workshop abstract data types & concurrency

October 10-14, 1983

Lecture Notes on Process Algebra

J. A. Bergstra , J. W. Klop

Centrum voor Wiskunde en Informatica

I. PROCESS ALGEBRA : PA.

In this chapter we will introduce the axiom system PA for process algebra without communication (and without internal steps); also we will introduce as semantics for PA several 'process algebras' of which the simplest one is the initial algebra of PA.

1. We start with giving, right away, the axiom system

PA

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

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

- $a, b, c, \ldots \in A$, the set of **atomic actions** (also called 'steps', or 'events'). A is also referred to as the **alphabet**.

Throughout these notes, we will assume that **A is finite**. In the axioms of PA, 'a' varies over A.

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

$+$	**alternative composition**, or sum
$\cdot$	**sequential composition**, or product
$\parallel$	**parallel composition**, or merge
$\between$	**left-merge**

The 'main' operators are $+, \cdot, \parallel$. Left-merge $\between$ is an auxiliary operator.

This concludes the description of the signature of PA.

The axioms of PA intend to describe processes; what exactly processes are, depends on which semantics we assign to PA. For the moment, we only have **process expressions** (or process terms): they are built from the $a \in A$ by means of $+, \cdot, \parallel, \between$. Examples of process expressions are:

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

We will employ some obvious notational conventions:

- xy stands for $x \cdot y$
- outermost brackets are omitted
- the operator $\cdot$ has the greatest binding power
- x^2 stands for xx, etc.
- $\between$, $\lfloor\!\lfloor$ bind stronger than $+$.

So the two process expressions above may be written as

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

2. Semantics of PA.

We will give three different **process algebras** for PA, that is, domains of processes which satisfy the axioms of PA. They are:

A_ω, the initial algebra of PA
$\mathcal{A}^\infty$, the graph model of PA
A^∞, the standard model of PA

2.1. We start with the simplest model, the initial algebra A_ω. The elements of A_ω are the process expressions modulo the equivalence given by PA. So, in A_ω, '$a+b$' and '$b+a$' and

'$a+b+a$' are the same. Likewise, the process expression

$$(aa \between b + cd)e$$

denotes in A_ω the same element as

$$(aa \between b)e + cde$$

or as

$$a(a \parallel b)e + cde$$

or as

$$a(a \between b + b \between a)e + cde$$

or as

$$a(ab + ba)e + cde$$

or as

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

Note that we have just proved that

$$PA \vdash (aa \between b + cd)e = a(abe + bae) + cde,$$

and that in the RHS the $\parallel$, $\between$ operators have vanished. This is a general

2.1.1. **FACT.** ***Using the axioms of PA as rewrite rules (from left to right) every process expression can be rewritten to an expression without $\parallel$ or $\between$.***

This entails that we can think of the elements of the initial algebra A_ω as: expressions built from atoms via $+$ and $\cdot$ only. (We may even suppose that these expressions are such that no further applications of the axioms A3 and A4 are possible.) Using this fact we arrive at the following convenient

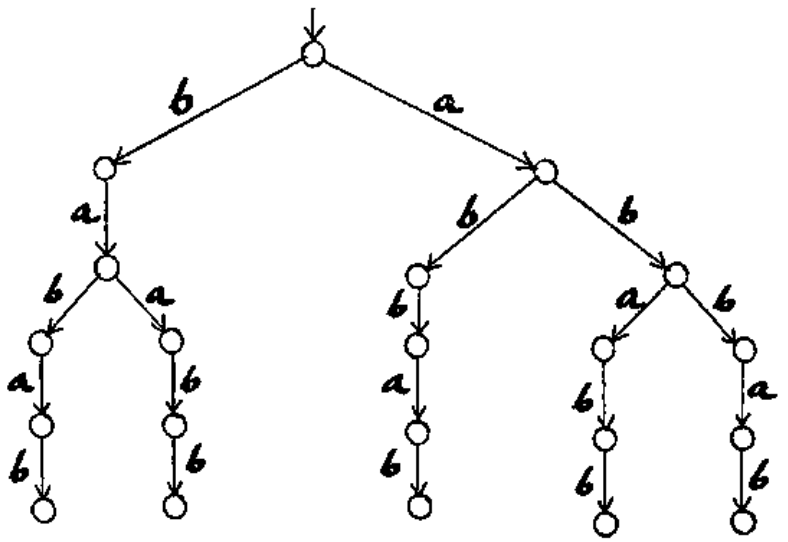
2.1.2. **FACT.** ***(Representation of elements of A_ω.) Modulo PA-equivalence, A_ω is inductively generated as follows:***

$$\left.\begin{array}{l} x_i \in A_\omega \quad (i=1,\dots,n) \\ a_i \in A \\ b_j \in A \quad (j=1,\dots,m) \end{array}\right\} \Longrightarrow \sum_{j=1}^{m} b_j + \sum_{i=1}^{n} a_i x_i \in A_\omega.$$

2.1.1.1. **EXAMPLE**, to illustrate Fact 2.1.1.

$$\begin{aligned}
&\mathbf{bab \between ab} = \\
&bab \between ab + ab \between bab = \\
&b(ab \between ab) + a(b \between bab) = \\
&b(ab \between ab + ab \between ab) + a(b \between bab + bab \between b) = \\
&b(ab \between ab) + a(bbab + b(ab \between b)) = \\
&b(a(b \between ab)) + a(bbab + b(ab \between b + b \between ab)) = \\
&\mathbf{b(a(bab+abb)) + a(bbab+b(abb+bab))}
\end{aligned}$$

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

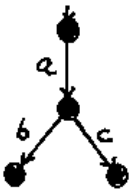


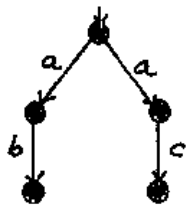
We will return to such trees later. First, we want to stress an important inequality in A_ω (or, 'non-derivability' in PA):

2.1.3. **FACT.** *In general, in* A_ω:

$$x(y+z) \neq xy + xz.$$

Of course special instances of the equation $x(y+z) = xy+xz$ may hold; e.g. $a(b+b) = ab + ab$ does hold. But the process expressions $a(b+c)$ and $ab+ac$,

with 'trees' 

and 

, differ in A_ω.

* 2.1.4. **EXERCISE / REMARK.**

If $A_\omega \models x+y = az$ for some $x,y,z \in A_\omega$, $a \in A$, then

$$A_\omega \models x = y$$

("$\models$" means: 'satisfies')
(The proof requires the Church-Rosser property of the PA-axioms viewed as rewrite rules.)

The next fact is easy to prove (especially (i)):

2.1.5. **FACT.** **For all process algebras:**

(i) $x \between y = y \between x$

(ii) $$\sum_{i=1}^{n} a_i x_i \between \sum_{j=1}^{m} b_j y_j =$$

$$\sum_{i=1}^{n} a_i (x_i \between \sum_{j=1}^{m} b_j y_j) + \sum_{j=1}^{m} b_j (y_j \between \sum_{i=1}^{n} a_i x_i).$$

2.1.5.1. **EXAMPLE.**

(i): $x \between y = x \lfloor y + y \lfloor x = y \lfloor x + x \lfloor y = y \between x$.

