Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.A. Bergstra, J.W. Klop

Process algebra for communication and mutual exclusion
Revised version

Department of Computer Science Report CS-R8409 May

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

PROCESS ALGEBRA FOR COMMUNICATION AND MUTUAL EXCLUSION
REVISED VERSION

J.A. BERGSTRA, J.W. KLOP
Centre for Mathematics and Computer Science, Amsterdam

Within the context of an algebraic theory of processes we provide an equational specification of process co-operation. We consider four cases: free merge or interleaving, merging with communication, merging with mutual exclusion of critical sections and synchronous process co-operation. The rewrite system behind the communication algebra is shown to be confluent and terminating (modulo its permutative reductions). Further, some relationships are shown to hold between the four concepts of merging.

1980 MATHEMATICS SUBJECT CLASSIFICATION: 68B10, 68C01, 68D25, 68F20.
1982 CR CATEGORIES: F.1.1, F.1.2, F.3.2, F.4.3. 69F11, 69F12, 69F32, 69F43
KEY WORDS & PHRASES: concurrency, communicating processes, process algebra, merge, critical regions, asynchronous co-operation, synchronus co-operation.
NOTE: This report will be submitted for publication elsewhere.
This report is a revised version of the report IW 218/83.

Report CS-R8409
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

0. INTRODUCTION

0.1. General motivation: process algebra.

Our aim is to contribute to the theory of concurrency, along the lines of an algebraic approach. The importance of a proper understanding of the basic issues concerning the behaviour of concurrent systems or processes, such as communication, is nowadays evident, and various formats have been proposed as a framework for concurrency. Without claiming historical precision, it seems safe to say that the proper development of an *algebra* of processes starts with the work of R. Milner (see his introductory work [33]) in the form of his Calculus of Communicating Systems (CCS). Milner states his aim in [35] in his own words as follows: *"In a definitive calculus there should be as few operators or combinators as possible, each of which embodies some distinct and intuitive idea, and which together give completely general expressive power."* In [35] Milner proposes SCCS (Synchronous CCS) based on four fundamental operators, and remarks: *"These four operators obey (as we show) several algebraic identities. It is not too much to hope that a class of these identities may be isolated as axioms of an algebraic 'concurrency' theory, analogous (say) to rings or vector spaces."* These two quotations denote precisely the general motivation underlying also the present paper.

0.2. Aims of the present paper.

More specifically, in this paper we propose an algebra of processes based on elementary actions and on the operators + (alternative composition or choice), · (sequential composition or product) and $\|$ (parallel composition or merge). It turns out that in order to obtain an algebraically more satisfactory set of axioms, much is gained with our introduction of an auxiliary operator $\lfloor\!\lfloor$ (left-merge) which drastically simplifies computations and has some desirable 'metamathematical' consequences (finite axiomatisability if the alphabet of elementary actions is finite; greater suitability for term rewriting analysis) and moreover enhances the expressive power (more processes definable). Using these operators we have a framework for processes whose parallel execution is simply by interleaving ('free' merge): this is the axiom system PA in Table 2 in Section 1. The axiom system ACP presented below in Table 3 is devised to cover also processes that can communicate,

by sharing of actions. To this end a constant δ for deadlock (or failure) is introduced, another operator: '|' (communication merge), and finally, an operator ∂_H for 'encapsulation' of a process. Also this system, ACP for Algebra of Communicating Processes, is a finite axiomatisation of its intended models (which we call process algebras).

Clearly there is a strong relation of the system ACP below to the system CCS of Milner. In Milner [33] some process domains are discussed which can be seen as models of ACP. Determining the precise relationship is a matter of detailed investigation. In advance to that, one might say that ACP is an alternative formulation of CCS, at least of a part of CCS. (In this paper we do not discuss the so-called 'τ-steps', or silent steps, obtained by abstraction from 'internal' steps.) Notably, several of the ACP operators differ from those in CCS:

(i) multiplication $\cdot$ is general (not only prefix multiplication),

(ii) NIL is absent in ACP,

(iii) δ, $\between$ and $|$ are not present in CCS.

The merge operator $\|$ is the same as in CCS, though it is differently (namely: finitely) axiomatised. In ACP we have no explicit relabeling operators as in CCS, or 'morphisms' as they are called in Milner [35], except the encapsulation operators ∂_H which play the role of 'restriction' in CCS and SCCS.

Also in ACP we have no τ-steps (silent steps) and not the well-known τ-laws (in Milner [32]) for them; they can be added consistently, and even conservatively, to ACP. The resulting axiom system ACP_τ is studied in [9]. In general, ACP does not address the complicated problem of 'hiding' or abstraction in processes.

The choices of these operators can be seen as design decisions; of course the basic insights into the algebraic nature of communicating processes are already stated in Milner's book [33]. Some of these design decisions are motivated by our wish to optimize the facility of doing calculations; some others to enhance the expressive power of the system. For instance, having general multiplication available enables one to give a specification of the process behaviour of Stack in finitely many equations which can be proved to be impossible with prefix multiplication (see [8]).

An explicit concern in the choice of the axiom systems has been an attempt to modularize the problems. Thus 'PA' is only about interleaving or as we prefer to call it, free merge, that is without communication, ACP moreover treats communication, AMP treats the merge of processes with the restriction of mutual exclusion of critical regions, and ACP_τ treats abstraction.

Apart from the general motivation to use the system ACP for specification and verification of processes, we have been concerned in subsequent work with the detailed investigation of several of the models of ACP, as well as mathematical properties of this axiom system itself. Also some extensions of ACP were studied. This brings us to stating the aim of this paper: it is the first of our series of papers consisting of the present one and [6-11] on process algebra, meant firstly to present the system ACP and secondly to establish some of its basic mathematical properties (notably consistency of the axioms and a normal form theorem for process expressions). In the concluding remarks we elaborate on some applications which have been realised in these subsequent papers.

Though our central interest in this paper is for the 'general purpose system' ACP, we have also formulated some other 'special purpose' axiom systems: AMP for merging with mutual exclusion of critical sections; ACMP, a join of ACP and AMP; and ASP for synchronous process coöperation. Some relationships between these systems are shown, e.g. an interpretation of ASP in ACMP and an 'implementation' of AMP and ASP in ACP.

0.3. Related approaches.

Since this is not a survey paper and since there are several approaches related to the present one, it is not possible to discuss them while doing them justice or giving a complete view. Yet we want to mention the following lines of investigation. Closest to the present work (and its subsequent work in [6-11]) is Milner's CCS, which was above briefly compared with the axioms below. Interestingly, Milner has proposed in [35] a system SCCS which supersedes CCS and which has as fundamental notion: synchronous process co-operation. It is argued that asynchronous process co-operation (as in CCS and ACP) is a subcase in some sense of the former one. The terminology synchronous versus asynchronous is used in a different sense by different authors: see Remark 6.5 near the end of this paper. Again, it would be very useful and interes-

ting to determine the precise mathematical relationships between those systems for synchrony and asynchrony; a start has been made in Milner [35].

Milner's work has been continued and extended in Hennessy & Plotkin [24] and a series of papers by Hennessy [16-20] in which a detailed and extensive investigation is carried out often using operational preorders as a means of establishing completeness results of various proof systems. Completeness here is w.r.t. the semantical notions of observational equivalence and/or versions of bisimulation. Hennessy [18,20] also studies the differentiations of '+' according to whether a choice is made by the process itself or by its environment. Further, the work of Hennessy and Milner obtains several results in terms of modal characterisations of observational equivalence [20-22]. (See also Graf & Sifakis [15] and Brookes & Rounds [13].)

Milne [30,31] presents the 'Dot calculus': here '•' is concurrent composition. The Dot calculus uses prefix multiplication as in the work of Milner and Hennessy (called 'guarding' here), operators $+, \oplus$ for choice (by environment resp. internal), Δ for deadlock as well as successful termination. In contrast to CCS as in [33], the Dot calculus supports not only binary communication but n-ary communication. (The latter is also present in subsequent work of Milner and Hennessy; and also in ACP.) The Dot calculus presents algebraic laws for its operators; for '•' these are rather different than the ones for the corresponding parallel composition operators in CCS and ACP.

In our view there is a noteworthy methodological difference between the approaches as mentioned above and the present one. Namely: it has been an explicit concern of us to state first a system of axioms for communicating processes (of course based on some a priori considerations of what features communicating processes should certainly have) and next study its models; the analogy with the axiomatic method in say group theory or the theory of vector spaces is clear. For instance, one can study a model of ACP containing only 'finitely branching' processes; or one might be interested in processes which admit infinite branchings (in the sense of '+'); or, one may study the process algebra of regular processes, i.e. processes with finitely many 'states' (cf. Milner [34], Bergstra & Klop [8]). Also, one may build process algebras based on the fundamental and fruitful notion of bisimulation (introduced by Park [39]), as is done in e.g. Milner [34,35]; or one may consider process algebras obtained by the purely algebraic construction of taking a

projective limit (of process algebras consisting of finitely deep processes). This list could be extended to some dozens of interesting process algebras, all embodying different possible aspects of processes. To the best of our knowledge, an explicit adherence to this axiomatic methodology as we are aiming at, is as yet not fully represented in related approaches to the understanding of concurrency.

