LANDELIJK PROJECT CONCURRENCY

syntactische, semantische en bewijstheoretische aspecten

SEMINARIUM

7 DECEMBER 1984 RIJKSUNIVERSITEIT LEIDEN

J.W.Klop

"Process Algebra"

INSTITUUT VOOR TOEGEPASTE WISKUNDE EN INFORMATICA

ALGEBRA OF COMMUNICATING PROCESSES - PART II

J.C.M. Baeten, J.A. Bergstra, J.W. Klop
Centre for Mathematics and Computer Science, Amsterdam

ABSTRACT

A survey of recent work in process algebra is given. We present various algebraic descriptions of concepts in concurrency, such as: axioms systems for asynchronous communication (stimulus-response mechanism or causality); mail along an order-preserving channel as well as mail along a non-order-preserving channel; actions with partially ordered priorities in order to model simple interrupt mechanisms; synchronous process cooperation; complete axiom systems for regular processes both with and without abstraction (yielding invisible steps or τ-steps). The axiom systems are presented as composed from more elementary building blocks. Also a short survey is given of some model constructions for these axiom systems; the main attention here is for models constructed from process graphs by means of several notions of bisimulation.

NOTE. This work was sponsored in part by ESPRIT contract METEOR.
These notes are not intended for publication.

INTRODUCTION

These notes continue a survey of process algebra of which the first part is in 'ALGEBRA OF COMMUNICATING PROCESSES' (CWI report CS-R8421). Part of the work surveyed below, concerning asynchronous communication (in sections 2.5.16 - 18), was done in collaboration with J.V. Tucker (Leeds University). *) The part about failure semantics (2.5.20, 2.5.22, 2.6.3) was done in collaboration with E.-R. Olderog (Universität Kiel). The first part of this survey in report CS-R8421 arose from a set of lecture notes for a workshop on concurrency and abstract data types organized by W.P. de Roever in Mook, autumn 1983. We have tried to make the present second part self-contained in the sense that exposure to the first part is helpful but not required.

The structure of the present notes is as follows. There are three chapters:

1. General remarks
2. Axiom systems
3. Models.

In chapter 1 we comment on such matters as motivations and methods, applications, related work.

In chapter 2 a genealogical tree is given of various axiomatisations for process algebra which we have studied; the progenitor is BPA (basic process algebra) which merely describes + and $\cdot$, the 'constructors' of the processes discussed below. Each axiom system describes (a set of) aspects or features of processes, e.g. synchronous communication by handshaking, or: cooperation in which actions are involved that may have priority over other actions, or: simple interleaving without communication, etc. These axiom systems are composed of simpler building blocks (e.g. the trio of Milner's τ-laws). These building blocks are discussed consecutively in chapter 2.

Each axiom system captures several models (or 'process algebras') which satisfy the axioms in that system. E.g. the axiom system PA (for 'free merge' or interleaving without communication) still leaves a lot of freedom in the choice of which model of processes one actually wants to consider (finite processes,

---

*) See Bergstra, Klop and Tucker (1984).

finitely branching processes, regular processes,...). This is the subject of chapter 3, in which also an overview is given of some model constructions for the axiom systems in chapter 2.

A word on terminology: 'process algebra' is used in three different senses. Firstly it denotes the activity of doing ... process algebra. Secondly, it denotes a model of any axiom system in chapter 2. So this is the same ambiguity as in the mathematical vernacular regarding the term 'algebra'. Thirdly and unfortunately, also one of the axiom systems below has the name PA (process algebra). (Reason: it was the first one that we considered.)

1.1. *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 Milner (see his introductory work, (Milner, 1980)) in the form of his calculus of communicating systems (CCS). Milner states his aim in (Milner, 1983) in his own words: "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." Milner (1983) 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.

1.2. *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), $\cdot$ (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 2.2.3*. The axiom system ACP presented below *in 2.2.5* 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 (1980) 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, $\lfloor\!\lfloor$, and $|$ are not present in CCS.

---

*) The text on this page and the next two is, with minor alterations, copied from the Introduction of the paper Bergstra and Klop (1984e) in Information and Control. The same holds for (part of) the References and some axiom systems below.

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 (1983), 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, 1980) for them; they can be added consistently, and even conservatively, to ACP. The resulting axiom system ACP_τ is studied in Bergstra and Klop (1984b). 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 (Milner, 1980). 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 Bergstra and Klop, 1984a).

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 tight 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
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.

1.3. *Related Approaches*

Closest to the present work is Milner's CCS, which was above briefly compared with the axioms below. Interestingly, Milner has proposed in (Milner, 1983) a system SCCS which supersedes CCS and which has as fundamental notion: synchronous process cooperation. It is argued that asynchronous process cooperation (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 *section 1.6*. Again, it would be very useful and interesting to determine the precise mathematical relationships between those systems for synchrony and asynchrony; a start has been made in Milner (1983).

Milner's work has been continued and extended in Hennessy and Plotkin (1980) and a series of papers by Hennessy (1981–1983) 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 (1982a, 1983) 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 (Hennessy, 1983; Hennessy and Milner, 1980, 1983). (See also Graf and Sifakis, 1984; and Brookes and Rounds, 1983.)

Milne (1982a, b), presents the "dot calculus": here · is concurrent composition. The dot calculus uses prefix multiplication as in the work of Milner and Hennessy (called "guarding" by Milne), operators $+$, $\oplus$ for choice (by environment resp. internal), Δ for deadlock as well as successful termination. In contrast to CCS as in (Milner, 1980), 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.

As some other related approaches which are less algebraical in spirit than the aforementioned (CCS, SCCS, dot calculus, ACP) and which have a more denotational style we mention the work of De Bakker and Zucker (1982a, b). 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 De Bakker and Zucker, 1982a) were our incentive to formulate PA as in Table II below. The processes of De Bakker and Zucker include several programming concepts which are not discussed in ACP. In De Bakker *et al.* (1983) 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 Hoare, 1978; 1980) have also been discussed from a uniform point of view in Olderog and Hoare (1983). For work discussing aspects of CCS and CSP, as well as connections between these two, we refer to Brookes (1983). Other work on concurrency in the denotational style includes Back and Mannila (1982a, b), Pratt (1982), and Staples and Nguyen (1983). Finally, Winskel (1983a, b) discusses communication formats in languages such as CCS, CSP.

1.4. Methodology

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 ours 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, 1982; Bergstra and Klop, 1984a). Also, one may build process algebras based on the fundamental and fruitful notion of bisimulation (introduced by Park (1981), as is done in, e.g., Milner (1982, 1983); 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 at which we are aiming, is not yet fully represented in related approaches to the understanding of concurrency.

1.5

CONNECTIONS WITH OTHER WORK PRESENTED IN LPC FRAMEWORK

There are several interesting connections, some already existing but mostly to be developed yet, with investigations presented at previous LPC meetings:

- regarding the denotational semantics of ('uniform') processes developed by de Bakker and Zucker, see some remarks above in 1.3.

- work is done by several people to connect process algebra as in Milner's CCS with Petri Net Theory (see notes of the presentations by Rozenberg, Thiagarajan); an example is Reisig (1984). A relative advantage of Petri nets seems to be that they are better suited to treat "true concurrency".

- work is going on (see e.g. Brookes (1983)) to connect process algebra as in CCS with Hoare's CSP. Cf. also the axiom system ACP_{RS} below, describing a simple form of failure semantics, a semantics recently assigned by Hoare e.a. to a stylized version of CSP. See Hoare, Brookes, Roscoe (1981). Languages as CSP are treated from a proof theoretical point of view by de Roever, Gerth and coworkers.

- Meyer (1984) develops calculi for 'free' merge of processes as well as the fair merge; as to the first one (free merge) there may be interesting connections with a proof system (PA_{LR}) below.*) Meyer's work is based on trace set semantics (LT, linear time semantics) as 'contrasted' with process semantics (BT, branching time semantics).

- It will be very useful to determine more of the relationships between LT semantics (trace semantics) and BT semantics (process semantics). Studies towards a determination of these relationships have been made in de Bakker, Bergstra, Klop, Meyer (1983), and are continued by De Bakker and coworkers. A better understanding of 'LT versus BT' will also be interesting in connection with the work on trace theory of the Eindhoven school (Rem (1983); Ebergen (1984); van de Snepscheut (1983)).

---

*) In fact, PA_{LR} is not explicitly displayed below. It is however easy to assemble:

$PA_{LR} = BPA_{LR}$ + M1-4. Or: $PA_{LR} = PA_{\tau LR}$ - T1-3, TI0-6, R3-5.

6. COOPERATION AND COMMUNICATION: THE DISTINCTION BETWEEN SYNCHRONOUS AND ASYNCHRONOUS

There does not seem to be a consensus as regards the use of the terms "synchronous" vs. "asynchronous". We have adopted the following terminology in which "cooperation" is distinguished from "communication".

Two regimes of cooperation can be distinguished:

synchronous cooperation

for instance in Milner's SCCS ('synchronous CCS'), ASP below, MEIJE: the regime of synchronous cooperation allows cooperating processes p and q to be executed in parallel with the same speed and timed on the same clock.

asynchronous cooperation

for instance in Hoare's CSP, CCS, ACP below. Asynchronous cooperation allows cooperating processes p and q to proceed with their own speed and timed by independent clocks only restricted by possible mutual interactions.

With communication we denote interaction between atomic actions of processes. Again two regimes can be distinguished:

synchronous communication

for instance in CSP, CCS, Ada: communication between actions a and b can take place only if both are performed simulaneously. This type of communication is often called handshaking.

asynchronous communication

for instance in CHILL: send and receive statements. Communication between actions a and b is consistent with b being performed after a.

Combining the above regimes one arrives at four possible combinations which can be used to roughly classify theoretical models of concurrency:

	synchronous communication	*asynchronous communication*
synchronous cooperation	*(SS)* SCCS ASCCS MEIJE ASP ACMP	*(SA)* no example known to us
asynchronous cooperation	*(AS)* CCS CSP Ada Petri nets ACP uniform processes	*(AA)* CHILL data flow networks restoring circuit logic $PA_\delta(\gamma)$, $PA_\delta(\mu_c^\sigma)$, $PA_\delta(\mu_c^M)$

ASCCS is from Milner (1983); according to Milner it gives a framework for 'asynchronous processes'. So it may be puzzling why we have classified ASCCS under (SS). The reason is that it is a subcalculus of SCCS, hence also employs synchronous cooperation and synchronous communication - even though asynchronously cooperating processes may be interpreted in ASCCS.

ACMP combines synchronous and asynchronous cooperation and has synchronous communication.

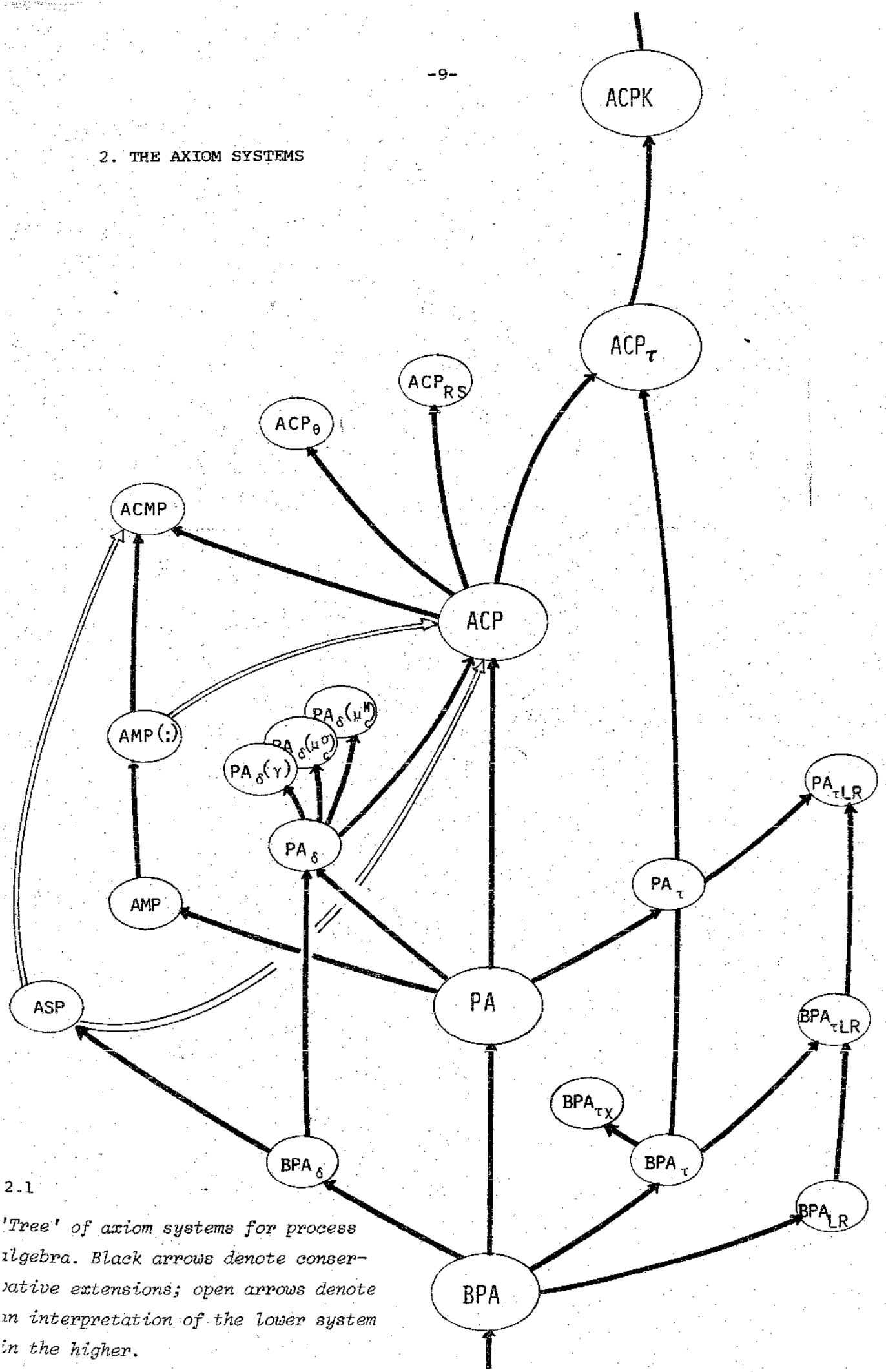
Restoring circuit logic is intended to describe the behaviour of circuits regardless of delays in the connecting wires. This delay insensitivity causes the classification under (AA).

As to MEIJE, we refer to Austry & Boudol (1984).

The combination (AA) is studied for instance using temporal logic in Pnueli (1977), Lamport (1979) and Koymans, Vytopil & de Roever (1983), Kuiper & de Roever (1982). Moreover, trace theories are used to describe the semantics of data flow networks (see Kahn (1974), Brock & Ackerman (1981) and the semantics of restoring circuit logic (see Ebergen (1984), Rem (1983) and van de Snepscheut (1983)).

A discussion of the case (AA) in an algebraic setting is absent to our knowledge. In Milne (1982b) and Bergstra & Klop (1983a) the (AA) case is reduced to the (AS) case for switching resp. data flow. We are not aware of any "direct" algebraic descriptions of the (AA) case other than those given by the axiom systems $PA_\delta(\gamma)$, $PA_\delta(\mu_c^M)$ and $PA_\delta(\mu_c^\sigma)$ below.

2. THE AXIOM SYSTEMS

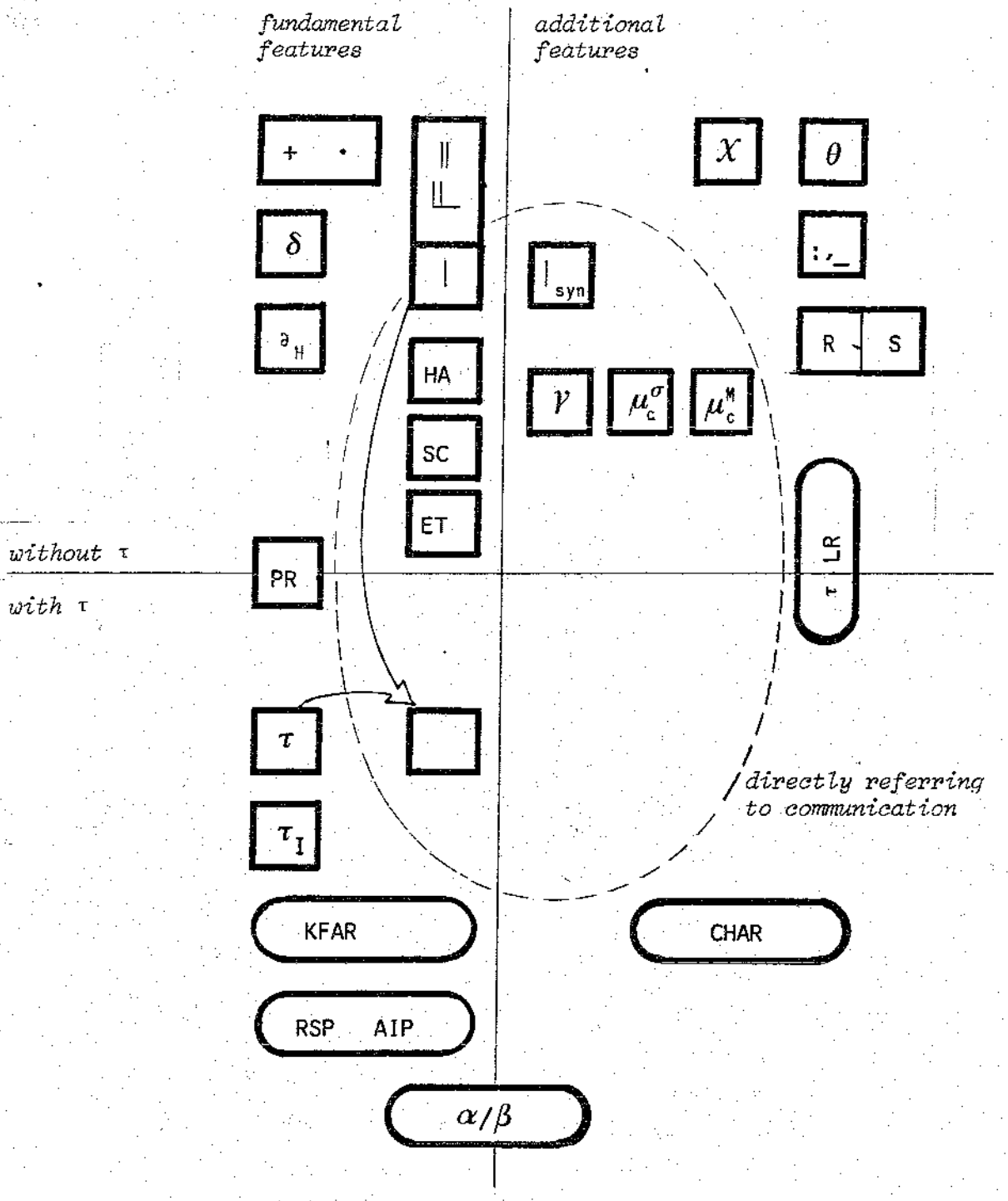


2.1

'Tree' of axiom systems for process algebra. Black arrows denote conservative extensions; open arrows denote an interpretation of the lower system in the higher.

2.2. NAMES OF THE AXIOM SYSTEMS

1. BPA *basic process algebra*
2. BPA_δ *basic process algebra with deadlock*
3. PA *process algebra*
4. PA_δ *process algebra with deadlock*
5. ACP *algebra of communicating processes*
6. BPA_τ *basic process algebra with abstraction*
7. PA_τ *process algebra with abstraction*
8. ACP_τ *algebra of communicating processes with abstraction*
9. BPA_{LR} *basic process algebra with linear recursion*
10. $BPA_{\tau LR}$ *basic process algebra with abstraction and linear recursion*
11. $PA_{\tau LR}$ *process algebra with abstraction and linear recursion*
12. AMP *process algebra with tight regions*
13. $AMP(:)$ *process algebra with tight regions and tight multiplication*
14. $ACMP$ *algebra of communicating processes with tight regions and tight multiplication*
15. ASP *algebra of synchronous processes*
18. $PA_\delta(\gamma)$ *process algebra with causality relations*
16. $PA_\delta(\mu_c^\sigma)$ *process algebra for ordered mail along channel c*
17. $PA_\delta(\mu_c^M)$ *process algebra for unordered mail along channel c*
19. ACP_θ *algebra of communicating processes with priorities*
20. ACP_{RS} *algebra of communicating processes with readies and failures*
21. $ACPK$ *algebra of communicating processes with Koomen's fair abstraction rule*
22. $BPA_{\tau X}$ *basic process algebra with abstraction and chaos*
23. ATS *algebra of trace sets*



1.3.

Building blocks of the various axiom systems for process algebra. A square or rectangle is a block of some axioms (always equational); a figure ⟨⟩ denotes a rule (or a block of rules) or a block of conditional axioms.