(ii): Let $p = a_1 x_1 + a_2 x_2$ and $q = b_1 y_1 + b_2 y_2$.
Then $p \between q = p \lfloor q + q \lfloor p =$
$(a_1 x_1 + a_2 x_2) \lfloor q + (b_1 y_1 + b_2 y_2) \lfloor p =$
$a_1 x_1 \lfloor q + a_2 x_2 \lfloor q + b_1 y_1 \lfloor p + b_2 y_2 \lfloor p =$
$a_1 (x_1 \between q) + a_2 (x_2 \between q) + b_1 (y_1 \between p) + b_2 (y_2 \between p)$.

We have now constructed our first process algebra as semantics of PA, the initial algebra A_ω. It contains as elements process expressions modulo PA-equality; these elements can also be thought of as finitely branching and finitely deep process trees.

The fact that the processes in A_ω are only finitely deep, means that we cannot find solutions p in A_ω for recursive definitions like

$$p = a \cdot p$$

for, p would be $aaaa\ldots$ or a^ω.

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

2.2. Process graph algebras.

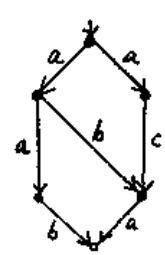
A **process graph** (sometimes called: transition diagram) over a set of atoms A is a **rooted, directed multigraph** whose edges are labeled by elements of A. Process graphs may be infinite and may contain cycles. **Process trees** are special cases: they are acyclic process graphs where no subgraph is shared (and containing no multiple edges).

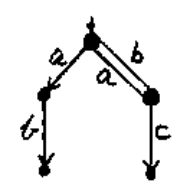
Some examples will clarify these concepts.

2.2.1. EXAMPLES.

(i) A finite, acyclic process graph, also a process tree: see the figure on p. 5

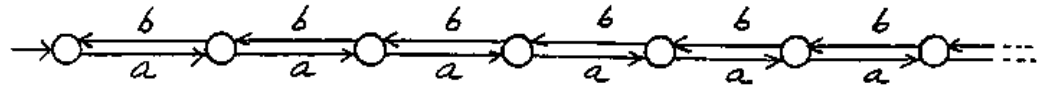
(ii) A finite, acyclic process graph, but not a tree:



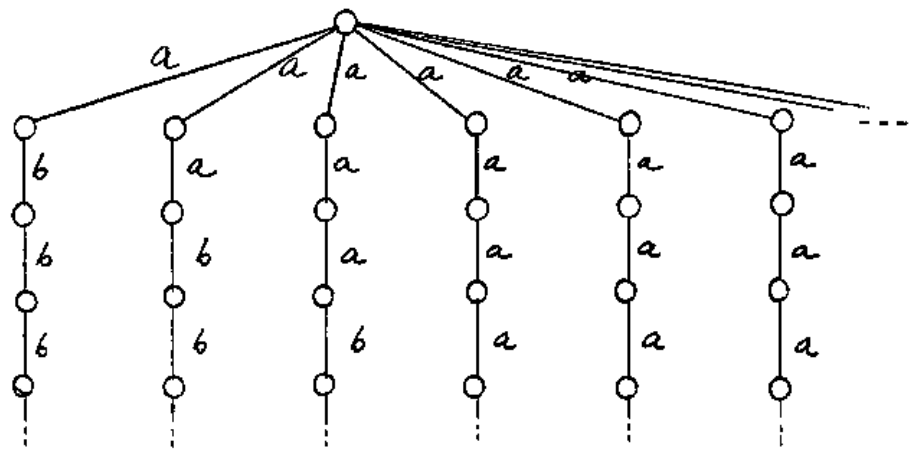


(iii) A finite process graph containing a cycle:

(iv) An infinite process graph with cycles:

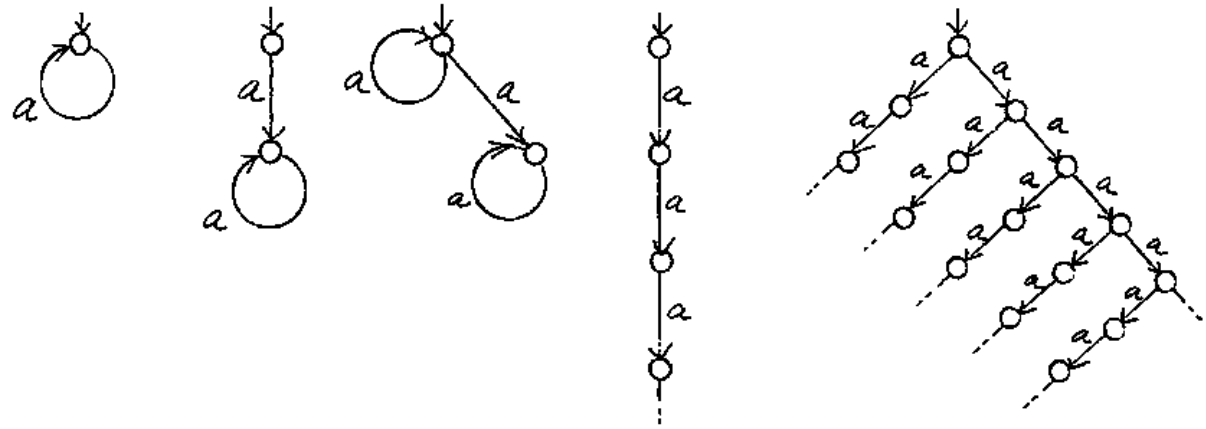


(v) An infinite process graph, also a process tree:



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

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



An elegant notion of R. Milner and/or D. Park, called **bisimulation**, does indeed identify these graphs.

2.2.2. Bisimulation of process graphs

is defined as follows. Let g_1, g_2 be process graphs with node sets $NODES(g_1)$, $NODES(g_2)$. Let s_0, t_0 be the roots of g_1, g_2 respectively.

Then **g_1, g_2 are bisimilar**, in symbols:

$$g_1 \leftrightarrows g_2$$

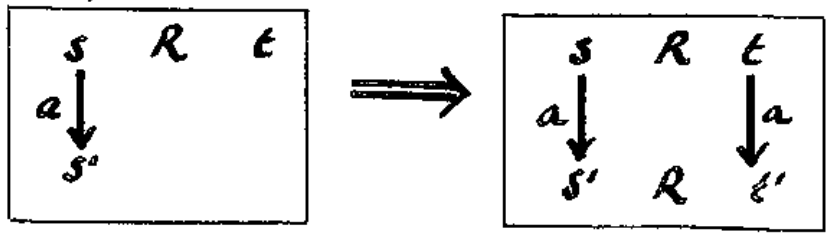
if it is possible to pair off the nodes of g_1, g_2 in such a way that from a node in g_1 which is paired with a node in g_2, the same steps are possible.