As some other related approaches which are less algebraical in spirit than the fore-mentioned (CCS, SCCS, Dot calculus,ACP) and which have a more denotational style we mention the work of De Bakker & Zucker [3,4]. They have studied several process domains as solutions of domain equations, using topological techniques and concepts such as metrical completion, compactness. In fact, their domain of 'uniform' processes and a question thereabout (see [3]) were our incentive to formulate PA as in Table 2 below. The processes of De Bakker and Zucker include several programming concepts which are not discussed in ACP. In De Bakker e.a. [5] the central issue of LT (linear time) versus BT (branching time), which determines the essential difference between trace sets and processes, has been studied. Denotational models for communicating processes as in Hoare's CSP (see [31,32]) have also been discussed from a uniform point of view in Olderog & Hoare [38]. For work discussing aspects of CCS and CSP, as well as connections between these two, we refer to Brookes [12]. Other work on concurrency in the denotational style includes Back & Mannila [1,2], Pratt [40] and Staples & Nguyen [42]. Finally, Winskel [43,44] discusses communication formats in languages such as CCS, CSP.

1. PRELIMINARIES: PROCESSES WITH ALTERNATIVE AND SEQUENTIAL COMPOSITION

Let A be a finite collection (alphabet) of atomic actions a,b,c,... . (We insist on a finite alphabet to safeguard the algebraic nature of the present work; specifically we wish to avoid here infinite sums whose algebraic specification is much less obvious than that of finite sums.)

Finite processes are generated from the atomic processes in A using the two 'basic' operations:

+ : *alternative composition (choice)*

. : *sequential composition (product)*

The following equational laws will hold for finite processes. (BPA stands for basic process algebra.)

BPA

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

Table 1.

Here x,y,z vary over processes. Often $x \cdot y$ is written as xy. The initial term algebra of these equations is $(A_\omega, +, \cdot)$. The elements of this algebra will be called "basic terms", i.e. terms modulo A1-5.

The main source of process algebra in this style is Milner [33]. Exactly the above processes occur as finite uniform processes in De Bakker & Zucker [3,4]. After adding an extra equation: $x(y + z) = xy + xz$, one obtains a version of trace theory as described in Rem [41].

For $n \geqslant 1$ we have the approximation map $\pi_n : A_\omega \longrightarrow A_\omega$, inductively described by

$$\pi_n(x + y) = \pi_n(x) + \pi_n(y)$$

$$\pi_n(a) = a$$

$$\pi_1(ax) = a$$

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

Interestingly, if $A_n = \{\pi_n(p) \mid p \in A\}$ then $(A_n, +_n, \cdot_n)$ is another model of BPA. Here the operations $+_n$ and $\cdot_n$ are defined by

$$x +_n y = \pi_n(x + y)$$

and likewise for product.

Infinite processes can be obtained as a projective limit, called A^∞, of the structures A_n. Technically this means that A^∞ is the set of all sequences $p = (p_1, p_2, p_3, \ldots)$ with $p_i \in A_i$ and $p_i = \pi_i(p_{i+1})$. Such sequences are called projective sequences. The operations '+' and '•' on A^∞ are defined component-wise:

$$(p+q)_n = (p)_n + (q)_n,$$

$$(p \cdot q)_n = \pi_n((p)_n \cdot (q)_n)$$

where $(p)_n$ is the n-th component of p. Thus we obtain the process algebra $(A^\infty, +, \cdot)$. On A^∞ a metric exists:

$$d(p,q) = \begin{cases} 0 \text{ if } p = q \\ 2^{-n} \text{ with n minimal such that } (p)_n \neq (q)_n \text{ if } p \neq q. \end{cases}$$

(A^∞, d) is a complete metric space, in fact it is the metric completion of (A_ω, d). The operations + and • are continuous. (A^∞, d) was introduced in De Bakker & Zucker [3]. Milner [34] uses charts modulo bisimulation (from Park [39]) to obtain infinite processes from finite ones. Working with trace sets under the extra assumption $x(y+z) = xy + xz$, this metric occurs in Nivat [36]. In De Bakker et. al. [5] the connections between (A^∞, d) and its corresponding trace space are investigated.

The processes discussed so far are provided with a bare minimum of structure. The crux of the algebraic method lies in algebraically defining new operators over the given process domains that will correspond to important process composition principles. We will describe operators corresponding to the following composition principles:

(i) *free merge* (Section 2)

(ii) merging with *communication* (Section 3)

(iii) merging processes with *mutual exclusion* for critical sections (Section 4)

(iv) merging with *communication and mutual exclusion* (Section 5)

(v) merging with *synchronous communication* (Section 6).

2. FREE MERGE: THE AXIOM SYSTEM PA

The result of merging processes p and q is $p \| q$. For algebraic reasons (finite axiomatisability and ease of computation) an auxiliary operation $\lfloor\!\lfloor$ (left merge) is used. The process $p \lfloor\!\lfloor q$ stands for the result of merging p and q but with the constraint that the first step must be one from p. Both operations $\|$ and $\lfloor\!\lfloor$ are specified on $(A_\omega, +, \cdot)$ by the equations M1-4 of the axiom system PA in Table 2:

PA

$x + y = y + x$	A1
$x + (y + z) = (x + y) + z$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$	M1
$a \lfloor\!\lfloor x = ax$	M2
$ax \lfloor\!\lfloor y = a(x \| y)$	M3
$(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	M4

Table 2.

We call the set of axioms A1-5 (i.e. BPA) together with M1-4: PA. This axiom system describes the interleaving of processes without communication, or as we prefer to call it, the free merge of processes. In Table 2, x,y,z vary over all processes (i.e. elements of an algebra satisfying PA), while 'a' is a variable over A. (This means that M2,3 are axiom schemes, having finitely many axioms as instances.)

Again the operations are extended to A^∞ coördinate-wise:

$$(p_1,p_2,\ldots) \,\|\, (q_1,q_2,\ldots) = (\pi_1(p_1 \| q_1), \pi_2(p_2 \| q_2), \ldots)$$

and likewise for $\lfloor\!\lfloor$. We omit the proof that these are indeed projective sequences, i.e. that

$$\pi_n(\pi_{n+1}(p_{n+1} \| q_{n+1})) = \pi_n(p_n \| q_n),$$

and likewise for $\lfloor\!\lfloor$. It also follows that $\|$ and $\lfloor\!\lfloor$ are continuous w.r.t. the metric d.

3. MERGING WITH COMMUNICATION: THE AXIOM SYSTEM ACP

In order to describe communication we will need a distinguished symbol $\delta \in A$, describing deadlock or failure. It is subject to the axioms $x + \delta = x$ and $\delta x = \delta$ (A6,7 in Table 3 below); δ can be seen intuitively as the 'action' by which a process acknowledges that it is stagnating.

Now, starting with $(A_\omega, +, \cdot)$ plus a communication function $.|. : A \times A \to A$ which describes the effect of sharing (simultaneously executing) two atomic actions, three operations $\|$, $\lfloor\!\lfloor$ and $|$ are defined on A_ω. Here '$|$', the communication merge, extends the given communication function. The operators $\|$ and $\lfloor\!\lfloor$ coincide with the analogous operators defined in Section 2 if the effect of a communication $a|b$ is always δ (i.e. no two atomic actions communicate).

For the communication function we require commutativity, associativity and $\delta|a = \delta$ for all $a \in A$ (resp. C1,2,3 in Table 3). The actions c for which there exists an action c' such that $c|c' \neq \delta$ are called subatomic or communication actions.

Furthermore, $\|$, $\lfloor\!\lfloor$ and $|$ are specified by the axioms CM1-9 in Table 3. (See next page.) Table 3 contains the axiom system ACP, for Algebra of Communicating Processes. Here the subset $H \subseteq A$ is a parameter of ∂_H, the encapsulation operator. Its function is to encapsulate a process p w.r.t. H, that is: $\partial_H(p)$ cannot communicate with its environment via communication actions in H. In Table 3, 'a' and 'b' range over the alphabet A.

Note that in general $\partial_H(x\|y) \neq \partial_H(x)\|\partial_H(y)$. Thus ∂_H is a homomorphism on $(A_\omega, +, \cdot, \delta)$, the initial algebra of axioms A1-7, but not on $(A_\omega, +, \cdot, \|, \lfloor\!\lfloor, |, \delta)$.

An important observation concerning the difference between processes and trace sets is exhibited in the following example. Let $A = \{a, c_1, c_2, c, \delta\}$ and let $c_1|c_2 = c$. All other communications result in δ. Now, writing ∂ for $\partial_{\{c_1,c_2\}}$, we have

$$\partial(a(c_1 + c_2) \| c_1 = ac, \text{ and } \partial((ac_1 + ac_2) \| c_1) = ac + a\delta$$

so the second process $ac_1 + ac_2$ has a deadlock possibility in some context where the first one, $a(c_1 + c_2)$, has not.

As before $\|$, $\between$, $|$ and ∂_H can be extended to continuous operations on (A^∞, d).

This formalism includes both message passing and synchronisation. In Milner [33] and De Bakker & Zucker [3,4] synchronisation is modeled by having $a|b = \tau$ whenever $a|b \neq \delta$, τ denoting a silent move. (In this paper we will not consider τ-steps.)

