting the handshaking axiom:

$$x | y | z = \delta$$

which is satisfied by both CSP and CCS. The handshaking axiom implies that *all proper communications are binary.*

Under the hypotheses of standard concurrency and the handshaking axiom we can prove the following fact which is a generalization of the $ACP$-axiom $CM1$:

THEOREM 2.2 (MILNER). $x_1 \| \ldots \| x_k = \sum_i x_i \mathbin{\|\!\lfloor} X_k^i + \sum_{i \neq j} (x_i | x_j) \mathbin{\|\!\lfloor} X_k^{i,j}$.

Here $X_k^i$ is obtained by merging $x_1, \ldots, x_k$ except $x_i$, and $X_k^{i,j}$ is obtained by merging $x_1, \ldots, x_k$ except $x_i, x_j$ $(k \geqslant 3)$. Thus, e.g. for $k=3$:

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

### 2.3. *Networks of processes communicating by handshaking*

Imagine a process $P$ (figure 18) whose events have a certain spatial position $\alpha, \beta, \gamma$ as well as a data content $d$ — so the actions of $P$ are pairs $(\alpha, d)$, $(\alpha, d')$, $(\beta, d)$, ..., for simplicity written as $\alpha_d, \alpha_{d'}, \beta_d, \ldots$ . E.g. let $\mathcal{D} = \{0,1\}$ be the data domain and let $P$ be given by the recursion equation

$$P = \alpha_0 \beta_1 \gamma_0 P.$$

Next, consider a network of such proceses as in figure 19, where the nodes $D, M, N, C$ are given by

$$\begin{aligned}
D &= (\alpha_0 \beta_0 \beta_0 + \alpha_1 \beta_1 \beta_1) D\\
M &= [(\beta_0 + \zeta_0)\gamma_0 + (\beta_1 + \zeta_1)\gamma_1] M\\
C &= [\gamma_0(\eta_0 \epsilon_0 + \epsilon_0 \eta_0) + \gamma_1(\eta_1 \epsilon_1 + \epsilon_1 \eta_1)] C\\
N &= (\epsilon_0 \eta_1 + \epsilon_1 \eta_0) N.
\end{aligned}$$

So $D$ is the process which doubles an 'incoming' 0 into 00, likewise for 1; $M$ is the merge process which relays the signals 0,1 in order of entrance at $\beta$ or $\xi$; $C$ is the copy process which relays an incoming signal to both $\eta$ and $\epsilon$, in either order; and $N$ is the process which inverts an incoming signal.

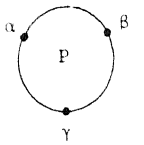


FIGURE 18

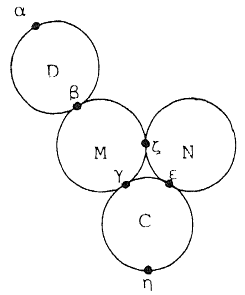


FIGURE 19

The positions $\alpha$, ...,$\zeta$ will be called *ports*; $\beta,\gamma,\epsilon,\zeta$ are *internal* ports. As suggested by figure 19 with its sharing of the internal ports, the processes $D$, $M$, $C$, $N$ cannot operate freely but are constrained by each other: an action $\beta_0$ of $D$ is now only an 'intended' action (a subatomic action) needing the same action $\beta_0$ of $M$ for the actual passing or 0 along port $\beta$. Let us denote this actual event by $\beta^\circ$; likewise $\beta_1^\circ$ denotes passing a 1 at $\beta$, etc. (In fact, the word 'passing' is misleading since it suggests a *direction* of flow which, interestingly, disappears at this level of analysis.)

Intuitively, it is clear that the example network has an operational semantics which is a process in $A^\infty$ or $\mathbf{A}^\infty$ over the alphabet

$$A = \{\alpha_d,\beta_d^\circ,\gamma_d^\circ,\epsilon_d^\circ,\xi_d^\circ,\eta_d | d \in \mathfrak{D}\}.$$

Now this process can be defined as

$$\partial_H(D \| M \| C \| N)$$

where $H = \{\beta_d,\gamma_d,\epsilon_d,\zeta_d | d \in \mathfrak{D}\}$ and where the communication function is defined by: $a|a = a^\circ$ for all $a \in H$ and these are the only proper communications. The operational semantics of the network can now be computed using $ACP$ to any desired depth. This computation can be speeded up by using the Milner Expansion Theorem 2.2. (In fact, for this example the resulting process is regular, that is: given by a finite process graph.)

Before discussing the operational semantics of dataflow networks through networks with *channels* (which were not considered above; there processes are 'directly' connected), we will make some remarks on the present definition of the operational semantics of networks communicating by handshaking.

*2.3.1. Handshaking.* Handshaking, implicitly introduced above by the example network, is understood here as follows. A network consisting of nodes $P_1,...,P_n$ communicates by handshaking if each port $\alpha$ of $P_i$ $(i=1,..,n)$ is either external (i.e. not connected to any other port) or connected to precisely one port of another process. Here '$\alpha$ is connected to $\beta$' means that $\alpha_d$ only communicates properly with $\beta_d$ (so if $\alpha_d|\gamma_e \neq \delta$, then $\gamma=\beta$ and $e=d$).

*2.3.2. Symmetrical handshaking.* Symmetrical handshaking was used in the example above; here a port $\alpha$ is either external or connected to $\alpha$. By the handshaking convention, a port $\alpha$ can be shared by two processes at most.

The example network can just as well be treated using *asymmetrical* handshaking, as in

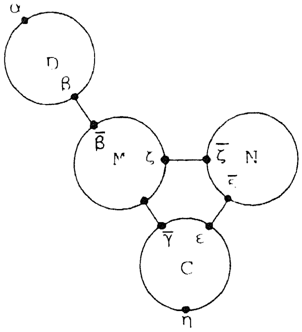


FIGURE 20

where $\beta \neq \bar{\beta}$, etc., and communication is given by $\beta_d|\bar{\beta}_d = \beta_d^\circ$, etc. This is the format used in MILNER [14], where many examples of networks communicating by handshaking are given. One can prove an adequacy theorem for asymmetrical communication, in the sense that communication by handshaking can always be taken to be 1-1 and asymmetrical without loss of defining power. This statement will be made more precise in subsection 3.8.

Our example network was phrased in terms of symmetrical handshaking, to minimize the notational overhead. For *regular* processes (the property 'regular' is the subject of the next section), as all the nodes $D,M,C,N$ in the example are, this works perfectly well. If the nodes are not regular and given by recursion equations containing $\|$, then asymmetrical communication must be chosen; otherwise undesired 'auto-communications' may occur when evaluating the recursive definition.

The condition in our definition of handshaking is a bit severe. One can safely allow a port to be shared by more than two processes, still requiring proper communications to be binary.

EXAMPLE 2.4. Let $\mathfrak{D}=\{0\}$, $I_{\alpha\beta}=\alpha_0\beta_0 I_{\alpha\beta}$, likewise $I_{\beta\gamma}$, $I_{\gamma\alpha}$. Let $T=\alpha_0$. Let communication be given by $a|a=a^\circ$ for $a\in H=\{\alpha_0,\beta_0,\gamma_0\}$.

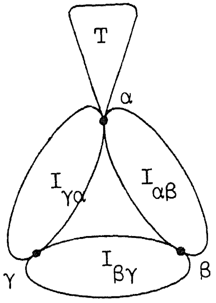


FIGURE 21

In the resulting total process $\partial_H(T\|I_{\alpha\beta}\|I_{\beta\gamma}\|I_{\gamma\alpha})$ the datum 0 is inserted by $T$ and then cycles clockwise through the ring of processors (which are buffers with capacity 1).

A more interesting and fundamental deviation of the handshaking requirements is introduced by MILNER [15].

*2.3.3. Synchronous versus asynchronous processes.* Process co-operation as described above is *asynchronous*, in the sense of MILNER [15] where a study is made of synchronicity vs. asynchronicity, and where it is argued that synchronous co-operation is the more fundamental of the two.

A *synchronous* network of processes is one where at the pulses of an (imaginary) universal clock all ports exhibit activity simultaneously. As an example consider the following network consisting of two *NOR* circuits; the example is from MILNER [15] with a slight adaptation and serves to demonstrate our claim that synchronous networks can be treated to a large extent within *ACP*. The *NOR* circuit (figure 22) is defined by

$$NOR(k) = \sum_{i,j\in\{0,1\}} (\alpha_i|\beta_j|\gamma_k)NOR(i\downarrow j) \quad (k=0,1)$$

Here $i\downarrow j=1 \Leftrightarrow i=j=0$, and the $\alpha_i|\beta_j|\gamma_k$ are actions which can be perceived simultaneously at the ports $\alpha,\beta,\gamma$. E.g. $\alpha_0|\beta_0|\gamma_1$ is the simultaneous passing (or rather, occurrence) of 0 at $\alpha$, 0 at $\beta$ and 1 at $\gamma$.