2.4. ABBREVIATIONS AND OPERATORS MENTIONED IN DIAGRAM OF BUILDING BLOCKS:

$+$	*alternative composition; sum*
$\cdot$	*sequential composition; multiplication; product*
$\|\|$	*parallel composition; merge (in absence of communication: 'free' merge)*
$\|\_$	*left-merge*
$\|$	*communication merge*
$\|_{syn}$	*merge for synchronous process cooperation*
δ	*deadlock or failure*
∂_H	*encapsulation operator w.r.t. H, a subset of the set of actions*
HA	*handshaking axiom*
SC	*axioms for standard concurrency*
ET	*expansion theorem*
PR	*projection axioms*
τ	*Milner's symbol for a silent action*
τ_I	*abstraction operator w.r.t. I, a subset of the set of actions*
θ	*operator realizing execution of a process in which the actions have certain partially ordered priorities*
:	*tight multiplication*
_	*tight region operator*
R	*axiom R describing readiness semantics*
S	*axiom S describing (in combination with S) failure semantics*
LR	*linear recursion*
τLR	*linear recursion with abstraction*
KFAR	*Koomen's fair abstraction rule*
RSP	*recursive specification principle*
AIP	*approximation induction principle*
α/β	*alphabet calculus*
μ_c^σ	*operator for asynchronous communication: ordered mail along channel c*
μ_c^M	*operator for asynchronous communication: unordered mail along c*
γ	*operator for asynchronous communication: causality*
^	*execution operator*
CHAR	*CHAOS abstraction rule*
χ	*Hoare's process CHAOS*

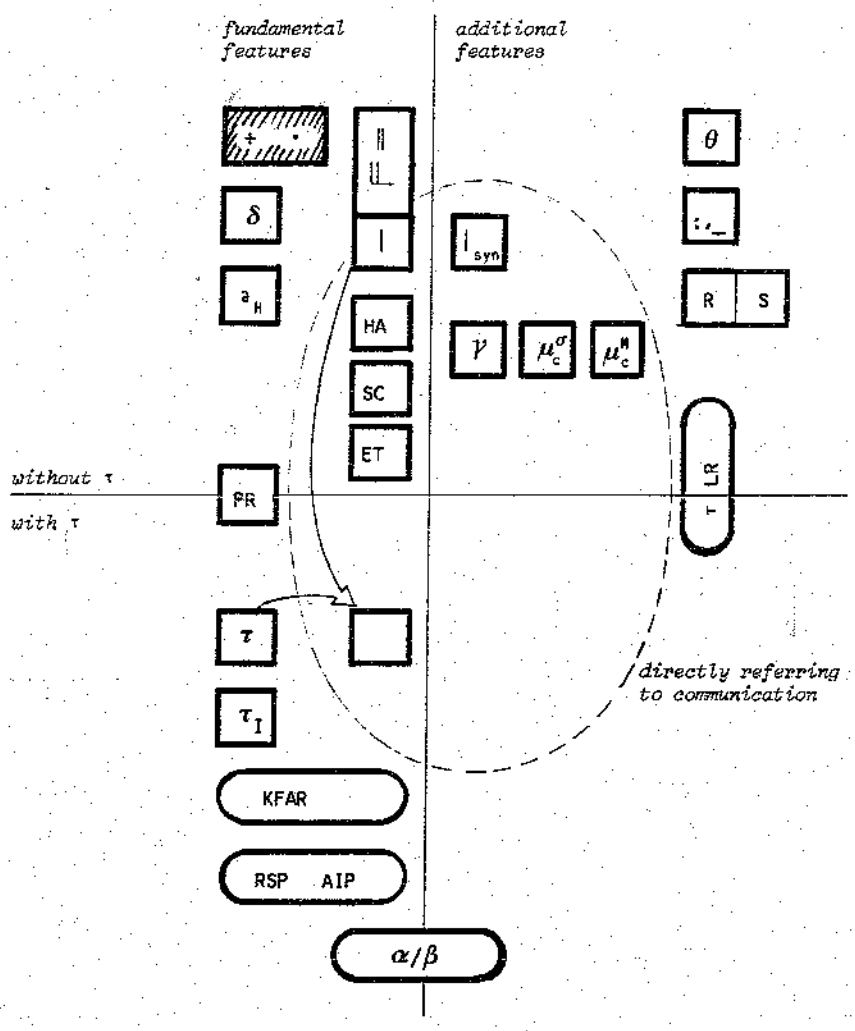
2.5.0. THE ALPHABET

The alphabet A from which processes are built, contains symbols a,b,c,... for atomic actions, also called: events, or: steps, or: atoms.

A may contain a distinguished element δ (deadlock). Another distinguished element which A may contain is τ (Milner's symbol for the invisible action). In the latter case we also use the notation A_τ ($= A \cup \{\tau\}$) with a,b,c,... ranging over A and u,v,... over A_τ.

The alphabet is always supposed to be finite. Often this requirement can be relaxed, but it helps to keep matters more 'algebraical'.

2.5.1.



BPA

$x+y=y+x$	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

basic process algebra

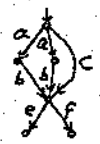
2.5.1, comments

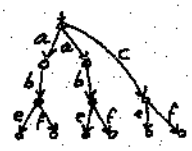
BPA, basic process algebra, consists of the building block +,. stating the properties that the basic constructors +,. of processes have: associativity of + and ., commutativity of + and the law of idempotency A3.

A1-3 say that the choices presenting themselves in a state of the process, form a set. The algebraical formalism can only speak about finite sets (there is no infinite sum operator, which would be much harder to specify in an algebraical way).(In the semantics the set of choices may have any cardinality depending on the process algebra (a model of some axiomatic system) under consideration.)

There is no distributive law $z(x+y) = zx+zy$ required. Of course it can be added: then one obtains a simple version of trace theory. A more useful algebraisation of trace theory is described in the system ATS (2.5.23).

The axioms A1,2,5 have an initial algebra which is isomorphic to a domain of process graphs $\mathbb{H}$, consisting of finite acyclic process graphs: e.g. $(ab+ab+c).(e+f)$ corresponds to figure (1).

(1) 

(2) 

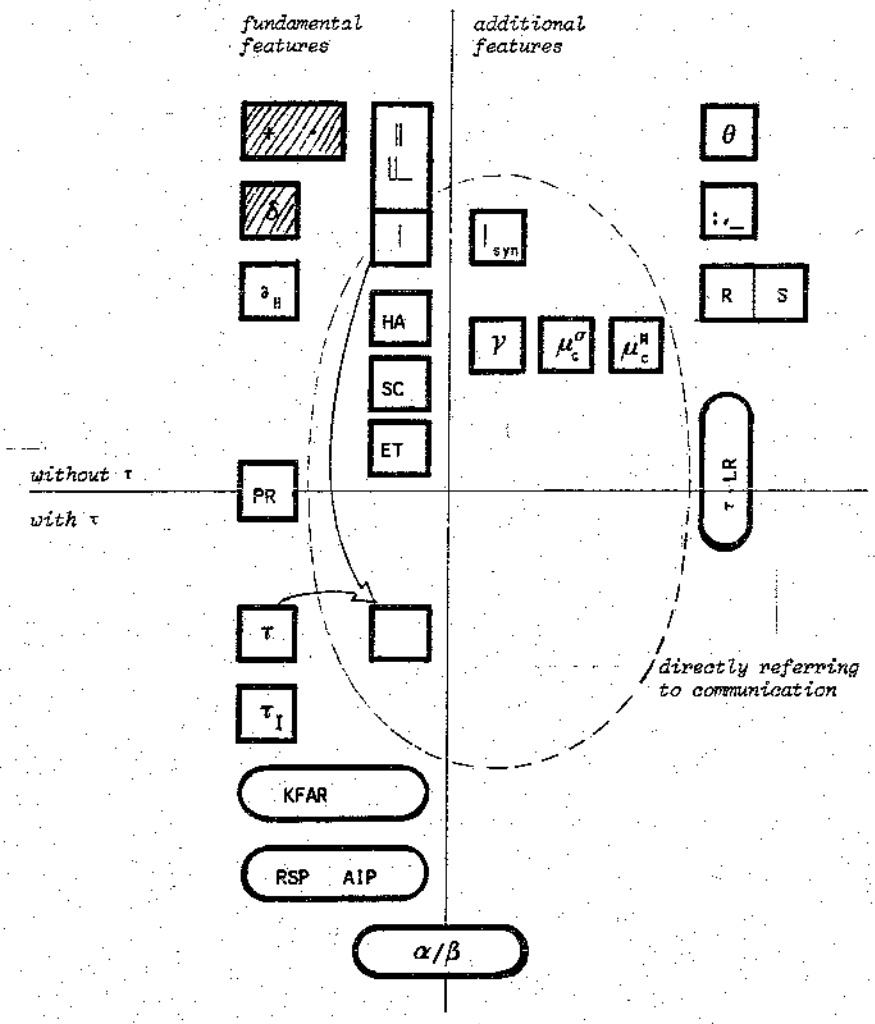
(3) 

The axioms A1,2,4,5 have an initial algebra isomorphic to the subdomain of $\mathbb{H}$ consisting of trees: e.g. the above term would correspond to figure (2). A1-5, that is BPA, has an initial algebra corresponding to finite process trees in which no two subtrees are the same. The above term would correspond to figure (3).

The operator + is as in Milner's CCS (Calculus of Communicating Systems): unlike the operators $\Box$,$\sqcap$ in Hoare's CSP (Communicating Sequential Processes) it does not say how the choice between a,b in a+b is made (by external influences, or by internal non-determinism).

The operator . is present in CSP and the process theory of De Bakker & Zucker but not in CCS where a weaker form of product (prefix multiplication) is used. In prefix multiplication, only a.x for an atom a and a process x can be formed. So (a+b).c would not be well-formed. The reason for adopting general multiplication is its greater defining power for infinite processes.

2.5.2.



BPA_δ

$x + y = y + x$	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

basic process algebra with deadlock

2.5.2, comments

δ is the new building block. Axioms A6,7 state that δ is deadlock, or the 'action' in which a process acknowledges stagnation.

δ is not the same as NIL in CCS; NIL satisfies only A6.

δ occurs in De Bakker & Zucker (1982a,b) as $\emptyset$.

Example: $a(bc + bc\delta)$ is a process in which one 'branch' terminates successfully and one unsuccessfully, in δ.

The presence of a process as δ in $\delta x = \delta$ can be made plausible as follows (where some ingredients from an axiom system below, in casu ACP, are used):

Let $p = \partial_{\{a,b\}}(a \| b)$ be such that actions a,b do not communicate.

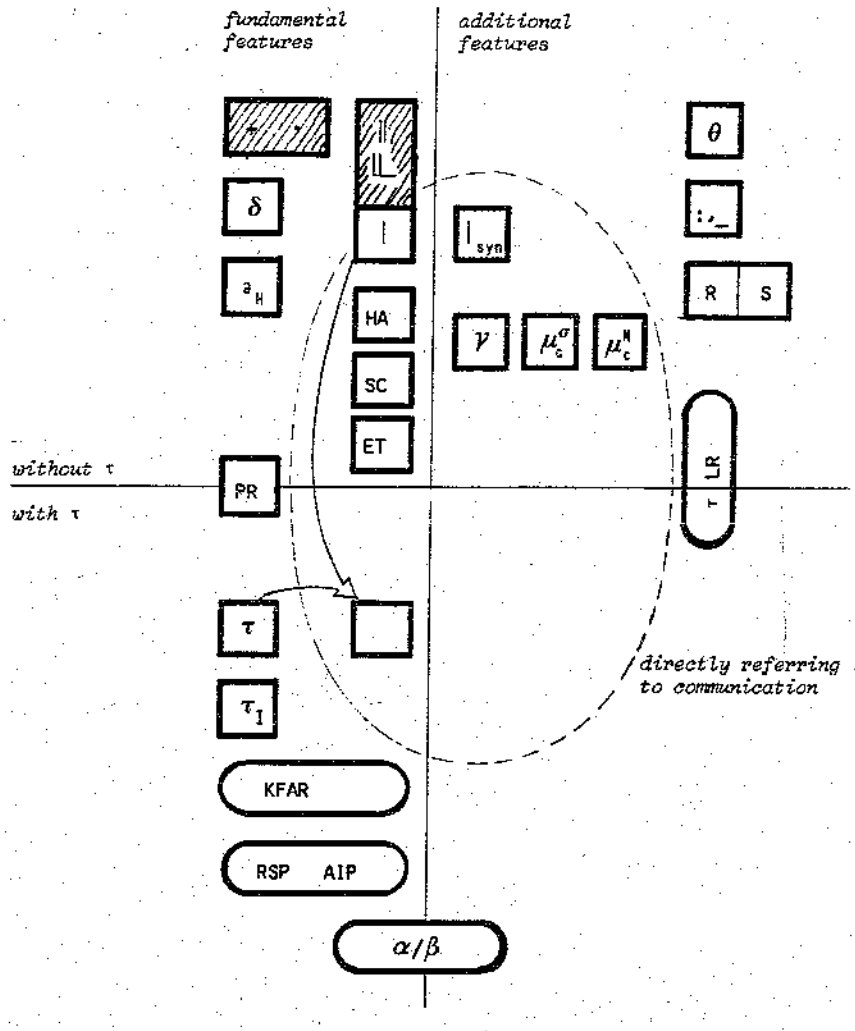
Then in

$$\partial_{\{a,b\}}(a \| b) \cdot x$$

control does not reach x, i.e. $\partial_{\{a,b\}}(a \| b) \cdot x = \partial_{\{a,b\}}(a \| b)$.

Note that in prefix multiplication, the product $p \cdot x$ cannot be formed.

2.5.3.



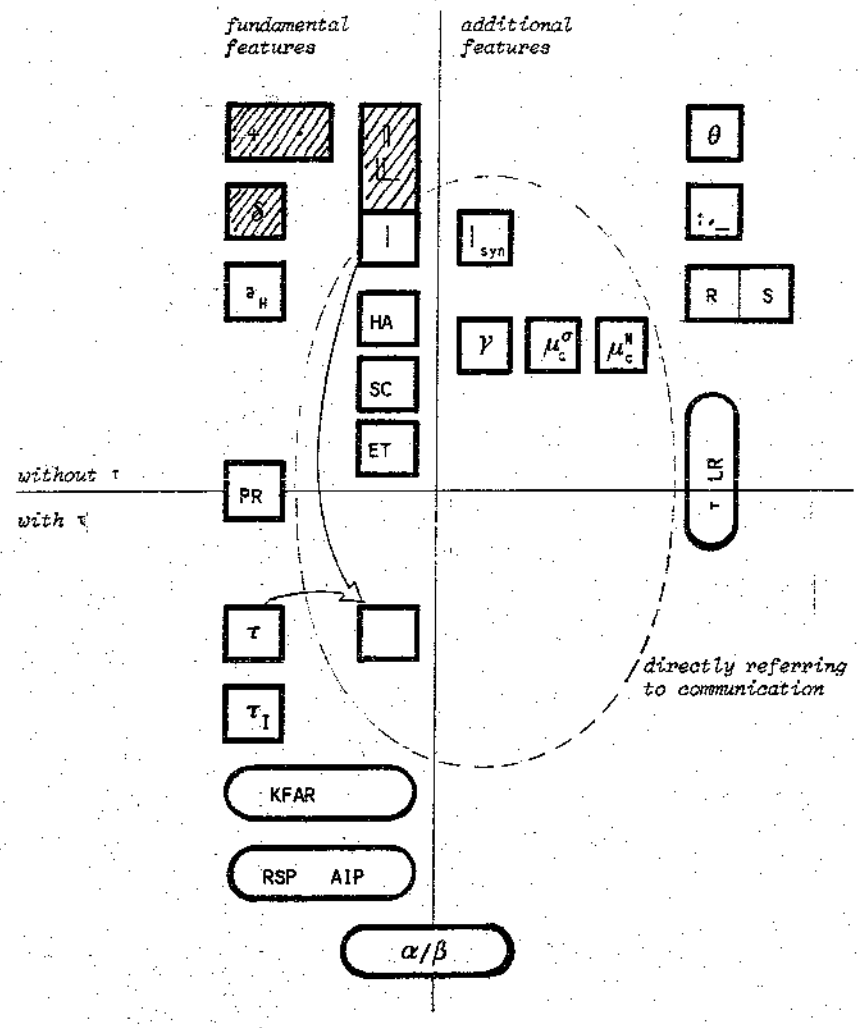
The axiom system *PA* consists of the following list of axioms:

$x+y = y+x$	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
$x \between y = x \lfloor y + y \lfloor x$	M1
$a \lfloor x = a\cdot x$	M2
$ax \lfloor y = a(x \between y)$	M3
$(x+y) \lfloor z = x \lfloor z + y \lfloor z$	M4

process algebra

Comments and examples: see survey report CS-R8421 chapter 1.

2.5.4.

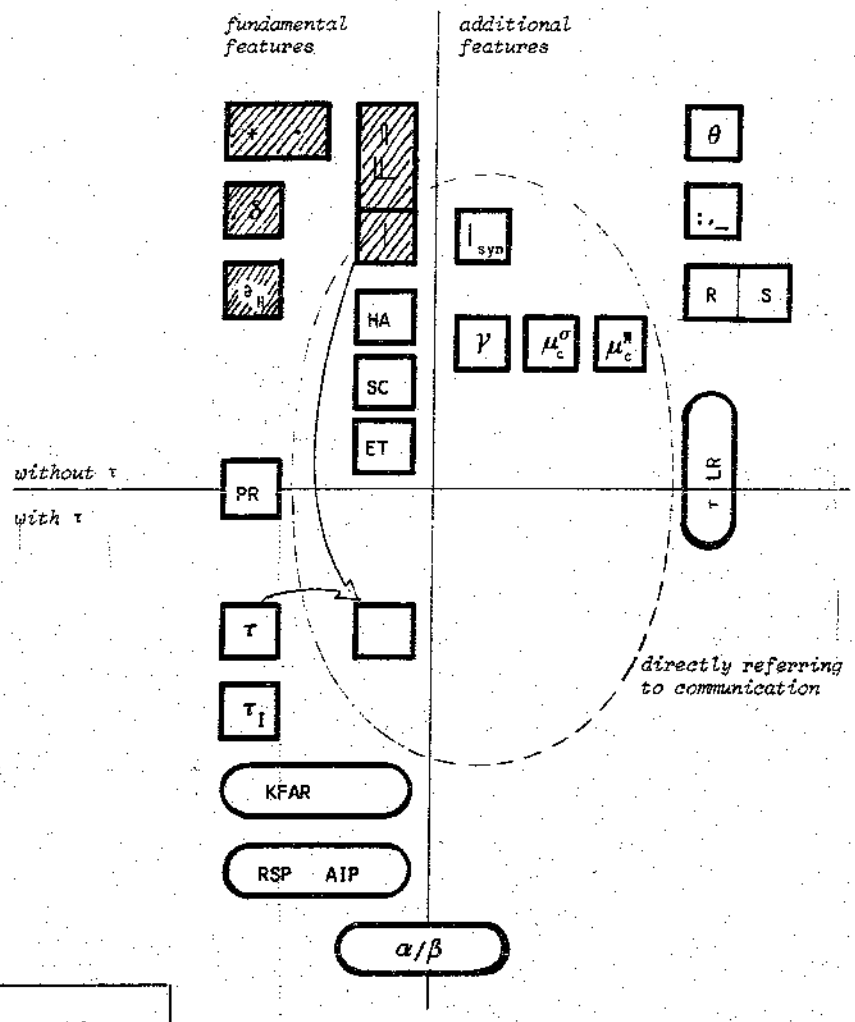


PA_δ

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$x(y + z) = xy + xz$	A4
$(xy)z = x(yz)$	A5
$x + \delta = x$	A6
$\delta x = \delta$	A7
$x \parallel y = x \between y + y \between x$	M1
$a \between x = ax$	M2
$(ax) \between y = a(x \parallel y)$	M3
$(x + y) \between z = x \between z + y \between z$	M4

process algebra with deadlock

1.5.5.



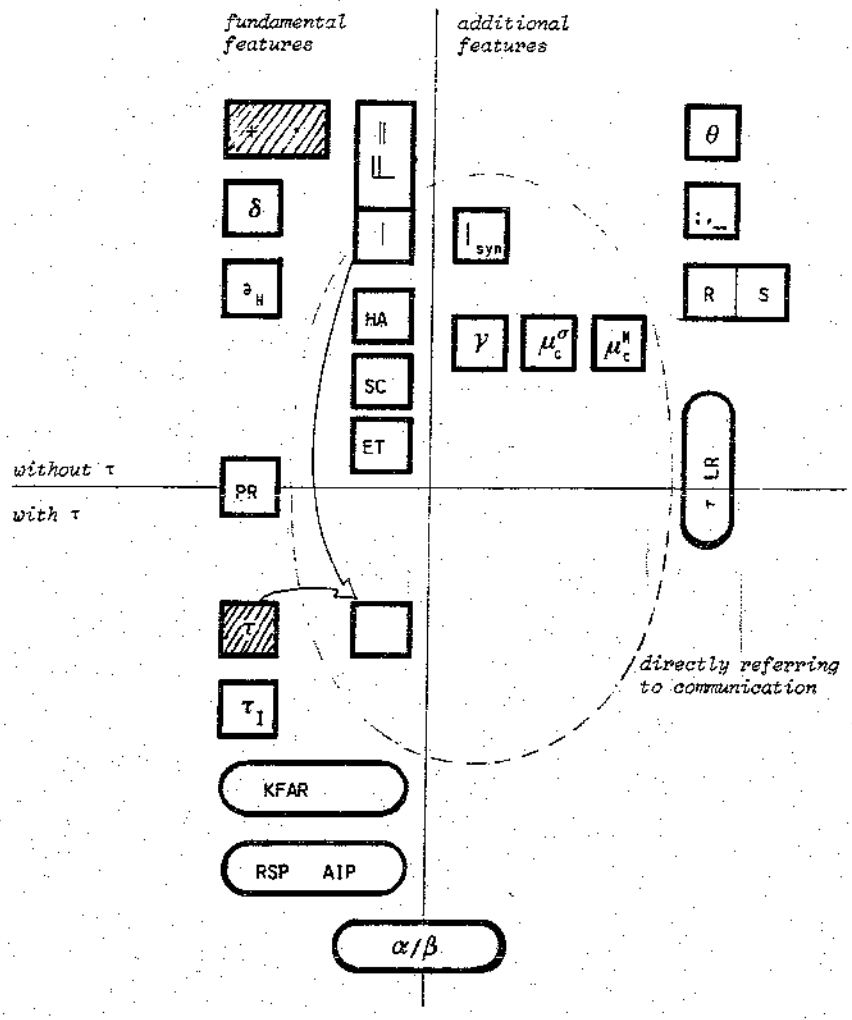
ACP

$x+y = y+x$	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
$x\between y = x\between y+y\between x+x\mid y$	CM1
$a\between x = a\cdot x$	CM2
$(ax)\between y = a(x\between y)$	CM3
$(x+y)\between z = x\between z+y\between z$	CM4
$(ax)\mid b = (a\mid b)\cdot x$	CM5
$a\mid(bx) = (a\mid b)\cdot x$	CM6
$(ax)\mid(by) = (a\mid b)\cdot(x\between y)$	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

Algebra of communicating processes

Comments and examples: see survey report CS-R8421 chapter 2.

2.5.6.