ACP

$x + y = y + x$	A1
$x + (y + z) = (x + y) + z$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x + \delta = x$	A6
$\delta x = \delta$	A7
$a\|b = b\|a$	C1
$(a\|b)\|c = a\|(b\|c)$	C2
$\delta\|a = \delta$	C3
$x\|\|y = x \between y + y \between x + x\|y$	CM1
$a \between x = ax$	CM2
$(ax) \between y = a(x\|\|y)$	CM3
$(x + y) \between z = x \between z + y \between z$	CM4
$(ax)\|b = (a\|b)x$	CM5
$a\|(bx) = (a\|b)x$	CM6
$(ax)\|(by) = (a\|b)(x\|\|y)$	CM7
$(x + y)\|z = x\|z + y\|z$	CM8
$x\|(y + z) = x\|y + x\|z$	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
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$	D4

Table 3.

3.1. Remark: a comparison with some operators in related work.

(i) Milne [30] employs an operator Δ with the axiom $x + \Delta = x$, as our A6. However, Δ denotes there not only deadlock but also successful termination. The same is the case for Milner's constant NIL in [33]. On the other hand, δ as in Table 3 corresponds precisely to the 'empty' process $\emptyset$ in the domain of uniform processes of De Bakker & Zucker [3,4]. There a process ends (in a terminating branch) either in a stop process p_0 (successfully) or in $\emptyset$ (deadlock).

(ii) Requirements on communication similar to C1-3 are found in Hennessy [17], except that δ is absent there but a unit element 1 is present. I.e. $<A,|,1>$ is an abelian monoid. See also Milner [35], who has similar postulates, viz. $<A,|>$ is an abelian semi-group; he also works with $<A,|,1,^->$ as a commutative group.

(iii) In Hennessy & Plotkin [24] a definition corresponding to the equation CM1: $x||y = x\mathbin{\llcorner} y + y\mathbin{\llcorner} x + x|y$ occurs.

(iv) In Hennessy [16] an auxiliary operator γ is used which is related to our auxiliary operators $\llcorner$ and $|$ as follows:

$$x \gamma y = x\mathbin{\llcorner} y + x|y.$$

Then one has

$$x||y = x \gamma y + y \gamma x;$$

also γ is linear in its left component:

$$(x+y) \gamma z = x \gamma z + y \gamma z$$

(This follows by axioms CM4,8 in Table 3.) The operator γ does not seem to yield a finite axiomatisation, however. Of course in the absence of communication, i.e. $x|y = \delta$, so that ACP 'reduces to' PA, the operators γ and $\llcorner$ coincide.

3.2. ACP seems to provide a concise formulation of the algebraic essence of communication. Therefore we review its structure in detail here. We will show that the new operators are indeed well-defined by A6,7,CM1-9,D1-4 over A1-5 + C1-3. We will to this end rearrange ACP into a TRS (term rewrite system) which is shown to be confluent and strongly terminating modulo the permutative reductions A1,2. As a consequence we find that each term built from

A by $+, \cdot, \|, \between, |, \partial_H$ can be proved equal to a unique term in A_ω in ACP.

Finally we prove that $\|$ is associative, as well as several other useful identities in Theorem 3.3.

For technical reasons we associate to each $a \in A$ a unary operator a^* which acts as follows:

$$a^*x = a \cdot x.$$

(That is, we consider the restriction to prefix-multiplication as in Milner [33-35]. For finite processes, as we will consider in the following analysis, general multiplication and prefix-multiplication are equivalent. Working with prefix-multiplication frees us from considering the permutative axiom A5, which is bothersome in a term rewriting analysis, in Table 3.)

On the term system generated by $A, +, \cdot, \|, \between, |, a^* \ (a \in A), \partial_H$ we introduce two norms $|.|$ and $\|.\|$. Here intuitively $|S|$ computes an upper bound for the path lengths in S and $\|S\|$ computes an upper bound of the number of (nontrivial) summands in which S decomposes. (See Table 4.)

$\|a\| = 1$	$\|\|a\|\| = 1$
$\|a^*x\| = 1 + \|x\|$	$\|\|a^*x\|\| = 1$
$\|x \cdot y\| = \|x\| + \|y\|$	$\|\|x \cdot y\|\| = \|\|x\|\|$
$\|x + y\| = \max(\|x\|, \|y\|)$	$\|\|x + y\|\| = \|\|x\|\| + \|\|y\|\|$
$\|x \| y\| = \|x\| + \|y\| - 1$	$\|\|x \| y\|\| = \|\|x\|\| \cdot \|\|y\|\|$
$\|x \between y\| = \|x\| + \|y\|$	$\|\|x \between y\|\| = \|\|x\|\|$
$\|x \| y\| = \|x\| + \|y\|$	$\|\|x \| y\|\| = \|\|x\|\| + \|\|y\|\| + \|\|x\|\| \cdot \|\|y\|\|$
$\|\partial_H(x)\| = \|x\|$	$\|\|\partial_H(x)\|\| = \|\|x\|\|$

Table 4.

Now consider the following term rewrite system RACP (which will only be needed for the proof of Theorem 3.3) in Table 5 below. Here in RCM5'-7 the symbol $c_{a,b}$ denotes the atom $a|b \in A$. The axioms C1-3 of ACP translate into the commutativity and associativity of c and $c_{\delta,a} = \delta$ for all $a \in A$.

RACP

$x+y \to y+x$	RA1
$x+(y+z) \to (x+y)+z$	RA2
$(x+y)+z \to x+(y+z)$	RA2'
$x+x \to x$	RA3
$(x+y)\cdot z \to x\cdot z + y\cdot z$	RA4
$a\cdot x \to a*x$	RA5'
$(a*x)\cdot y \to a*(x\cdot y)$	RA5
$x+\delta \to x$	RA6
$\delta*x \to \delta$	RA7
$x \| y \to x \lfloor\!\lfloor y + y \lfloor\!\lfloor x + x \mid y$	RCM1
$a \lfloor\!\lfloor x \to a*x$	RCM2
$(a*x) \lfloor\!\lfloor y \to a*(x \| y)$	RCM3
$(x+y) \lfloor\!\lfloor z \to x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	RCM4
$a \mid b \to c_{a,b}$	RCM5'
$(a*x) \mid b \to c^*_{a,b} x$	RCM5
$a \mid b*x \to c^*_{a,b} x$	RCM6
$(a*x) \mid (b*y) \to c^*_{a,b}(x \| y)$	RCM7
$(x+y) \mid z \to x \mid z + y \mid z$	RCM8
$x \mid (y+z) \to x \mid y + x \mid z$	RCM9
$\partial_H(a) \to a$ if $a \notin H$	RD1
$\partial_H(a) \to \delta$ if $a \in H$	RD2
$\partial_H(x+y) \to \partial_H(x) + \partial_H(y)$	RD3
$\partial_H(x\cdot y) \to \partial_H(x)\cdot\partial_H(y)$	RD4
$\partial_H(a*x) \to a*\partial_H(x)$ if $a \notin H$	RD1'
$\partial_H(a*x) \to \delta*\partial_H(x)$ if $a \in H$	RD2'

Table 5.

In the following theorem, $=_R$ denotes convertibility in RACP (i.e. the equivalence relation generated by $\to$).

3.3. THEOREM. *For all* ACP*-terms without variables:*

(i) $ACP \vdash S = T \iff S =_R T$

(ii) $ACP \vdash S = S'$ *for some* S' *not containing* $\|, \lfloor\!\lfloor, \mid, \partial_H$

(iii) $ACP \vdash S' = S'' \iff A1\text{-}7 \vdash S' = S''$ *for* S', S'' *not containing* $\|, \lfloor\!\lfloor, \mid, \partial_H$

(iv) $S\cdot(T\cdot U) =_R (S\cdot T)\cdot U$

(v) RACP *is weakly confluent, working modulo* A1,2.

(vi) RACP *is strongly terminating, modulo* A1,2.

(vii) RACP *is confluent (has the Church-Rosser property).*

PROOF. We start with (vi) and we introduce the auxiliary notion of the multiset of direct subterms DS(T) of a term T:

$$DS(a) = \emptyset$$
$$DS(a*x) = DS(x)$$
$$DS(x+y) = DS(x) \dot\cup DS(y)$$
$$DS(x \Box y) = \{x \Box y\} \dot\cup DS(x) \dot\cup DS(y) \quad \text{(here } \Box \text{ is } \cdot, \|, \lfloor\!\lfloor, \text{ or } |\text{)}$$
$$DS(\partial_H(x)) = DS(x)$$

Here $\dot\cup$ denotes the multiset union. Let [S] be the mapping from terms to $\omega\times\omega$ defined by

$$[S] = (|S|, \|S\|).$$

This mapping is extended to multisets over terms, thus producing multisets over $\omega\times\omega$:

$$[V] = \{[S] \mid S \in V\}.$$

On $\omega \times \omega$ there is the lexicographic well-ordering $<$ which induces a well-ordering $\ll$ on finite multisets over $\omega \times \omega$. We now observe that along a reduction path

$$T_0 \xrightarrow[R_0]{} T_1 \xrightarrow[R_1]{} T_2 \xrightarrow[R_2]{} \cdots$$

we have

$[DS(T_i)] \gg [DS(T_{i+1})]$ if R_i is not RA1, RA2, RA2', and

$[DS(T_i)] = [DS(T_{i+1})]$ if R_i is RA1, RA2 or RA2'.