This is rather vague. More precisely:

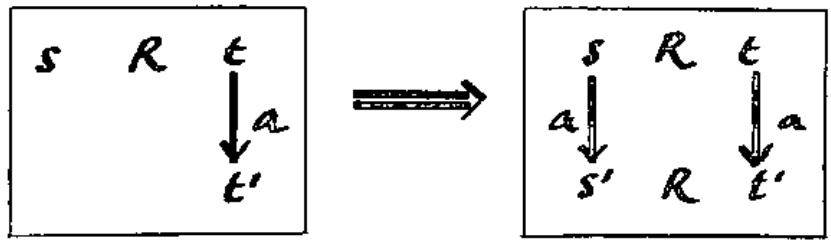
- if there is a relation $R \subseteq NODES(g_1) \times NODES(g_2)$ such that

  - $s_0 \; R \; t_0$ (the roots are related)

  - in the situation $s \; R \; t$ where $s \xrightarrow{a} s'$ is an edge of g_1, t a node of g_2, there must be an edge $t \xrightarrow{a} t'$ such that $s' \; R \; t'$.

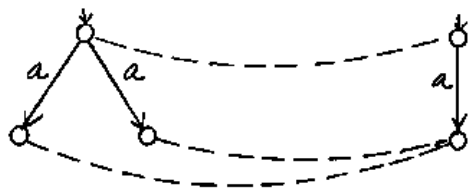


- vice versa (with the role of g_1, g_2 interchanged):

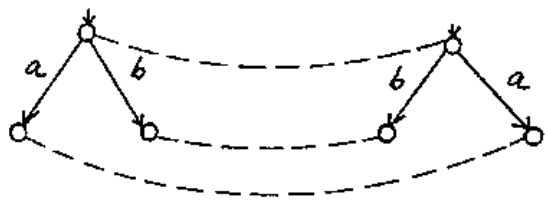


2.2.2.1. **EXAMPLES.**

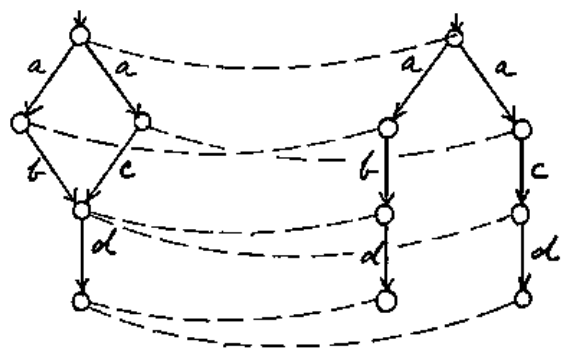
(i)



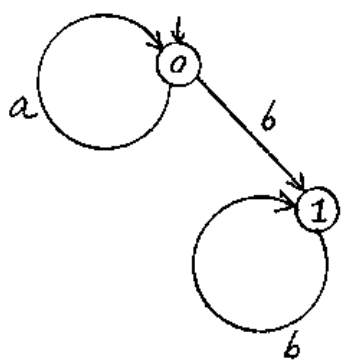
(ii)

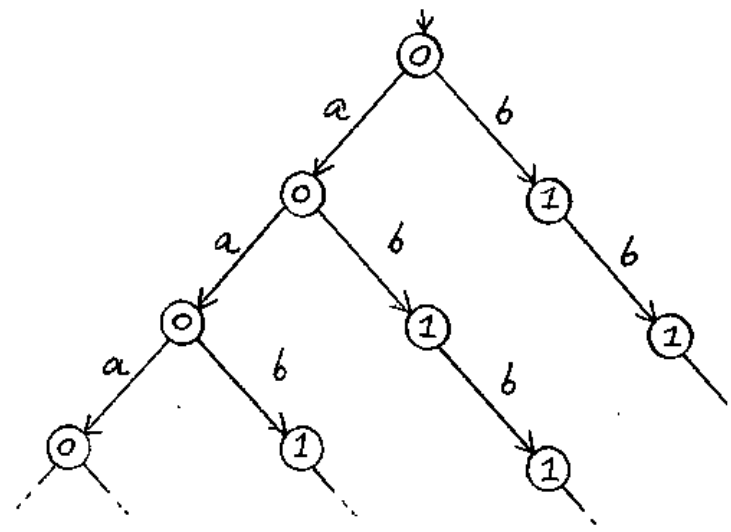


(iii)

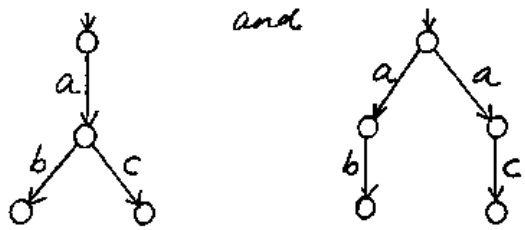


(iv)





(v) A non-example:



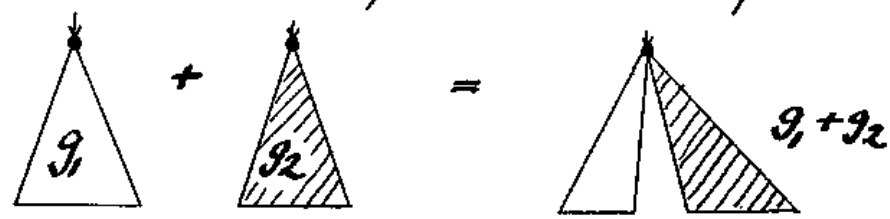
Note that **unfolding** (unwinding) a process graph respects bisimilarity.

We can now define the second process algebra for PA, called **the process graph algebra A^∞**, as follows. The elements of A^∞ are

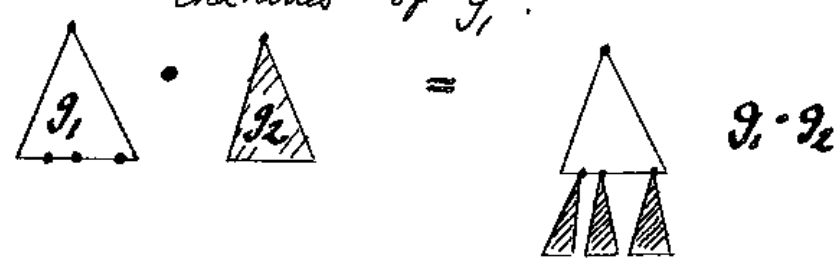
> **the finitely branching process graphs (except the trivial one with one node and no edges) modulo bisimulation.**

The operations $+, \cdot, \|, \lfloor\!\lfloor$ on A^∞ are defined as follows.

$g_1 + g_2$ is obtained by glueing together the roots of g_1 and g_2. Here we must and may suppose that the roots of g_1, g_2 are acyclic (not lying on a cycle). We can always realize this by unwinding.