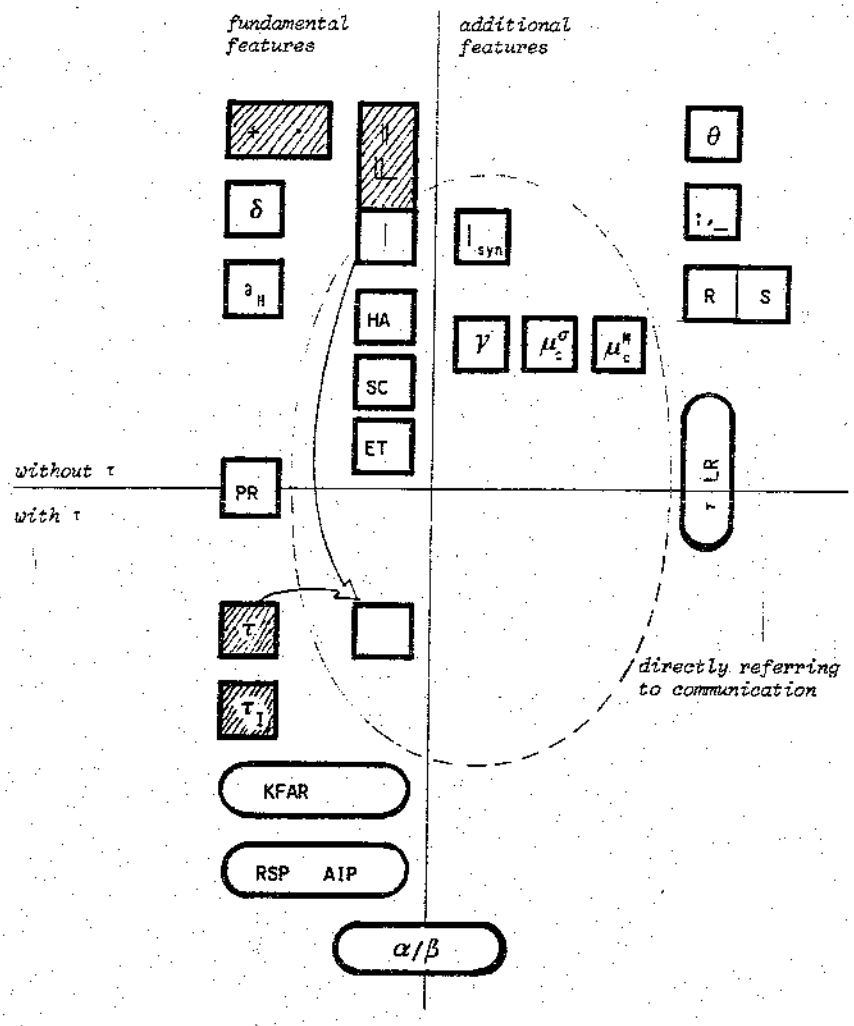
BPA_τ:

$x+y=y+x$	A1
$(x+y)+z=x+(y+z)$	A2
$x+x=x$	A3
$(x+y)z=xz+yz$	A4
$(xy)z=x(yz)$	A5
$x\tau=x$	T1
$\tau x+x=\tau x$	T2
$a(\tau x+y)=a(\tau x+y)+ax$	T3

basic process algebra with abstraction

Comments: see 2.5.8.

2.5.7.



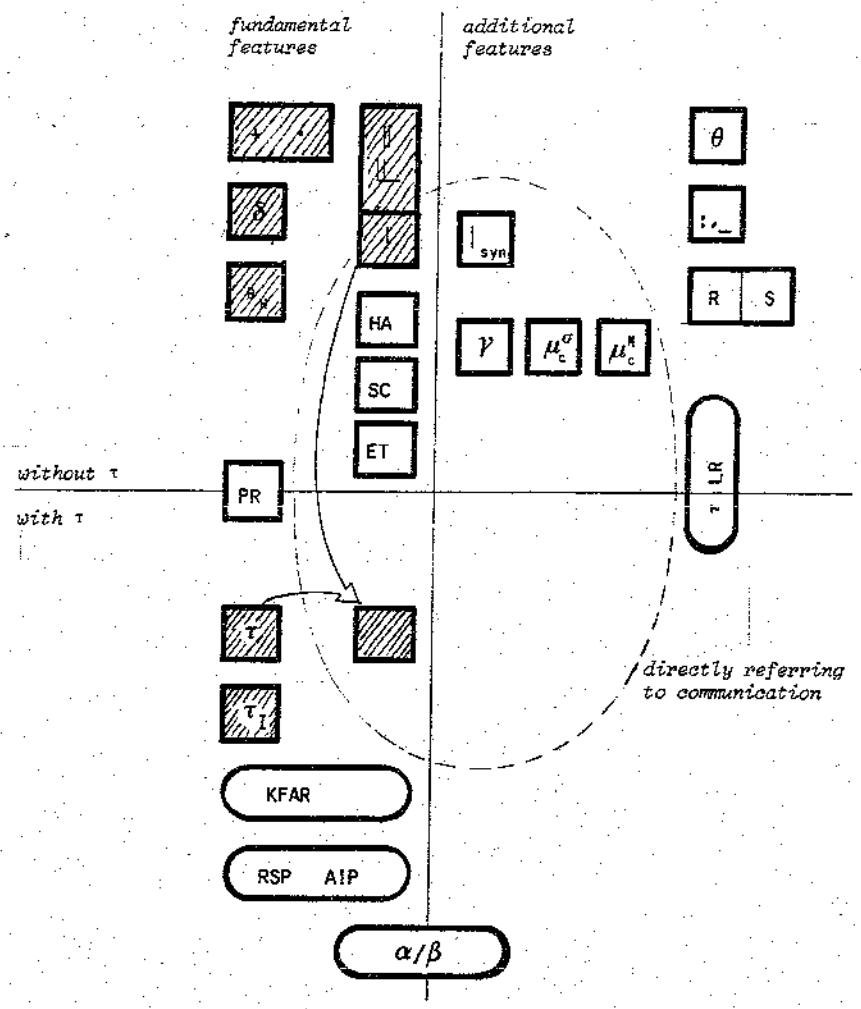
PA_τ

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

process algebra with abstraction

Comments: see 2.5.8; and survey part I (report CS-R8421)

2.5.8.



ACP_τ

$x+y = y+x$	A1	$x\tau = x$	T1
$x+(y+z) = (x+y)+z$	A2	$\tau x + x = \tau x$	T2
$x+x = x$	A3	$a(\tau x+y) = a(\tau x+y)+ax$	T3
$(x+y)z = xz+yz$	A4		
$(xy)z = x(yz)$	A5		
$x+\delta = x$	A6		
$\delta x = \delta$	A7		
$a\|b = b\|a$	C1		
$(a\|b)\|c = a\|(b\|c)$	C2		
$\delta\|a = \delta$	C3		
$x\|\|y = x\lfloor\!\lfloor y + y\lfloor\!\lfloor x + x\|y$	CM1		
$a\lfloor\!\lfloor x = ax$	CM2	$\tau\lfloor\!\lfloor x = \tau x$	TM1
$(ax)\lfloor\!\lfloor y = a(x\|\|y)$	CM3	$(\tau x)\lfloor\!\lfloor y = \tau(x\|\|y)$	TM2
$(x+y)\lfloor\!\lfloor z = x\lfloor\!\lfloor z + y\lfloor\!\lfloor z$	CM4	$\tau\|x = \delta$	TC1
$(ax)\|b = (a\|b)x$	CM5	$x\|\tau = \delta$	TC2
$a\|(bx) = (a\|b)x$	CM6	$(\tau x)\|y = x\|y$	TC3
$(ax)\|(by) = (a\|b)(x\|\|y)$	CM7	$x\|(\tau y) = x\|y$	TC4
$(x+y)\|z = x\|z + y\|z$	CM8		
$x\|(y+z) = x\|y + x\|z$	CM9		
		$\partial_H(\tau) = \tau$	DT
		$\tau_I(\tau) = \tau$	TI1
$\partial_H(a) = a$ if $a \notin H \subseteq A$	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

2.5.8, comments

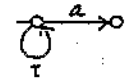
ACP_τ contains, as compared to ACP, three new building blocks.

τ T1-3 are Milner's τ-laws. These equations are maybe not very revealing at first sight - but they correspond (already in BPA_τ) precisely to an elegant notion of process equivalence: $r\tau$-bisimulation (not treated here). ($r\tau$-bisimulation is also known as observational congruence in Milner (1980).)

In fact, CCS as in Milner (1980) can be seen as the reduct of ACP_τ after leaving out $\between$, $|$, and τ_I.

□ is the 'spin-off' of the combination of the τ-axioms T1-3 and the axioms for $\between$ and $|$ (and hence has no parallel in CCS):

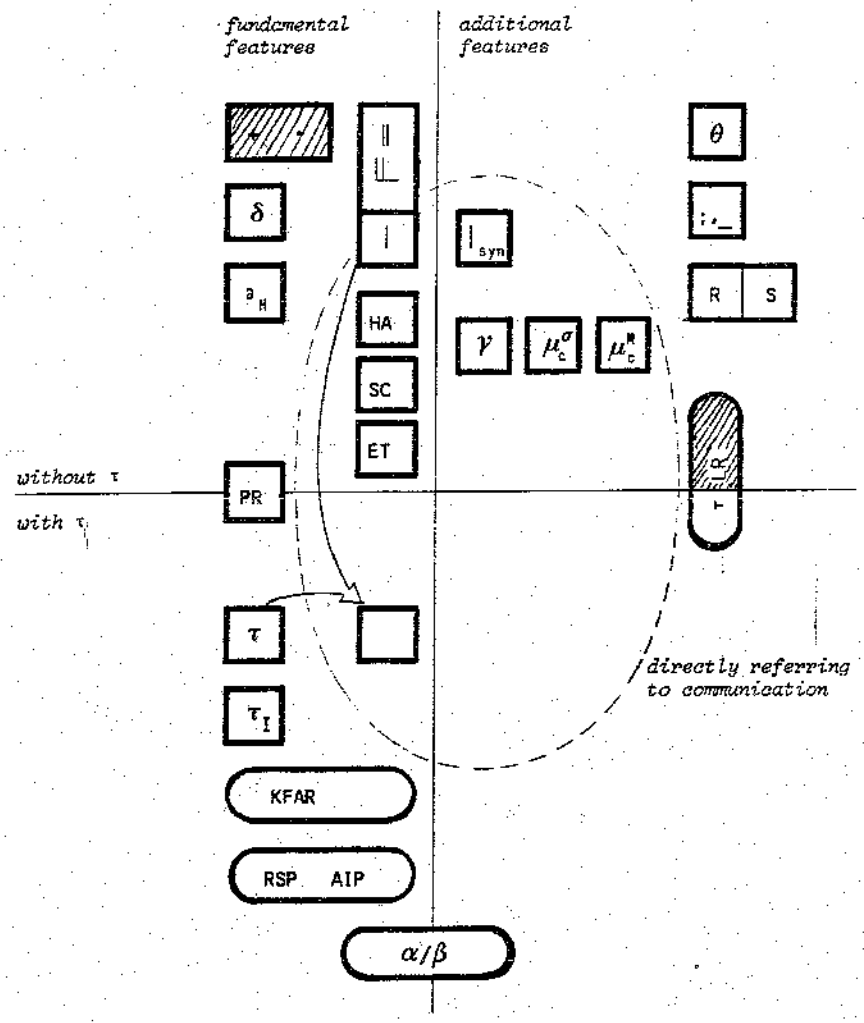
$\tau \between x = \tau x$	TM1
$(\tau x) \between y = \tau(x \between y)$	TM2
$\tau \mid x = \delta$	TC1
$x \mid \tau = \delta$	TC2
$(\tau x) \mid y = x \mid y$	TC3
$x \mid (\tau y) = x \mid y$	TC4

τ_I τ_I is an operator (the abstraction operator) of fundamental importance for this style of process algebra. While in CCS, a pair of communicating atoms a,b yields τ at once: $a|b = \tau$, here this takes two steps and we distinguish the effect of communication from abstraction. A communication yields an internal step i (internal but still visible), which can be abstracted to the invisible step τ. This is crucial because otherwise processes as $\underline{X}$ =  simply cannot be specified (= defined recursively).

For, 'defining' $\underline{X}$ by the recursion equation $X = a + \tau X$ does not work since it turns out that such equations (in which not all guards are non-τ atoms) may have many non-intended solutions. E.g. every $\underline{X} = \tau(a + q)$, q arbitrary, satisfies $X = a + \tau X$ as can be shown by a simple computation using the τ-laws. However, using τ_I it is easy to specify $\underline{X}$:

$$Y = a + iY$$
$$X = \tau_{\{i\}}(Y).$$

2.5.9.



BPA_{LR}	$x+y=y+x$	A1
	$(x+y)+z=x+(y+z)$	A2
	$x+x=x$	A3
	$(x+y)z=xz+yz$	A4
	$(xy)z=x(yz)$	A5
	$\dfrac{x_i=<X_i \mid E>, i=1,\ldots,n}{x_1=T_1(\vec{x})}$	R1
	$\dfrac{x_i=T_i(\vec{x}), i=1,\ldots,n}{x_1=<X_1 \mid E>}$ $T_i(\vec{X})$ *is A−guarded*	R2

basic process algebra with linear recursion

2.5.9, comments

BPA_{LR} is merely a reformulation of Milner's system M (in Milner(1984)):

M:	$x+0=x$	A0
	$x+y=y+x$	A1
	$(x+y)+z=x+(y+z)$	A2
	$x+x=x$	A3
	$\mu X.T(X)=\mu Y.T(Y)$	$\mu 0$
	$\mu X.T(X)=T(\mu X.T(X))$	$\mu 1$
	$\frac{x=T(x)}{x=\mu X.T(X)}\ T(X)$ guarded	$\mu 2$
	$\mu X(X+T)=\mu X(T)$	$\mu 3$

The presence of 0 (the NIL of CCS) and the use of prefix multiplication is not essential here, in the context of regular processes. The notation of μ-calculus is equivalent to our construct$\langle X_1 | E\rangle$, denoting a system E of recursion equations: $E = \{X_i = t_i(X_1,..,X_n) \mid i=1,..,n\}$.

For instance, $\mu X\ (aX)$ is in BPA_{LR}: $\langle X \mid X = aX\rangle$;

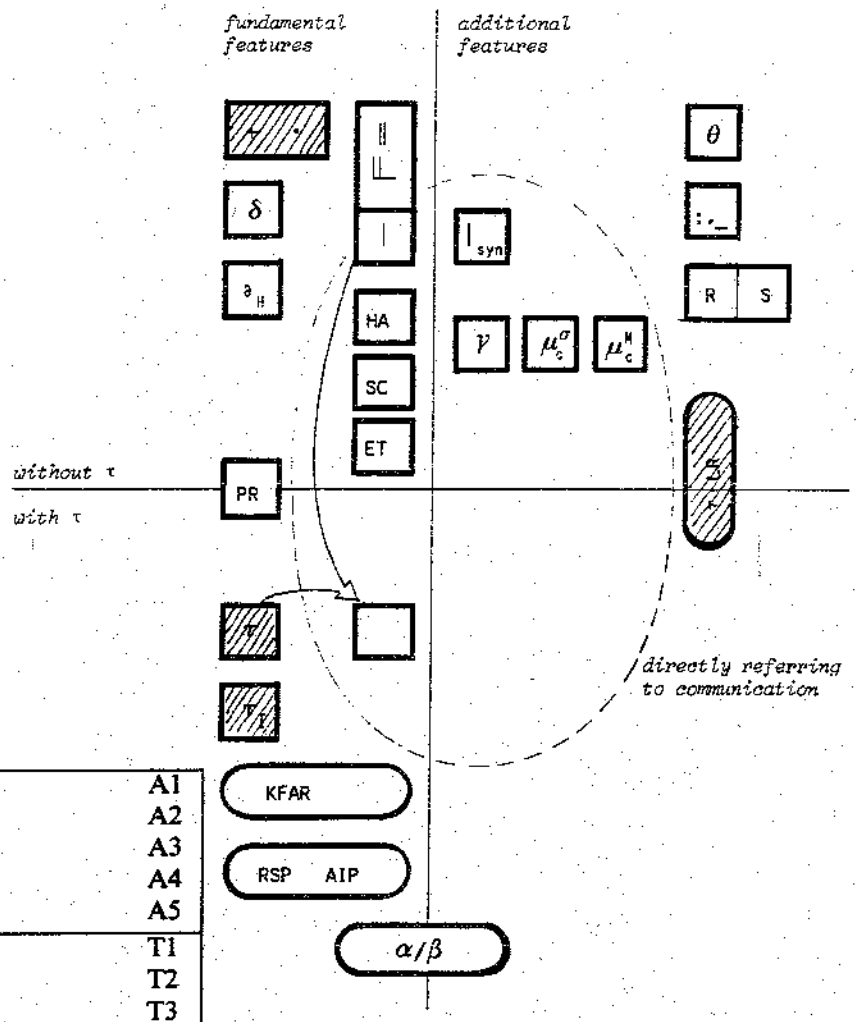
$(\mu X\ (a(\mu Y\ (bY))$ is $\langle X \mid X = aY,\ Y = bY\rangle$.

Example: $M \vdash \mu X\ (aX) = \mu Y\ (aY + \mu Z\ (aZ))$.

$BPA_{LR} \vdash \langle X \mid X = aX\rangle = \langle Y \mid Y = aY + aZ,\ Z = aZ\rangle$.

For a completeness result (due to Milner for M), see the table in chapter 3.

2.5.10.

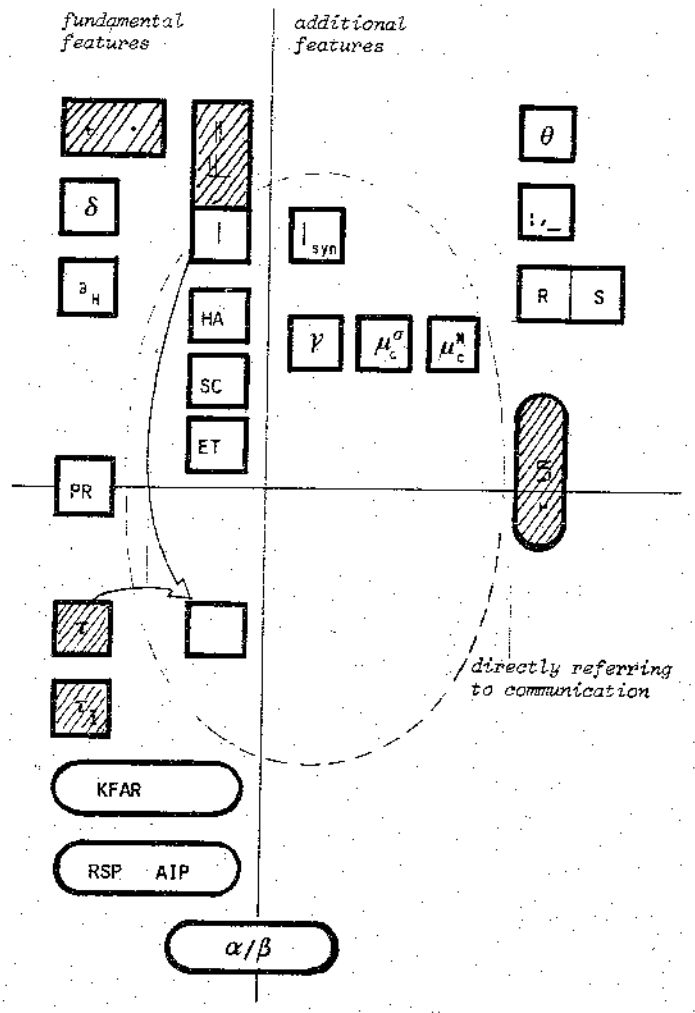


$BPA_{\tau LR}$

$x+y=y+x$	A1
$(x+y)+z=x+(y+z)$	A2
$x+x=x$	A3
$(x+y)z=xz+yz$	A4
$(xy)z=x(yz)$	A5
$x\tau=x$	T1
$\tau x+x=\tau x$	T2
$a(\tau x+y)=a(\tau x+y)+ax$	T3
$\tau_I(X)=X$	TI0
$\tau_I(\tau)=\tau$	TI1
$\tau_I(a)=\tau$ if $a\in I$	TI2
$\tau_I(a)=a$ if $a\notin I$	TI3
$\tau_I(x+y)=\tau_I(x)+\tau_I(y)$	TI4
$\tau_I(\tau y)=\tau\cdot\tau_I(y)$	TI5′
$\tau_I(a y)=\tau_I(a)\cdot\tau_I(y)$	TI5″
$\tau_I(<X_1\mid E>)=<X_1\mid\tau_I(E)>$	TI6
$\dfrac{x_i=<X_i\mid E>,i=1,\ldots,n}{x_1=T_1(\vec{x})}$	R1
$\dfrac{x_i=T_i(\vec{x}),i=1,\ldots,n}{x_1=<X_1\mid E>}\ T_i(\vec{X})$ is A − guarded	R2
$\dfrac{BPA_{\tau,I}\vdash E=E'}{<X_1\mid E>=<X_1\mid E'>}$	R3
$\dfrac{\tau<X_k\mid E>=\tau<X_l\mid E>\text{ for some } k,l.\ k\neq l}{<X_1\mid E>=<X_1\mid E_{-k},X_k=T_k(\vec{X})+\tau X_l>}$	R4
$<X_1\mid E_{-k},X_k=\tau X_k>=<X_1\mid E_{-k},X_k=\tau>$	R5

basic process algebra with abstraction and linear recursion

2.5.11.



$PA_{\tau LR}$

$x+y=y+x$	A1
$(x+y)+z=x+(y+z)$	A2
$x+x=x$	A3
$(x+y)z=xz+yz$	A4
$(xy)z=x(yz)$	A5
$x \mathbin{\|} y = x \mathbin{\lfloor} y + y \mathbin{\lfloor} x$	M1
$(ax) \mathbin{\lfloor} y = a(x \mathbin{\|} y)$	M2
$a \mathbin{\lfloor} y = ay$	M3
$(x+y) \mathbin{\lfloor} z = x \mathbin{\lfloor} z + y \mathbin{\lfloor} z$	M4
$x\tau = x$	T1
$\tau x + x = \tau x$	T2
$a(\tau x + y) = a(\tau x + y) + ax$	T3
$\tau_I(X) = X$	TI0
$\tau_I(\tau) = \tau$	TI1
$\tau_I(a) = \tau$ if $a \in I$	TI2
$\tau_I(a) = a$ if $a \notin I$	TI3
$\tau_I(x+y) = \tau_I(x) + \tau_I(y)$	TI4
$\tau_I(\tau \cdot y) = \tau \cdot \tau_I(y)$	TI5′
$\tau_I(a \cdot y) = \tau_I(a) \cdot \tau_I(y)$	TI5″
$\tau_I(<X_1 \mid E>) = <X_1 \mid \tau_I(E)>$	TI6
$\dfrac{x_i = <X_i \mid E>, i = 1,\ldots,n}{x_1 = T_1(\vec{x})}$	R1
$\dfrac{x_i = T_i(\vec{x}), i = 1,\ldots,n}{x_1 = <X_1 \mid E>}$ $T_i(\vec{X})$ is A − guarded	R2
$\dfrac{BPA_{\tau_I} + E = E'}{<X_1 \mid E> = <X_1 \mid E'>}$	R3
$\dfrac{\tau<X_k \mid E> = \tau<X_\ell \mid E> \text{ for some } k,\ell \neq 1}{<X_1 \mid E> = <X_1 \mid E_{-k}, X_k = T_k(\vec{X}) + \tau X_\ell>}$	R4
$<X_1 \mid E_{-k}, X_k = \tau X_k> = <X_1 \mid E_{-k}, X_k = \tau>$	R5

2.5.11, comments

Example. The following example can already be treated in the subsystem PA_{LR} of $PA_{\tau LR}$, resulting from omitting the axioms and rules referring to τ. So PA_{LR} is BPA_{LR} (2.5.9) plus M1-4 for $\|, \lfloor\!\lfloor$.