From this observation strong termination of RACP modulo A1 and A2 follows.

Instead of a proof of the observation we provide two characteristic examples.

(1) $a\cdot x \to a*x$. Then:

$[DS(a\cdot x)] = [a\cdot x] \dot\cup [DS(x)]$ and $[DS(a*x)] = [DS(x)]$.

Now $[a\cdot x]$ majorizes each element of $[DS(x)]$ because

$[S] \in [DS(x)] \Rightarrow |S| \leq |x| \Rightarrow |S| < |a\cdot x|$. Hence $[DS(a\cdot x)] \gg [DS(a*x)]$.

(2) $x||y \to x\llcorner y + y\llcorner x + x|y$. Then:

$[DS(x||y)] = [x||y] \dot{\cup} [DS(x)] \dot{\cup} [DS(y)]$ and

$[DS(x\llcorner y + y\llcorner x + x|y)] = [x\llcorner y] \dot{\cup} [DS(x)] \dot{\cup} [DS(y)] \dot{\cup}$
$[y\llcorner x] \dot{\cup} [DS(x)] \dot{\cup} [DS(y)] \dot{\cup}$
$[x|y] \dot{\cup} [DS(x)] \dot{\cup} [DS(y)]$.

Again $[x||y]$ majorizes all of $[x\llcorner y]$, $[y\llcorner x]$, $[x|y]$, $[DS(x)]$, $[DS(y)]$, the first three in width and the second two in depth.

An alternative proof of termination can be given by ranking all occurrences of $||, \llcorner, |$ by the $|.|$-norm of the term of which they are the leading operator. Using this extended set of operators a recursive path ordering can be found which is decreasing in all rewrite steps except the first three (RA1,RA2,RA2'). See DERSHOWITZ [14]. A proof along this line has been given in [9].

Proof of (v). RACP is weakly confluent modulo $\sim$, the congruence generated by A1 and A2. (We are here working in congruence classes and reductions have the form $[S]^{\sim} \to [S']^{\sim}$ whenever $S \to S'$.) This is a matter of some 400 straightforward verifications. (Of course left to the reader as an exercise.)

Proof of (vii). Working modulo $\sim$ RACP is strongly terminating in view of (vi). Now combining (v) and (vi) and using Newman's Lemma (see [29], Lemma 5.7.(1) or [28] where more information about reduction modulo equivalence can be found) we find that RACP is confluent modulo $\sim$ and consequently it is confluent because the reductions generating $\sim$ are symmetric.

Proof of (ii). This follows immediately from (vi).

Proof of (iv). First one proves the associativity of $\cdot$ for terms not containing $||, \llcorner, |, \partial_H$ using induction on the structure of S. The result then immediately follows using (ii).

Proof of (i). $S =_R T \implies ACP \vdash S = T$ is immediate. For the other direction one uses (iv).

Proof of (iii). If $ACP \vdash S' = S''$ then by (i) $S' =_R S''$ and by (vii) for some S''': $S' \twoheadrightarrow S'''$ and $S'' \twoheadrightarrow S'''$ (here $\twoheadrightarrow$ is the transitive reflexive closure of $\to$). Now because S' and S'' are free of $||, \llcorner, |, \partial_H$ we see that $S' \twoheadrightarrow S''' \twoheadleftarrow S''$ is just a proof in A1,...,A7.

3.4. THEOREM. *The following identities hold in* $(A_\omega,+,\cdot,\|,\between,|,\partial_H)$:

(1) $x|y = y|x$

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

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

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

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

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

PROOF. All proofs use induction on the structure of x,y,x written as a term over $(A,+,\cdot)$, which is justified by Theorem 3.3.(2). We write

$$x = \sum_i a_i x_i + \sum_j a'_j$$
$$y = \sum_k b_k y_k + \sum_\ell b'_\ell$$
$$z = \sum_m c_m z_m + \sum_n c'_n .$$

(1) and (2) are proved in a simultaneous induction.

$$x|y = \sum(a_i|b_k)(x_i\|y_k) + \sum(a_i|b'_\ell)x_i + \sum(a'_j|b_k)y_k + \sum(a'_j|b'_\ell) =$$
$$= \sum(b_k|a_i)(y_k\|x_i) + \sum(b'_\ell|a_i)x_i + \sum(b_k|a'_j)y_k + \sum(b'_\ell|a'_j) =$$
$$y|x.$$

Here we use C1 and the induction hypothesis for $x_i\|y_k = y_k\|x_i$.

(2): $x\|y = x\between y + y\between x + x|y = y\between x + x\between y + y|x = y\|x$.

The proof of (3),...,(6) is also done using one simultaneous induction.

(3): Write $x = x' + x''$ where $x' = \sum a_i x_i$ and $x'' = \sum a'_j$. Likewise $y = y' + y''$ and $z = z' + z''$. Then

$$x|(y|z) = x'|(y'|z') + x'|(y''|z') + x'|(y'|z'') + x'|(y''|z'') +$$
$$x''|(y'|z') + x''|(y''|z') + x''|(y'|z'') + x''|(y''|z'').$$

Now $x'|(y'|z') = \sum(a_i|(b_k|c_m))(x_i||(y_k\|z_m)) =$

$\sum((a_i|b_k)|c_m)((x_i||y_k)\|z_m) =$

$(x'|y')|z'.$

Here we used C2 and the induction hypothesis for (6). The other summands of $x|(y|z)$ are treated similarly. Hence $x|(y|z) = (x|y)|z$.

(4): $(x\,\underline{\|}\,y)\,\underline{\|}\,z = ((\sum a_i x_i + \sum a'_j)\,\underline{\|}\,y)\,\underline{\|}\,z = (\sum a_i(x_i\|y) + \sum a'_j.y)\,\underline{\|}\,z =$

$\sum a_i((x_i\|y)\|z) + \sum a'_j(y\|z) =$ (induction hypothesis on (6))

$\sum a_i(x_i||(y\|z)) + \sum a'_j(y\|z) =$

$(\sum a_i x_i + \sum a'_j)\,\underline{\|}\,(y\|z) =$

$x\,\underline{\|}\,(y\|z).$

(5): Let $x = x' + x''$ and $y = y' + y''$ as in the proof of (3). Then

$x|(y\,\underline{|}\,z) = x'|(y'\,\underline{|}\,z) + x'|(y''\,\underline{|}\,z) + x''|(y'\,\underline{|}\,z) + x''|(y''\,\underline{|}\,z).$

Now $x'|(y'\,\underline{|}\,z) = (\sum a_i x_i)|(\sum b_k y_k)\,\underline{|}\,z) = (\sum a_i x_i)|(\sum b_k(y_k\|z)) =$

$\sum(a_i|b_k)(x_i||(y_k\|z)) =$ (induction hypothesis on (6))

$\sum(a_i|b_k)((x_i||y_k)\|z) = (\sum(a_i|b_k)(x_i||y_k))\,\underline{|}\,z =$

$(x'|y')\,\underline{|}\,z.$

The other three summands are treated similarly. Hence $x|(y\,\underline{|}\,z) = (x|y)\,\underline{|}\,z$.

(6) Write $A_x(y,z) = x\,\underline{|}\,(y\|z)$ and $B_x(y,z) = (y|z)\,\underline{|}\,x$. Then:

$x||(y\|z) = x\,\underline{|}\,(y\|z) + (y||z)\,\underline{|}\,x + x|(y||z) =$

$A_x(y,z) + (y\,\underline{|}\,z)\,\underline{|}\,x + (z\,\underline{|}\,y)\,\underline{|}\,x + (y|z)\,\underline{|}\,x + x|(y\,\underline{|}\,z) + x|(z\,\underline{|}\,y) + x|(y|z) =$

$A_x(y,z) + y\,\underline{|}\,(z||x) + z\,\underline{|}\,(y||x) + B_x(y,z) + (x|y)\,\underline{|}\,z + (x|z)\,\underline{|}\,y + x|(y|z) =$

$A_x(y,z) + A_y(z,x) + A_z(y,x) + B_x(y,z) + B_z(y,x) + B_y(x,z) + x|(y|z).$ (*)

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

$A_z(y,x) + A_y(x,z) + A_x(y,z) + B_z(y,x) + B_x(y,z) + B_y(z,x) + z|(y|x) =$

$A_x(y,z) + A_y(x,z) + A_z(y,x) + B_x(y,z) + B_y(z,x) + B_z(y,x) + (x|y)|z$

which equals (*) using the commutativity of the A's and B's and the induction hypothesis on $(x|y)|z$.