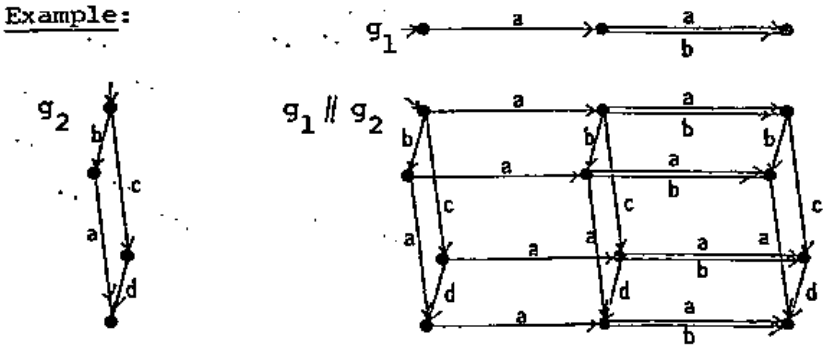


$g_1 \cdot g_2$ is obtained by appending g_2 to all endnodes of g_1.

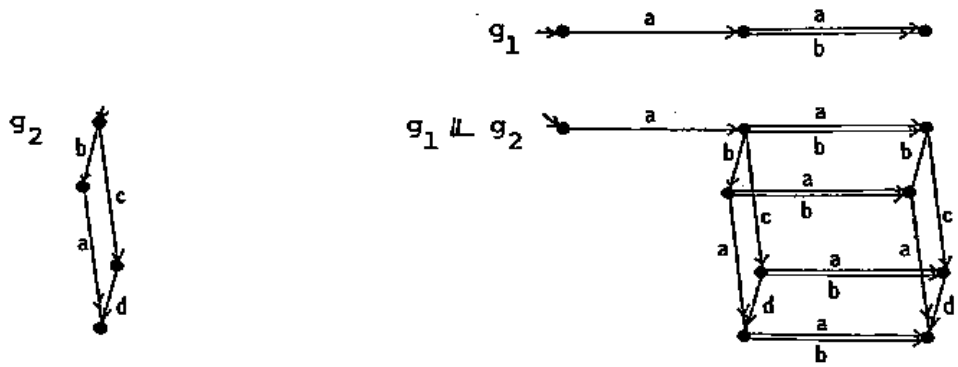


$g_1 \| g_2$ is obtained as the 'cartesian product' graph, as in the figure:

Example:



$g_1 \perp\!\!\!\perp g_2$ is obtained as a subgraph of $g_1 \| g_2$ as in the example:



(Here we have to take care that the roots are (made) acyclic again.)

It is now easy to prove the following

2.2.3. **FACT.**

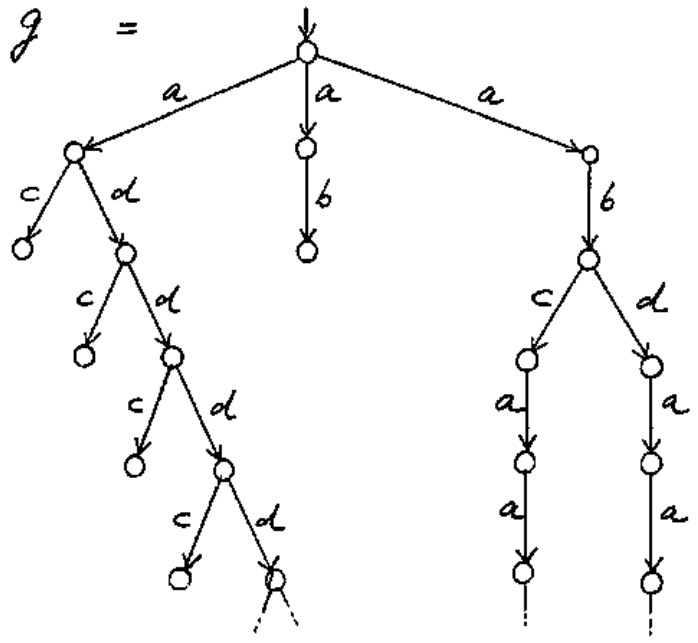
(i) $\mathbb{A}^\infty$ is a process algebra (a model of PA)

(ii) The finite elements in $\mathbb{A}^\infty$ constitute a subalgebra which is (isomorphic to) A_ω.

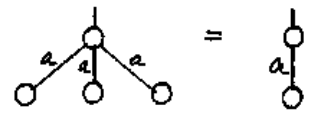
We will now describe how each element ('process') in $\mathbb{A}^\infty$ can be <u>approximated</u> by a <u>sequence of finite processes</u>:

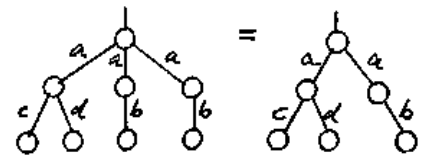
Let $g \in A^\infty$. Since we are working in A^∞ modulo bisimulation, we may suppose g is unwound and that g is a process tree. We will define the approximations (or **projections**) $(g)_n$ for each $n \geq 1$, via the following example.

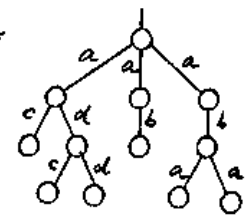
g =



Then g has projections $(g)_1, (g)_2, (g)_3, \ldots$ obtained by cutting off g's tree at level resp. $1, 2, 3, \ldots$. Here:

$(g)_1$ =  = a

$(g)_2$ =  = $a(c+d) + ab$

$(g)_3$ =  = $a(c+d(c+d)) + ab + aba$

$\vdots$

Note that we, obviously, have:

$$(g)_n = ((g)_{n+1})_n .$$

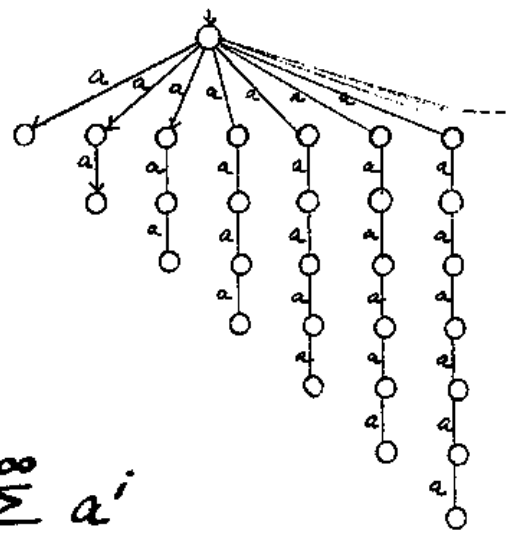
Now we have the following pleasant fact:

2.2.4. **FACT.** *Let $g, h \in A^\infty$. Then:*

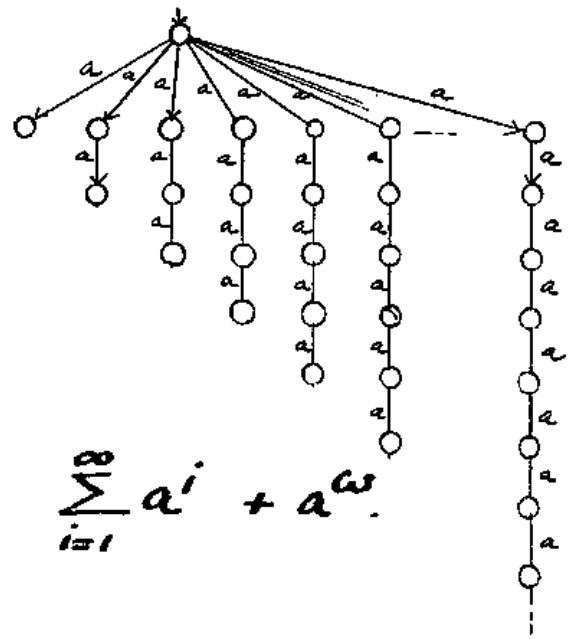
$$g = h \iff \forall n \ (g)_n = (h)_n .$$

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

Note that the condition 'finitely branching' is essential here. For, the following two graphs g, h are <u>not</u> bisimilar, but have the same finite approximations:



$\sum_{i=1}^{\infty} a^i$



$\sum_{i=1}^{\infty} a^i + a^\omega .$

* **Proof of Fact 2.2.4.**

It is interesting to see how this condition 'finitely branching' is used in the proof of Fact 2.2.4: it makes an appeal to <u>König's Lemma</u> possible.