Now consider the network as in figure 23, where $NOR'$ is a copy of $NOR$ obtained by renaming the indicated ports. So

$$NOR'(k) = \sum_{i,j\in\{0,1\}} (\bar{\beta}_k|\bar{\gamma}_i|\lambda_j)NOR'(i\downarrow j).$$

Communication is given by $(\alpha_i|\beta_j|\gamma_k)\underline{|}(\bar{\beta}_j|\bar{\gamma}_k|\lambda_{\underline{l}})=\alpha_i|\gamma_k^\circ|\beta_j^\circ|\lambda_l$; all other communications result in $\delta$. Further, $H=\{\alpha_i|\beta_j|\gamma_k,\ \bar{\gamma}_i|\beta_j|\lambda_k \mid i,j,k\in\{0,1\}\}$.

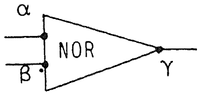


FIGURE 22

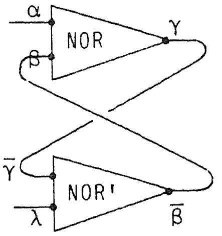


FIGURE 23

Then the network of Figure 23 has as semantics: $\partial_H(NOR(k)\|NOR'(l))$, in the initial state $k,l$. Abbreviating this expression by $X(k,l)$ we compute using the axioms of $ACP$:

$$\begin{aligned} X(k,l) &= \partial_H(NOR(k)\mathbin{|\!\!\lfloor} NOR'(l))+\partial_H(NOR'(l)\mathbin{|\!\!\lfloor} NOR(k)) \\ &+ \partial_H(NOR(k)|NOR'(l))=\delta+\delta+\partial_H(NOR(k)|NOR'(l)) \\ &= \partial_H(\sum_{i,j}(\alpha_i|\beta_j|\gamma_k)NOR(i{\downarrow}j) \mid \sum_{i,j}(\bar{\beta}_l|\bar{\gamma}_i|\lambda_j)NOR'(i{\downarrow}j)) \\ &= \partial_H(\sum_{i,j}(\alpha_i|\beta_l^\circ|\gamma_k^\circ|\lambda_j)[NOR(i{\downarrow}l)\|NOR'(k{\downarrow}j)] \\ &= \sum_{i,j}(\alpha_i|\beta_l^\circ|\gamma_k^\circ|\lambda_j)X(i{\downarrow}l,\ k{\downarrow}j) \end{aligned}$$

which is a system of four recursion equations, describing the intuitively expected process. The difference with Milner's approach via SCCS (see [15]) is the use of $\delta$: not only does it serve to remove the undesired interleaving results, also it is used to express that certain composite actions are incompatible.

A more direct axiomatization of synchronous processes, related to Milner's SCCS, can be given by omitting the interleaving part of $ACP$, that is: replace $CM1$ by $x\|y=x|y$, and erase CM2-4. We will not study this axiomatization here, however.

*2.4. Dataflow networks.* We will return now to the case of networks communicating by handshaking. Above, the connections between ports were directionless and thought of as relaying the data instantaneously. These port connections are *not* channels as used in dataflow networks; e.g. a channel like *Queue* does not relay its messages instantaneously. So let us consider networks such as the one in figure 24, where the arrow-shaped figures denote channels. We will consider as channels: *Queue, Bag* and *Stack.* Now an important realization is that channels and nodes are in fact the same type of entities: both are processes.

Hence this simple form of dataflow is nothing more than a network communicating by handshaking as treated above. The only difficulty is that the processes Queue, Bag and Stack are rather complicated: they are not regular. In the next section we will consider recursion equations within $ACP$ (in fact, even within $PA$) for Bag and Stack, and discuss some of their properties.

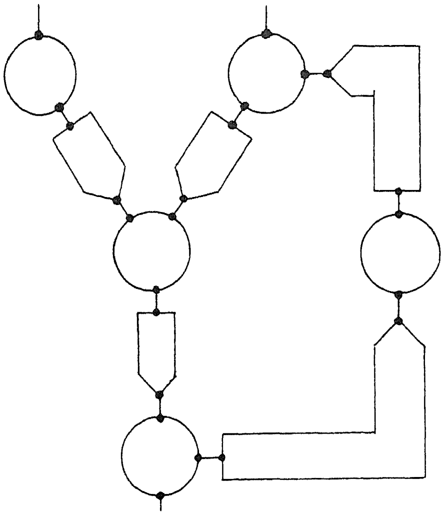


FIGURE 24

For *Queue* the situation is essentially more complicated. If one admits infinite systems of equations, Queue can be defined in $PA$ as the first component of the solution of such an infinite system of equations.

One can prove (see [10]) that Queue cannot be defined recursively by a finite system of recursion equations over $PA$.

If one allows extensions of the $PA$ formalism, there are two ways of specifying Queue. The first method is via auxiliary operators $\wedge\!\!\!\!/$ and $\underline{\triangle}$, that can be axiomatized by finitely many equations (like $\lfloor\!\lfloor$ is finitely axiomatized). Then Queue can be recursively defined over $PA$ extended with these new operators. (See [10].) The second method uses process graphs defined by means of abstract data types; see [7].

## 3. Recursively defined processes

In the previous sections we have used, occasionally, some processes which were defined as the solutions of recursion equations; namely, the iteration $p^{\omega}$ of $p$ (as the solution of $X=pX$) and the $\omega$-merge $p^{\overset{\omega}{\lfloor\!\lfloor}}$ of $p$ (as the solution of $X=p\lfloor\!\lfloor X$; see 1.8.).

In this section we will consider this important specification method for processes in a more systematic way. This will produce some criteria as to which processes in $A^{\infty}$ can be defined recursively; also it will give us some other process algebras.

In the course of these considerations the concept of a *finitely generated* process algebra will prove to be an important concept. Likewise, the concept of a *regular* process plays a prominent role: this is a process corresponding to a finite transition diagram (i.e. having a finite canonical process graph), possibly with cycles. First we need two technical concepts.

### *3.1. Linear terms and guarded terms*

Let $X_1,\ldots,X_n$, be variables ranging over processes. Given the signature of *PA* or that of *ACP*, two kinds of terms containing variables $X_1,\ldots,X_n$ are of particular importance:

(i) *Linear terms.* Linear terms are inductively defined as follows:
- atoms $a, \delta$ and variables $X_i$ are linear terms,
- if $T_1$ and $T_2$ are linear terms then so are $T_1+T_2$ and $aT_1$ (for $a \in A$).

An equation $T_1=T_2$ is called linear if $T_1,T_2$ are linear.

(ii) *Guarded terms.* The *unguarded* terms are inductively defined as follows:
- $X_i$ is unguarded,
- if $T$ is unguarded then so are $T+T'$, $T{\cdot}T'$, $\partial_H(T)$, $T\|T'$, $T \mathbin{\|\!\!\_} T'$, $T|T'$ (for every $T'$).

A term $T$ is guarded if it is not unguarded. Note that we introduced 'formal' variables $X_1,\ldots,X_n$; they are meant as the 'unknowns' in recursion equations. The formal variables should not be confused with the metavariables $x,y,\ldots$ which occur in the axioms of *PA* and *ACP*.

Mostly, we will be interested in *finite* systems $E$ of equations. In this section we will always require that $E$ is a *guarded* system of equations. (I.e. the RHS's of the equations in $E$ are guarded.) We will first consider the case of *linear* $E$, which gives us the regular processes.

### *3.2. Regular processes*

As defined in Section 1, an element $p \in A^\infty$ has a canonical process graph, with the subprocesses as nonterminal nodes and 'o' as terminal node. Now we define:

(i) $p \in A^\infty$ is *regular* if $Sub(p)$ is finite;
(ii) $r(A^\infty)$ is the collection of the regular processes in $A^\infty$.

The next fact is immediate.

THEOREM 3.1. *The following statements are equivalent:*
(i) *$p$ is regular*
(ii) *$Sub(p)$ is finite*
(iii) *$G(p)$ is finite*
(iv) *$p$ is the first component of the solution vector of a finite, guarded, linear system of equations.*

Moreover, $r(A^\infty)$ is closed under all operations (in the signature of *PA* as well as that of *ACP*); it is a process algebra whose position relative to the previous ones is as follows: $A_\omega \subseteq r(A^\infty) \subseteq \mathbf{A}^\infty \subseteq A^\infty$.