Let $\underline{X}$ abbreviate $\langle X \mid X = a(X + b)\rangle$, and let $\underline{Y}$ abbreviate $\langle Y \mid Y = cY\rangle$.
(In the well-known notation of μ-calculus, we would have $\underline{X} \equiv \mu X\,(a(X + b))$ and $\underline{Y} \equiv \mu Y\,(cY)$.)

Then:

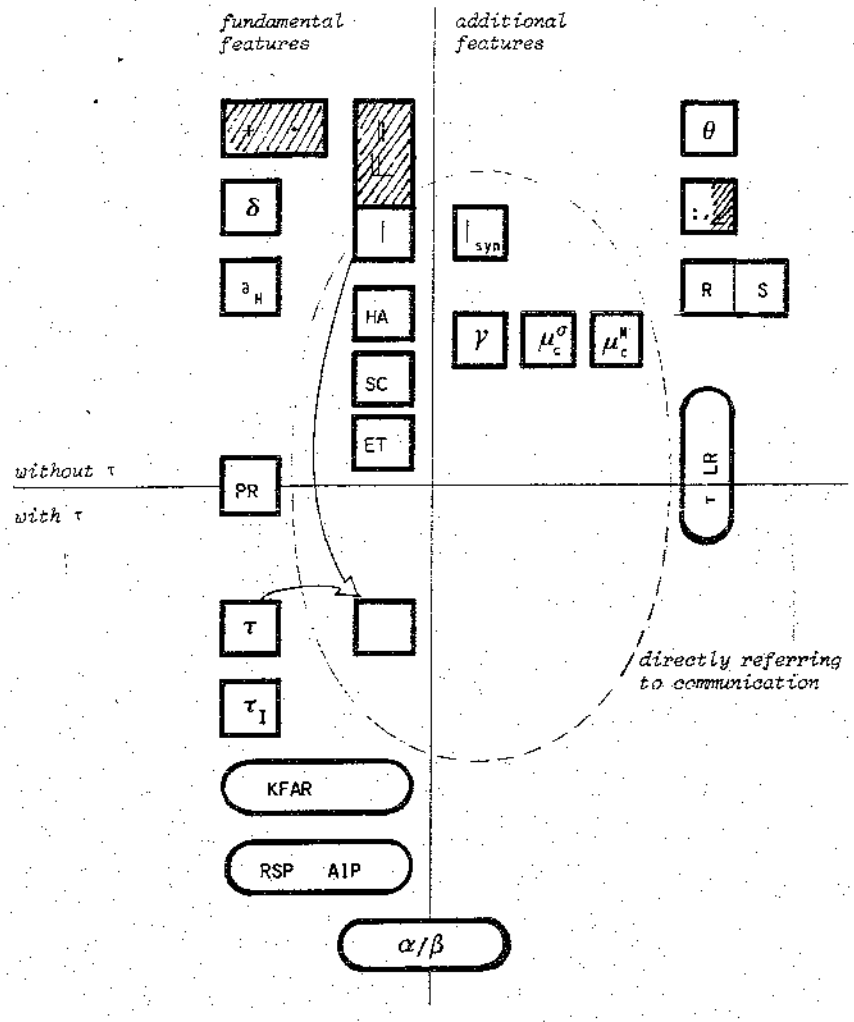
$$\underline{X}\|\underline{Y} = \underline{X}\lfloor\!\lfloor\underline{Y} + \underline{Y}\lfloor\!\lfloor\underline{X} = a(\underline{X}+b)\lfloor\!\lfloor\underline{Y} + c\underline{Y}\lfloor\!\lfloor\underline{X} =$$
$$a((\underline{X}+b)\|\underline{Y}) + c(\underline{Y}\|\underline{X})$$
$$(\underline{X}+b)\|\underline{Y} = (\underline{X}+b)\lfloor\!\lfloor\underline{Y} + \underline{Y}\lfloor\!\lfloor(\underline{X}+b) =$$
$$\underline{X}\lfloor\!\lfloor\underline{Y} + b\lfloor\!\lfloor\underline{Y} + \underline{Y}\lfloor\!\lfloor(\underline{X}+b) =$$
$$a(\underline{X}+b)\lfloor\!\lfloor\underline{Y} + b\underline{Y} + c\underline{Y}\lfloor\!\lfloor(\underline{X}+b) =$$
$$a((\underline{X}+b)\|\underline{Y}) + b\underline{Y} + c(\underline{Y}\|(\underline{X}+b)).$$

So, putting $\underline{U} \equiv \underline{X}\|\underline{Y}$ and $\underline{V} \equiv (\underline{X}+b)\|\underline{Y}$ we have found guarded linear recursion equations for $\underline{U},\underline{V},\underline{Y}$.
Hence by R2:

$$\underline{U} \equiv \underline{X}\|\underline{Y} = \langle U \mid U = aV + cU,\ V = aV + bY + cV,\ Y = cY\rangle.$$

For a completeness result about $PA_{\tau LR}$: see table in chapter 3.

2.5.12.



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		
$x \parallel y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$	M1		
$a \lfloor\!\lfloor x = ax$	M2	$\underline{x} \lfloor\!\lfloor y = \underline{x} \cdot y$	TRM1
$ax \lfloor\!\lfloor y = a(x \parallel y)$	M3	$\underline{x} \cdot y \lfloor\!\lfloor z = \underline{x}(y \parallel z)$	TRM2
$(x+y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	M4		
$\underline{a} = a$	TR1	$\phi(a) = a$	F1
$\underline{x+y} = \underline{x} + \underline{y}$	TR2	$\phi(x+y) = \phi(x) + \phi(y)$	F2
$\underline{\underline{x}} = \underline{x}$	TR3	$\phi(\underline{x}) = \phi(x)$	F3
		$\phi(x \cdot y) = \phi(x) \cdot \phi(y)$	F4

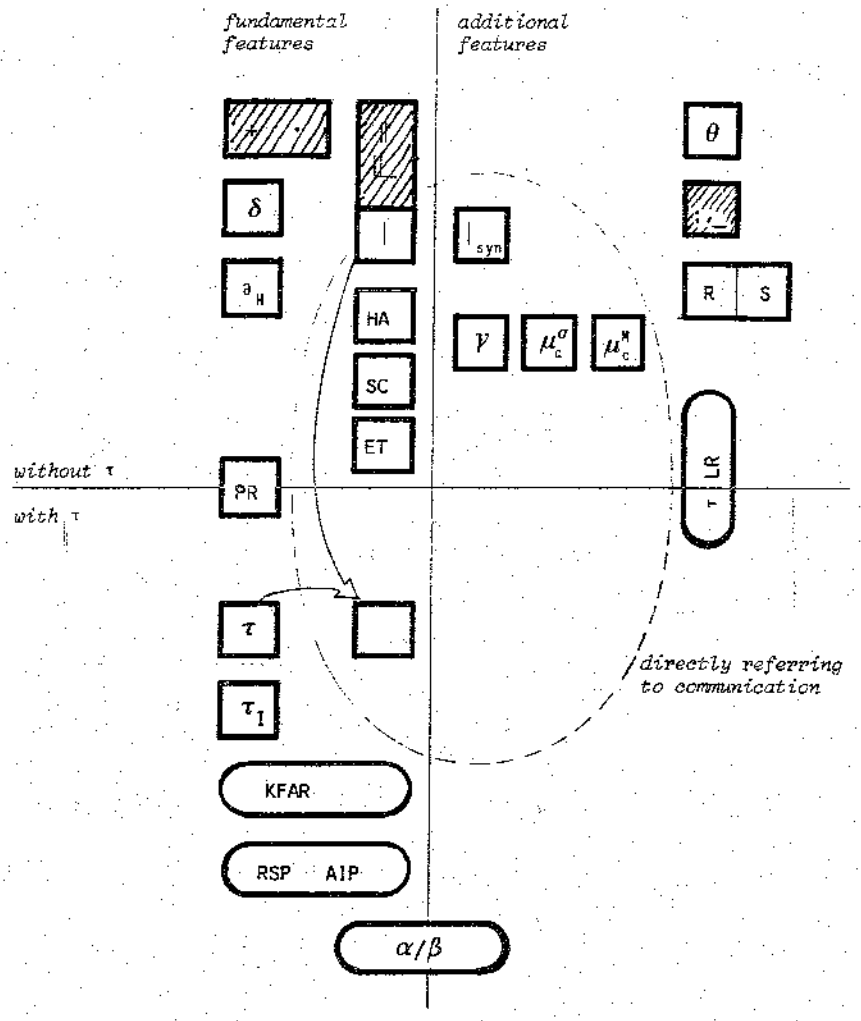
The Tight Region Operator

In the framework of ACP as introduced above, one can treat process cooperation where processes have tight regions which are to be executed without any interruption. This is substantially more complicated than the following more direct way: 2.5.12 contains an axiom system AMP for processes with tight regions without communication. It is an extension of the axiom system PA for free merge in 2.5.3 : the additions in the signature consist of an unary operator $x \mapsto \underline{x}$, the tight region operator (in the literature $\underline{x}$ is also denoted as $\langle x \rangle$), and an inverse operator ϕ which removes the constraints of tight regions. Intuitively, the underlined parts in a process expression (the tight regions) are to be executed in a cooperation 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 2.5.12 :

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

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

2.5.13.



AMP(:)

$x + y = y + x$	A1		
$(x + y) + z = x + (y + z)$	A2		
$x + x = x$	A3		
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4	$(x + y) : z = x : z + y : z$	AT1
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5	$(x : y) : z = x : (y : z)$	AT2
		$(x : y) \cdot z = x : (y \cdot z)$	AT3
		$(x \cdot y) : z = x \cdot (y : z)$	AT4
$x \parallel y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x$	M1		
$a \mathbin{\lfloor\!\lfloor} y = ay$	M2		
$ax \mathbin{\lfloor\!\lfloor} y = a(x \parallel y)$	M3	$(a : x) \mathbin{\lfloor\!\lfloor} y = a : (x \mathbin{\lfloor\!\lfloor} y)$	TRM
$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$	M4		
$\underline{a} = a$	TR1	$\phi(a) = a$	F1
$\underline{x + y} = \underline{x} + \underline{y}$	TR2	$\phi(x + y) = \phi(x) + \phi(y)$	F2
$\underline{\underline{x}} = \underline{x}$	TR3	$\phi(\underline{x}) = \phi(x)$	F3
$\underline{x \cdot y} = \underline{x} : \underline{y}$	TR4	$\phi(x \cdot y) = \phi(x) \cdot \phi(y)$	F4
$\underline{x : y} = \underline{x} : \underline{y}$	TR5	$\phi(x : y) = \phi(x) \cdot \phi(y)$	F5

process algebra with tight regions and tight multiplication

2.5.13, comments

Tight Multiplication

A shortcoming in expressive power of the tight region operator in AMP is that it does not allow us 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. 2.5.13 contains an axiom system AMP(:) which is an extension of AMP by this new operator and corresponding axioms.

The axiom system AMP(:) is redundant when only *finite* processes are considered: then "$\_$" 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 other, e.g., TR3.

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).

Remark. Note that the axioms in 2.5.13 for AMP:

$$\underline{x} \lfloor\!\lfloor \underline{y} = \underline{xy} \qquad \text{(TRM1)}$$

$$\underline{xy} \lfloor\!\lfloor \underline{z} = \underline{x}(\underline{y} \| \underline{z}) \qquad \text{(TRM2)}$$

and their immediate consequence

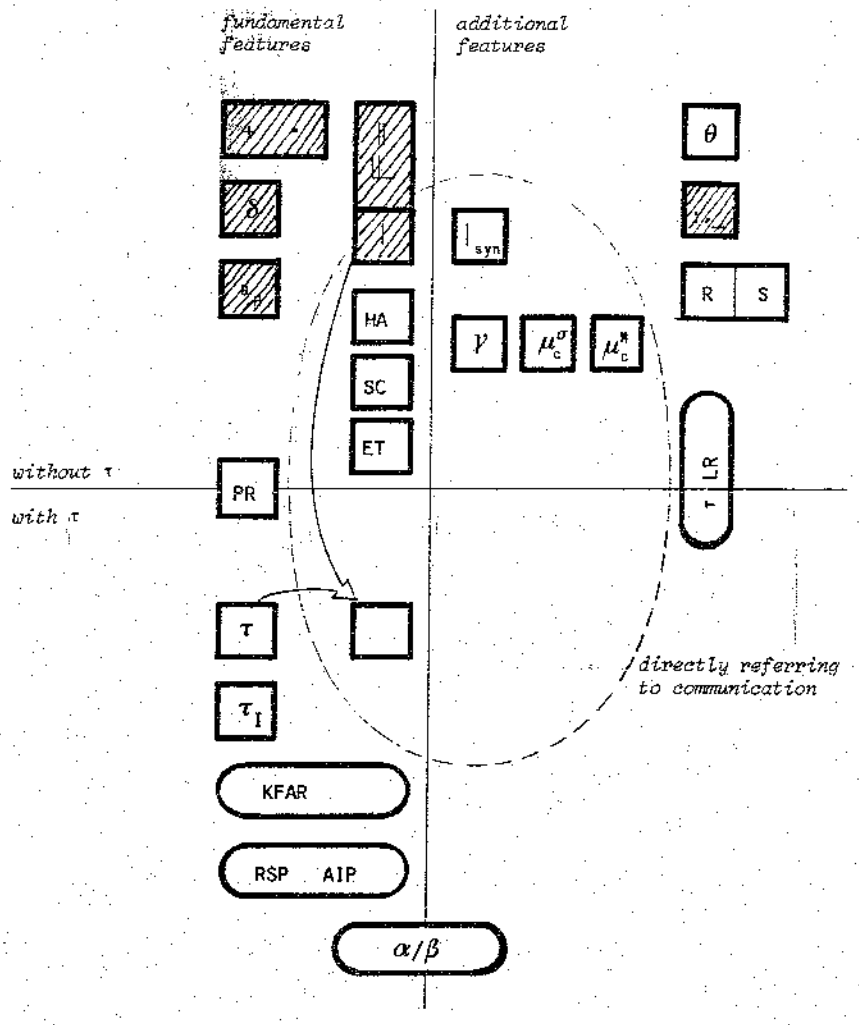
$$\underline{x} \| \underline{y} = \underline{x}\,\underline{y} + \underline{y}\,\underline{x} \qquad \text{(Proposition in 2.5.12)}$$

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

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

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

2.5.14.



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	$(x:y)\cdot z=x:(y\cdot z)$	AT3
$(x+y)z=xz+yz$	A4	$(x\cdot y):z=x\cdot(y:z)$	AT4
$(xy)z=x(yz)$	A5	$\delta:x=\delta$	AT5
$x+\delta=x$	A6		
$\delta x=\delta$	A7	$(a:x)\between y=a:(x\between y+x\mid y)$	CTRM1
		$(a:x)\mid(b:y)=(a\mid b):(x\mid y)$	CTRM2
		$(a:x)\mid(by)=(a\mid b):(x\between y+x\mid y)$	CTRM3
		$(a:x)\mid b=(a\mid b):x$	CTRM4
$a\mid b=b\mid a$	C1		
$(a\mid b)\mid c=a\mid(b\mid c)$	C2		
$a\mid\delta=\delta$	C3		
$x\parallel y=x\between y+y\between x+y\mid x$	CM1	$\underline{a}=a$	TR1
$a\between x=ay$	CM2	$\underline{x+y}=\underline{x}+\underline{y}$	TR2
$ax\between y=a(x\parallel y)$	CM3	$\underline{\underline{x}}=\underline{x}$	TR3
$(x+y)\between z=x\between y+y\between z$	CM4	$\underline{x\cdot y}=\underline{x}:\underline{y}$	TR4
$x\mid y=y\mid x$	CM5*	$\underline{x:y}=\underline{x}:\underline{y}$	TR5
$a\mid by=(a\mid b)y$	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	$\phi(x\cdot y)=\phi(x)\cdot\phi(y)$	F4
$\partial_H(a)=\delta$ if $a\in H$	D2	$\phi(x:y)=\phi(x)\cdot\phi(y)$	F5
$\partial_H(x+y)=\partial_H(x)+\partial_H(y)$	D3		
$\partial_H(x\cdot y)=\partial_H(x)\cdot\partial_H(y)$	D4		

Merging with Communication and Mutual Exclusion of Tight Regions: ACMP

The facilities of merge with communication (ACP) and merge with mutual exclusion of tight regions (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 2.5.14 . The left column contains ACP with a slight alteration for convenience: CM5* is added which saves us some axioms. The right column consists of the axioms in AMP(:) (see 2.5.13 .) for the operators :, _, and ϕ, where the axiom

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

is now "extended" to

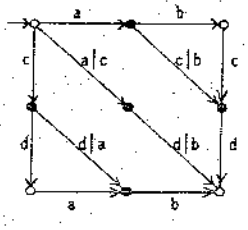
$$(a : x) \mathbin{\lfloor\!\lfloor} y = a : (x \mathbin{\lfloor\!\lfloor} y + x \mid y) \qquad \text{CTRM1.}$$

The axiom CTRM1 can be understood as follows: The process $(a : x) \mathbin{\lfloor\!\lfloor} y$ has a double commitment: $\mathbin{\lfloor\!\lfloor}$ insists that the first step in the cooperation 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 $a : (x \mathbin{\lfloor\!\lfloor} y + x \mid y)$. After a, the required step from x may be an "autonomous" step of x, as in $x \mathbin{\lfloor\!\lfloor} y$, or a simultaneous step in x and y, as in $x \mid y$. (Note that when communication is absent, i.e., $x \mid y = \delta$, CTRM1 specializes to TRM.) Moreover axiom AT5 is new and so are CTRM2–CTRM4 which specify : versus |.

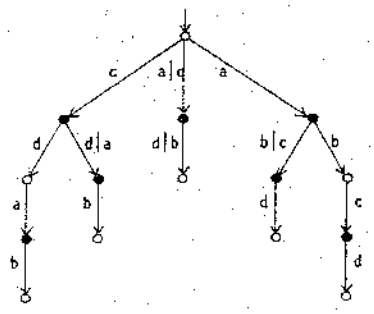
By means of a tedious prooftheoretic analysis analogous to the one for ACP one can prove consistency of ACMP and that ACMP is a conservative extension of both ACP and AMP(:). Also associativity of $\|$ holds for ACMP; intuitively this can be seen via a graph representation of closed ACMP-terms as in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

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

Example. $a : b \| c : d = a : b \mathbin{\lfloor\!\lfloor} c : d + c : d \mathbin{\lfloor\!\lfloor} a : b + a : b \mid c : d = a : (bc : d + b \mid c : d) + c : (da : b + d \mid a : b) + (a \mid c) : (b \mid d) = a : (bc : d + (b \mid c) : d) + c : (da : b + (d \mid a) : b) + (a \mid c) : (b \mid d)$. There is a simple graphical method for evaluating such expressions, as suggested by Fig. 1a. (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 Fig. 1b. (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 Bergstra and Klop, 1983a).

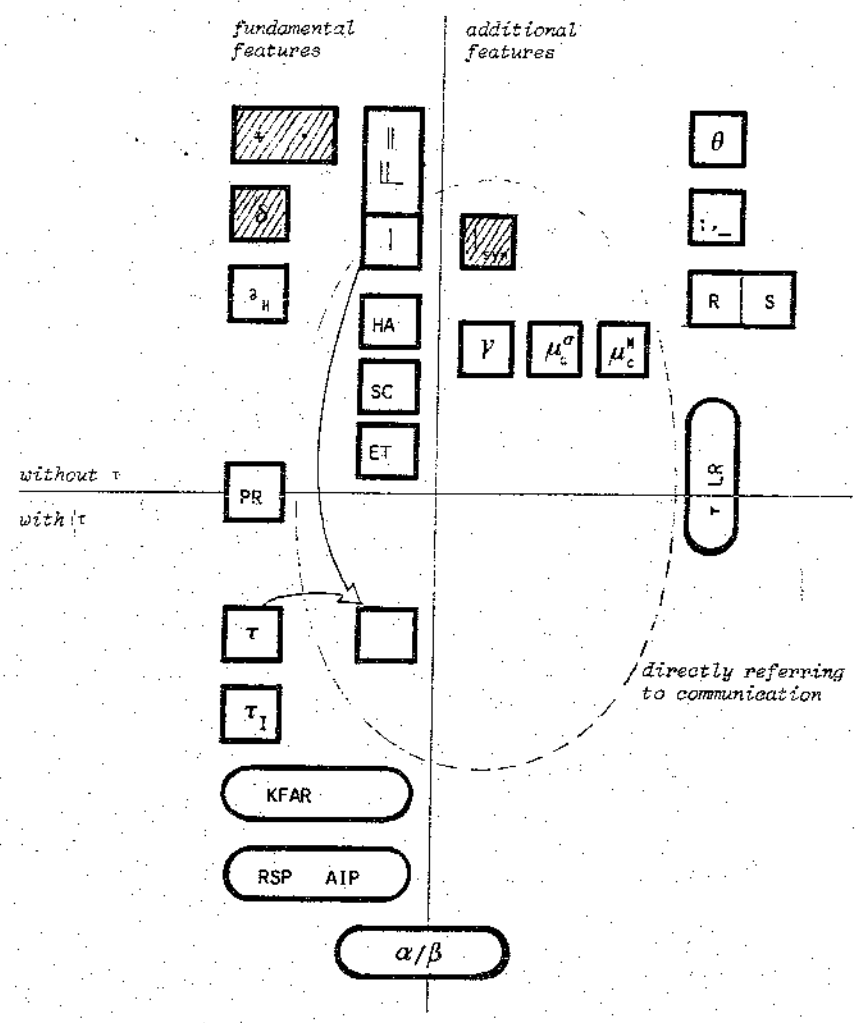


1 (a)



1 (b)

2.5.15.