The proof of $\Rightarrow$ is routine.

The proof of $\Leftarrow$ can be sketched as follows:

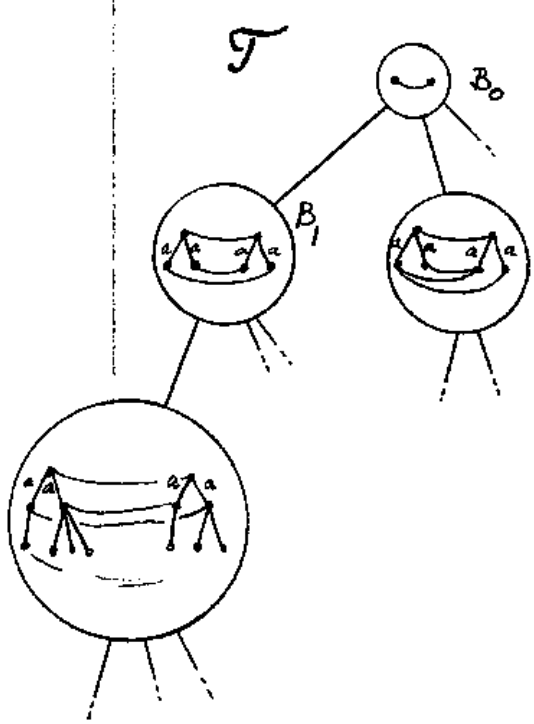
Suppose $(g)_n = (h)_n$ for all n; i.e. we have bisimulations $\mathcal{B}_n$ between the finite trees $(g)_n$ and $(h)_n$, for all n.

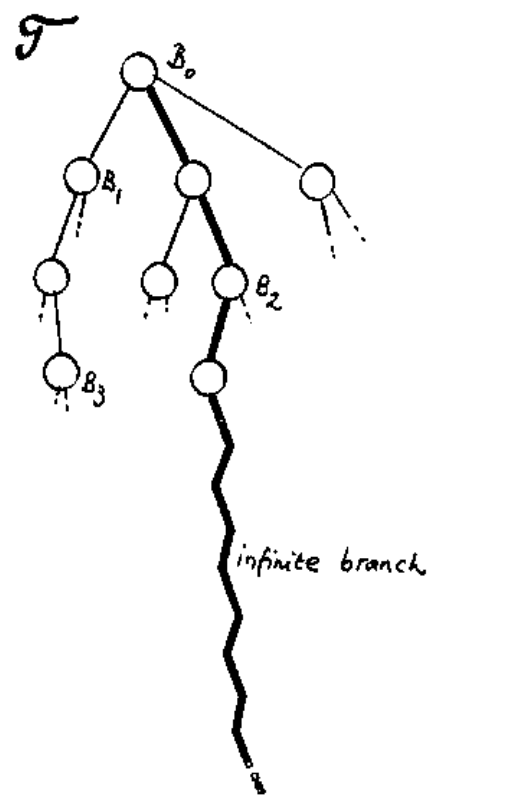
Consider the tree $\mathcal{T}$ of <u>all</u> bisimulations between $(g)_n$, $(h)_n$; that is, on level n of $\mathcal{T}$ we find all possible bisimulations between $(g)_n$ and $(h)_n$.

The ordering in $\mathcal{T}$ is: <u>extension of bisimulations</u>. I.e. Bisimulation $\mathcal{B}$ is extended by bisimulation $\mathcal{B}'$ if $\mathcal{B}$ results from $\mathcal{B}'$ by cutting off some levels of the trees which are bisimilar by $\mathcal{B}'$.

The $\mathcal{B}_n$ that we are given, do not necessarily lie on one single branch of $\mathcal{T}$. But they do ensure that $\mathcal{T}$ is infinite. Moreover, <u>$\mathcal{T}$ is finitely branching</u> because the $(g)_n$, $(h)_n$ are finite, and their are only finitely many bisimulations between finite trees. (In turn, the $(g)_n$, $(h)_n$ are finite since g, h are finitely branching.)

Hence, by König's Lemma, $\mathcal{T}$ has an infinite branch. Taking the union of the bisimulations along this infinite branch, we have a bisimulation between g and h. Hence in $\mathbb{A}^\infty$, $g = h$.





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

2.2.5. the standard model A^∞ for PA.

Remembering that an element $g \in A^\infty$ gave rise to a sequence $((g)_1, (g)_2, \ldots, (g)_n, \ldots)$ which determines g and for which $(g)_n = ((g)_{n+1})_n$, we now define without any reference to graphs:

> a projective sequence is a sequence $(p_1, p_2, p_3, \ldots, p_n, \ldots)$ of elements of A_ω such that $p_n = (p_{n+1})_n$

Of course we must define the projections $(\ )_n : A_\omega \to A_\omega$:

$$(a)_n = a$$
$$(ax)_1 = a$$
$$(ax)_{n+1} = a(x)_n$$
$$(x+y)_n = (x)_n + (y)_n$$

Now we define

the elements of A^∞ are the projective sequences.

The operations $+, \cdot, \between, \perp\!\!\!\perp$ on A^∞ are defined coördinate-wise, thus:

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

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

Note the subscripts $\uparrow$, necessary to ensure that the result from applying the operation $\square$ ($\in \{+, \cdot, /\!/, \mathbb{L}\}$) is again a projective sequence. (The simple proof employs the fact that

$$(p \square q)_n = ((p)_n \square (q)_n)_n \; . \quad)$$

2.2.6. **EXAMPLE**.

(i) $(a, a+a^2, a+a^2+a^3, \dots, \sum_{i=1}^{n} a^i, \dots) \in A^\infty$.
We will refer to this element as

$$\sum_{i=1}^{\infty} a^i \; .$$

This notation is not formally defined, however.

(ii) Call $a^\omega = (a, a^2, a^3, \dots)$.

Then $a^\omega \cdot b^\omega = ((a \cdot b)_1, (a^2 \cdot b^2)_2, \dots, (a^n \cdot b^n)_n, \dots) =$

$(a, a^2, \dots, a^n, \dots) = a^\omega$.