EXAMPLE 3.1.
(1) Let $X$ be the solution of $X=a+bX$. Then $G(X)$ is as in figure 25, with a tree representation as in figure 26. Note that $Sub(X)=\{X\}$. As a projective sequence, $X=(a+b, a+b(a+b), a=b(a+b(a+b)),\ldots)$. $X$ is a regular process.

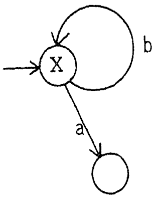


FIGURE 25

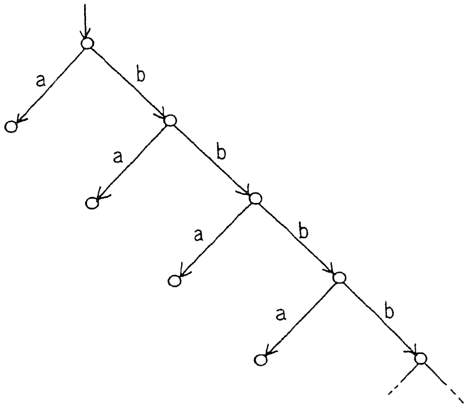


FIGURE 26

(2) Let $E_{X,Y}$ be $\begin{cases} X = aY+c \\ Y = bX+dY+e \end{cases}$

Then the regular solution $\underline{X}$ has the canonical process graph

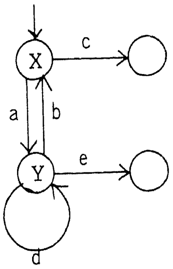


FIGURE 27

(3) The following process $\underline{X}$ is not regular.

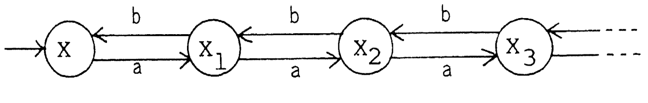


FIGURE 28

It is the first component of the solution vector of the *infinite* system of linear equations

$$\begin{cases} X = X_0 = aX_1 \\ X_{n+1} = aX_{n+2} + bX_n \quad (n \geqslant 0) \end{cases}$$

That $\underline{X}$ is indeed not regular follows from the realization that there are infinitely many subprocesses (all the $\underline{X}_n$, $n \geqslant 0$, are pairwise different).

### *3.3. Recursively defined processes*

We now define in full generality the concept of a *recursively defined process.* Let $X = \{X_1, \ldots, X_n\}$ be a set of process names (formal variables). We will consider terms over $X$ composed from atoms $a \in A$ and the operators in the signature of $PA$ or that of $ACP$.

A sytem $E_X$ of *guarded fixed point equations* (or *guarded recursion equations*) for $X$ is a set of $n$ equations $\{X_i = T_i(X_1, \ldots, X_n) | i = 1, \ldots, n\}$ with $T_i(X)$ a guarded term. There is the standard result:

THEOREM 3.2. *Each system $E_X$ of guarded fixed point equations has a unique solution in $(A^\infty)^n$.*

We define $p \in A^\infty$ to be *recursively definable* if there exists a system $E_X$ of guarded fixed point equations over $X$ with solution $(p, q_1, \ldots, q_{n-1})$. With $R(A^\infty)$ we denote the *subalgebra of recursively defined processes.* The relative position of this second new process algebra $R(A^\infty)$ is as follows:

$$A_\omega \subseteq r(A^\infty) \subseteq R(A^\infty) \subseteq \mathbf{A}^\infty \subseteq A^\infty$$

both for $PA$ and $ACP$. All inclusions are proper.

There is an algebra of some interest which is strictly intermediate between $r(A^\infty)$ and $R(A^\infty)$: it is the process algebra of *uniformly finitely branching* processes. These are processes having canonical process graphs where all the nodes have a uniformly bounded outdegree.

EXAMPLE 3.2. The following is a system of nonlinear guarded recursion equations:

$$\begin{cases} X = aX(X+b) \\ Y = bYXY \end{cases}$$

Likewise $X = a(b \| X)$ is a nonlinear equation. (The process graph of the solution $\underline{X}$ is the one in figure 28.)

A useful fact is the following. Call a process *perpetual* if all its traces are infinite. Then:

THEOREM 3.3. *Let $E_X$ be a system of guarded recursion equations using only $+$ and $\cdot$. Suppose the solutions $X$ are perpetual. Then they are regular.*

An example suggests the simple proof:

$$E_{X,Y,Z} \begin{cases} X = aYX(X+Y) + bYYZ \\ Y = bYY + cZX \\ Z = c(ZX + dZX) \end{cases}$$

Now a short inspection of $E_{X,Y,Z}$ reveals that the solutions $\underline{X}, \underline{Y}, \underline{Z}$ are perpetual. So in products in $E_{X,Y,Z}$ one can erase every factor following $X$, $Y$, $Z$ (since for all $p$, $\underline{X}\,p = \underline{X}$, etc.) I.e. the system of equations

$$E'_{X,Y,Z}\begin{cases} X = aY + bY \\ Y = bY + cZ \\ Z = c(Z + dZ) \end{cases}$$

has the same solutions $\underline{X}$, $\underline{Y}$, $\underline{Z}$. But since $E'_{X,Y,Z}$ is a *linear* system, these solutions are regular.

We will now consider recursion equations for the processes corresponding to Bag, Stack and Counter.

*3.4. Bag*

Let $\alpha$ -▭- $\beta$ be a bag with input port $\alpha$ and output port $\beta$. (Here 'bag' is considered as a channel which does not preserve, like Queue does, the order of the incoming data. So the contents of Bag can be imagined as a multiset or bag.) Consider a finite data domain $D$. Then the actions to be performed by Bag are, in our earlier notation, $\alpha_d$ and $\beta_d$ $(d \in D)$. For notational convenience we write $d$ instead of $\alpha_d$ and $\underline{d}$ instead of $\beta_d$.

Let $B$ be the initial state of Bag: the empty bag. Now let action $d$ be executed, that is: $d$ is added to the bag. The result is a bag *with the commitment of eventually giving d as output,* i.e. performing action $\underline{d}$. We claim on intuitive grounds that this bag-with-commitment-$\underline{d}$ is $\underline{d} \| B$. This leads to the equation for $B$:

$$B = \sum_{d \in D} d(\underline{d} \| B).$$

Alternatively: consider the process $\Sigma_d d\underline{d}$. Then it is (again intuitively) clear that $B$ is the $\omega$-merge:

$$B = \sum_d d\underline{d} \parallel \sum_d d\underline{d} \parallel \sum_d d\underline{d} \parallel \ldots = (\sum_d d\underline{d})^{\underline{\omega}}.$$

So

$$B = (\sum_{d \in D} d\underline{d}) \mathbin{|\!\!\lfloor} B$$

which indeed is equivalent to the first recursion equation for $B$, by using the axioms of $PA$ for $\mathbin{|\!\!\lfloor}$.

A third definition:

$$\begin{cases} B_d = d\underline{d} \mathbin{|\!\!\lfloor} B_d = d(\underline{d} \| B_d) \\ \\ B = \|_d B_d \quad (d \in D) \end{cases}$$

How can one verify that these equations for $B$ indeed describe the intended Bag?

(a) By computing the corresponding canonical process graph and 'validating this against the intuition';

(b) by the more rigorous method employed in [7], which consists of giving a specification of $B$ in terms of abstract data types and *proving* the equation given here correct w.r.t. that specification. Here we will not discuss that method.

We proceed with (a). First, consider the singleton data domain $D=\{d\}$. Then $B=d(\underline{d}\,\|B)$, and now-writing

$$B_0 = B,\ B_{n+1}=\underline{d}\,\|B_n=\underline{d}^n\,\|B$$

one proves immediately

$$\begin{cases} B_0 = dB_1 \\ B_{n+1} = dB_{n+2}+\underline{d}\,B_n \quad (n\geqslant 0). \end{cases}$$

(PROOF:

$$\begin{aligned} B_{n+1} &=\underline{d}\,\|B_n=\underline{d}\mathbin{|\!\!\lfloor} B_n+B_n\mathbin{|\!\!\lfloor}\underline{d}=\underline{d}\,B_n+(dB_{n+1}+\underline{d}\,B_{n-1})\mathbin{|\!\!\lfloor}\underline{d} \\ &=\underline{d}\,B_n+dB_{n+1}\mathbin{|\!\!\lfloor}\underline{d}+\underline{d}\,B_{n-1}\mathbin{|\!\!\lfloor}\underline{d}=\underline{d}\,B_n+d(B_{n+1}\|\underline{d})+\underline{d}\,(B_{n-1}\|\underline{d}) \\ &=\underline{d}\,B_n+dB_{n+2}+\underline{d}\,B_n.) \end{aligned}$$

This yields the canonical process graph

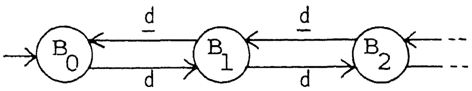


FIGURE 29

The general case $B=\Sigma_d d(\underline{d}\,\|B)$ is, as process graph, obtained by merging these 'singleton-bags' $B_d$. So if $D=\{a,b\}$, the canonical process graph of $B=a(\underline{a}\,\|B)+b(\underline{b}\,\|B)$ is:

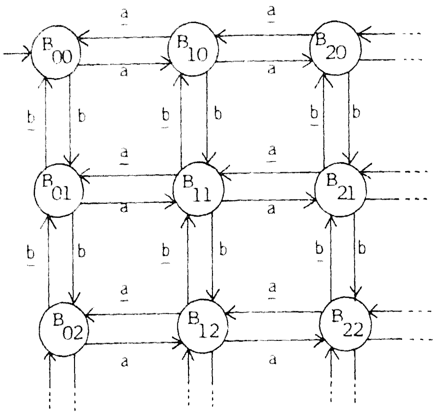


FIGURE 30

We will return to Bag later in this section.