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

algebra of synchronous processes

2.5.15, comments

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 [1983]). ACP, just as CCS, describes the asynchronous co-operation of processes. The axiom system ASP in 2.5.15 above 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

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

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

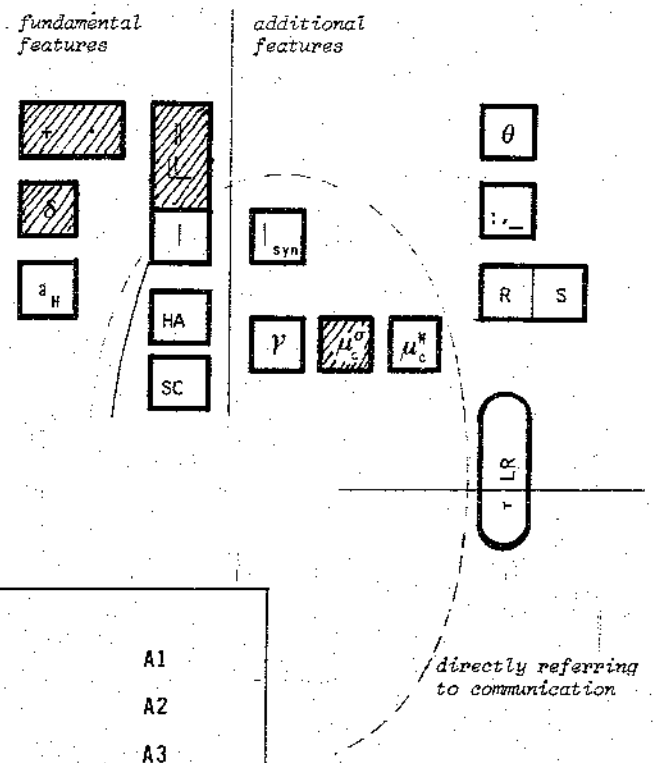
ASP bears a strong resemblance to Milner's SCCS [1983] (see also Hennessy [1981]); the most notable difference is δ which does part of the work done in SCCS by restriction operators. (In SCCS 'incompatibility' of atoms a,b cannot be expressed, so that certain superfluous subprocesses of a co-operation must be pruned away after the evaluation of the co-operation by a restriction operator. In ASP this incompatibility is stated as $a|b = \delta$.) Another notable difference is that SCCS admits also infinite sums.

Milner [1983] 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.

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

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.

2.5.16



$PA_\delta(\mu_c^\sigma, B, D)$

$x + y = y + x$	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
$x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$	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
$\mu_c^\sigma(e) = e$	M01
$\mu_c^\sigma(ex) = e \cdot \mu_c^\sigma(x)$	M02
$\mu_c^\sigma(c\uparrow d) = c \Uparrow d$	M03
$\mu_c^\sigma(c\uparrow d \cdot x) = c \Uparrow d \cdot \mu_c^{d*\sigma}(x)$	M04
$\mu_c^{\sigma*d}(c\uparrow d) = c \Downarrow d$	M05
$\mu_c^{\sigma*d}(c\uparrow d \cdot x) = c \Downarrow d \cdot \mu_c^\sigma(x)$	M06
$\mu_c^\sigma(c\uparrow d) = \delta$ if $d \neq last(\sigma)$ or $\sigma = \varepsilon$	M07
$\mu_c^\sigma(c\uparrow d \cdot x) = \delta$ if $d \neq last(\sigma)$ or $\sigma = \varepsilon$	M08
$\mu_c^\sigma(x + y) = \mu_c^\sigma(x) + \mu_c^\sigma(y)$	M09

$(a \in A,\ e \in E,\ \sigma \in D^*)$

process algebra for ordered mail along channel c

2.5.16, comments

μ_c^σ

The alphabet. Let B be a finite set of actions, D a finite set of data, c a special symbol (for 'channel'). For all $d \in D$ there are actions

$c \uparrow d$ *(send d via channel c: potential action)*

$c \Uparrow d$ *(send d via c: past action)*

$c \downarrow d$ *(receive d via c: potential action)*

$c \Downarrow d$ *(receive d via c: past action)*

The distinction between $c \uparrow d$ and $c \Uparrow d$ may be slightly unusual; $c \uparrow d$ indicates an intended (potential, future) action while $c \Uparrow d$ denotes a realised (actual, past) action. Likewise for $c \downarrow d$ and $c \Downarrow d$.

(Note that this distinction is implicitly also present in ACP: there a communication has the form $a|b = c$. Now the communication actions can be seen as potential actions, while the communication result c is an actual action.)

Notation: $c \uparrow D = \{c \uparrow d \mid d \in D\}$. Likewise for $c \Uparrow D$, etc.

Now we define the alphabet to be

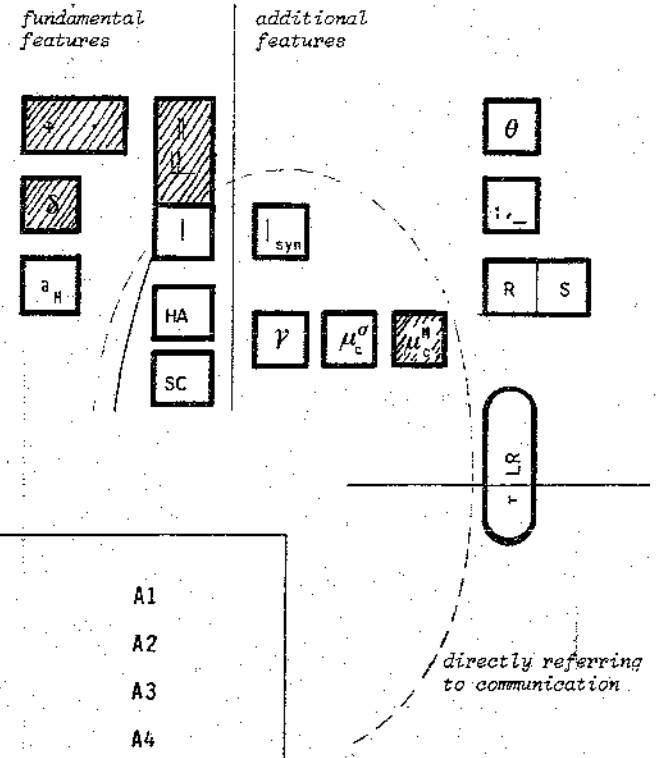
$$A = B \cup \{\delta\} \cup (c \uparrow D) \cup (c \Uparrow D) \cup (c \downarrow D) \cup (c \Downarrow D) .$$

Encapsulation.

Let D^* be the set of *sequences* σ of data $d \in D$. The empty sequence is denoted by ε. Concatenation of sequences σ, τ is denoted as $\sigma * \tau$; especially if $\sigma = \langle d_1, \ldots, d_n \rangle$ $(n \geq 0)$ then $d * \sigma = \langle d, d_1, \ldots, d_n \rangle$, and $\sigma * d = \langle d_1, \ldots, d_n, d \rangle$. Further, if $n \geq 1$, $last(\sigma) = d_n$.

Now for each $\sigma \in D^*$ there is an *encapsulation operator* $\mu_c^\sigma : \mathcal{P} \to \mathcal{P}$ where $\mathcal{P}$ is the domain of processes (i.e. the elements of a process algebra satisfying the axioms above). If x is a process, then $\mu_c^\sigma(x)$ denotes the process obtained by requiring that the channel c is initially containing a data sequence σ and that no communications with c are performed outside x.
Phrased otherwise: $\mu_c^\sigma(x)$ is the result of *partial execution* of x w.r.t. c with initial contents σ. (Here 'execution' refers to the transformation of a potential action like $c \uparrow d$ into a past action $c \Uparrow d$; moreover such a transformation has a side-effect on the contents of c as specified by the axioms.)

2.5.17



$PA_\delta(\mu_c^M, B, D)$

$x + y = y + x$	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
$x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$	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
$\mu_c^M(e) = e$	MN01
$\mu_c^M(ex) = e.\mu_c^M(x)$	MN02
$\mu_c^M(c\uparrow d) = c \Uparrow d$	MN03
$\mu_c^M(c\uparrow d \,.\, x) = c \Uparrow d \,.\, \mu_c^{M \cup \{d\}}(x)$	MN04
$\mu_c^{M \cup \{d\}}(c\uparrow d) = c \Downarrow d$	MN05
$\mu_c^{M \cup \{d\}}(c\uparrow d \,.\, x) = c \Downarrow d \,.\, \mu_c^M(x)$	MN06
$\mu_c^M(c\uparrow d) = \delta$ if $d \notin M$	MN07
$\mu_c^M(c\uparrow d \,.\, x) = \delta$ if $d \notin M$	MN08
$\mu_c^M(x + y) = \mu_c^M(x) + \mu_c^M(y)$	MN09

($a \in A$, $e \in E$, M a multiset over D)

process algebra for unordered mail along channel c

2.5.17, comments

μ_c^M

For the bag-like channel the situation is very much the same except that a data sequence σ is now a *multiset* of data. We denote a finite multiset of $d \in D$ by 'M'. Now for all finite multisets M over D we introduce again an encapsulation or partial execution operator

$$\mu_c^M : \mathcal{P} \to \mathcal{P}$$

Example

(i) $\mu_c^{\emptyset}(c{\uparrow}d \cdot c{\uparrow}d) = c{\Uparrow}d \cdot c{\Downarrow}d$

(ii) $\mu_c^{\emptyset}(c{\uparrow}d \cdot \sum_{u \in D} c{\uparrow}u) = c{\Uparrow}d \cdot c{\Downarrow}d$

(iii) $\mu_c^{\emptyset}(c{\uparrow}d \parallel c{\uparrow}d) = c{\Uparrow}d \cdot c{\Downarrow}d$

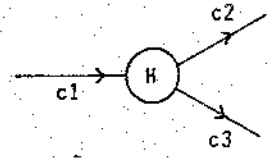
(iv) $\mu_c^{\emptyset}(c{\uparrow}d1 \cdot c{\uparrow}d2 \cdot \sum_{u \in D} c{\uparrow}u \cdot \sum_{u \in D} c{\uparrow}u) =$

$c{\Uparrow}d1 \cdot c{\Uparrow}d2 \cdot (c{\Downarrow}d1 \cdot c{\Downarrow}d2 + c{\Downarrow}d2 \cdot c{\Downarrow}d1)$

(v) Let $D = D1 \cup D2$, $D1 \cap D2 = \emptyset$,

and $H = \left(\sum_{d \in D1} c1{\uparrow}d \cdot c2{\uparrow}d + \sum_{d \in D2} c1{\downarrow}d \cdot c3{\uparrow}d \right) \cdot H$

Then H separates the D1 messages from the D2 messages.



(vi) Let $d1 \neq d2$. Then:

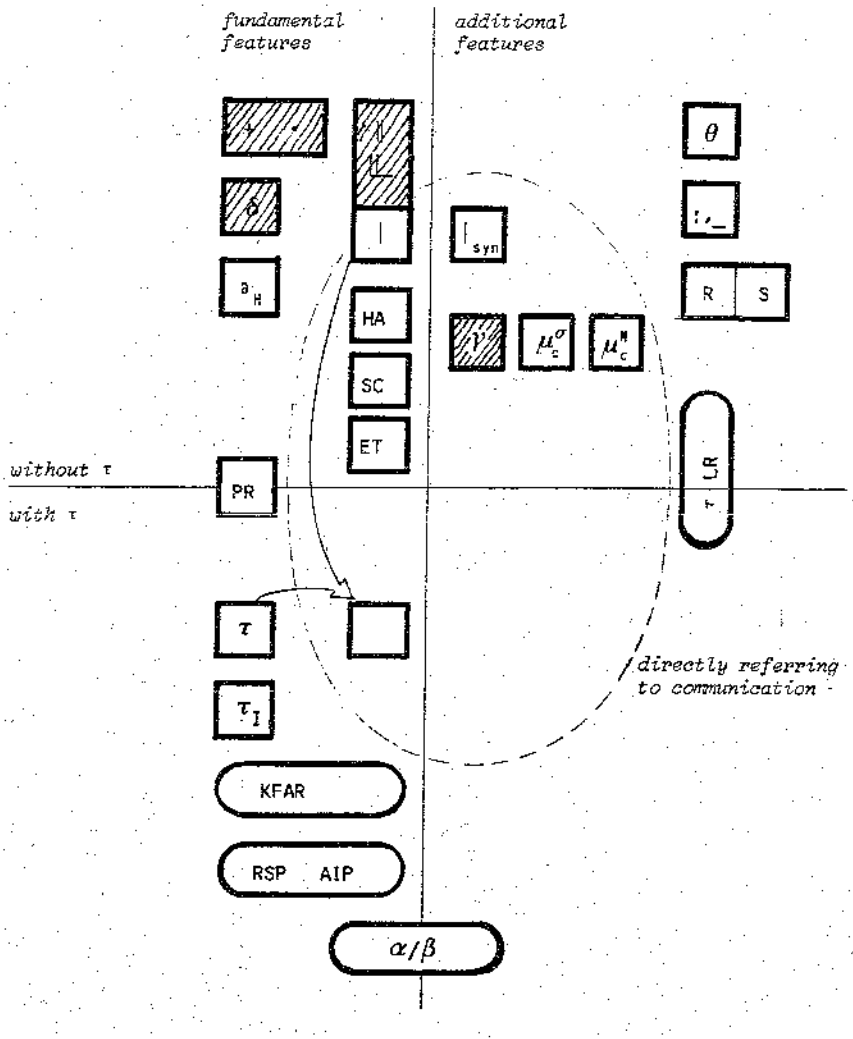
$\mu_c^{\emptyset}(c{\uparrow}d1 \cdot c{\uparrow}d2) = c{\Uparrow}d1 \cdot \delta$

$\mu_c^{\emptyset}(c{\uparrow}d1 \parallel c{\uparrow}d2) = c{\Uparrow}d1 \cdot \delta$

$\mu_c^{\emptyset}(c{\uparrow}d2 \between c{\uparrow}d1) = \delta$.

Remark. Notice that there is no guarantee that after a send action $c{\uparrow}d$ the corresponding receive action $c{\downarrow}d$ will ever be performed. Thus the send action *enables* the receive action but does not *force* its execution. This holds for both mechanisms: queue-like channel and bag-like channel.

2.5.18



$PA_\delta(\gamma,\hat{})$ over atoms A with causality relation R

$x + y = y + x$	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
$x \parallel y = x \between y + y \between x$	M1
$a \between y = ay$	M2
$ax \between y = a(x \parallel y)$	M3
$(x + y) \between z = x \between z + y \between z$	M4
$\hat{\delta} = \delta$	G1
$\hat{a} = \hat{a}$	G2
$\gamma^E(a) = \hat{a}$ if $a \in E$ or $a \notin \mathrm{Ran}(R)$	G3
$\gamma^E(a) = \delta$ if $a \notin E$ and $a \in \mathrm{Ran}(R)$	G4
$\gamma^E(ax) = \gamma^E(a) \cdot \gamma^{(E - \{a\}) \cup R(a)}(x)$	G5
$\gamma^E(x + y) = \gamma^E(x) + \gamma^E(y)$	G6

2.5.18, comments

γ CAUSALITY

In the previous section, the action c⇑d is the 'actualised form' of c↛d and likewise c⇓d is c↛d after execution or actualisation. Moreover, a causal effect is involved: c↛d causes c↛d. These concepts will be made explicit in the present section.

4.1. Actualisation. On the alphabet A we postulate an operator $\hat{}: A \to A$, such that $\hat{\delta} = \delta$ and $\hat{\hat{a}} = \hat{a}$ for all $a \in A$. The action $\hat{a}$ is called the *actualisation* of a. Writing $B = A - \hat{A}$, where $\hat{A} = \{\hat{a} \mid a \in A\}$, A is partitioned as follows:

$$A = B \cup \hat{A}.$$

4.2. Causal relations. On the set B of not yet completed actions we have a binary relation R encoding the causal relations between such actions. Instead of $(a,b) \in R$ we write:

$$a \Vdash b,$$

in words: "a causes b". Further notations are:
Dom(R) for the *domain* of R, i.e. $Dom(R) = \{b \mid \exists b'\ b' \Vdash b\}$, and
Ran(R) for the *range* of R, i.e. $Ran(R) = \{b \mid \exists b'\ b' \Vdash b\}$.
So Dom(R) contains the 'causes' or 'stimuli' and Ran(R) the 'effects' or 'responses'. Note that an action can be both a cause and an effect. Finally, $R(b) = \{b' \mid b \Vdash b'\}$, the set of effects of b.

4.3. Encapsulation. Let $b \in B$. Performing b has two consequences: b is changed into $\hat{b}$, and all $b' \in R(b)$ (i.e. caused by b) are now *enabled*. The operator which takes care of the execution of b (or in another phrasing, which changes the view from 'internal' to 'external') and which takes into account which actions are enabled, is the *encapsulation operator* γ^E. Here $E \subseteq B$. The intuitive meaning of γ^E is: $\gamma^E(x)$ is the process where all causal effects take place within x, i.e. actions within x are neither enabled nor disabled by actions outside x and conversely. Moreover, initially the actions $\in E$ are enabled.

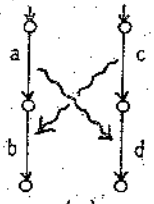
Example. (i) Suppose $a\|\!\vdash d$, $c\|\!\vdash b$ (see Figure 6(a)). Then

$\gamma^{\emptyset}(ab\|cd) = \gamma^{\emptyset}(a(b\|cd)) + \gamma^{\emptyset}(c(d\|ab)) = \hat{a}\gamma^{\{d\}}(b\|cd) + \ldots =$

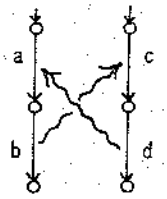
$\hat{a}\gamma^{\{d\}}(bcd + c(d\|b)) + \ldots = \hat{a}(\delta + \hat{c}\gamma^{\{d,b\}}(db + bd)) + \ldots =$

$\hat{a}\hat{c}(\hat{d}\hat{b} + \hat{b}\hat{d}) + \hat{c}\hat{a}(\hat{b}\hat{d} + \hat{d}\hat{b}) = (\hat{a}\|\hat{c})(\hat{b}\|\hat{d})$.

(ii) Suppose $d\|\!\vdash a$, $b\|\!\vdash c$ (see Figure). Then $\gamma^{\emptyset}(ab\|cd) = \delta$.



(a)



(b)

Note that circular causal relations (such as in this example (ii)) yield deadlock. Here an action a must be considered to cause also the actions accessible from a ('later' than a). (Indeed, we have $a \cdot b = \gamma^{\emptyset}(a\|b)$ for $a\|\!\vdash b$.)

(iii) Let X and Y be the two infinite processes recursively defined by $X = abX$ and $Y = cdY$; so $X = (ab)^{\omega}$ and $Y = (cd)^{\omega}$. Suppose $a\|\!\vdash c$ and $d\|\!\vdash b$. Then

$$\gamma^{\emptyset}(X\|Y) = \gamma^{\emptyset}(a(bX\|Y) + c(dY\|X)) = \hat{a}\gamma^{\{c\}}(bX\|Y) + \delta =$$

$$\hat{a}\gamma^{\{c\}}(b(X\|Y) + c(dY\|bX)) = \hat{a}(\delta + \hat{c}\gamma^{\emptyset}(dY\|bX)) =$$

$$\hat{a}\hat{c}(\hat{d}\gamma^{\{b\}}(Y\|bX) + \delta) = \hat{a}\hat{c}\hat{d}\gamma^{\{b\}}(b(X\|Y) + Y\mathbin{\lfloor\!\lfloor} bX) =$$

$$\hat{a}\hat{c}\hat{d}\hat{b}\gamma^{\emptyset}(X\|Y).$$

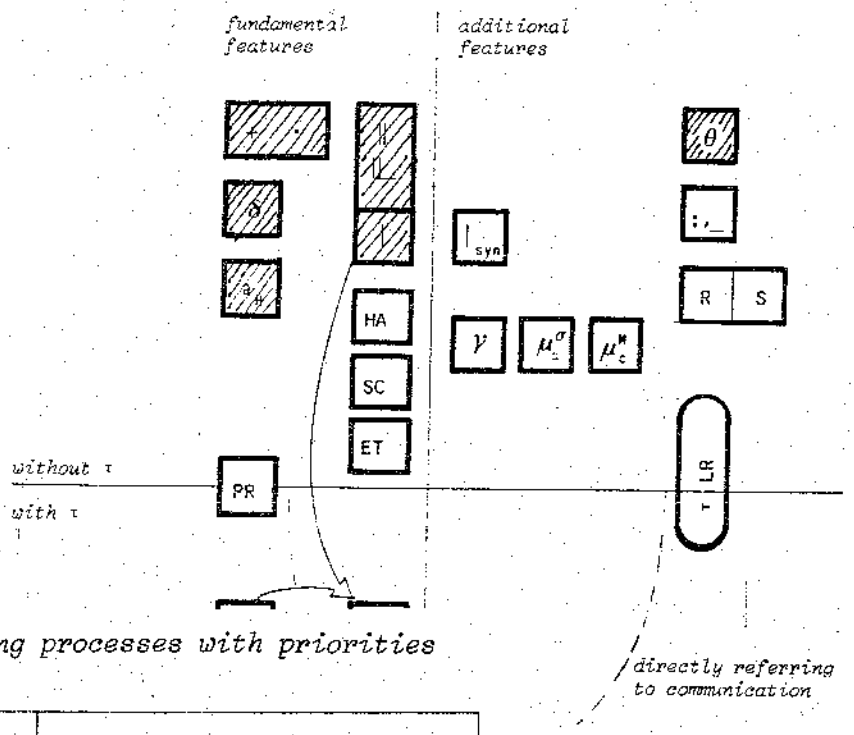
Hence $\gamma^{\emptyset}(X\|Y) = (\hat{a}\hat{c}\hat{d}\hat{b})^{\omega}$.