(iii) $\left(\sum_{i=1}^{\infty} a^i\right) \cdot \left(\sum_{i=1}^{\infty} b^i\right) = (a, a+a^2, a+a^2+a^3, \dots) \cdot$

$\cdot (b, b+b^2, b+b^2+b^3, \dots) = (a, ab+a^2, a(b+b^2)+a^2b+a^3, \dots$

(iv) $a^\omega /\!/ b^\omega = ((a /\!/ b)_1, (a^2 /\!/ b^2)_2, \dots, (a^n /\!/ b^n)_n, \dots,$

$(a+b, a(a+b)+b(a+b), \dots)$.

2.2.7. **FACT**. A^∞ *is a model of PA.*

* 2.2.8 **EX.** Note that the cardinality of A^∞ (A finite, as always in these notes) is $2^{\aleph_0}$, even if A is a singleton.

Now one may ask how A^∞ and $\mathbb{A}^\infty$ compare. The answer is that A^∞ is an extension of $\mathbb{A}^\infty$: it contains all the processes in $\mathbb{A}^\infty$ but also some processes which are <u>not</u> finitely branching, like $\sum_{i=1}^{\infty} a^i$ above. Strictly speaking, we have not defined when an element of A^∞, a projective sequence, is finitely branching or not.

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

2.2.9 The collection of **subprocesses of p** is given by

$$\begin{array}{l} p \in Sub(p) \\ ax \in Sub(p) \Rightarrow x \in Sub(p) \\ ax+y \in Sub(p) \Rightarrow x \in Sub(p) \end{array}$$

From the subprocesses of p (which may be thought of as the 'states' of the process) we can assemble a process graph G_p. This process graph is in some respects 'canonical' (it is the most 'economic' or 'minimal' process graph for p) and so will be called

the canonical process graph G_p for p.

It is defined as follows:

- the set of nodes of G_p is $Sub(p) \cup \{\sigma\}$
- the root of G_p is p
- the edges of G_p are given by:

(1) $a \in Sub(p) \Rightarrow a \xrightarrow{a} 0$ is edge,
(2) $ax \in Sub(p) \Rightarrow ax \xrightarrow{a} x$ is edge,
(3) $ax+y \in Sub(p) \Rightarrow ax+y \xrightarrow{a}$ is edge.

So now the statement that $\sum_{i=1}^{\infty} a^i$ $(= (a, a+a^2, \ldots))$ is infinitely branching makes sense: it is meant that $G_{\sum_{i=1}^{\infty} a^i}$ is so.

* 2.2.10. **REMARK**. Some interesting finite process algebras (models of PA) which were not introduced above, can be obtained as follows.

Define

$$A_n = \{(p)_n \mid p \in A_\omega\}$$

and define as operations $+_n, \cdot_n, \|_n, \lfloor\!\lfloor_n$ on A_n (a finite set.

$$x \,\square_n\, y = (x \,\square\, y)_n$$

where $\square_n$ is $+_n, \cdot_n, \|_n, \lfloor\!\lfloor_n$ and $\square \in \{+, \cdot, \|, \lfloor\!\lfloor\}$ are the operations from A_ω.

Then $A_n(+_n, \cdot_n, \|_n, \lfloor\!\lfloor_n) \models PA$.

Now A^∞ can be defined simply as the projective limit of the algebras A_n $(n \geq 1)$.

2.2.10.1. **EX.** $A_n \models [(a^n \| b^n) + a^n] \| b^n = a^n \| b^n = (a+b)^n$

$A^\infty \models [(a^\omega \| b^\omega) + a^\omega] \| b^\omega = (a^\omega \| b^\omega) = (a+b)^\omega$.

(This example shows that it is possible that merging a process with another can result in less branches; the branch a^ω in $(a^\omega \| b^\omega) + a^\omega$, that is the full binary

a-b-tree plus the branch a^ω, is "merged away".)

2.2.11. **REMARK.** From the axioms of PA, the commutativity of $\|$:

$$x \| y = y \| x$$

follows immediately. The associativity

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

does <u>not</u> follow from PA. (Indeed one can construct a process algebra with non-associative merge operator.)

However, in the process algebras which are introduced above (A_ω, A_n, A^∞, $A^{\infty\omega}$) the associativity does hold. A proof, by induction on the structure of the elements, can be given simultaneously with a proof of the useful identity

$$x \mathbin{\llcorner} (y \| z) = x \mathbin{\llcorner} (y \| z).$$

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

$$\begin{aligned} x + 0 &= x \\ 0x &= x \\ x0 &= x. \end{aligned}$$

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

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

and we certainly do not want the identity of ab and $ab + b$.

However, with the added proviso in axiom A4 :

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

(and adding $0 \between x = 0$, $x \between 0 = x$). this inconsistency is removed and we have a consistent (better : conservative) extension of PA.

Yet we will not pursue this option, since we have no need for 0. (Cf. also the remark about partial orders, 2.2.13, Also, we like to keep the 'equational format' of PA intact.

* 2.2.13. **REMARK on the existence of a cpo-structure for process algebras.**

It would be most convenient to have a cpo-structure for process algebras such as A_ω, A^∞. One could think of adding an element 0 as in the previous Remark, to function as the least element in a supposed partial order $\leq$ on A_ω, A^∞. Moreover, such a p.o. should respect substitution (i.o.w. be monotone in the operations).

However, a partial order on A_ω or A^∞ (extended with 0) with these properties:

$$0 \leq p$$

$$p \leq q \Rightarrow s(p) \leq s(q)$$

(where $s(\ )$ is some 'context'), does not exist.
For if there was such a p.o. $\leq$, then:

$$a a = a a + 0 \leq a a + a = a a + a 0 \leq a a + a a = a a .$$

Hence $a a = a a + a$, a contradiction.

2.2.14. **EX.** A language L over the alphabet A is a subset of A^*, the set of all finite words over A. Let $\mathbb{L}(A)$ be the collection of non-empty languages not containing the empty word. Define for $L_1, L_2 \in \mathbb{L}(A)$ the operations $+, \cdot, \between, \lfloor\!\lfloor$ as follows:

$$L_1 + L_2 = L_1 \cup L_2$$
$$L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$$
$$L_1 \between L_2 = \bigcup \{w_1 \between w_2 \mid w_1 \in L_1, w_2 \in L_2\}$$ where $w_1 \between w_2$ is the 'shuffle' of the words w_1, w_2 (e.g. $ab \between cd = \{abcd, acbd, cabd, cadb, cdab, acdb\}$)
$$L_1 \lfloor\!\lfloor L_2 = \bigcup \{w_1 \between w_2 \mid w_1 \in L_1, w_2 \in L_2\}$$ where $w_1 \between w_2$ is the subset of $w_1 \between w_2$ of words starting with a 'step' of w_1.

Then $\mathbb{L}(A)$ is a process algebra which satisfies in addition to PA the law

$$z(x+y) = zx + zy.$$

(In fact, $\mathbb{L}(A)$ is the initial algebra of $PA + \{z(x+y) = zx + zy\}$.)

* 2.2.15. **REMARK.** In the algebra $\mathbb{L}(A)$ of finite traces of EX. 2.2.14. there is a partial order $\leqslant$ with the properties

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

Namely: $x \leqslant y \Leftrightarrow x \subseteq y$.