### 3.5. *Stack*

For convenience, let $D=\{a,b\}$. As with Bag, $'a'$ denotes the event of pushing $'a'$ on the stack, $'\underline{a}'$ of popping $'a'$ from the stack; likewise for $b$. Now Stack can be defined thus:

$$\begin{cases} S = TS \\ T = aT_a + bT_b \\ T_a = \underline{a} + TT_a \\ T_b = \underline{b} + TT_b\,. \end{cases}$$

Here $T$ is *Terminating Stack,* which must terminate as soon as it is again empty. Further, $T_a$ is $T$ containing an $'a'$, $T_b$ likewise. $S$ is the iteration $T^{\omega}$ of $T$; so $S$ is the intended perpetual process Stack. Essentially, this recursive definition of Stack occurs also in HOARE [12].

The recursive definition of Stack above involves the definition of a nonperpetual process, in casu $T$. This is essential: *Stack $S$ cannot be derived recursively (over $+$ and $\cdot$) without a non-perpetual auxiliary process.* For, if it could, then Theorem 3.3 would entail that $S$ is regular, an obvious contradiction. A consequence is that $S$ cannot be defined recursively (over $+$ and $\cdot$) in one equation.

The canonical process graphs of $S$ and $T$ are as in figure 31 and 32.

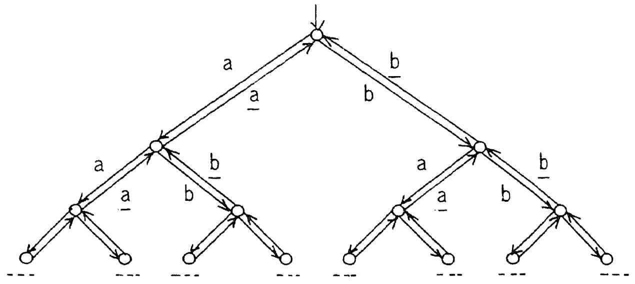


FIGURE 31

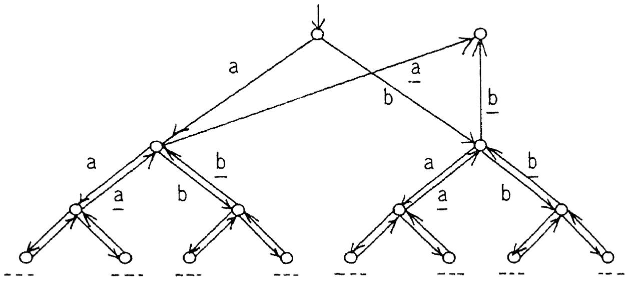


FIGURE 32

*3.6. Counter*

We will consider a simple counter $C$ without test for zero. The equation for $C$ is obtained by the one for Stack, with $D = \{a\}$:

$$\begin{cases} C = TC \\ T = aT_a \\ T_a = \underline{a} + TT_a \end{cases}$$

or, after eliminating $T$ and writing $D = T_a$:

$$\begin{cases} C = aDC \\ D = \underline{a} + aDD. \end{cases}$$

$G(C)$ is determined as follows: writing $C_n = D^n C$ one easily computes

$$\begin{cases} C_0 = aC_1 \\ C_{n+1} = \underline{a}\, C_n + aC_{n+2} \end{cases}$$

which determines the same process graph as for the singleton-bag above. So we have the interesting fact that $C$ is also the solution of

$$C = a(\underline{a} \,\|\, C).$$

This leads to the question whether it is also possible in the case of the general bag (over an arbitrary but finite data domain $D$) to eliminate $\|$ in its recursive definition in favour of $+, \cdot$ (and possibly using more equations). The answer is no, if $D$ contains at least two elements. For the lengthy proof see [8].

### *3.7. Criteria for recursive definability*

THEOREM 3.4. *A process which is recursively defined only with + and ·, and which has an infinite branch, must have an eventually periodic infinite branch.*

EXAMPLE 3.3. The process *babaabaaabaaaabaaaaab...* cannot recursively be defined over + and ·.

THEOREM 3.5.

(i) *If* $p \in R(A^\infty)(+,\cdot,\|,\lfloor\!\lfloor)$, *that is: p can recursively be defined in the signature of PA, then Sub(p) is finitely generated (in the usual algebraic sense) over* $+,\cdot,\|,\lfloor\!\lfloor$.

(ii) *Likewise for the reduced signature* $+,\cdot$.

The last fact (ii) can be used to prove that Bag over a non-singleton domain cannot be recursively defined by + and alone; one must prove that Sub(Bag) (i.e. the collection $B_{mn}$ in figure 30, if $D=\{a,b\}$) cannot be finitely generated using +, only.

### *3.8. 1-1 communication*

We conclude this section with a theorem stating that binary communication may always be supposed to have a certain simple form.

Consider the alphabet $A=E\cup H$ where $H$ is the set of communication actions, so $H=\{a\in A \mid \exists b\; a|b\neq\delta\}$. Let communication be binary: $a|b|c=\delta$ for all $a,b,c\in A$.

We claim that without loss of defining power (on the external processes, where 'external' refers to $E^\infty$), the communication mechanism $H, |$ can be replaced by a 1-1 communication mechanism $H^*, |^*$. This means: there is a map $^{-}: H^*\rightarrow H^*$, such that $\bar{\bar{a}}=a$ and such that all proper communications have the form $a|\bar{a}=b$.

Let us be more precise about the phrase 'without loss of defining power on external processes'. The situation is as in figure 33:

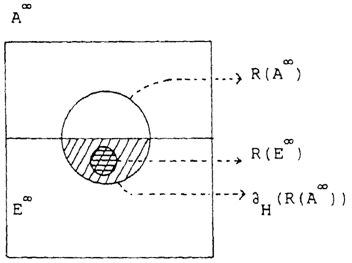
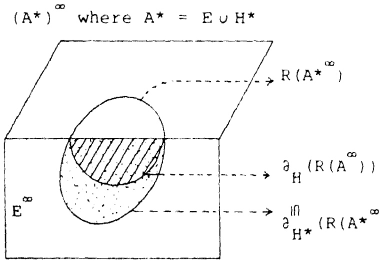


FIGURE 33



$(A^*)^\infty$ where $A^* = E\cup H^*$
$R(A^{*\infty})$
$\partial_H(R(A^\infty))$
$\cong$
$\partial_{H^*}(R(A^{*\infty}))$
$E^\infty$

FIGURE 34

In the original setting with $H$ and $|$ (see figure 33), the communication mechanism is able to define the 'external' processes (i.e. in $E^{\infty}$) contained in $\partial_H(R(A^{\infty}))$. This $\partial_H(R(A^{\infty}))$ is a subalgebra of $A^{\infty}$; it contains a subalgebra $R(E^{\infty})$, the external processes recursively definable without communication. Here the difference $\partial_H(R(A^{\infty}))-R(E^{\infty})$ is nonempty; i.e. communication yields more expressive power.

Now it is possible to replace $H,|$ by $H^{*},|^{*}$ such that the situation in figure 34 is obtained. That is, the new communication mechanism given by $H^{*}$, $|^{*}$ recursively defines at least as many processes as the old mechanism.

It is not hard to obtain as a next step an 1-1 *asymmetrical* communication function without impairing the expressive power. (A communication function is asymmetrical if for all $a$: $a|a=\delta$.) In fact, this is the communication format chosen in MILNER [14].

## 4. HIDING INTERNAL STEPS IN FINITE PROCESES

In this last section we will discuss the very fundamental problem of abstraction of internal steps ('hiding'). In a process one may wish to distinguish internal and external behaviour and to abstract from the former; obviously the availability of adequate abstraction mechanisms is of crucial importance for a hierarchical construction of systems.