Remark. There is an interesting connection between the 'spatial' notion of encapsulation (as effectuated by the operators ∂_H in ACP, μ_c^{σ}, μ_c^{M} in the mail mechanisms of 2.5.16, 17 and the present γ^{E} for causality) and the 'temporal' notion of execution. In some sense, one could say:

encapsulation = execution.

Indeed, an encapsulated process can be thought to be already executed since no further interactions with an environment are possible.

2.5.19.



ACP_θ *algebra of communicating processes with priorities*

$x + y = y + x$	A1	$a \lhd b = a$	if not $(a<b)$	P1
$x + (y + z) = (x + y) + z$	A2	$a \lhd b = \delta$	if $a<b$	P2
$x + x = x$	A3	$x \lhd yz = x \lhd y$		P3
$(x + y)z = xz + yz$	A4	$x \lhd (y + z) = (x \lhd y) \lhd z$		P4
$(xy)z = x(yz)$	A5	$xy \lhd z = (x \lhd z)y$		P5
$x + \delta = x$	A6	$(x + y) \lhd z = x \lhd z + y \lhd z$		P6
$\delta x = \delta$	A7			
$a \mid b = b \mid a$	C1			
$(a \mid b) \mid c = a \mid (b \mid c)$	C2			
$\delta \mid a = \delta$	C3			
$x \mid y = x \between y + y \between x + x \mid y$	CM1	$\theta(a) = a$		TH1
$a \between x = ax$	CM2	$\theta(xy) = \theta(x) \cdot \theta(y)$		TH2
$ax \between y = a(x \mid y)$	CM3	$\theta(x+y) = \theta(x) \lhd y + \theta(y) \lhd x$		TH3
$(x + y) \between z = x \between z + y \between z$	CM4			
$(ax) \mid b = (a \mid b)x$	CM5			
$a \mid (bx) = (a \mid b)x$	CM6			
$(ax) \mid (by) = (a \mid b)(x \mid y)$	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			
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$	D4			

2.5.19, comments

This axiom system presents a new piece of syntax:

together with defining equations P1-6, TH1-3. Based on a partial order $>$ on atoms (in which δ is always the least element), an operator θ is defined: $\theta(x)$ is a context of x inside which, in a choice, action a has priority over b whenever $a > b$. In order to give a finite axiomatisation of θ, an auxiliary operator $\triangleleft$ is used. In words: $x \triangleleft y$ is "x unless y". The process y acts as a filter on the initial steps of x: every initial step of x dominated by an initial step of y, is 'killed' (replaced by δ). E.g. $a \triangleleft b = a$ unless b has priority over a, in which case $a \triangleleft b = \delta$.

Examples: assume $b < a$ and $c < a$. Then:

(i) $\theta(a+b) = \theta(a) \triangleleft b + \theta(b) \triangleleft a = a \triangleleft b + b \triangleleft a = a + \delta = a$.

(ii) $\theta(b+c) = \theta(b) \triangleleft c + \theta(c) \triangleleft b = b \triangleleft c + c \triangleleft b = b + c$.

(iii) $\theta(b(a+c)) = \theta(b).\theta(a+c) = b(\theta(a) \triangleleft c + \theta(c) \triangleleft a) =$

$b(a \triangleleft c + c \triangleleft a) = b(a + \delta) = ba$.

(iv) $\theta(a+b+c) = \theta(a) \triangleleft (b+c) + \theta(b+c) \triangleleft a =$

$(\theta(a) \triangleleft b) \triangleleft c + (\theta(b) \triangleleft c + \theta(c) \triangleleft b) \triangleleft a = (a \triangleleft b) \triangleleft c + (b \triangleleft c + c \triangleleft b) \triangleleft a =$

$a \triangleleft c + (b+c) \triangleleft a = a + \delta = a$.

(v) $\theta(a+b+c) = \theta(a+b) \triangleleft c + \theta(c) \triangleleft (a+b) = (\theta(a) \triangleleft b + \theta(b) \triangleleft a) \triangleleft c$

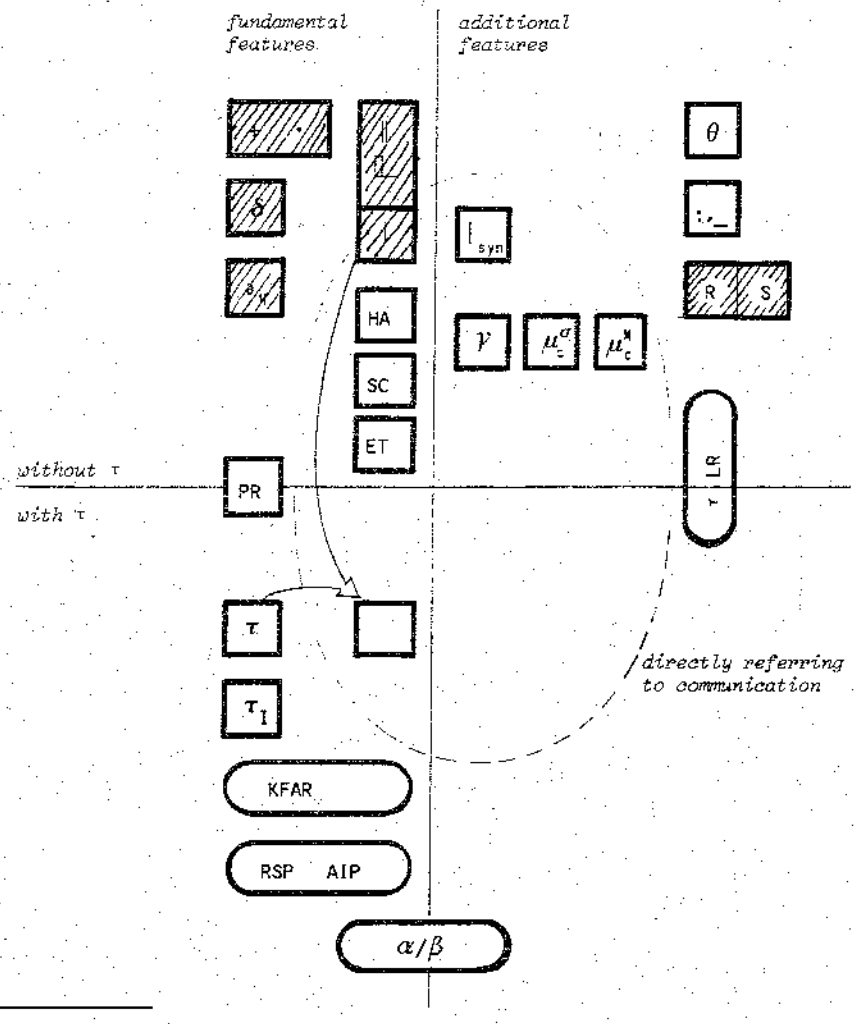
$+ \theta(c) \triangleleft a) \triangleleft b = (a \triangleleft b + b \triangleleft a) \triangleleft c + (c \triangleleft a) \triangleleft b = (a + \delta) \triangleleft c + \delta \triangleleft b =$

$a \triangleleft c + \delta \triangleleft c + \delta = a + \delta + \delta = a$.

((iv) and (v) show that an expression can be broken down in several ways. This is why a term rewriting analysis yielding confluency is required.) ACP_θ is a conservative extension of ACP. An elimination theorem for the newly defined operators $\theta, \triangleleft$ is proved with the help of the method of recursive path orderings, a termination proof technique.

ACP_θ has been used to model several simple interrupt mechanisms, involving recursively defined processes.

2.5.20.



ACP_{RS}

$x + y = y + x$	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
$\delta \mid a = \delta$	C3
$x \parallel y = x \between y + y \between x + x \mid y$	CM1
$a \between x = ax$	CM2
$(ax) \between y = a(x \parallel y)$	CM3
$(x + y) \between z = x \between z + y \between z$	CM4
$(ax) \mid b = (a \mid b)x$	CM5
$a \mid (bx) = (a \mid b)x$	CM6
$(ax) \mid (by) = (a \mid b)(x \parallel y)$	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
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$	D4

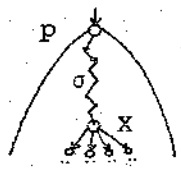
algebra of communicating processes

with readies and failures

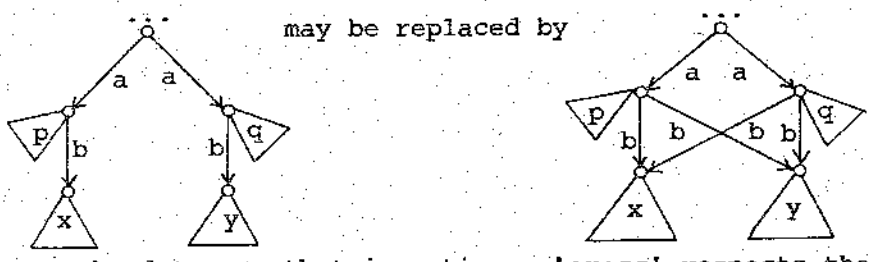
a(bx + p) + a(by + q) = a(bx + by + p) + a(bx + by + q)	R1
a(b + p) + a(by + q) = a(b + by + p) + a(b + by + q)	R2
ax + a(y + z) = ax + a(y + z) + a(x + y)	S

2.5.20, comments

R,S Readiness and failure semantics can be seen as a kind of 'enriched trace' semantics. Two processes p,q are ready equivalent if they have the same ready pairs; here (σ,X) is a ready pair of p if there is a path in p (starting from the root) yielding the finite trace σ and such that $X \subseteq A$ is the set of immediate possible further steps.



The readiness axioms R1,2 give (in the simple setting of ACP, without abstraction) a complete description of the readiness semantics of finite processes. The axiom R1 amounts, pictorially, to placing a 'cross' in the graph of the process:



(It is not hard to see that inserting a 'cross' respects the readiness semantics.)

Failure equivalence equates more processes than readiness equivalence does. (I.e. readiness equivalence $\Rightarrow$ failure equivalence.) The concept refers to 'negative' information about possible further actions: if (σ,X) is a ready pair, any (σ,Y) with $Y \subseteq A-X$ is a failure pair. Processes p,q are failure equivalent if they have the same failure pairs.

It turns out that failure equivalence for finite processes (without τ) is completely described by ACP_{RS}. Axiom S has as consequences the 'saturation' or 'convexity' axioms

$$ax + ay = ax + a(x + y) + ay$$
$$ax + a(x + y + z) = ax + a(x + y) + a(x + y + z).$$

(Vice versa S can be derived from these two.)
For S also a simple graphical representation exists, which is useful in the completeness proof.

$x+y = y+x$	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		
$a\|b = b\|a$	C1		
$(a\|b)\|c = a\|(b\|c)$	C2		
$\delta\|a = \delta$	C3		
$x\|\|y = x\llcorner y+y\llcorner x+x\|y$	CM1		
$a\llcorner x = ax$	CM2	$\tau\llcorner x = \tau x$	TM1
$(ax)\llcorner y = a(x\|\|y)$	CM3	$(\tau x)\llcorner y = \tau(x\|\|y)$	TM2
$(x+y)\llcorner z = x\llcorner z+y\llcorner z$	CM4	$\tau\|x = \delta$	TC1
$(ax)\|b = (a\|b)x$	CM5	$x\|\tau = \delta$	TC2
$a\|(bx) = (a\|b)x$	CM6	$(\tau x)\|y = x\|y$	TC3
$(ax)\|(by) = (a\|b)(x\|\|y)$	CM7	$x\|(\tau y) = x\|y$	TC4
$(x+y)\|z = x\|z+y\|z$	CM8		
$x\|(y+z) = x\|y+x\|z$	CM9		
		$\partial_H(\tau)=\tau$	DT
		$\tau_I(\tau) = \tau$	TI1
$\partial_H(a) = a$ if $a\notin H\subseteq A$	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

Handshaking axiom:

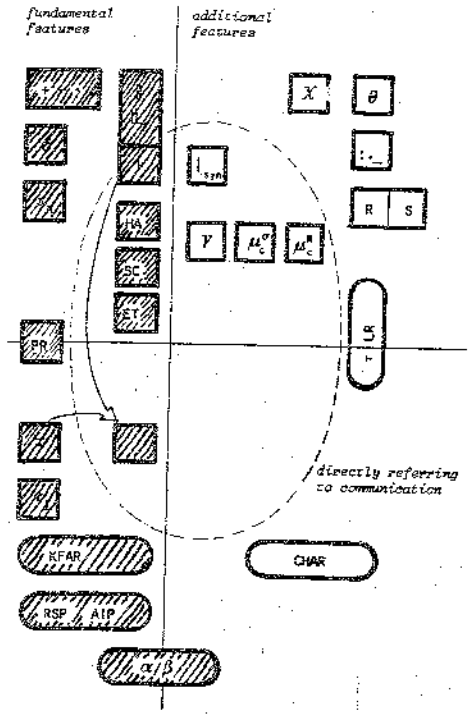
$$x|y|z = \delta$$

Expansion theorem

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

Axioms of standard concurrency:

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



ACPK

algebra of communicating processes with Koomen's fair abstraction rule

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

$\pi_m(a)=a$ $\qquad$ $\pi_m(\tau)=\tau$

$\pi_1(ax)=a$ $\qquad$ $\pi_m(\tau x)=\tau\cdot\pi_m(x)$

$\pi_{m+1}(ax)=a\pi_m(x)$

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

RSP $\quad \dfrac{E(x,-)\qquad E(y,-)}{x = y}$ E guarded

AIP $\quad \dfrac{\forall n\ \pi_n(x)=\pi_n(y)\qquad E(x,-)}{x = y}$ E guarded, no abstraction

$\dfrac{\alpha(x)\|(\alpha(y)\cap H)\subseteq H}{\partial_H(x\|y)=\partial_H(x\|\partial_H(y))}$	CA1	$\dfrac{\alpha(x)\|(\alpha(y)\cap I)=\emptyset}{\tau_I(x\|y)=\tau_I(x\|\tau_I(y))}$	CA2
$\dfrac{\alpha(x)\cap H=\emptyset}{\partial_H(x)=x}$	CA3	$\dfrac{\alpha(x)\cap I=\emptyset}{\tau_I(x)=x}$	CA4
$\dfrac{H=H_1\cup H_2}{\partial_H(x)=\partial_{H_1}\circ\partial_{H_2}(x)}$	CA5	$\dfrac{I=I_1\cup I_2}{\tau_I(x)=\tau_{I_1}\circ\tau_{I_2}(x)}$	CA6
$\dfrac{H\cap I=\emptyset}{\tau_I\circ\partial_H(x)=\partial_H\circ\tau_I(x)}$	CA7		

2.5.21, comments

KFAR A proof rule which is vital in algebraic computations for system verification, is Koomen's fair abstraction rule. It was used by C.J. Koomen of Philips Research in a formula-manipulation system based on CCS and defined explicitly in Bergstra - Klop (1984c) where it served to give an algebraical verification of a simple version of the Alternating Bit Protocol. In the name KFAR, the adjective 'fair' refers to the bias that bisimulation has towards a fair execution of τ-paths in the sense that, after finitely many executions of a τ-cycle, an alternative not on the τ-cycle must be chosen (if there is no alternative the result is δ).

The formal version of KFAR (which is in fact parametrised by $k \geq 1$) is:

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

where the subscripts n, n+1 are elements of $\mathbb{Z}_k = \{0,1,\ldots,k-1\}$ in which addition is modulo k. The x_n, x_{n+1}, y_n, y_m are metavariables ranging over the process algebra under consideration. The i_n are elements of A; we always require $I \subseteq A$, so i_n cannot be τ. This is essential, as we shall show. We conceive the i_n (or more general, the elements of I) as <u>internal</u> steps (but <u>not</u>, as τ is, invisible or silent) which can be abstracted to yield τ-steps.

We will explain the rule by some examples.

(i)
$$KFAR_1 \quad \frac{x = ix}{\tau_{\{i\}}(x) = \tau\delta}$$

(Note that this case is covered by $KFAR_i$ since $x = ix = ix + \delta$.)

(ii)
$$KFAR_1 \quad \frac{x = a + ix}{\tau_{\{i\}}(x) = \tau a}$$

(iii)
$$KFAR_3 \quad \frac{x = iy + p, \quad y = jz + q, \quad z = kx + r}{\tau_{\{i,j,k\}}(x) = \tau(\tau_{\{i,j,k\}}(p + q + r))}$$

We remark that it is essential that the $i_0, i_1, \ldots, i_{k-1}$-"cycle" appearing in the hypothesis of KFAR is <u>not</u> a cycle of τ-steps. Indeed, the version of KFAR:

"If $\forall n \in \mathbb{Z}_k \quad x_n = \tau x_{n+1} + y_n$, then $x_n = \tau(\Sigma y_m)$"

would simply be false: from $x = a + \tau x$ it does not follow that $x = \tau a$, as already remarked above in .

(iv) The following example of an application of KFAR is also used in 2.6.3.

Consider processes x_0, x_1, x_2 such that

$$x_0 = ax_1 + ax_2$$

$$x_1 = c + bx_2$$

$$x_2 = d + bx_1$$

Then $\tau_{\{b\}}(x_0) = a.\tau_{\{b\}}(x_1) + a.\tau_{\{b\}}(x_2)$. Now $x_1 = bx_2 + c$, $x_2 = bx_1 + d$ can be used as the hypothesis of $KFAR_2$, yielding:

$$\tau_{\{b\}}(x_1) = \tau^{\bullet}\tau_{\{b\}}(c + d) = \tau(c + d)$$

$$\tau_{\{b\}}(x_2) = \tau^{\bullet}\tau_{\{b\}}(c + d) = \tau(c + d)$$

Hence $\tau_{\{b\}}(x_0) = a\tau(c+d) + a\tau(c+d) = a\tau(c+d) = a(c+d)$.

Milner remarks in his book (Milner (1980)) that it is a bit curious that a τ-loop  τ is set equal to τ by the notion of bisimulation (or rather observational congruence) used there.

In this respect it is interesting to see that the sole adoption of τ_I and its axiom TI5: $\tau_I(xy) = \tau_I(x).\tau_I(y)$, forces us to refrain from this choice to treat infinite terminal τ-loops in this 'benevolent' way. Namely, consider

$$x = ix$$
$$y = xa.$$

Then clearly $x = y$ (in fact this can be proved in ACPK from the approximation induction principle AIP). However:

$$\tau_{\{i\}}(x) = \tau_{\{i\}}(y) = \tau_{\{i\}}(xa) = \tau_{\{i\}}(x).\tau_{\{i\}}(a) = \tau_{\{i\}}(x).a$$

The equality $\tau_{\{i\}}(x) = \tau_{\{i\}}(x).a$ shows that the 'τ-loop' $\tau_{\{i\}}(x)$ certainly is not equal to τ, since $\tau \neq \tau a$.

So what is $\tau_{\{i\}}(x)$? The equation $\tau_{\{i\}}(x).a = \tau_{\{i\}}(x)$ suggests that $\tau_{\{i\}}(x)$ is something 'like δ'. Indeed, as example (i) above shows, an application of KFAR tells us that $\tau_{\{i\}}(x) = \tau\delta$; and $\tau\delta = \tau\delta.a$.

(This state of affairs is the reason that in $BPA_{\tau LR}$', 2.5.10, we had to replace axiom TI5 by its prefix multiplication form: axioms TI5' and TI5". In $BPA_{\tau LR}$ the 'Milner option' prevails.)

There exists a model of ACPK ($\mathcal{G}_K/\leftrightarrows_{r\tau\delta}$, see chapter 3) with the following properties:

(1) every infinite system of guarded recursion equations has a unique solution in the model;

(2) the solution of an infinite, effectively representable system of guarded recursion equations can already be obtained as the solution of a finite system of guarded recursion equations.

A remark as to the consistency of ACPK: in contrast to the cases of the axiom systems PA, ACP, ACP_τ and some others, establishing the consistency of ACPK via a rewrite rule analysis seems totally unfeasible. Instead, consistency was proved by providing the graph model mentioned above.

For comments on HA (handshaking axiom), ET (expansion theorem) and SC (axioms of standard concurrency), see survey part I (report CS-R8421), p.19.

AIP The approximation induction principle AIP does not hold in the unrestricted form $\forall n\ \pi_n(x) = \pi_n(y) \Rightarrow x = y$ (in the above mentioned graph model).

However, AIP does hold if one of the processes x,y is restricted to be bounded and finitely branching. Here bounded means: having no infinite τ-paths. Now this condition of being bounded and finitely branching can be expressed in an algebraical way:

if x is the solution of a guarded system of recursion equations, it is bounded and finitely branching. Hence the extra assumption E(x,-) in the premiss of AIP.

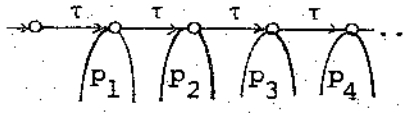
(Explanation of the notation E(x,-): $E(x,\vec{y})$ means that $x,\vec{y}$ solve the system E of equations. The $\vec{y}$ are the auxiliary 'variables'. Now E(x,-) abbreviates $\exists y\ E(x,\vec{y})$.)

$$\text{RSP}\quad \frac{E(x,-) \qquad E(y,-)}{x = y}\ E \text{ guarded}$$

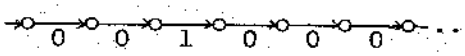
$$\text{AIP}\quad \frac{\forall n\ \pi_n(x) = \pi_n(y) \qquad E(x,-)}{x = y}\ E \text{ guarded, no abstraction}$$

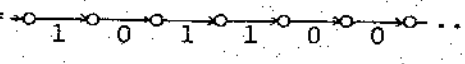
$\pi_m(a)=a$	$\pi_m(\tau)=\tau$
$\pi_1(ax)=a$	$\pi_m(\tau x)=\tau\cdot\pi_m(x)$
$\pi_{m+1}(ax)=a\pi_m(x)$	
$\pi_m(x+y)=\pi_m(x)+\pi_m(y)$	

PR The projection axioms say that the n-th projection operator π_n cuts off process x to depth n - with the understanding that τ-steps are 'transparent'. I.e. a τ-step does not raise the depth.

Example: let x be 

where p_n = reversed binary representation of n followed by 0^ω.
So e.g.

p_9 = 

p_{13} = 

Then $\pi_1(x) = \tau(0 + 1)$, $\pi_2(x) = \tau(00 + 01 + 10 + 11)$, etc.