However, in A_ω or A^∞ a p.o. with these mentioned properties does not exist. For, consider

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

Then $a(b+c) = a(b+c) + ab$, a contradiction.

* 2.2.16. **EX.** Show that one can add a constant $\bullet$ ("black hole") to PA together with the rules

$$x + \bullet = x\bullet = \bullet x = \bullet$$

and obtain a conservative extension $PA_\bullet$ (i.e. no more identities between expressions without $\bullet$ are proved by $PA_\bullet$ than by PA).

* 2.2.17. **Open problem.** Prove that the open theory of PA, i.e. the set of all equations $t(\vec{x}) = s(\vec{x})$ (possibly containing variables $\vec{x}$) provable by PA, is decidable.

* 2.2.18. **Open problem.** Prove that the theory of A^∞ (i.e. all equations $t = s$ between process expressions without variables such that $A^\infty \models t = s$) is decidable.

* 2.2.19. **Open problem.** Prove that "PA without $\between$"

is not finitely axiomatizable (in contrast to the finite axiomatization given by PA).

Here "PA without $\between$" is: PA without the axioms mentioning $\between$, and extended with axioms for $\parallel$ such as the identities in Fact 2.1.5 (p. 6).

More precisely: if $A_\omega(+, \cdot, \parallel)$ is the reduct of $A_\omega(+, \cdot, \parallel, \between)$ to show that the theory of $A_\omega(+, \cdot, \parallel)$, i.e. the set of true equations between closed terms, is not finitely axiomatizable.

* 2.2.20. **REMARK. Solving equations in A^∞.**

The theme of solving recursion equations and systems of recursion equations will be considered more systematically in Chapter V of these notes. There, we will impose on such (systems of) equations the condition that they are 'guarded'.

Here, we want to mention a theorem about solutions of recursion equations and systems of them in full generality – at the price of unicity of solutions.

2.2.20.1. **FACT.** Let $E_X =$

$$\begin{cases} X_1 = T_1(X_1, \dots, X_n) \\ \vdots \\ X_n = T_n(X_1, \dots, X_n) \end{cases}$$

be a system of equations for $X_1, \dots, X_n$ (without any condition on the terms T_i).

Then E_X has a solution $(P_1, \dots, P_n)$ in A^∞ (and also in each of the A_m, $m \geq 1$, and in $\mathbb{A}^\infty$).

2.2.20.2. **FACT.** Let $X = T(X)$ be a recursion equation for X. Then a solution for X, say P, can be obtained in A^∞ ($\mathbb{A}^\infty$) as the limit of the iteration sequence

$$Q, T(Q), T(T(Q)), \dots, T^n(Q), \dots$$

for arbitrary Q.

(Here $\lim_{k\to\infty} T^k(Q) = P$ means: $\forall n \, \exists m \, (T^m(Q))_n = (P)_n$.)

2.2.20.3. **EX.**

(i) Show that $X = p \,/\!/\, X$ has several (i.e. more than one) solutions in A^∞.

(ii) By contrast, the equation

$$X = p \between X$$

has a unique solution in A^∞ which we write as

$$p^{\overline{\overline{\omega}}},$$

the "ω-merge" of p. It is the limit of the iteration sequence

$$p,\ p \between p,\ p \between p \between p,\ \ldots,\ p^{\overline{\overline{n}}},\ \ldots.$$

(It is also the limit of the iteration sequence

$$p,\ p \between p,\ p \between p \between p,\ \ldots.$$

Note that we do not need brackets in $p \between p \between p \ldots \between p$, even though $\between$ is not associative.)

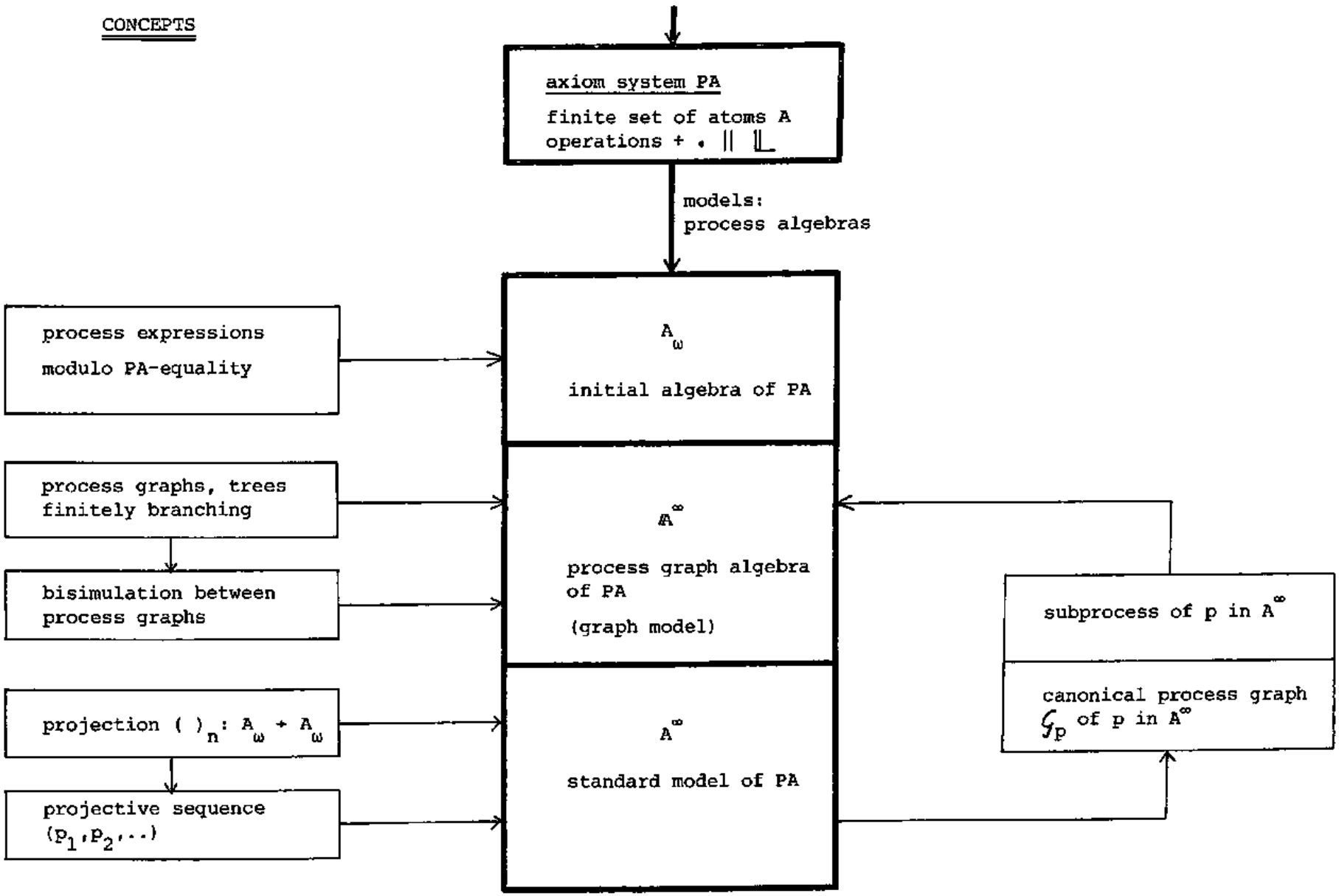
* 2.2.21. **EX.** How many processes are there, even over a singleton action alphabet $A = \{a\}$?