In *trace semantics,* which may be viewed as the theory of $ACP$ augmented with the axiom $x(y+z)=xy+xz$, the abstraction problem seems easy: abstracting from the internal (or silent) steps $\tau$ (in Milner's notation) from a trace such as $ab\tau ac\tau\tau a$ results simply in $abaca$.

Also for *synchronous processes* as described in MILNER [15] abstraction from internal steps is easy: in a composite action (i.e. a simultaneous action of *all* ports, internal and external, of the network in consideration), say $e_1|e_2|e_3|i_1|i_2$ where $i_1,i_2$ are internal, the result after 'hiding' the internal steps is $e_1|e_2|e_3$. (The point here is that each composite action has a nonempty external part, so that hiding does not hide the whole action — therefore the choice structure is left intact.)

However, trace semantics does not respect and reflect deadlock behaviour; and synchronous process co-operation is in our view a special case of the more general mechanism of asynchronous process co-operation, cf. subsection 2.3.3. (MILNER [15] argues the reverse point of view, though.)

For asynchronous processes the initial temptation to treat internal or silent steps $\tau$ as above, like the unit element in group theory, that is via equations $x\tau=\tau x=x$, leads at once to difficulties in the presence of communication. Namely, the processes $a(\tau b+c)$ and $a(b+c)$ have different deadlock behaviour: let $c,c'$ be communication atoms such that $c|c'=c^{\circ}$ is the only proper communication (so $a|c'=\tau|c'=...=\delta$). Then for the context

$$C[\ ]=\partial_{\{c,c'\}}[\ldots\|c']$$

we have

$$C[a(\tau b+c)] = a(\tau\delta+c^{\circ})$$

$$C[a(b+c)] = ac^{\circ}.$$

In this section we will treat abstraction of internal steps for asynchronous processes. We will deal only with *finite* processes; here the theory exhibits some clarity. For infinite processes the situation is at present much less clear — for some comments see our 'concluding remarks' (4.3) at the end of this section.

### *4.1. Hiding internal steps in finite processes without communication: $PA_\tau$*

*4.1.1. Bisimulation modulo internal steps.* From now on, we consider the alphabet $A\cup\{\tau\}$, where $\tau$ is the silent or invisible step. A *trace* $\sigma$ is a possibly empty finite string over $A\cup\{\tau\}$ (thus $\sigma\in(A\cup\{\tau\}^{*})$. With $e(\sigma)$ we denote the trace $\sigma$ where all $\tau$-steps are erased.

Consider a finite acyclic process graph $g$ over $A\cup\{\tau\}$. A path $\pi{:}s_0 \twoheadrightarrow s_k$ in $g$ is a sequence of the form

$$s_0 \overset{l_0}{\underset{h_0}{\to}} s_1 \overset{l_1}{\underset{h_1}{\to}} \ldots \overset{l_{k-1}}{\underset{h_{k-1}}{\to}} s_k$$

($k\geqslant 0$) where the $s_i$ are nodes, the $h_i$ are edges between $s_i$ and $s_{i+1}$, and each $l_i$ is the label of edge $h_i$. (The $h_i$ are needed because we work with multigraphs.) The trace $trace(\pi)$ associated to this path is just $l_0l_1...l_{k-1}$.

DEFINITION 4.1. A *bisimulation modulo* $\tau$ between two finite acyclic process graphs $g_1$ and $g_2$ is a relation $R$ on NODES($g_1$)$\times$ NODES($g_2$) satisfying the following conditions:

(i) (ROOT($g_1$), ROOT($g_2$))$\in R$,

(ii) Domain ($R$) = NODES($g_1$) and Codomain($R$) = NODES($g_2$),

(iii) For each pair $(s_1,s_2)\in R$ and for each path $\pi_1$: $s_1 \twoheadrightarrow t_1$ in $g_1$ there is a path $\pi_2$: $s_2 \twoheadrightarrow t_2$ in $g_2$ such that $(t_1,t_2)\in R$ and $e(trace(\pi_1))=e(trace(\pi_2))$. (See figure 35.)

(iv) Likewise for each pair $(s_1,s_2)\in R$ and for each path $\pi_2$: $s_2 \twoheadrightarrow t_2$ in $g_2$ there is a path $\pi_1$: $s_1 \twoheadrightarrow t_1$ in $g_1$ such that $(t_1,t_2)\in R$ and $e(trace(\pi_1))=e(trace(\pi_2))$. (See figure 36.)

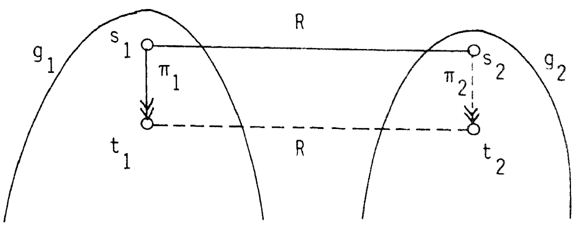


FIGURE 35

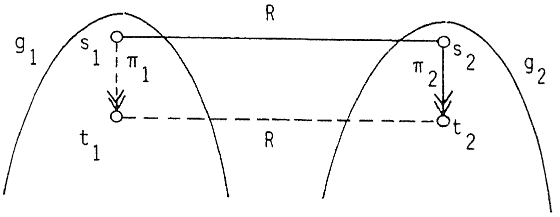


FIGURE 36

Process graphs $g_1,g_2$ are bisimilar modulo $\tau$ if there is a bisimulation modulo $\tau$ between $g_1,g_2$. Notation: $g_1 \underline{\leftrightarrow}_\tau g_2$.

The notion of bisimulation modulo $\tau$ specializes to the notion of bisimulation $\underline{\leftrightarrow}$ introduced in Section 1, where $\tau$ is not around. For technical reasons it is convenient to work with *rooted* bisimulation modulo $\tau$: here a root cannot be related to a nonroot node. If $g_1,g_2$ are bisimilar in this sense, we write $g_1 \underline{\leftrightarrow}_{r,\tau} g_2$. Also this notion of bisimulation specializes to $\underline{\leftrightarrow}$ in Section 1 (see 1.2.2.1).

EXAMPLES 4.1. $a\tau b \underline{\leftrightarrow}_{r,\tau} ab$ (see figure 37); $ab \underline{\leftrightarrow}_{r,\tau} a\tau(\tau b+\tau\tau b)$ (see figure 38); $a(\tau b+b) \underline{\leftrightarrow}_{r,\tau} ab$ (see figure 39); $c(a+b) \underline{\leftrightarrow}_{r,\tau} c(\tau(a+b)+a)$ (see figure 40).

A negative example: see figure 41. This was the example in the introduction to this section. The heavy line denotes where it is not possible to continue a construction of the bisimulation.

Another negative example: $a(\tau b+c) \not\underline{\leftrightarrow}_{r,\tau} a(b+c)+ab$.

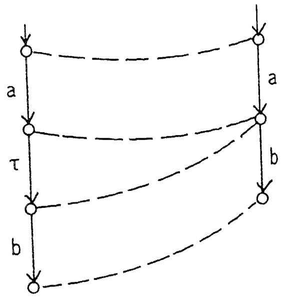


FIGURE 37

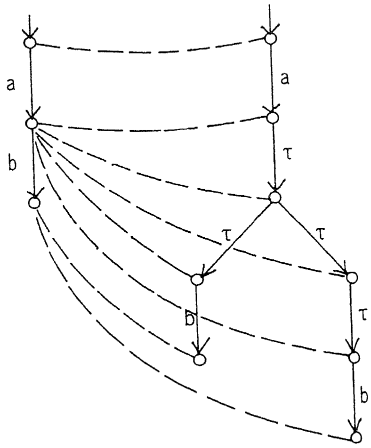


FIGURE 38

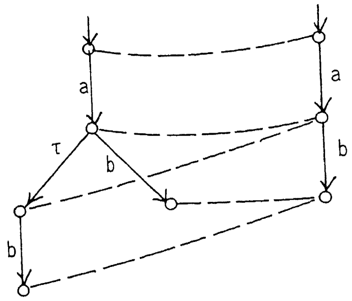


FIGURE 39

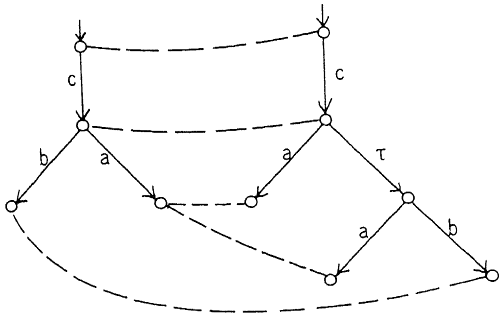


FIGURE 40

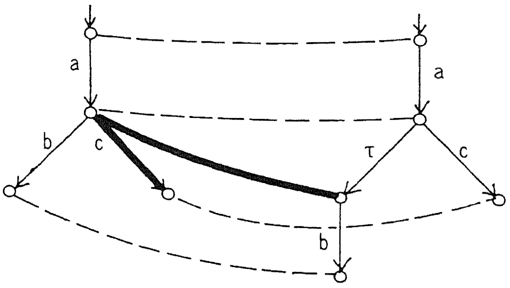


FIGURE 41

THEOREM 4.1. *Rooted bisimulation modulo $\tau$ is preserved by the operators* $+, \cdot, \lfloor\!\lfloor, \|$ *on finite acyclic process graphs.*

(This would not be true for bisimulation modulo $\tau$ without the 'rooted' condition. E.g. $a \underline{\leftrightarrow}_{\tau} a$, $\tau b \underline{\leftrightarrow}_{\tau} b$ but $a+\tau b \not\underline{\leftrightarrow}_{\tau} a+b$. Note that $\tau b \not\underline{\leftrightarrow}_{r,\tau} b$.)