3.5. Remark. The identity (4) in Theorem 3.3 also holds for the operator γ in Hennessy [16] (discussed above in Remark 3.1(iv)); indeed this identity $(x \gamma y) \gamma z = x \gamma (y \| z)$ occurs in [16]. Note that the identity follows from Theorem 3.4 and the definition of γ, that is $x \gamma y = x \between y + x|y$, as follows:

$$(x \gamma y) \gamma z = (x \between y) \gamma z + (x|y) \gamma z =$$

$$(x \between y) \between z + (x \between y)|z + (x|y) \| z + (x|y)|z = \text{(Theorem 3.4)}$$

$$x \between (y \| z) + x|(z \between y) + x|(y \between z) + x|(y|z) = \text{(CM9)}$$

$$x \between (y \| z) + x|(z \between y + y \between z + y|z) = \text{(CM1)}$$

$$x \between (y \| z) + x|(y \| z) = x \gamma (y \| z).$$

3.6. Remark. Note that Theorem 3.4 (2,4,5) hold a fortiori for the initial algebra of PA in Table 2, since PA is the specialisation of ACP where communication is absent ($x|y = \delta$).

4. MERGING WITH MUTUAL EXCLUSION OF CRITICAL REGIONS: AMP

4.1. The critical region operator.

In the framework of ACP as introduced above, one can treat process co-operation where processes have critical sections which are to be mutually excluded in the co-operation. This is substantially more complicated (see Remark 4.2.3 below) than the following more direct way: Table 6 contains an axiom system AMP for processes with mutually excluded regions. It is an extension of the axiom system PA for free merge in Table 2: the additions in the signature consist of an unary operator $x \longmapsto \underline{x}$, the critical region operator (in the literature $\underline{x}$ is also denoted as $\langle x \rangle$), and an inverse operator ϕ which removes the constraints of critical regions. Intuitively, the underlined parts in a process expression (the critical regions) are to be executed in a co-operation as a single atomic step - that is, no interruption by an action from a parallel process is possible. Indeed we have as an immediate consequence of axioms CRM1 and M1 in Table 6:

4.1.1. PROPOSITION. $\underline{x}\|\underline{y} = \underline{x}\cdot\underline{y} + \underline{y}\cdot\underline{x}$.

Note that in general $\underline{x\|y} \neq \underline{x}\|\underline{y}$.

AMP

$x + y = y + x$	A1		
$(x + y) + z = x + (y + z)$	A2		
$x + x = x$	A3		
$(x + y)z = xz + yz$	A4		
$(xy)z = x(yz)$	A5		
$x\|y = x\lfloor y + y\lfloor x$	M1		
$a\lfloor x = ay$	M2	$\underline{x}\lfloor y = \underline{x}.y$	CRM1
$ax\lfloor y = a(x\|y)$	M3	$\underline{x}.y\lfloor z = \underline{x}(y\|z)$	CRM2
$(x + y)\lfloor z = x\lfloor z + y\lfloor z$	M4		
$\underline{a} = a$	CR1	$\phi(a) = a$	F1
$\underline{x + y} = \underline{x} + \underline{y}$	CR2	$\phi(x + y) = \phi(x) + \phi(y)$	F2
$\underline{\underline{x}} = \underline{x}$	CR3	$\phi(\underline{x}) = \phi(x)$	F3
		$\phi(x.y) = \phi(x).\phi(y)$	F4

Table 6.

A prooftheoretical analysis of AMP can be given analogous to the one in Section 3 for ACP, resulting in the following theorem:

4.1.2. THEOREM. *(i) Using the axioms* M1-4, CR1-3, CRM1,2, F1-4 *as rewrite rules from left to right, every closed term* T *in the signature of* AMP *can be proved equal to a unique basic term* T' *(i.e. a term built from* +,· *only and modulo* A1-5*).*
(ii) AMP *is a conservative extension of* PA. *Hence* AMP *is consistent.*

Writing n(T) for the unique basic term T' as in Theorem 4.1.2(i), it is easy to assign the ('intuitively' correct) semantics $M_{AMP}(T)$ in $(A_\omega,+,\cdot)$ to a closed AMP-term T:

$$\mathcal{M}_{AMP}(T) = [\![n(\phi(T))]\!]$$

where $[\![\ ]\!]$ is the semantics of basic terms in $(A_\omega,+,\cdot)$. E.g.:

$$\mathcal{M}_{AMP}(\underline{ab} \,||\, \underline{cd}) = abcd + cdab.$$

4.2. Tight multiplication.

A shortcoming in expressive power of the critical region operator in AMP is that it does not allow to specify a process $a\cdot(b\cdot x + c\cdot y)$ with the restriction that only after the first step 'a' and before the subprocess $bx + cy$ no interruption by a parallel process is possible. Therefore we consider a binary operator ':' ('tight' multiplication) with the interpretation that x:y is like $x\cdot y$ but with the proviso that in a merge, no step from a parallel process can be interleaved between x and y. Then $a:(b\cdot x + c\cdot y)$ is the process intended above. Table 7 contains an axiom system AMP(:) which is an extension of AMP by this new operator and corresponding axioms.

AMP(:)

$x + y = y + x$	A1		
$(x + y) + z = x + (y + z)$	A2		
$x + x = x$	A3		
$(x + y).z = x.z + y.z$	A4	$(x + y):z = x:z + y:z$	AT1
$(x.y).z = x.(y.z)$	A5	$(x:y):z = x:(y:z)$	AT2
		$(x:y).z = x:(y.z)$	AT3
		$(x.y):z = x.(y:z)$	AT4
$x \,\|\| y = x \,\llcorner\, y + y \,\llcorner\, x$	M1		
$a \,\llcorner\, y = ay$	M2		
$ax \,\llcorner\, y = a(x \,\|\| y)$	M3	$(a:x) \,\llcorner\, y = a:(x \,\llcorner\, y)$	CRM
$(x + y) \,\llcorner\, z = x \,\llcorner\, z + y \,\llcorner\, z$	M4		
$\underline{a} = a$	CR1	$\phi(a) = a$	F1
$\underline{x + y} = \underline{x} + \underline{y}$	CR2	$\phi(x + y) = \phi(x) + \phi(y)$	F2
$\underline{\underline{x}} = \underline{x}$	CR3	$\phi(\underline{x}) = \phi(x)$	F3
$\underline{x.y} = \underline{x}:\underline{y}$	CR4	$\phi(x.y) = \phi(x).\phi(y)$	F4
$\underline{x:y} = \underline{x}:\underline{y}$	CR5	$\phi(x:y) = \phi(x).\phi(y)$	F5

Table 7.

The axiom system AMP(:) is redundant when only finite processes are considered: then '$\lfloor\!\lfloor$' can be eliminated in favor of ':' (but not, as just remarked, reversely), and also for finite processes some of the axioms in AMP(:) can be proved inductively from the others, e.g. CR3.

The operator ':' has distinct advantages above '_': apart from its greater expressive power, it is more suitable for a treatment of infinite processes, both via projective sequences (as used above) and via bisimulation (not considered here).

A prooftheoretical analysis can be given analogous to the one in Section 3 for ACP and yielding a result analogous to Theorem 4.1.2. Likewise each closed AMP(:)-term T has an obvious semantics $M_{AMP(:)}(T)$ in $(A_\omega,+,\cdot)$, similarly to the case of AMP. (We will drop the subscript AMP(:) sometimes.)

Example: M(a:b || c:d) = abcd + cdab.

Note that M is a homomorphism w.r.t. + and $\cdot$, but not w.r.t. ||.

As before we have by a simple inductive proof:

4.2.1. THEOREM. *For all* x,y,z *in the initial algebra of* AMP(:) *we have:*
(i) $(x\lfloor\!\lfloor y)\lfloor\!\lfloor z = x\lfloor\!\lfloor (y\|z)$
(ii) $(x\|y)\|z = x\|(y\|z)$.

4.2.2. Remark. Note that the axioms in Table 6 for AMP:

$$\underline{x}\lfloor\!\lfloor y = \underline{xy} \qquad \text{(CRM1)}$$
$$\underline{xy}\lfloor\!\lfloor z = \underline{x}(y\|z) \qquad \text{(CRM2)}$$

and their immediate consequence

$$\underline{x}\|\underline{y} = \underline{x}\,\underline{y} + \underline{y}\,\underline{x} \qquad \text{(Proposition 4.1.1)}$$

can now be proved in AMP(:) from the axiom

$$(a:x)\lfloor\!\lfloor y = a:(x\lfloor\!\lfloor y) \qquad \text{(CRM)},$$

for finite closed terms (using an induction on term formation).

4.2.3. Remark. AMP(:) can be 'implemented' by ACP in the following sense. Let P,Q,R be closed AMP(:)-terms (the general case involving terms $P_1,..,P_n$ is similarly treated). Then we have in $(A_\omega,+,\cdot,\delta)$, the initial algebra of A1-7:

$$M_{AMP(:)}(P\|Q\|R)\bullet\delta = M_{ACP}(\partial_H(\underset{\rightarrow}{P}'\|\underset{\leftrightarrow}{Q}'\|\underset{\rightarrow}{R}'\|C)") \qquad (*)$$

where $M_{AMP(:)}$, defined above, yields the semantics in $(A_\omega,+,\bullet,\delta)$ of the AMP(:)-term $P\|Q\|R$ and M_{ACP} is the semantics of the ACP-term $\partial_H(\underset{\rightarrow}{P}'\|\underset{\leftrightarrow}{Q}'\|\underset{\rightarrow}{R}'\|C)"$ in that algebra. Here the terms $\underset{\rightarrow}{P}',\underset{\leftrightarrow}{Q}',\underset{\rightarrow}{R}'$ and C are defined as follows.