Let y be x with 0,1 interchanged. Now note the curious fact that for all n, $\pi_n(x) = \pi_n(y)$ while clearly $x \neq y$ (since every branch in x consists eventually of 0's, and of y of 1's). This shows that the unrestricted form of AIP (see below) does not hold.

That τ-steps do not raise the depth is a consequence of the τ-laws T1-3: e.g. with projection operators for which τ is not transparent, we would have an immediate contradiction: τ = "π_1"(τa) = "π_1"($\tau a + a$) = $\tau + a$.

α/β

$\frac{\alpha(x)\mid(\alpha(y)\cap H)\subseteq H}{\partial_H(x\|y)=\partial_H(x\|\partial_H(y))}$	CA1	$\frac{\alpha(x)\mid(\alpha(y)\cap I)=\emptyset}{\tau_I(x\|y)=\tau_I(x\|\tau_I(y))}$	CA2
$\frac{\alpha(x)\cap H=\emptyset}{\partial_H(x)=x}$	CA3	$\frac{\alpha(x)\cap I=\emptyset}{\tau_I(x)=x}$	CA4
$\frac{H=H_1\cup H_2}{\partial_H(x)=\partial_{H_1}\circ\partial_{H_2}(x)}$	CA5	$\frac{I=I_1\cup I_2}{\tau_I(x)=\tau_{I_1}\circ\tau_{I_2}(x)}$	CA6
	$\frac{H\cap I=\emptyset}{\tau_I\circ\partial_H(x)=\partial_H\circ\tau_I(x)}$	CA7	

$\alpha(x)$ is the alphabet of process x. Although this notion may seem trivial, it is in fact possible to give recursive definitions of processes for which it is undecidable to determine what the alphabet is.

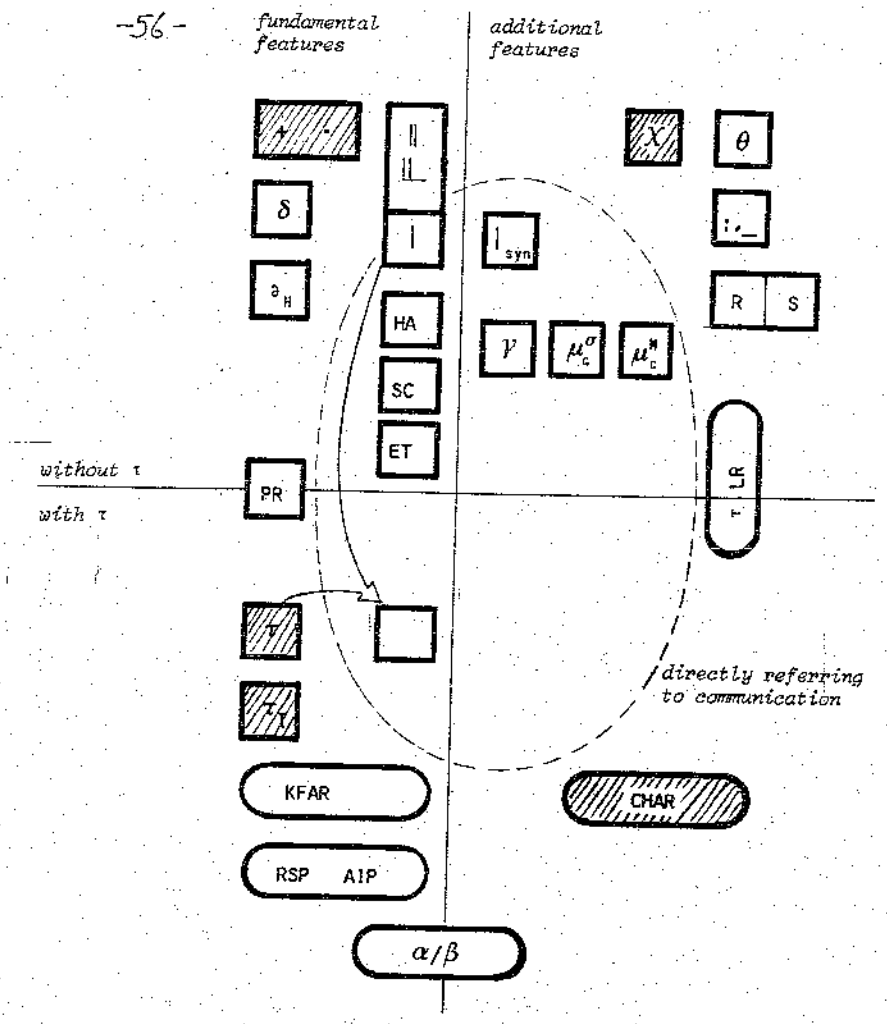
The conditional axioms CA1-7 hold for all finite ACP_τ-terms. In the present setting of ACPK, they are postulated for all processes including infinite ones.

E.g. CA4 says: if no action from I occurs in x, then abstracting the actions in I has no effect on x.

CA2: this axiom allows one to commute abstraction and parallel composition (in appropriate circumstances). CA2 is indispensable for the algebraic verification of systems with three or more components in parallel.

The conditional axioms CA1-7 were used in verifications of equations like "BAG ‖ BAG = BAG", and more general "BAG^{Π} = BAG".

2.5.22.



$x+y = y+x$	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		
		$\tau_I(\tau) = \tau$, $\tau_I(\chi) = \chi$	TI1
		$\tau_I(a) = a$ if $a \notin I$	TI2
		$\tau_I(a) = \tau$ if $a \in I$	TI3
		$\tau_I(x+y) = \tau_I(x)+\tau_I(y)$	TI4
		$\tau_I(xy) = \tau_I(x).\tau_I(y)$	TI5
$\chi + x = \chi$	CH1	$\tau * \chi = \chi$	CHT
$\chi * x = \chi$	CH2		

CHAR $\dfrac{x_n = i_n * x_{n+1} + y_n \quad (n \in \mathbb{N},\ i_n \in I)}{\tau_I(x_0) = \chi}$

$BPA_{\tau\chi}$ *basic process algebra with abstraction and chaos*

2.5.22, comments.

$BPA_{\tau\chi}$ can easily be extended to $PA_{\tau\chi}$ and probably to $PA_{\tau\chi}$ + R,S; but an extension to '$ACP_{\tau\chi}$' would be highly problematic since χ and the communication merge seem to be incompatible. (Is $\chi|x = \chi$ or $\chi|x = \delta$? The axiom $\tau|x = \delta$ in ACP_τ prescribes the second possibility. But then

$$(\chi + x)|y = \chi|y + x|y = \delta + x|y = x|y$$

while also

$$(\chi + x)|y = \chi|y = \delta$$

yielding the contradiction $x|y = \delta$.)

CHAR declares a process having an infinite τ-trace starting from the root, equal to χ (chaos). The rule is (in a different formalism) from Brookes, Hoare, Roscoe (1984).

Note the difference with KFAR in ACPK (2.5.21) which works much more 'refined' (i.e. gives a lot of information about processes which are equal to chaos χ by CHAR).
It is interesting that failure semantics as in Brookes,Hoare,Roscoe (1984) cannot work with KFAR but must adopt the crude rule CHAR in order to escape inconsistency. See further comments in 2.6.3.

2.5.23.

ATS *algebra of trace sets*

$x+y = y+x$		A1
$x+(y+z) = (x+y)+z$		A2
$x+x = x$		A3
$a(x+y) = ax+ay$		A4'
$x+\varepsilon = x$		E1
$\varepsilon x = x$		E2
$a\varepsilon = a$		E3
$(x+y)\|\|z = x\|\|z + y\|\|z$		TSM1
$z\|\|(x+y) = z\|\|x + z\|\|y$		TSM2
$ax \| by = a(x\|by) + b(ax\|y)$	if $f(a,b) \neq$	TSM3
$ax \| by = a(x\|by) + b(ax\|y) + c(x\|y)$	if $f(a,b) = c$	TSM4
$\varepsilon\|\|x = x$		TSM5
$x\|\|\varepsilon = x$		TSM6
$\partial_H(\varepsilon) = \varepsilon$		DTS1
$\partial_H(ax) = a\partial_H(x)$	if $a \notin H$	DTS2
$\partial_H(ax) = \varepsilon$	if $a \in H$	DTS3
$\partial_H(x+y) = \partial_H(x) + \partial_H(y)$		DTS4
$\tau_I(\varepsilon) = \varepsilon$		TI1
$\tau_I(ax) = a\tau_I(x)$	if $a \notin I$	TI2
$\tau_I(ax) = \varepsilon.\tau_I(x)$	if $a \in I$	TI3
$\tau_I(x+y) = \tau_I(x) + \tau_I(y)$		TI4

2.5.23, comments

ATS, algebra of trace sets, is the only axiom system in these notes not fitting in the above scheme of composing the before mentioned building blocks: ATS does not use a sinlgle building block from 2.5.3.

ε comes in as the result of abstraction (by τ_I). In traces, ε is very simple to deal with: E1-3. (Contrast this with τ and its much more complicated τ-laws T1-3.) The presence of ε prohibits the use of general multiplication as in BPA satsfying A4,5. Namely:

$$xy = (x+\varepsilon)y = xy + \varepsilon y = xy + y$$

would yield the undesirable equation $xy = xy + y$.

Instead, prefix multiplication is used. Axiom A4' states that we have to do with trace sets.

δ is absent in this version of trace theory.

No distinction is made between good and bad termination.

Merge ($\|$) is much simpler now; therefore the left-merge ($\lfloor\!\lfloor$) and communication merge ($|$) from ACP are not needed. The merge uses a partial communication function

$$f: (A-\{\varepsilon\})^2 \longrightarrow A$$

(partial since δ is absent).

Note that

$$ab = a(b+\varepsilon) = ab + a\varepsilon = ab + a.$$

In general one proves that terms in the initial algebra of AST, can be represented as the elements of $\{\varepsilon\} \cup$ the collection of finite non-empty prefix-closed sets of finite words over $A-\{\varepsilon\}$.

Another model of ATS is the universe of prefix closed trace sets, with finite traces; ε is $\{\lambda\}$ (λ being the empty trace). This model is the projective limit of the "modulo depth n" structures.

It would be interesting also to consider models of ATS containing infinite traces.

The abstraction operator τ_I is a renaming operator. Note that ∂_H, the encapsulation operator, is now no longer merely a renaming operator (as it is in ACP). This is because δ is missing and the role that δ played in encapsulation in ACP, is now directly built into ∂_H.

ATS plays a role in a comparison of LT and BT (linear time and branching time semantics). It turns out that for deadlock free and deterministic systems, ATS is just as adequate as ACP. Here 'deterministic' means: containing no subprocess $ax + ay$ $(x \neq y)$.

2.6. Impossible combinations.

The preceding pages suggest that the axiom systems are built in a modular way, and one could wonder whether the 23 building blocks do not simply yield 8388607 (= $2^{23}-1$) axiom systems. However, not every combination of building blocks is consistent. In this section some examples are given of pairs of incompatible building blocks.

2.6.1. $BPA_{\tau LR}$ cannot be extended with both δ (and its axioms A6,7) and KFAR. Namely:

$$BPA_{\tau LR} \vdash \langle X \mid X = \tau X \rangle = \tau;$$

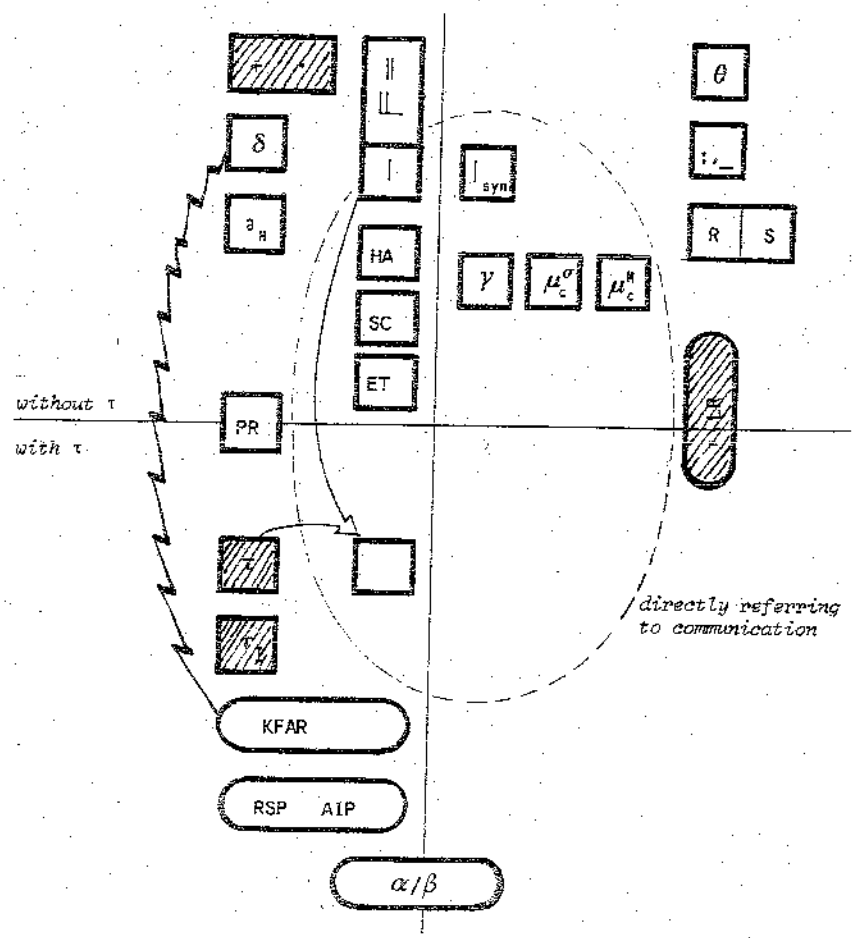
but in the presence of δ and KFAR we would have, abbreviating $\langle X \mid X = iX \rangle$ by ξ:

$$\vdash \langle X \mid X = \tau X \rangle = \tau_{\{i\}}(\xi) = \tau$$

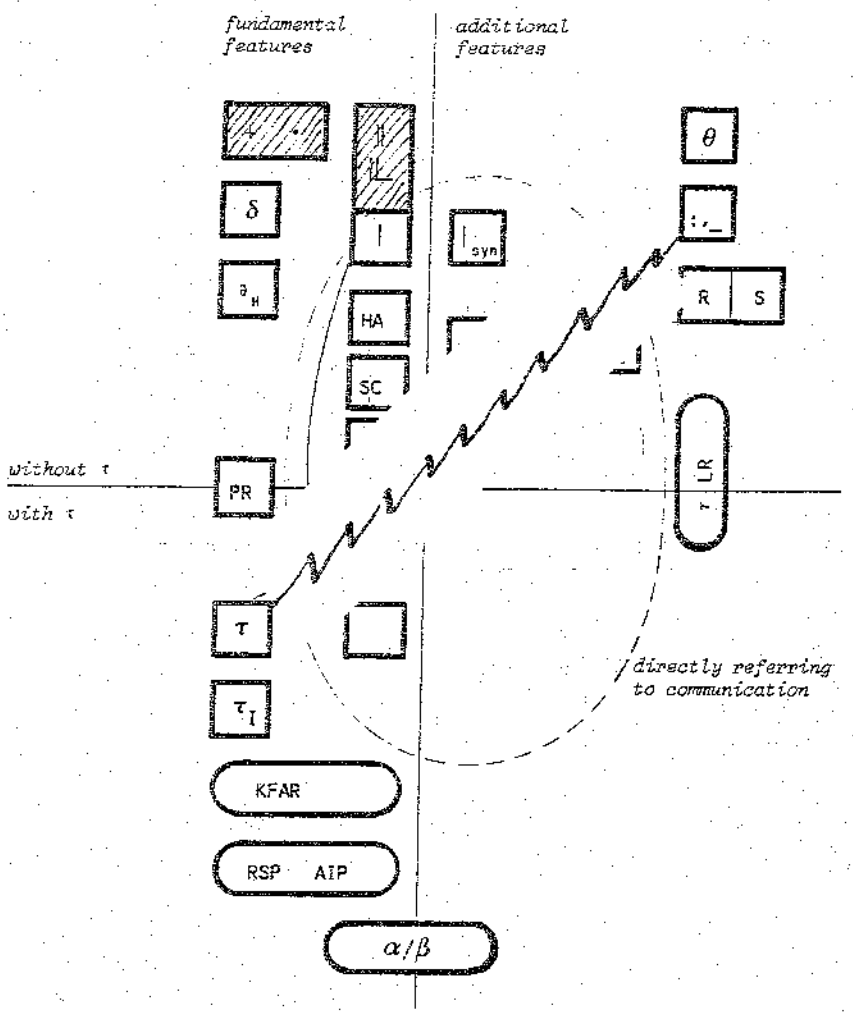
while on the other hand by KFAR, since $\xi = i\xi = i\xi + \delta$:

$$\tau_{\{i\}}(\xi) = \tau \cdot \tau_{\{i\}}(\delta) = \tau\delta$$

yielding the contradiction $\tau = \tau\delta$.

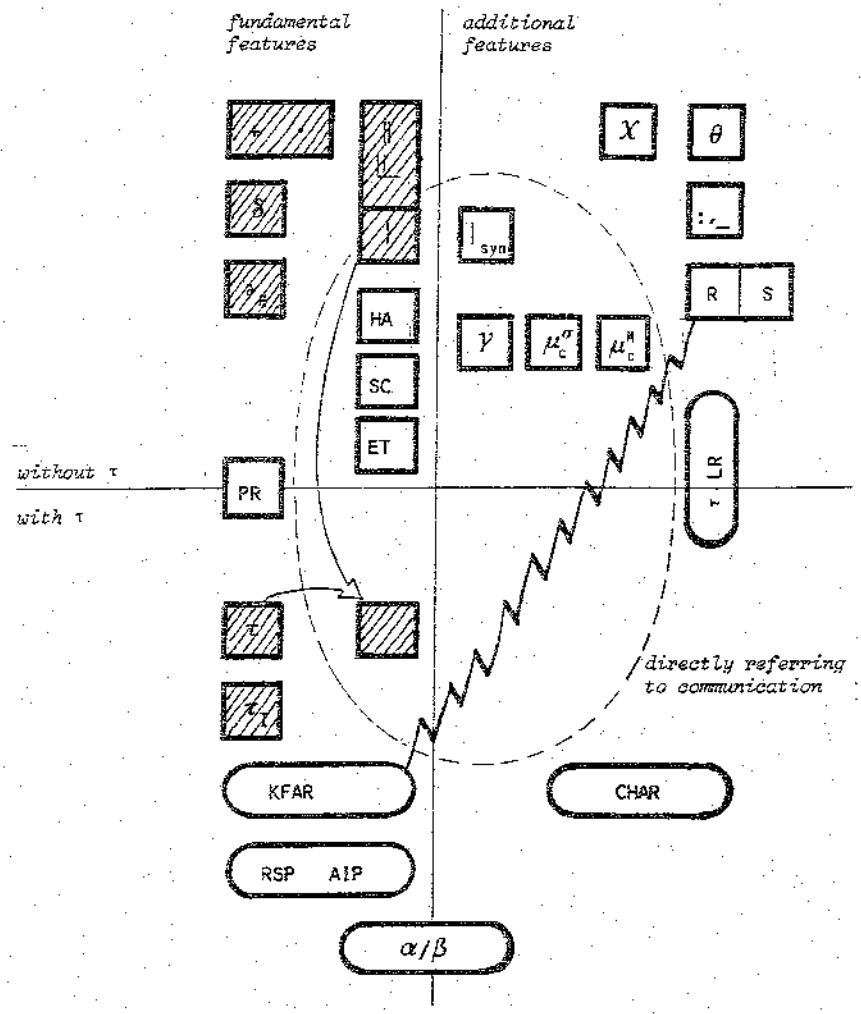


2.6.2.



Given PA, it is at least problematic to add both abstraction (τ) and tight multiplication (:).

2.6.3. THE INCONSISTENCY OF KFAR WITH FAILURE SEMANTICS



It turns out that ACPK, algebra of communicating processes with Koomen's fair abstraction rule, rejects failure semantics, i.e. ACPK + R,S is inconsistent.

The inconsistency can be pinpointed to the system

$$PA_\tau + RSP + KFAR + R$$

where PA_τ is as in 2.5.7, RSP is the recursive specification principle (see comments 2.5.21) stating that guarded systems of recursion equations have unique solutions and R is the 'readiness' axiom in 2.5.20.

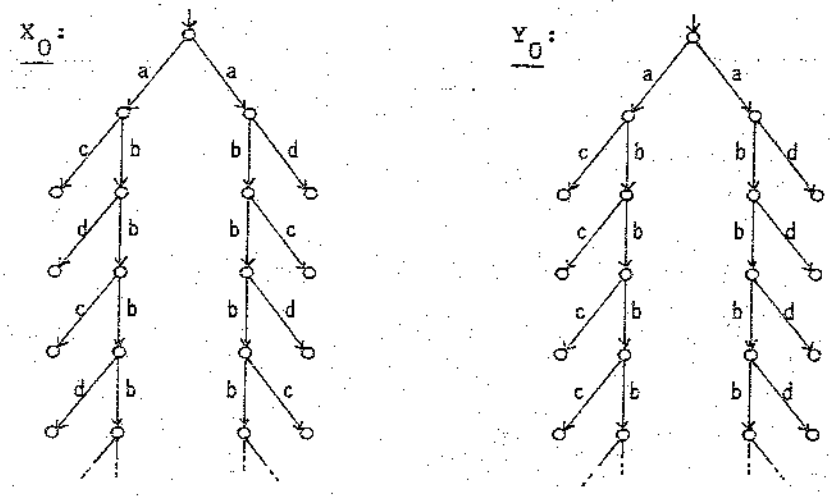
All these assumptions, except for KFAR, are valid for failure semantics (see Brookes, Hoare, Roscoe (1984)), in the absence of communication to simplify matters. It is important to notice that the contradiction, derived below, is entirely insensitive to how failure semantics works with processes containing τ-steps.