COROLLARY 4.1. *The relation 'bisimilar modulo $\tau$' is a congruence on* $A_{\omega}(+, \cdot, \|, \lfloor\!\lfloor)$.

*4.1.2 Axioms for abstraction.* A beautiful result in MILNER [14] is that the semantical notion of $\underline{\leftrightarrow}_{r,\tau}$ congruence on finite processes *can be treated algebraically,* namely by three simple equations: Milner's $\tau$-laws T1, T2, T3. Added to $PA$ we obtain $PA_{\tau}$ as in Table 5, where the abstraction operator $\tau_I$ serves to 'internalize' steps. (Here $a \in A_{\tau}$.)

$PA_{\tau}$

| | | | |
|---|---|---|---|
| $x+y = y+x$ | A1 | $x\tau = x$ | T1 |
| $x+(y+z) = (x+y)+z$ | A2 | $\tau x + x = \tau x$ | T2 |
| $x+x = x$ | A3 | $a(\tau x + y) = a(\tau x + y) + ax$ | T3 |
| $(x+y)z = xz+yz$ | A4 | | |
| $(xy)z = x(yz)$ | A5 | | |
| | | $\tau_I(a) = a$ if $a \notin I$ | TI1 |
| $x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$ | M1 | $\tau_I(a) = \tau$ if $a \in I$ | TI2 |
| $a \lfloor\!\lfloor x = ax$ | M2 | $\tau_I(x+y) = \tau_I(x) + \tau_I(y)$ | TI3 |
| $(ax) \lfloor\!\lfloor y = a(x \| y)$ | M3 | $\tau_I(xy) = \tau_I(x) \cdot \tau_I(y)$ | TI4 |
| $(x+y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$ | M4 | | |

TABLE 5

THEOREM 4.2.
(i) *$PA_\tau$ is conservative over PA (the latter with actions from A).*
(ii) *The initial algebra of $PA_\tau$ is iomorphic to $A_\omega/\underline{\leftrightarrow}_{r,\tau}$ (the initial algebra of PA modulo the congruence of rooted bisimulation modulo $\tau$).*

*Stated differently: the $\tau$-laws T1-3 are a complete axiomatization of rooted bisimulation modulo $\tau$.*

Part (i) of the theorem states that $PA_\tau$ does not identify processes not containing $\tau$ which differ w.r.t. $PA$.

EXAMPLE 4.2. If the $\tau$-laws constitute a congruence, then since $PA_\tau \vdash a\tau = a$ we must also have $PA_\tau \vdash a\tau \lfloor\!\lfloor b = a \lfloor\!\lfloor b$. Indeed:

$$a\tau \lfloor\!\lfloor b \;=\; a(\tau \| b) = a(\tau b + b\tau) = a(\tau b + b) = a\tau b = ab = a \lfloor\!\lfloor b.$$

The following derivable identity is often useful:

PROPOSITION 4.1. $PA_\tau \vdash \tau(x+y)+x \;=\; \tau(x+y)$.

PROOF. $\tau(x+y) \;=\; \tau(x+y)+x+y = \tau(x+y)+x+y+x = \tau(x+y)+x.$

In [6] a proof of Theorem 4.2. is given along the following line. On the set of finite acyclic process graphs a reduction procedure is defined which simplifies the graph (lessens its number of edges and nodes) and which is *sound* for $\underline{\leftrightarrow}_{r,\tau}$. A *normal* proces graph is one in which no further reduction steps are possible. A *rigid* process graph is one which admits only the *trivial* rooted bisimulation *with itself*. (E.g. $a\tau b + ab$ is not rigid since it admits the nontrivial 'auto-bisimulation' as in figure 42.)

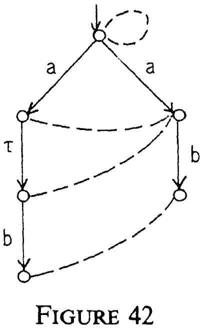


FIGURE 42

Now one can prove that (i) normal process graphs are rigid and (ii) rigid bisimilar process graphs must be identical. This together with the soundness yields the confluency property for the graph reduction procedure (the explicit confluency proof in [6] is in fact superfluous), which in turn implies the completeness of the graph reduction procedure w.r.t. $\underline{\leftrightarrow}_{r,\tau}$.

An example to see how the graph reduction procedure translates into the axioms T1-3: one of the reduction steps consists of replacing in a graph a part as in figure 43 (i) by the part in figure 43 (ii) (i.e. deleting an $a$-step).

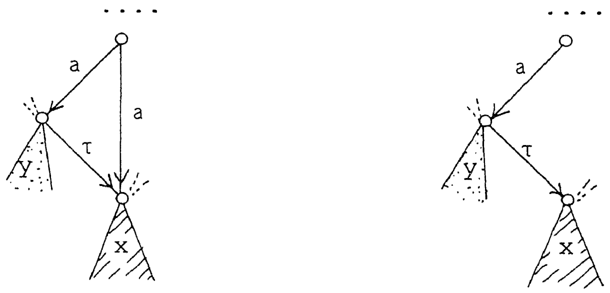


FIGURE 43 (i) (ii)

In terms of terms this amounts to $a(\tau x+y)+ax=a(\tau x+y)$.

We remark that the confluency result mentioned above only holds for the graph reduction procedure; when T1-3 are viewed as reduction rules (in whatever direction), together with a restatement of $PA$ as a rewrite system (i.e. choosing the direction left to right in all but the axioms for commutativity and associativity) confluency does *not* hold.

Before extending the $PA_\tau$-formalism with communication, we mention the following curious fact (which is significant for some choices in the development of the present theory):

PROPOSITION 4.2. *The equation $X=a+\tau X$ has infinitely many solutions in the initial model of $PA_\tau$.*

PROOF. If $p$ is a solution, then also $\tau(p+q)$ is a solution for arbitrary $q$:

$$\begin{aligned} a+\tau\tau(p+q) &= a+\tau(p+q)=a+p+\tau(p+q) \\ &= a+a+\tau p+\tau(p+q) \\ &= a+\tau p+\tau(p+q) = p+\tau(p+q)=\tau(p+q). \end{aligned}$$

Therefore, since $\tau a$ is a solution (by T1 and T2), $\tau(\tau a+q)$ solves the equation for arbitrary $q$. This proves the proposition.

Although we do not treat infinite processes here, we note as a corollary from this proposition that recursion equations, guarded by atoms from $A\cup\{\tau\}$, are no longer an adequate specification mechanism for infinite processes as they do not have unique solutions.

### *4.2. Hiding internal steps in finite processes with communication: $ACP_\tau$*

The virtue of the $\tau$-laws T1-3 is not yet fully realized in $PA_\tau$; it is more realized in the presence of communication — indeed the motivation for rejecting some alternative to the $\tau$-laws as in the example in the introduction to this section was stated in terms of communication behaviour. Therefore we want to combine $ACP$ with the $\tau$-laws; the result is the axiom system $ACP_\tau$ in Table 6.

It turns out that (apart from the $\tau$-laws) the atom $\tau$ must also in the axioms concerning '$|$' be treated differently from the $a \in A$; otherwise some desirable congruence properties are lost. Namely, a term as $\tau\tau a|\tau\tau b$ will be evaluated in $ACP_\tau$ as $a|b$ (and not as $(\tau|\tau)$ $(\tau a\|\tau\tau b)$ as $ACP$ would prescribe).

$ACP_\tau$

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

TABLE 6

Here the alphabet is $A \cup \{\tau\}$; and $a,b$ in Table 6 vary over $A$ only. In the renaming operators $\partial_H$, $\tau_I$ we require $\tau \notin H$ and $\delta \notin I$, since these constants should not be renamed.