(i) $\underset{\rightarrow}{P}$ results from P by replacing every substring "a:" by "$\underset{\rightarrow}{a}\bullet$" where $\underset{\rightarrow}{a}$ is a new atom.
E.g. $a_1:(a_2\bullet a_3 + a_4:a_5)$ yields $\underset{\rightarrow}{a}_1\bullet(a_2\bullet a_3 + \underset{\rightarrow}{a}_4\bullet a_5)$.
Likewise for Q,R.

(ii) $\underset{\rightarrow}{P}',\underset{\leftrightarrow}{Q}',\underset{\rightarrow}{R}'$ are copies of $\underset{\rightarrow}{P},\underset{\leftrightarrow}{Q},\underset{\rightarrow}{R}$ obtained by renaming such that their alphabets are pairwise disjoint. Say $\underset{\rightarrow}{P}'$ contains only actions $\underset{\rightarrow}{a}_i,a_j$; $\underset{\leftrightarrow}{Q}'$ contains only actions $\underset{\rightarrow}{b}_k,b_\ell$, and $\underset{\rightarrow}{R}'$ only $\underset{\rightarrow}{c}_m,c_n$.

(iii) The control process C has alphabet $\{\alpha,\underset{\rightarrow}{\alpha},\beta,\underset{\rightarrow}{\beta},\gamma,\underset{\rightarrow}{\gamma}\}$ and is recursively defined by

$$\begin{cases} C = C_\alpha + C_\beta + C_\gamma \\ C_\alpha = \alpha\bullet C + \underset{\rightarrow}{\alpha}\bullet C_\alpha \\ C_\beta = \beta\bullet C + \underset{\rightarrow}{\beta}\bullet C_\beta \\ C_\gamma = \gamma\bullet C + \underset{\rightarrow}{\gamma}\bullet C_\gamma \end{cases}$$

(iv) The communication function to be used in evaluating the merges in the RHS of (*) is given by

$$\underset{\rightarrow}{\alpha}\,|\,\underset{\rightarrow}{a}_i = a_i^\circ, \quad \alpha|a_j = a_j^\circ$$

and likewise for β,γ. All other communications equal δ. H contains all communication actions $\alpha,\underset{\rightarrow}{\alpha},\ldots,\underset{\rightarrow}{a}_i,a_j,\ldots$.

Further, $\partial_H(\ldots)"$ in the RHS of (*) denotes a suitable renaming of $\partial_H(\ldots)$ into the original alphabets of P,Q,R.

Finally, the presence of δ in the LHS of (*) is due to the fact that C has no finite branches.

Working in ACP_τ as in [9] or in CCS [33], so that τ-steps are available, the control process C can be simplified (essentially to binary semaphores) as Milner [33] shows in an example. There a process as a:(b•c:f + d:e:f) would be syntactically converted to $\pi a(b\phi\pi cf\phi + def\phi)$, with $C = \bar{\pi}\bar{\phi}C$ and $\pi|\bar{\pi} =$

$\phi \mid \overline{\phi} = \tau$. However, the problem is then to get rid of the τ-steps in order to obtain an equation like (*) above.

5. MERGING WITH COMMUNICATION AND MUTUAL EXCLUSION: ACMP

The facilities of merge with communication (ACP) and merge with mutual exclusion of critical sections (AMP(:)) can be joined in a smooth way. (This is not self-evident; e.g. it seems not clear at all how to join tight multiplication as in AMP(:) with τ-steps.)

The result of this join is the axiom system ACMP in Table 8. The left column contains ACP with a slight alteration for convenience: CM5* is added (cf. Table 3 and 8) which saves us some axioms. The right column consists of the axioms in AMP(:) (see Table 7) for the operators :, $\lfloor$ and ϕ where the axiom

$$(a:x) \lfloor\!\lfloor y = a:(x \lfloor\!\lfloor y) \qquad \text{CRM}$$

is now 'extended' to

$$(a:x) \lfloor\!\lfloor y = a:(x \lfloor\!\lfloor y + x \mid y) \qquad \text{CTM1}.$$

The axiom CTM1 can be understood as follows. The process $(a:x) \lfloor\!\lfloor y$ has a double commitment: '$\lfloor\!\lfloor$' insists that the first step in the coöperation between a:x and y is taken from a:x and ':' insists that after performing 'a' a step from x must follow without interruption. This double restraint is respected in $a:(x \lfloor\!\lfloor y + x \mid y)$. After 'a', the required step from x may be an 'autonomous' step of x, as in $x \lfloor\!\lfloor y$, or a simultaneous step in x and y, as in $x \mid y$. (Note that when communication is absent, i.e. $x \mid y = \delta$, CTM1 specializes to CRM.) Moreover axiom AT5 is new and so are CTM2-4 which specify ':' versus '|'.

By means of a tedious prooftheoretic analysis analogous to the one for ACP one can prove consistency of ACMP and that ACMP is a conservative extension of both ACP and AMP(:). Also associativity of '||' holds for ACMP; intuitively this can be seen via a graph representation of closed ACMP-terms as in Example 5.1 below.

It turns out that the combination of asynchronous co-operation as in ACP with 'tight' multiplication as in AMP(:) is able to give an interpretation of synchronous co-operation. This will be stated more precisely in the next section where a direct axiomatisation of synchronous co-operation is given.

ACMP

$x+y = y+x$	A1	$(x+y):z = x:z + y:z$	AT1
$(x+y)+z = x+(y+z)$	A2	$(x:y):z = x:(y:z)$	AT2
$x+x = x$	A3	$(x:y).z = x:(y.z)$	AT3
$(x+y)z = xz+yz$	A4	$(x.y):z = x.(y:z)$	AT4
$(xy)z = x(yz)$	A5	$\delta:x = \delta$	AT5
$x+\delta = x$	A6	$(a:x)\mathbin{\lfloor\!\lfloor} y = a:(x\mathbin{\lfloor\!\lfloor} y + x\mid y)$	CTM1
$\delta x = \delta$	A7	$(a:x)\mid(b:y) = (a\mid b):(x\mid y)$	CTM2
		$(a:x)\mid(by) = (a\mid b):(x\mathbin{\lfloor\!\lfloor} y + x\mid y)$	CTM3
		$(a:x)\mid b = (a\mid b):x$	CTM4
$a\mid b = b\mid a$	C1		
$(a\mid b)\mid c = a\mid(b\mid c)$	C2		
$a\mid\delta = \delta$	C3		
$x\parallel y = x\mathbin{\lfloor\!\lfloor} y + y\mathbin{\lfloor\!\lfloor} x + y\mid x$	CM1	$\underline{a} = a$	CR1
$a\mathbin{\lfloor\!\lfloor} x = ay$	CM2	$\underline{x+y} = \underline{x}+\underline{y}$	CR2
$ax\mathbin{\lfloor\!\lfloor} y = a(x\parallel y)$	CM3	$\underline{\underline{x}} = \underline{x}$	CR3
$(x+y)\mathbin{\lfloor\!\lfloor} z = x\mathbin{\lfloor\!\lfloor} y + y\mathbin{\lfloor\!\lfloor} z$	CM4	$\underline{x.y} = \underline{x}:\underline{y}$	CR4
$x\mid y = y\mid x$	CM5*	$\underline{x:y} = \underline{x}:\underline{y}$	CR5
$a\mid by = (a\mid b)y$	CM6		
$ax\mid by = (a\mid b)(x\parallel y)$	CM7	$\phi(a) = a$	F1
$(x+y)\mid z = x\mid z + y\mid z$	CM8	$\phi(x+y) = \phi(x)+\phi(y)$	F2
		$\phi(\underline{x}) = \phi(x)$	F3
$\partial_H(a) = a$ if $a \notin H$	D1	$\phi(x.y) = \phi(x).\phi(y)$	F4
$\partial_H(a) = \delta$ if $a \in H$	D2	$\phi(x:y) = \phi(x).\phi(y)$	F5
$\partial_H(x+y) = \partial_H(x)+\partial_H(y)$	D3		
$\partial_H(x.y) = \partial_H(x).\partial_H(y)$	D4		

Table 8.

5.1. Example. $a:b \parallel c:d = a:b \mathbin{\lfloor\!\lfloor} c:d + c:d \mathbin{\lfloor\!\lfloor} a:b + a:b \mid c:d =$

$a:(bc:d + b\mid c:d) + c:(da:b + d\mid a:b) + (a\mid c):(b\mid d) =$

$a:(bc:d + (b\mid c):d) + c:(da:b + (d\mid a):b) + (a\mid c):(b\mid d).$

There is a simple graphical method for evaluating such expressions, as sug-

gested by Figure 1(a). (This is moreover relevant since it enables us to define simple graph models for ACMP; we will not do so here.) In the figure black nodes indicate tight multiplication. After 'unraveling' shared subgraphs we arrive at the correct evaluation of a:b || c:d, as in Figure 1(b).