2.6.3.1. THE CONTRADICTION

Consider the following systems of guarded recursion equations:

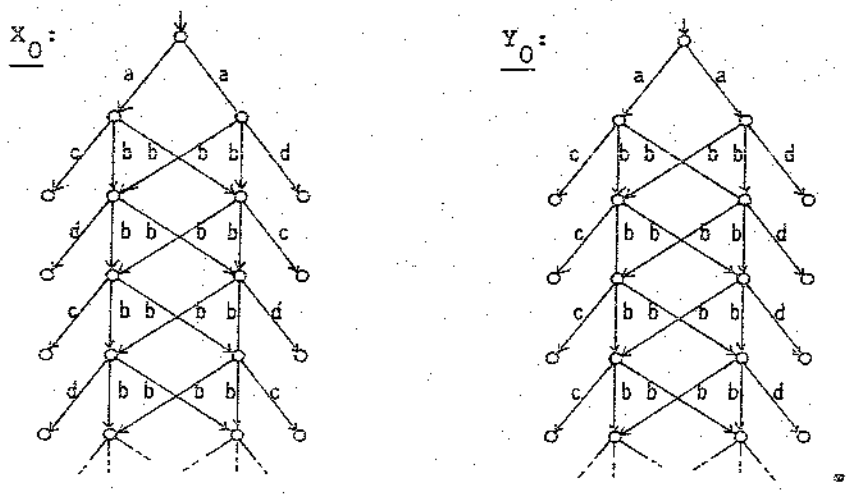
$$\begin{cases} X_0 = aX_1 + aX_2 \\ X_1 = c + bX_2 \\ X_2 = d + bX_1 \end{cases} \qquad \begin{cases} Y_0 = aY_1 + aY_2 \\ Y_1 = c + bY_1 \\ Y_2 = d + bY_2 \end{cases}$$

These systems have solutions with the following tree representations:



CLAIM: X_0 and Y_0 are failure equivalent.

Intuitively, this can be seen using the pictorial representation of axiom R (see 2.5.20) which amounts to placing 'crosses':



Now the graphs are in fact isomorphic. Hence $X_0 = Y_0$.
(It is easy to prove formally that $X_0 = Y_0$, using R and RSP.)

However, with KFAR:

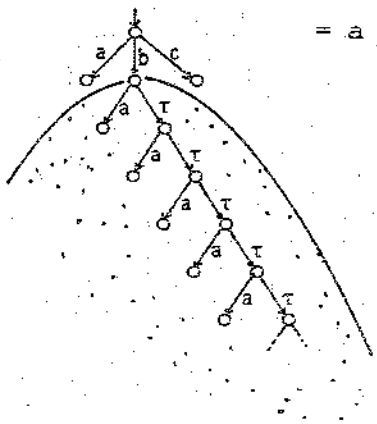
$$\tau_{\{b\}}(\underline{X}_0) = a(c+d) \quad \text{(this is an example in 2.5.20)}$$

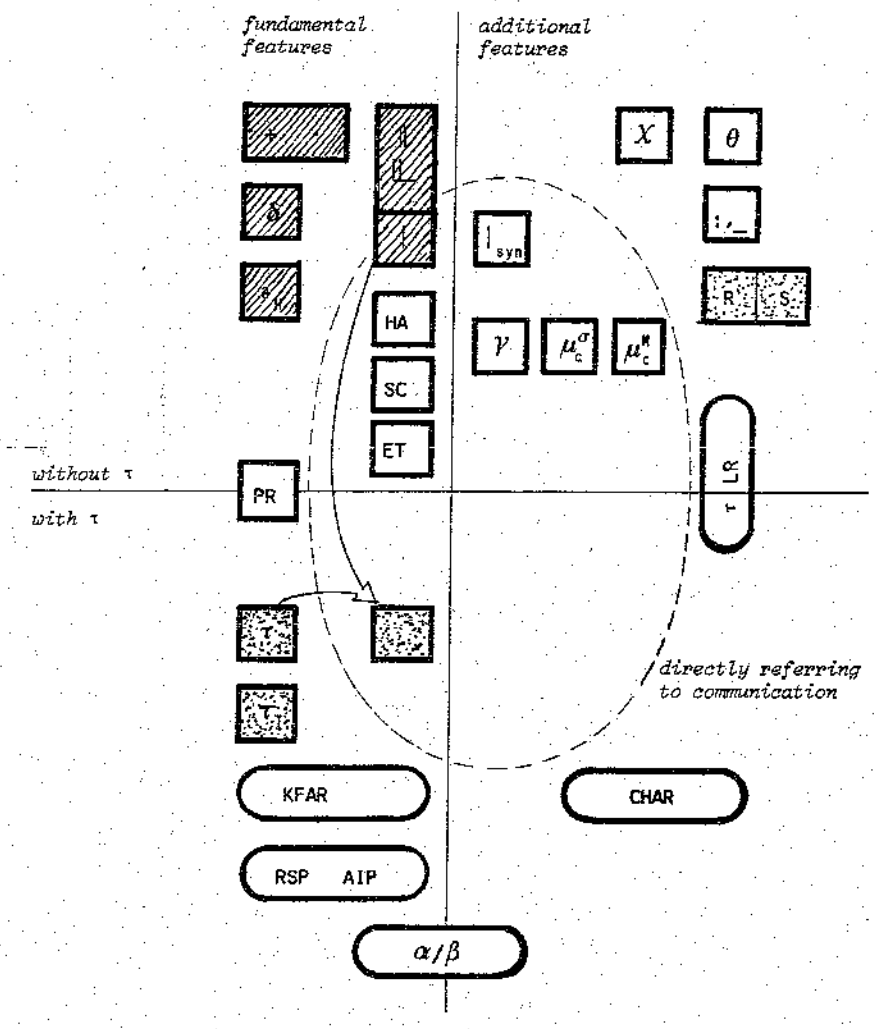
$$\tau_{\{b\}}(\underline{Y}_0) = ac + ad$$

which is a contradiction as these processes are not failure equivalent.

2.6.3.2. PARADOX. There is a paradox involved here:
(1) failure semantics contains bisimulation semantics, in the sense that bisimilar processes are failure equivalent. In particular, Milner's τ-laws are valid in failure semantics.
(2) KFAR is semantically supported by bisimulation semantics: the rule is valid in process models constructed via bisimulation.
(3) The contradiction above shows that failure semantics and KFAR exclude each other.

So, why is failure semantics as in Brookes, Hoare, Roscoe (1984) not simply inconsistent? The answer is that in failure semantics as in BHR(1984) a drastic amputation is performed of this problem area in which the inconsistency would appear, by requiring that a process with an infinite τ-path from the root, is equal to CHAOS (in our notation: χ, see $BPA_{\tau\chi}$ in 2.5.22 above.)

E.g.  $= a + b.\chi + c$ (Cf. the rule CHAR in $BPA_{\tau\chi}$.)



There are several more combinations which seem interesting. One of them is as indicated in the diagram above: in the presence of ACP, combining abstraction and failure semantics. Axioms involving τ and failure semantics are found in Brookes (1983).

SYNTAX	SEMANTICS (MODELS)				
axiom system	initial algebra	finite algebras	projective limit	process graph models: finite elements	process graph models: infinite elements
1. BPA	$I(BPA) = (A_\omega, +, .)$	$(A_n, +_n, \cdot_n)$	$(A^\infty, +, .)$	$\mathbb{H}(+,.)/\leftrightarrows$	$\mathbb{A}^\infty = \mathbb{G}_\omega/\leftrightarrows$; $\mathbb{G}_K/\leftrightarrows$
2. PA	$I(PA) = (A_\omega, +, ., \|\|, \between)$	$(A_n, +_n, \cdot_n, \|\|_n, \between_n)$	$(A^\infty, +, ., \|\|, \between)$	$\mathbb{H}(+,.,\|\|,\between)/\leftrightarrows$	$\mathbb{A}^\infty =$,, ,,
3. ACP	$I(ACP) = (A_\omega, +, ., \|\|, \between, \|, \delta, \partial_H)$	$(A_n$, etc.)	$(A^\infty$, etc.)	$\mathbb{H}/\leftrightarrows$	$\mathbb{A}^\infty$,, ,,
4. ACP_τ	$I(ACP_\tau) = (A_\omega, +, ., \|\|, \between, \|, \delta, \tau, \partial_H, \tau_I)$			$\mathbb{H}/\leftrightarrows_{rr\tau}$	
5. ACP_τ + rules					graphs mod. $\leftrightarrows_{rr\tau\delta}$
6. BPA_δ	$I(BPA_\delta) = (A_\omega, +, ., \delta)$				
7. PA_δ	$I(PA_\delta) = (A_\omega, +, ., \|\|, \between, \delta)$				
8. BPA_τ	$I(BPA_\tau)$			$\mathbb{H}(+,.,\tau)/\leftrightarrows_{rr\tau}$	
9. BPA_{LR}	$I(BPA_{LR})$				$\mathbb{R}(+,.)/\leftrightarrows$
10. $BPA_{\tau LR}$	$I(BPA_{\tau LR})$				$\mathbb{R}(+,.,\tau,\tau_I)/\leftrightarrows_{rr\tau}$
11. $PA_{\tau LR}$	$I(PA_{\tau LR})$				$\mathbb{R}(..)/\leftrightarrows_{rr\tau}$
12. ACP_{RS}	$I(ACP_{RS})$				
13. ASP	$I(ASP)$			$\mathbb{H}$ modulo failure equivalence	
⋮	⋮				

comments on the table

A box around a model denotes that it is completely axiomatised by the axiom system on the same line. So the boxes around the initial algebras I(...) denote trivial completeness results; the boxes in the two right columns denote real completeness theorems.

$\mathbb{H}$ is the domain of acyclic and finite process graphs; $\mathbb{R}$ consists of finite process graphs (corresponding to regular processes); and $\mathbb{G}_\kappa$ contains process graphs whose branching degree in every node is less than cardinal κ.
For the operations on process graphs, see survey part I (report CS-R8421).

The process graph domains turn into models after dividing out the appropriate congruences. In our case these are several notions of bisimulation:
$\leftrightarrows$, $(\leftrightarrows_\tau)$, $\leftrightarrows_{r\tau}$, $\leftrightarrows_{r\tau\delta}$.
The first one is ordinary bisimulation (called strong equivalence in Milner (1980), the second one is Milner's observational equivalence (not a congruence), the third one, rooted τ-bisimulation, coincides with Milner's observational congruence and the last one takes moreover good and bad termination into account.
The first three notions of bisimulation are explained in survey part I.

REFERENCES (See also additional references on page 70.)

BACK, R. J. R., AND MANNILA, H. (1982a), A refinement of Kahn's semantics to handle nondeterminism and communication, *in* ACM Conf. on Principles of Distributed Computing, Ottawa.

BACK, R. J. R., AND MANNILA, H. (1982b), "On the Suitability of Trace Semantics for Modular Proofs of Communicating Processes," Dept. of Computer Science, University of Helsinki.

DE BAKKER, J. W., AND ZUCKER, J. I. (1982a), Denotational semantics of concurrency, *in* "Proc. 14th ACM Sympos. on Theory of Computing," 153–158.

DE BAKKER, J. W., AND ZUCKER, J. I. (1982b), Processes and the denotational semantics of concurrency, *Inform. Control* **54**, No. 1/2, 70–120.

DE BAKKER, J. W., BERGSTRA, J. A., KLOP, J. W., AND MEYER, J.-J. CH. (1983), Linear time and branching time semantics for recursion with merge, *in* "Proc. 10th Int. Colloq. Automat. Lang. & Programming, Barcelona" (J. Diaz, Ed.), Lecture Notes in Computer Science No. 154, 39–51, Springer-Verlag, New York/Berlin; expanded version, *Theoret. Comput. Sci.*, in press.

BERGSTRA, J. A. AND KLOP, J. W. (1983a), "A Process Algebra for the Operational Semantics of Static Data Flow Networks," Report IW222/83, Mathematisch Centrum, Amsterdam.

BERGSTRA, J. A. AND KLOP, J. W. (1983b), "An Algebraic Specification Method for Processes over a Finite Action Set," Report IW 232/83, Mathematisch Centrum, Amsterdam.

BERGSTRA, J. A. AND KLOP, J. W. (1984a), The algebra of recursively defined processes and the algebra of regular processes, *in* "Proc. 11th Int. Colloq. Automat. Lang. & Programming, Antwerpen" (J. Paredaens, Ed.), Lecture Notes in Computer Science No. 172, 82–94, Springer-Verlag, New York/Berlin.

BERGSTRA, J. A. AND KLOP, J. W. (1984b), "Algebra of Communicating Processes with Abstraction," Report CS-R8403, Centrum voor Wiskunde en Informatica, Amsterdam.

BERGSTRA, J. A. AND KLOP, J. W. (1984c), "Verification of an Alternating Bit Protocol by Means of Process Algebra," Report CS-R8404, Centrum voor Wiskunde en Informatica, Amsterdam.

BERGSTRA, J. A. AND KLOP, J. W. (1984d), "Fair FIFO Queues Satisfy an Algebraic Criterion for Protocol Correctness," Report CS-R8405, Centrum voor Wiskunde en Informatica, Amsterdam.

BROOKES, S. D. (1983), On the relationship of CCS and CSP, *in* "Proc. 10th Int. Colloq. Automat. Lang. & Programming, Barcelona" (J. Diaz, Ed.), Lecture Notes in Computer Science No. 154, 83–96, Springer-Verlag, New York/Berlin.

BROOKES, S. D. AND ROUNDS, W. C. (1983), Behavioural equivalence relations induced by programming logics, *in* "Proc. 10th Int. Colloq. Automat. Lang. & Programming, Barcelona" (J. Diaz, Ed.), 97–108, Lecture Notes in Computer Science, Springer-Verlag, New York/Berlin.

DERSHOWITZ, N. (1982), Orderings for term-rewriting systems, *Theoret. Comput. Sci.* **17**, 279–301.

GRAF, S. AND SIFAKIS, J. (1984), A modal characterization of observational congruence on finite terms of CCS, *in* "Proc. 11th Int. Colloq. Automat. Lang. & Programming, Antwerpen" (J. Paredaens, Ed.), Lecture Notes in Computer Science No. 172, 222–234, Springer-Verlag, New York/Berlin.

HENNESSY, M. (1981a), "On the Relationship Between Time and Interleaving," Univ. of Edinburgh.

HENNESSY, M. (1981b), A term model for synchronous processes, *Inform. Control* **51**, 58–75.

HENNESSY, M. (1982a), "Synchronous and Asynchronous Experiments on Processes," Report CSR-125-82, Univ. of Edinburgh.

HENNESSY, M. (1982b), "Axiomatising Finite Delay Operators," Report CSR-124-82, Univ. of Edinburgh.

HENNESSY, M. (1983), "A model for Nondeterministic Machines," CSR-135-83, Univ. of Edinburgh.

HENNESSY, M. AND MILNER, R. (1980), On observing nondeterminism and concurrency, *in* "Proc. 7th Int. Colloq. Automat. Lang. & Programming," 299–309, Lecture Notes in Computer Science No. 85, Springer-Verlag, New York/Berlin.

HENNESSY, M. AND MILNER, R. (1983), "Algebraic Laws for Nondeterminism and Concurrency," Report CSR-133-83, Univ. of Edinburgh; *J. Assoc. Comput. Mach.*, in press.

HENNESSY, M. AND DE NICOLA, R. (1982) "Testing Equivalences for Processes" Report CSR-123-82, Univ. of Edinburgh.

HENNESSY, M. AND PLOTKIN, G. (1980), A term model for CCS, *in* "Proc. 9th Mathematical Foundations of Computer Science" (P. Dembiński, Ed.), Lecture Notes in Computer Science No. 88, Springer Verlag, New York/Berlin.

HOARE, C. A. R. (1978), Communicating sequential processes, *Comm. ACM* **21** 666–677.

HOARE, C. A. R. (1980), A model for communicating sequential processes, *in*, "On the Construction of Programs" (R. M. McKeag and A. M. McNaghton, Eds.), pp. 229–243, Cambridge Univ. Press, London/New York.

HOARE, C., BROOKES, S., AND ROSCOE, W. (1981), "A Theory of Communicating Sequential Processes," *J. Assoc. Comput. Mach.* **31**, No. 3, 560–599.

HUET, G. (1980), Confluent reductions: Abstract properties and applications to term rewriting systems, *J. Assoc. Comput. Mach.* **27**, No. 4, 797–821.

KLOP, J. W. (1980), "Combinatory Reduction Systems," Mathematical Centre Tracts No. 127, Mathematisch Centrum, Amsterdam.

MILNE, G. (1982a), Abstraction and nondeterminism in concurrent systems, 3rd International Conference on Distributed Systems, Florida, Oct. 1982, IEEE. p. 358–364.

MILNE, G. (1982b), "CIRCAL: A Calculus for Circuit Description," INTEGRATION, the VLSI journal **1** (1983), 121–160.

MILNE, G. AND MILNER, R. (1979), Concurrent processes and their syntax, *J. Assoc. Comput. Mach.* **26**, No. 2, 302–321.

MILNER, R. (1980), "A Calculus of Communicating Systems," Lecture Notes in Computer Science No. 92, Springer Verlag, New York/Berlin.

MILNER, R. (1984), A complete inference system for a class of regular behaviours, *J. Comput. and Syst. Sci.* **28** 439–466.

MILNER, R. (1983), Calculi for synchrony and asynchrony, *Theoret. Comput. Sci.* **25** (1983), p. 267–310.

NIVAT, M. (1979), Infinite words, infinite trees, infinite computations, *in* "Foundations of Computer Science III.2" (J. W. de Bakker and J. van Leeuwen, Eds.), pp. 3–52, Mathematical Centre Tracts No. 109, Mathematisch Centrum, Amsterdam.

NIVAT, M. (1980), Synchronization of concurrent processes, *in* "Formal Language Theory" (R. V. Book, Ed.), pp. 429–454, Academic Press, New York.

OLDEROG, E.-R. AND HOARE, C. A. R. (1983), Specification-oriented semantics for communicating processes, *in* "Proc. 10th Int. Colloq. Automat. Lang. & Programming, Barcelona," 561–572, Lecture Notes in Computer Science No. 154, Springer-Verlag New York/Berlin; expanded version, Technical Monograph PRG-37, Oxford Univ. Comput. Lab., February 1984.

PARK, D. M. R. (1981), Concurrency and automata on infinite sequences, *in* "Proc. 5th GI (Gesellschaft für Informatik) Conference, Lecture Notes in Computer Science No. 104, Springer-Verlag, New York/Berlin.

PRATT, V. R. (1982), On the composition of processes, *in* "Proc. 9th ACM Sympos. on Principles of Programming Languagues," pp. 213–223.

REM, M. (1983), Partially ordered computations, with applications to VLSI design, *in* "Proc. 4th Advanced Course on Foundations of Computer Science, Part 2" (J. W. de Bakker and J. van Leeuwen, Eds.), 1–44, Mathematical Centre Tracts No. 159, Mathematisch Centrum, Amsterdam.

STAPLES, J. AND NGUYEN, V. L. (1983), "A Fixpoint Semantics for Nondeterministic Data Flow," Report No. 48, Dept. of Comput. Sci., Univ. of Queensland, Australia.

WINSKEL, G. (1983a), Event structure semantics for CCS and related languages, *in* "Proc. Int. Colloq. Automat. Lang. & Programming, 561–576, Lecture Notes in Computer Science No. 140, Springer-Verlag, New York/Berlin.

WINSKEL, G. (1983b), Synchronisation trees, *in* "Proc. 10th Int. Colloq. Automat. Lang. & Programming, Barcelona" (J. Diaz, Ed.), 695–711, Lecture Notes in Computer Science No. 154, Springer Verlag, New York/Berlin.

continued on next page

AUSTRY, D. and BOUDOL, G., Algèbre de processus et synchronisation, TCS 30 (1984), 91-131.

DE BAKKER, J.W., MEYER, J.-J.Ch., OLDEROG, E.-R. and ZUCKER, J.I. (1984), Transition systems, infinitary languages and the semantics of uniform concurrency, Report IR-95, Vrije Universiteit, Amsterdam.

BERGSTRA, J.A., HEERING, J., KLINT, P. and KLOP, J.W., Een analyse van de case-study HμP, mimeographed notes, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

BERGSTRA, J.A. and KLOP, J.W.(1984e), Process algebra for synchronous communication, Information and Control Vol.60, No.1-3, 109-137.

BERGSTRA, J.A., KLOP, J.W. and TUCKER, J.V. (1984), Process algebra with asynchronous communication mechanisms, Report CS-R8410, Centrum voor Wiskunde en Informatica, Amsterdam.

BROCK, J.D. and ACKERMAN, W.B.(1981), Scenarios: A model of non-determinate computation, in: Proc. Formalization of Programming Concepts (eds. J. Díaz and I. Ramos), 252-259, Springer LNCS 107.

EBERGEN, J., On VLSI design, NGI-SION Proceedings 1984, 144-150, Nederlands Genootschap voor Informatica, Amsterdam.

KAHN, G. (1974), The semantics of a simple language for parallel programming, in: Proc. IFIP 74, North-Holland, Amsterdam 1974.

KOYMANS, R., VYTOPIL, J. and DE ROEVER, W.P.(1983), Real-time programming and asynchronous message passing, in: Proc. of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal 1983.

KUIPER, R. and DE ROEVER, W.P. (1982), Fairness assumptions for CSP in a temporal logic framework, IFIP WG2 Working Conference on the Formal Description of Programming Concepts, Proc., Garmisch 1982.

LAMPORT, L. (1979), 'Sometime' is sometimes 'NOT never', tutorial on the temporal logics of programs, SRI International CSL-86, 1979.

MEYER, J.-J.Ch. (1984a), Fixed points and arbitrary and fair merge of a fairly simple class of processes, Part 1: the arbitrary merge; Report IR-89, Vrije Universiteit, Amsterdam.

MEYER, J.-J.Ch. (1984b), Fixed points and the arbitrary and fair merge of a fairly simple class of processes, Part 2 (the fair merge), Report IR-92, Vrije Universiteit, Amsterdam.

REISIG, W. (1984), Partial order semantics versus interleaving semantics for CSP-like languages and its impact on fairness, in: Proc. Int. Colloq. Automat. Lang. & Programming (J.Paredaens, Ed.), 403-413, Springer LNCS No. 172.

PNUELI, A. (1977), The temporal logic of programs, in: Proc. 19th Ann. Symp. on Foundations of Computer Science, IEEE, 46-57.

DE SIMONE, R. (1984), On MEIJE and SCCS: infinite sum operators vs. non-guarded definitions, TCS 30 (1984), 133-138.

VAN DE SNEPSCHEUT, J.L.A. (1983), Trace Theory and VLSI design, Ph.-D. Thesis, Eindhoven University of Technology, 1983.