In order to discuss some properties of $ACP_\tau$, we begin with establishing a graph model for $ACP_\tau$.

*4.2.1. The model of finite acyclic process graphs for $ACP_\tau$.* Consider, as in 4.1.1, the collection $\mathcal{X}$ of finite acyclic process graphs over $A \cup \{\tau\}$. In Theorem 4.2 (ii) it was stated that $\mathcal{X}/\underline{\leftrightarrow}_{r,\tau}$, i.e. the collection of finite acyclic graphs modulo rooted $\tau$-bisimulation, is (isomorphic to) the initial algebra of $PA_\tau$. (In fact, we used a loose formulation there by not distinguishing $\mathcal{X}$ from $A_\omega$.) We will now do the same in the context of $ACP_\tau$.

The operations $\|, \mathbin{\|\!\!\_}, |, \partial_H, \tau_I$ on $\mathcal{X}$ are defined as follows. The definition of $\partial_H$ and $\tau_I$ is clear — their effect is merely renaming some atoms (labels at the edges) into $\delta$ resp. $\tau$. The definition of $\|$ and $\mathbin{\|\!\!\_}$ is also easy: it is analogous to that for $ACP$ (see 2.1.2) with the additional communication $\tau|a=\delta$ for all $a \in A$ and $\tau|\tau=\delta$. The communication merge $g_1|g_2$ is different now:

$$g_1|g_2 = \sum\{(s \longrightarrow s')\,(g_1\|g_2)_{s'} \,|\, s \longrightarrow s' \text{ is a maximal com. step in } g_1\|g_2\}.$$

Here $(g)_s$ denotes the subgraph of $g$ with root $s$ ($\in$ NODES $(g)$) and 'maximal' refers to the accessibility ordering on EDGES$(g)$ (i.e. $s_1 \longrightarrow s_2$ is greater in this ordering than $s_3 \longrightarrow s_4$ if $s_3$ can be reached from $s_2$). A 'communication step' in $g_1\|g_2$ is one obtained as a 'diagonal' edge $\xrightarrow{a|b}$, resulting from the communication of $\xrightarrow{a}$and $\xrightarrow{b}$

The structure $X = \mathcal{X}(+, \,, \|, \mathbin{\|\!\!\_}, |, \partial_H, \tau_I, \delta, \tau)$ is not yet a model of $ACP_\tau$. It has a homomorphic image which is a model of $ACP_\tau$, and which is obtained by dividing out $\underline{\leftrightarrow}_{r,\tau}$, rooted $\tau$-bisimulation. To define $\underline{\leftrightarrow}_{r,\tau}$ on the elements of $\mathcal{X}$, we must extend the definition of $\underline{\leftrightarrow}_{r,\tau}$, given before, such that the presence of $\delta$'s in graphs is taken into account: this is done as above in 2.1.2, so that in effect we work with '$\delta$-normal graphs'.

Now one can prove the important fact:

LEMMA 4.1.
(i) *Rooted $\tau$-bisimulation is a congruence on* X.
(ii) $\mathcal{X}/\underline{\leftrightarrow}_{r,\tau} \models ACP_\tau$.

To prove this, we use results in [6] stating that $\underline{\leftrightarrow}_{r,\tau}$ can be analyzed into some elementary graph reductions which have the confluency property. Denoting the subset of axioms $A1-7$, $T1-3$ of $ACP_\tau$ by $AT$, we have, also essentially from [6], the following proposition.

PROPOSITION 4.2. *Let $t,s$ be terms built from $A \cup \{\tau\}$ by $+$ and only. Then:*

$$\mathcal{X}/\underline{\leftrightarrow}_{r,\tau} \models t=s \Rightarrow AT \vdash t=s.$$

Now consider $\Sigma = ACP_\tau - AT$, the set of axioms of $ACP_\tau$ minus $AT$. This set of axioms gives rise to a rewrite system (in fact on equivalence classes of terms modulo the associativity and commutativity axioms, $A$ 1,2,5) by choosing in every axiom the direction from left to right. Let $\xrightarrow{\Sigma}$be one step reduction, and $\overset{\Sigma}{\twoheadrightarrow}$ be the transitive reflexive closure of $\xrightarrow{\Sigma}$. The reductions in $\Sigma$ are confluent and terminating. Let $\xrightarrow{\Sigma}$denote reduction to the unique normal form. (Note that these normal forms are built by $+, \cdot$ only.) Then, applying Proposition 4.2 on $t_3, t_4$ in the diagram of figure 44, (together with Lemma 4.1 (ii)) we have immediately:

LEMMA 4.2.

(i) *I.e. if $ACP_\tau \vdash t_1 = t_2$, then $t_1$ and $t_2$ can be reduced by means of the rewrite rules (from left to right) associated to the axioms in $ACP_\tau - AT$ to normal forms $t_3, t_4$ which are convertible via the $AT$-axioms.*

(ii) *Every term $t$ can be proved equal in $ACP_\tau$ to a term $t'$ built from $A \cup \{\tau\}$ by $+$ and$\cdot$ only; moreover, $t'$ is unique modulo $\overset{\leftrightarrow}{=}_{r,\tau}$.*

$$\begin{tikzcd}
t_1 \arrow[r, equal, "ACP_\tau"'] \arrow[d, "ACP_\tau - AT"'] & t_2 \arrow[d, "ACP_\tau - AT"] \\
t_3 \arrow[r, equal, "\exists", "AT"'] & t_4
\end{tikzcd}$$

FIGURE 44

EXAMPLE 4.3. The following examples illustrate Lemma 4.2 (i):

$$\begin{tikzcd}
(\tau a + a)|b \arrow[r, phantom, "\underset{T2}{=}"] \arrow[d] & \tau a|b \arrow[dd] \\
\tau a|b + a|b \arrow[d] & \\
a|b + a|b \arrow[r, phantom, "="] & a|b
\end{tikzcd} \qquad \text{(i)}$$

$$\begin{tikzcd}
a\tau \mathbin{|\!\lfloor} b \arrow[rrrr, equal] \arrow[d] & & & & a \mathbin{|\!\lfloor} b \arrow[ddd] \\
a(\tau \| b) \arrow[d] & & & & \\
a(\tau \mathbin{|\!\lfloor} b + b \mathbin{|\!\lfloor} \tau + \tau|b) \arrow[d] & & & & \\
a(\tau b + b\tau + \delta) \arrow[r, phantom, "="] & a(\tau b + b\tau) \arrow[r, phantom, "="] & a(\tau b + b) \arrow[r, phantom, "="] & a\tau b \arrow[r, phantom, "="] & ab
\end{tikzcd} \qquad \text{(ii)}$$

$$\begin{tikzcd}
(\tau a + a)\mathbin{\|\!\_}b \arrow[d] & = & \tau a\mathbin{\|\!\_}b \arrow[d] \\
\tau a\mathbin{\|\!\_}b + a\mathbin{\|\!\_}b \arrow[d] & & \tau(a\|b) \arrow[ddd] \\
\tau(a\|b)+a\mathbin{\|\!\_}b \arrow[d] & & \\
\tau(a\mathbin{\|\!\_}b + b\mathbin{\|\!\_}a + a|b)+a\mathbin{\|\!\_}b \arrow[d] & & \\
\tau(ab+ba+a|b)+ab & \underset{(*)}{=} & \tau(ab+ba+a|b)
\end{tikzcd} \qquad \text{(iii)}$$

Here (*) is an instance of the (from $AT$) derivable rule $\tau(x+y)+x=\tau(x+y)$ as in Proposition 4.1.

A further corollary of Lemma 4.1 and 4.2 is:

THEOREM 4.3.
(i) $\mathcal{X}/\underline{\leftrightarrow}_{r,\tau}$ *is isomorphic to* $I(ACP_\tau)$, *the initial algebra of* $ACP_\tau$.
(ii) $ACP_\tau$ *is conservative over ACP (the latter over the alphabet A). I.e., for* $\tau$-*less terms* $t_1,t_2$: $ACP_\tau \vdash t_1=t_2 \Rightarrow ACP \vdash t_1=t_2$.