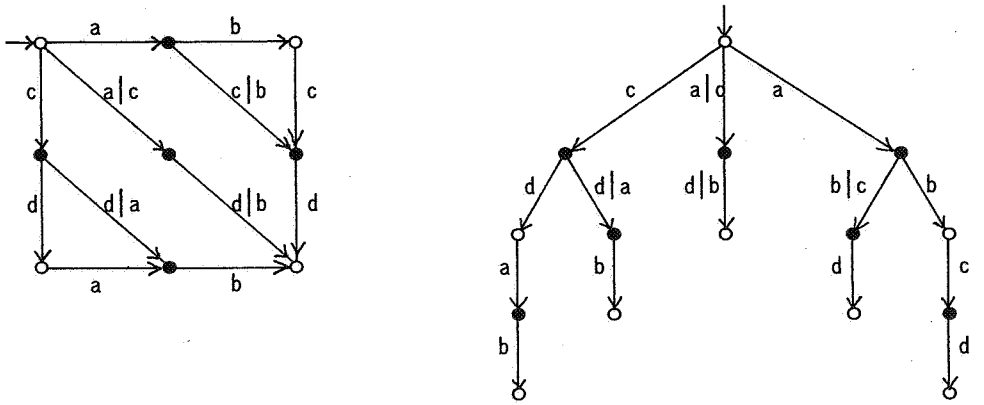

Figure 1. (a) (b)

(For the merge || in PA and ACP there are analogous ways: merging two process graphs in the PA sense consists of taking the full cartesian product graph; in ACP diagonal edges for the results of communication have to be added. See [6].)

6. SYNCHRONOUS CO-OPERATION: ASP

We will briefly comment in this section on the distinction between asynchronously versus synchronously co-operating processes (in the sense of Milner [35]). ACP, just as CCS, describes the asynchronous co-operation of processes. The axiom system ASP in Table 9 below describes synchronous co-operation of processes, in the sense that the co-operation of processes $P_1,\ldots,P_n$, notation $P_1|P_2|\ldots|P_n$, proceeds by taking in each of the P_i simultaneously steps on the (imaginary) pulses of a global clock.

Formally, the relation of ASP to ACP is clear: it originates by leaving out the results of the free merge, that is: in axiom CM1 of ACP

$$x||y = x\between y + y\between x + x|y$$

the first two summands are discarded (so that || is in effect |, the communication merge).

ASP

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	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
$a \mid \delta = \delta$	C3
$(x + y) \mid z = x \mid z + y \mid z$	SM1
$x \mid (y + z) = x \mid y + x \mid z$	SM2
$ax \mid by = (a \mid b)(x \mid y)$	SM3
$a \mid by = (a \mid b)y$	SM4
$ax \mid b = (a \mid b)x$	SM5

Table 9.

ASP bears a strong resemblance to Milner's SCCS [35] (see also Hennessy [17]); the most notable difference is δ which does part of the work done in SCCS by restriction operators. (In SCCS 'incompatibility' of atoms a,b cannot be expressed, so that certain superfluous subprocesses of a co-operation must be pruned away after the evaluation of the co-operation by a restriction operator. In ASP this incompatibility is stated as $a \mid b = \delta$.) Another notable difference is that SCCS admits also infinite sums.

Milner [35] gives an ingenuous implementation of asynchronous processes (as in CCS) in terms of SCCS, via some 'delay-operators' and argues that synchronous co-operation is a more fundamental notion than asynchronous co-operation. However, the reverse position can be argued too, since many synchronous processes can be implemented in ACP (see Remark 6.3).

Synchronous co-operation as axiomatised by ASP can be interpreted in ACMP, as the next theorem states (the routine proof is omitted).

6.1. THEOREM. *Let* x,y *be basic terms. Then* x|y *evaluates in* ASP *to the same basic term as* $\phi(\underline{x}|\underline{y})$ *in* ACMP.

Phrased differently, Theorem 6.1 says that in the algebra

$$\mathcal{A} = (A,+,\cdot,:,\|,\lfloor\!\lfloor,|,|*,\_,\phi,\partial_H,\delta)$$

which has as reducts

$$(A,+,\cdot,|*,\delta),$$

the initial algebra of ASP, and

$$(A,+,\cdot,\|,\lfloor\!\lfloor,|,:,\_,\phi,\partial_H,\delta),$$

the initial algebra of ACMP, we have:

$$\mathcal{A} \models x|*y = \phi(\underline{x}|\underline{y}).$$

6.2. Example. $\phi(\underline{ab}|\underline{cd}) = \phi(a:b|c:d) = \phi((a|c):(b|d)) = (a|c)(b|d) = ab|*cd$.

6.3. Remark. Another possibility, only slightly less direct than the interpretation in ACMP above, is to 'implement' ASP in ACP as follows. Let $P_1|\ldots|P_n$ be a closed ASP-term; the P_i are basic. Let $A_i \subseteq A$ be the set of actions occurring in P_i $(i=1,\ldots,n)$, and $H = A_1 \cup \ldots \cup A_n$.

Suppose that H does not contain results of H-communications:

$$H \cap (H|H \cup H|H|H \cup \ldots) = \emptyset$$

(Here $H|H = \{c \mid \exists\, a,b \in H \;\; a|b=c\}$, etc.) Then:

$$\mathcal{M}_{ASP}(P_1|\ldots|P_n) = \mathcal{M}_{ACP}(\partial_H(P_1\|\ldots\|P_n))$$

where $\mathcal{M}_{ASP}, \mathcal{M}_{ACP}$ denote the semantics of ASP-, resp. ACP-terms in the respective initial algebras.

6.4. Example. In ASP: $ab|cd = (a|c)(b|d)$. Suppose $a|c,\ b|d \notin \{a,b,c,d\} = H$, then also in ACP:

$$\partial_H(ab\|cd) = \partial_H(ab\lfloor\!\lfloor cd) + \partial_H(cd\lfloor\!\lfloor ab) + \partial_H(ab|cd) =$$

$$\partial_H(a(b\|cd) + \partial_H(c(d\|ab)) + \partial_H((a|c)(c\|d)) =$$

$$\delta + \delta + (a|c)(b|d) = (a|c)(b|d).$$

6.5. Remark: asynchronous communication.

There does not seem to be a consensus as regards the use of the terms 'synchronous' vs. 'asynchronous'. The terminology that we have adopted and used in the preceding pages, distinguishes 'co-operation' from 'communication' and is stated more explicitly as follows:
(i) ASP, SCCS have synchronous co-operation and synchronous communication;
(ii) ACP, CCS have asynchronous co-operation and synchronous communication.
A third format, not considered above but used in some programming languages, is 'asynchronous co-operation with asynchronous communication'. Here the communication is asynchronous in the sense that e.g. a process P sends a message c! to a process Q such that P can proceed while the message c! to Q is "on its way".

7. CONCLUDING REMARKS

We have introduced axiom systems as in the enclosed part of Figure 2:

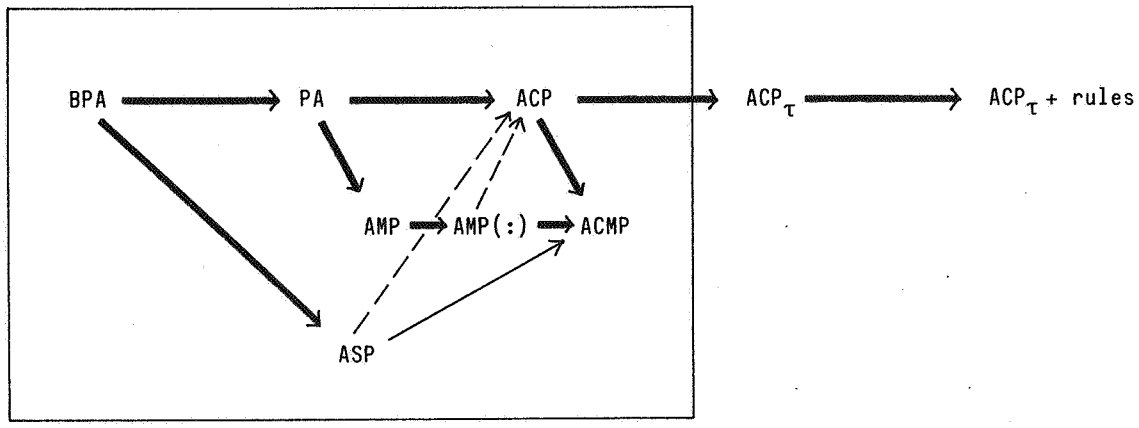

Figure 2.

Here each heavy arrow denotes a conservative extension, the arrow from ASP to ACMP denotes an 'interpretation' and the dashed arrows denote an 'implementation' (in the vague sense of a less direct interpretation).

For the main axiom system, ACP, basic properties such as consistency and an eleimination theorem have been proved. For the other systems similar results follow by a similar proof. It is claimed that ACP and the other axiom systems codify central concepts in concurrency: free merge, merge with com-

munication by action sharing, merge with mutual exclusion of critical regions, synchronous vs. asynchronous process co-operation. Also some of these concepts are shown to be related as indicated in the diagram in Figure 2.

Clearly, as we discussed in the Introduction, this work is strongly related to other algebraic approaches of concurrency. In this paper we did not study the effect of adding mechanisms for recursive definitions, such as μ-expressions (cf. Milner [34]), or systems of recursion equations (as in [8]). For each of the systems such an addition is possible; for BPA, PA and ACP the relative expressive power, after adding recursion facilities, is studied in [8]. For instance, one can show that the process B recursively defined by $B = (aa' \mp bb') \between B$ over PA cannot be recursively defined over BPA, i.e. without merge or left-merge. (B is the behaviour of a 'bag' over a data domain consisting of two elements.)

Also not touched in this paper is the problem of abstraction ('hiding'). In [9] an extension ACP_τ (see Figure 2) of ACP has been defined and studied, which basically consists of ACP plus Milner's τ-laws, in order to deal with abstraction of internal steps. An application of ACP yielding such internal steps, is given in [6], where the operational semantics of data flow networks is defined in terms of ACP. Further applications of ACP include finite specifications of the behaviours of processes like Stack, Bag and Queue, as well as algebraic verifications such as that the juxtaposition of two Bags is again equivalent to a Bag - after abstraction from internal steps. In [7] a connection between processes and abstract data types is investigated, with the purpose of providing the means of validating some process specifications against their abstract data types specifications.

In [10] a simple version of the Alternating Bit Protocol is proved correct in the framework of ACP_τ plus some extra rules, using only algebraical calculations.

There exists a rich modeltheory for ACP. In this paper we have only mentioned (apart from the obvious initial algebras) the projective limit algebra. A fruitful concept for building process algebras is the notion of bisimulation (see [39]) between process graphs. Process algebras obtained in this way are defined and studied in [9].

REFERENCES

[1] BACK, R.J.R. & H. MANNILA,
A refinement of Kahn's semantics to handle nondeterminism and communication,
ACM Conf. on Principles of Distributed Computing, Ottawa 1982.

[2] BACK, R.J.R. & H. MANNILA,
On the suitability of trace semantics for modular proofs of communicating processes,
Preprint, Dept. of Computer Science, University of Helsinki.

[3] DE BAKKER, J.W. & J.I. ZUCKER,
Denotational semantics of concurrency,
Proc. 14th ACM Symp. on Theory of Computing, p.153-158, 1982.

[4] DE BAKKER, J.W. & J.I. ZUCKER,
Processes and the denotational semantics of concurrency,
Information and Control, Vol.54, No.1/2, p.70-120, 1982.

[5] DE BAKKER, J.W., BERGSTRA, J.A., KLOP, J.W. & J.-J.CH. MEYER,
Linear time and branching time semantics for recursion with merge,
Proc. 10th ICALP, Barcelona 1983 (Ed. J. Díaz), Springer LNCS 154, p.39-51, 1983. Expanded version to appear in Theor. Comp. Sci.

[6] BERGSTRA, J.A. & J.W. KLOP,
A process algebra for the operational semantics of static data flow networks,
Report IW 222/83, Mathematisch Centrum, Amsterdam 1983.

[7] BERGSTRA, J.A. & J.W. KLOP,
An algebraic specification method for processes over a finite action set,
Report IW 232/83, Mathematisch Centrum, Amsterdam 1983.

[8] BERGSTRA, J.A. & J.W. KLOP,
The algebra of recursively defined processes and the algebra of regular processes,
To appear in Proc. 11th ICALP, Antwerpen, July 1984.

[9] BERGSTRA, J.A. & J.W. KLOP,
Algebra of Communicating Processes with abstraction,
Report CS-R8403, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[10] BERGSTRA, J.A. & J.W. KLOP,
Verification of an alternating bit protocol by means of process algebra,
Report CS-R8404, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[11] BERGSTRA, J.A. & J.W. KLOP,
Fair FIFO queues satisfy an algebraic criterion for protocol correctness,
Report CS-R8405, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[12] BROOKES, S.D.,
On the relationship of CCS and CSP,
Proc. 10th ICALP, Barcelona 1983 (ed. J. Díaz), Springer LNCS 154, p.83-96, 1983.

[13] BROOKES, S.D. & W.C. ROUNDS,
Behavioural equivalence relations induced by programming logics,
Proc. 10th ICALP, Barcelona 1983, Springer LNCS 154 (ed. J. Díaz), p.97-108, 1983.

[14] DERSHOWITZ, N.,
Orderings for term-rewriting systems,
Theoretical Computer Science 17 (1982) p.279-301.

[15] GRAF, S. & J. SIFAKIS,
A modal characterization of observational congruence on finite terms of CCS,
to appear in Proc. 11th ICALP, Antwerpen 1984.

[16] HENNESSY, M.,
On the relationship between time and interleaving,
Preprint, Univ. of Edinburgh.

[17] HENNESSY, M.,
A term model for synchronous processes,
Information and Control 51, p.58-75 (1981)

[18] HENNESSY, M.,
Synchronous and asynchronous experiments on processes,
Report CSR-125-82, Univ. of Edinburgh 1982.

[19] HENNESSY, M.,
Axiomatising finite delay operators,
Report CSR-124-82, Univ. of Edinburgh 1982.

[20] HENNESSY, M.,
A model for nondeterministic machines,
CSR-135-83, Univ. of Edinburgh 1983.

[21] HENNESSY, M. & R. MILNER,
On observing nondeterminism and concurrency,
Proc. 7th ICALP, Springer LNCS 85, p.299-309, 1980.

[22] HENNESSY, M. & R. MILNER,
Algebraic laws for nondeterminism and concurrency,
Report CSR-133-83, Univ. of Edinburgh 1983. To appear in JACM.

[23] HENNESSY, M. & R. DE NICOLA,
Testing equivalences for processes,
Report CSR-123-82, Univ. of Edinburgh 1982.

[24] HENNESSY, M. & G. PLOTKIN,
A term model for CCS,
Proc. 9th MFCS, Poland (1980), Springer LNCS 88.

[25] HOARE, C.A.R.,
Communicating Sequential Processes,
C.ACM 21 (1978), p.666-677.

[26] HOARE, C.A.R.,
A model for Communicating Sequential Processes,
in: "On the construction of programs" (eds. R.M. McKeag and A.M. McNaghton), Cambridge Univ. Press (1980), p.229-243

[27] HOARE, C., BROOKES, S. & W. ROSCOE,
A theory of communicating sequential processes,
Programming Research Group, Oxford University (1981). To appear in JACM.

[28] HUET, G.,
Confluent reductions: Abstract properties and applications to term rewriting systems,
J.ACM 27 (4) (1980),p.797-821.

[29] KLOP, J.W.
Combinatory Reduction Systems,
Mathematical Centre Tracts 127, Mathematisch Centrum, Amsterdam 1980.

[30] MILNE, G.,
Abstraction and nondeterminism in concurrent systems,
1982 IEEE, p.358-364.

[31] MILNE, G.,
CIRCAL: A calculus for circuit description,
Preprint, Univ. of Edinburgh 1982.

[32] MILNE, G. & R. MILNER,
Concurrent processes and their syntax,
JACM 26 (2), 1979, p.302-321.

[33] MILNER, R.,
A Calculus of Communicating Systems,
Springer LNCS 92, 1980.

[34] MILNER, R.,
A complete inference system for a class of regular behaviours,
Report CSR-111-82, Univ. of Edinburgh 1982.

[35] MILNER, R.,
Calculi for synchrony and asynchrony,
Theor. Comp. Sci. 25 (1983), p.267-310.

[36] NIVAT, M.,
Infinite words, infinite trees, infinite computations,
Foundations of Computer Science III.2 (J.W. de Bakker & J. van Leeuwen, eds.) p.3-52, Mathematical Centre Tracts 109, Mathematisch Centrum, Amsterdam 1979.

[37] NIVAT, M.,
Synchronization of concurrent processes,
Formal Language Theory (R.V. Book, ed.), p.429-454, Academic Press 1980.

[38] OLDEROG, E.-R. & C.A.R. HOARE,
Specification-oriented semantics for communicating processes,
Proc. 10th ICALP, Barcelona 1983, Springer LNCS 154, p.561-572.
Expanded version to appear as Technical Monograph PRG-37, Febr.1984, Oxford Univ. Comp. Lab.

[39] PARK, D.M.R.,
Concurrency and automata on infinite sequences,
Proc. 5th GI Conference, Springer LNCS 104, 1981.

[40] PRATT, V.R.,
On the composition of processes,
Proc. 9th ACM Symp. on Principles of Programming Languages, p.213-223 (1982).

[41] REM, M.
Partially ordered computations, with applications to VLSI design,
Proc. 4th Advanced Course on Foundations of Computer Science, Part 2 (eds. J.W. de Bakker and J. van Leeuwen), Mathematical Centre Tracts 159, p.1-44, Mathematisch Centrum, Amsterdam 1983.

[42] STAPLES, J. & V.L. NGUYEN,
A fixpoint semantics for nondeterministic data flow,
Report No.48, Dept. of Comp. Sci., Univ. of Queensland, Australia, 1983.

[43] WINSKEL, G.,
Event structure semantics for CCS and related languages,
Proc. ICALP 82, Springer LNCS 140, p.561-576, 1982.

[44] WINSKEL, G.,
Synchronisation trees,
Proc. 10th ICALP (ed. J. Díaz), Barcelona 1983, Springer LNCS 154, p.695-711.