A corollary of Theorem 4.3 (i) and the fact that $\|$ in $ACP_\tau$ behaves like $\|$ in $ACP$ is the associativity of $\|$:

PROPOSITION 4.3. $I(ACP_\tau)\vDash x\|(y\|z) \;=\; (x\|y)\|z$

In fact, $I(ACP_\tau)$ satisfies all 'axioms of standard concurrency' as in 2.2 (Table 4) except the second one. Although this second axiom $(x|y)\mathbin{\|\!\_}z=x|(y\mathbin{\|\!\_}z)$ does not hold in $I(ACP_\tau)$, as can be seen by evaluating $(a|\tau b)\mathbin{\|\!\_}c$ to $(a|b)c$ and $a|(\tau b\mathbin{\|\!\_}c)$ to $(a|b)c+(a|c)b+a|b|c$, a restricted form does hold in $I(ACP_\tau)$, namely:

$$(x|ay)\mathbin{\|\!\_}z \;=\; x|(ay\mathbin{\|\!\_}z).$$

In view of the linearity of $|$ and $\mathbin{\|\!\_}$ this can be rephrased as follows: $I(ACP_\tau)$ $\vDash(x|y)\mathbin{\|\!\_}z=\; x|(y\mathbin{\|\!\_}z)$ for *stable* $y$. Here $y$ is stable, in the terminology of MILNER [14], if $y$ admits no $\tau$-stap as a first step.

Some other useful identities in $I(ACP_\tau)$ are:

$$x\|\tau y \;=\; \tau x\|y=\tau(x\|y)$$

$$x\mathbin{\|\!\_}\tau y \;=\; x\mathbin{\|\!\_}y, \quad x\mathbin{\|\!\_}\tau=x.$$

For a binary communication mechanism (so that the handshaking axiom $x|y|z=\delta$ holds) we have analogous to the Milner Expansion Theorem 2.2:

THEOREM 4.4 (EXPANSION THEOREM FOR $ACP_\tau$). *Let $a|b|c=\delta$ for all $a,b,c \in A$. Then, in the notation of Theorem 2.2:*

$$I(ACP_\tau) \vdash x_1\|\ldots\|x_k = \sum_{1\leqslant i\leqslant k} x_i \lfloor\!\lfloor X_k^i + \sum_{1\leqslant i<j\leqslant k} (x_i|x_j) \lfloor\!\lfloor X_k^{i,j}.$$

This is not a straightforward generalization of Theorem 2.2, since our proof of that theorem employed the axioms of standard concurrency (in Table 4) of which, as remarked above, the second one does not hold in $I(ACP_\tau)$.

The diagram in figure 45 gives an impression of the modular construction of $ACP_\tau$. Here $\Sigma_1 \trianglelefteq \Sigma_2$ means that $\Sigma_2$ is a conservative extension of $\Sigma_1$; for each axiom system part of the signature (viz. the alphabet) is mentioned.

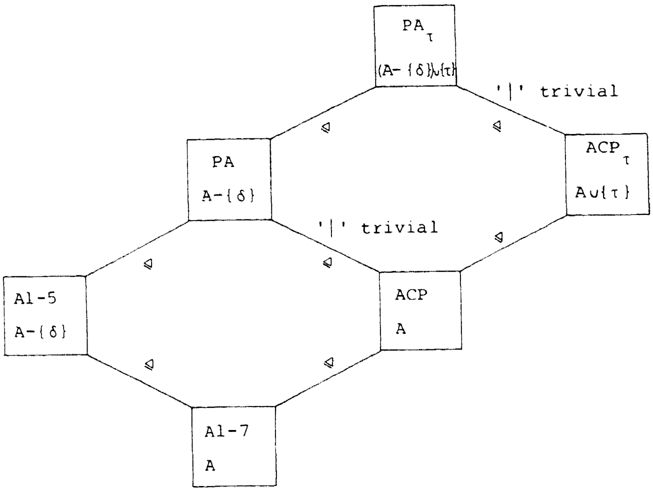


FIGURE 45

*4.3. Concluding remarks*

In [6] we have described an abstraction mechanism that is at least able to deal with the following situation: suppose two channels (say, bags $B_1,B_2$) are connected in series:

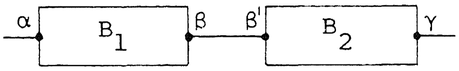


FIGURE 46

The result, clearly, is again a bag $B$; however in $B$ there are internal steps visible, viz. the passings of the data through the port connection $\beta-\beta'$. Now a minimal requirement for an adequate abstraction mechanism is that it can deal with such a simple situation: the mechanism should be able to hide the internal data transmissions and allow a proof that the connection of $B_1, B_2$ yields again a bag.

It is hard to find the 'canonical' extension of the above algebraic framework for finite processes with internal steps to infinite processes. This has to do with the possible presence of infinitely long traces of internal steps. E.g. the notion of bisimulation can be extended to the infinite case in several nonequivalent ways whose consequences are by no means immediately clear. One possibility, which is (formally) the straightforward generalization of $\underline{\leftrightarrow}_{r,\tau}$, admits the possibility of collapsing infinite $\tau$-traces; thus equating 'the' solution $X$ of the recursion equation $X = a + \tau X$ (which one would expect to be as in figure 47) with the finite process '$\tau a$'.

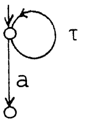


FIGURE 47

Two difficulties arise here, one technical, one conceptual. The technical problem was mentioned in the remark after Proposition 4.2: $X$ is *not* uniquely determined by $X = a + \tau X$. The conceptual problem is that equating $X$ with '$\tau a$' implies a certain fairness assumption, viz. that $X$ will not always take the option $\tau$. Interestingly, this built in fairness assumption can be used to attack the problem of protocol verification (where the fairness assumption is that a defective channel will not always be defective), as was pointed out to us by C.J. KOOMEN [13].

It is also possible to extend $\underline{\leftrightarrow}_{r,\tau}$ in another way, such that infinite $\tau$-traces cannot be collapsed. In that case '$\tau a$' and $X$ are different. It might be that here is a bifurcation point in the development of the theory.

However, for many purposes such as the one explained above (proving that composing two bags yields again a bag), one can work within the restricted algebra of finitely branching processes which are bounded in the sense of not having infinite $\tau$-traces. Here all 'reasonable' extensions of the concept of bisimulation coincide. In [6] an abstraction mechanism was worked out which essentially resides in this algebra of bounded processes.

Even though, maybe, the real interest is for infinite processes with invisible steps, it is certainly safe to say that an adequate algebraic framework to deal with them presupposes a clear understanding of such a framework for the finite case; and that was the subject of this last section.

## References

1. J.W. de Bakker, J.I. Zucker (1982). Denotational semantics of concurrency. *Proc. 14th ACM Symp. on Theory of Computing*, p. 153-158.
2. J.W. de Bakker, J.I. Zucker (1982). Processes and the denotational semantics of concurrency. *Information and Control, Vol. 54, No. 1/2*, 70-120.
3. J.A. Bergstra, J.W. Klop (1982). *Fixed Point Semantics in Process Algebras*, Department of Computer Science Technical Report IW 206/82, Mathematisch Centrum, Amsterdam.
4. J.A. Bergstra, J.W. Klop (1984). Process algebra for synchronous communication. *Information and Control 60 1-3*, 109-137.
5. J.A. Bergstra, J.W. Klop (1983). *A Process Algebra for the Operational Semantics of Static Data Flow Networks*, Department of Computer Science Technical Report IW 222/83, Mathematisch Centrum, Amsterdam.
6. J.A. Bergstra, J.W. Klop (1983). *An Abstraction Mechanism for Process Algebras*, Department of Computer Science Technical Report IW 231/83, Mathematisch Centrum, Amsterdam.
7. J.A. Bergstra, J.W. Klop (1983). *An Algebraic Specification Method for Processes over a Finite Action Set*, Department of Computer Science Technical Report IW 232/83, Mathematisch Centrum, Amsterdam.
8. J.A. Bergstra, J.W. Klop (1984). The algebra of recursively defined processes and the algebra of regular processes. J. Paredaens (ed.). *Proc. 11th ICALP Antwerpen*, Springer LNCS 172, 82-94.
9. J.A. Bergstra, J.W. Klop, J.V. Tucker (1983). Algebraic tools for system construction. E. Clarke, D. Kozen (eds.). *Logic of Programs, Proc. 1983*, Springer LNCS 164, 34-44.
10. J.A. Bergstra, J. Tiuryn (1983). *Process Algebra Semantics for Queues*, Department of Computer Science Technical Report IW 241/83, Mathematisch Centrum, Amsterdam.
11. M. Hennessy (1981). A term model for synchronous processes. *Information and Control, Vol. 51, No. 1*, 58-75.
12. C.A.R. Hoare (1980). A model for communicating sequential processes. R.M. McKeag, A.M. McNaghton (eds.). *On the construction of programs*, Cambridge University Press, 229-243.
13. C.J. Koomen (1983). Personal communication.
14. R. Milner (1980). *A Calculus for Communicating Systems*, Springer LNCS 92.
15. R. Milner (1983). Calculi for synchrony and asynchrony. *Theoretical Computer Science 25*, 267-310.
16. D.M.R. Park (1981). Concurrency and automata on infinite sequences. *Proc. 5th GI (Gesellschaft für Informatik) Conference*, Springer LNCS 104.