Show that the collection of such processes is a <u>proper class</u> in the sense of axiomatic set theory, by proving that if α, β are ordinals and $Tr(\alpha)$, $Tr(\beta)$ are their trees (as given by the $\in$-relation), then:

$$Tr(\alpha) \leftrightarrows Tr(\beta) \quad \Leftrightarrow \quad \alpha = \beta.$$

SUMMARY OF CHAPTER I, PROCESS ALGEBRA

CONCEPTS



SUMMARY OF CHAPTER I, PROCESS ALGEBRA

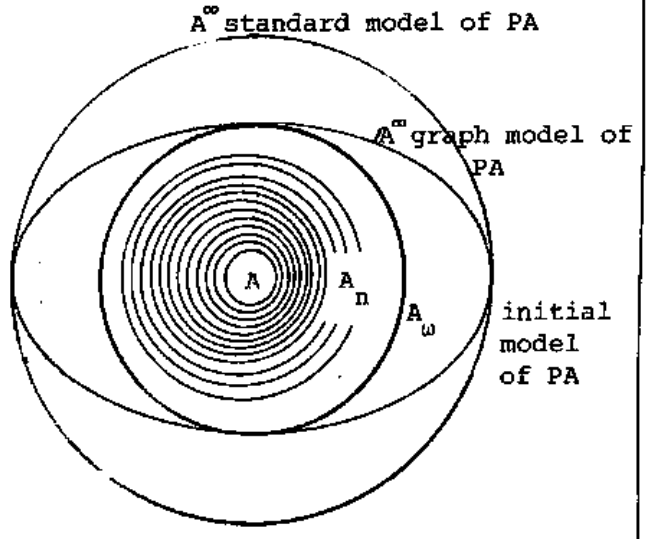
FACTS

eliminability of $\|, \between$ from PA-expressions
sum representation of elements of A_ω

$A_\omega, \mathcal{A}^\infty, A^\infty \not\models x(y+z) = xy+xz$
$A_\omega, \mathcal{A}^\infty, A^\infty \models$ $x\|y = y\|x$ $x\|(y\|z) = (x\|y)\|z$ $(x \between y) \between z = x \between (y\|z)$

$\mathcal{A}^\infty \models g = h \iff \forall n\ (g)_n = (h)_n$

modulo isomorphisms:



$A = A_1$

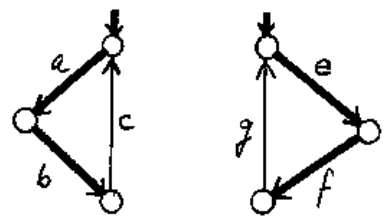
A_n: model of finite processes of depth at most n

II. PROCESS ALGEBRA WITH COMMUNICATION: ACP.

We will now extend the axiom-system PA of Chapter I with the facility of **communication** between processes. The communication will be modelled by **action sharing**. In PA all atomic actions were on equal footing, and capable of being performed without more. In **ACP, algebra of communicating processes**, we will introduce next to this kind of independent or autonomous actions, so-called **subatomic** actions which need one or more other subatomic actions in order to be executed. (Cf. the subatomic actions $C!t$ and $C?x$ in Hoare's CSP, whose simultaneous execution amounts to the assignment $x := t$.) The execution is then an 'ordinary' atomic action.

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

To get an idea, consider the following processes p and q.

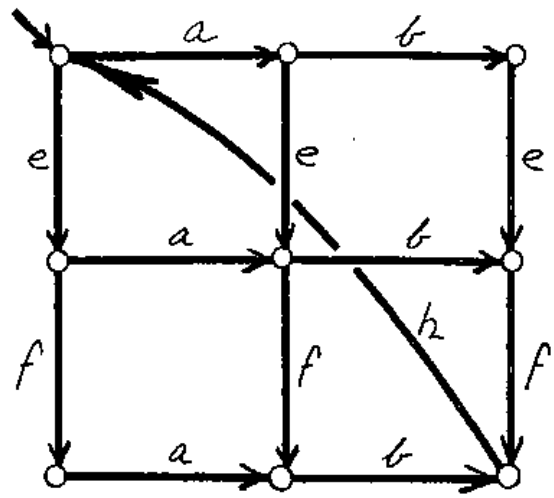


$$p = (abc)^\omega \qquad q = (efg)^\omega.$$

The heavy lines denote atomic actions, the steps c and g are subatomic actions and need each other to perform the action h, notation:

$$c \mid g = h.$$

Then the process r resulting from the coöperation of p and q would be



$$p \between q = r = \big([a(e(bf+fb)+bef)+e(g(bf+fb)+fab)]\big)$$

The axiom system ACP (on the next page) gives the means to compute the results of such communicating processes in an algebraic way.

ACP is an extension of PA, but not strictly in the sense that axioms are added; one axiom from PA (viz. M1) is adapted:

$x \between y$ is in ACP a sum of <u>three</u> terms, namely

$x \between\!\!\!\between y$, $y \between\!\!\!\between x$ and the new summand $x \mid y$.

Here $x \between\!\!\!\between y$ is, as in PA, "like $x \between y$" but taking its first step from x; likewise $y \between\!\!\!\between x$; and $x \mid y$ is like $x \between y$ but requires the first action to be the result of a communication (between a first subatomic 'step' of x and a first subatomic step of y).

This new operator '$\mid$' is called **communication merge**; on the set A of atoms and subatomic actions it is a binary function, **the communication function**, which is given a priori. It is commutative and associative. The precise choice of the communication functio

$x + y = y + x$	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
$a\|b = b\|a$	C1
$(a\|b)\|c = a\|(b\|c)$	C2
$\delta\|a = \delta$	C3
$x\|\|y = x\between y + y\between x + x\|y$	CM1
$a\between x = a\cdot x$	CM2
$(ax)\between y = a(x\|\|y)$	CM3
$(x + y)\between z = x\between z + y\between z$	CM4
$(ax)\|b = (a\|b)\cdot x$	CM5
$a\|(bx) = (a\|b)\cdot x$	CM6
$(ax)\|(by) = (a\|b)\cdot(x\|\|y)$	CM7
$(x + y)\|z = x\|z + y\|z$	CM8
$x\|(y + z) = x\|y + x\|z$	CM9
$\partial_H(a) = a \quad$ if $a \notin H$	D1
$\partial_H(a) = \delta \quad$ 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

Axioms of ACP

varies with the application of ACP which one has in mind - just as the choice of the alphabet A. Thus ACP is in fact parametrized by A and by the communication function

$$| : A \times A \to A .$$

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

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

The first is a constant δ, which is a 'zero' for + and moreover satisfies

$$\delta x = \delta \qquad (A7)$$

It stands for **deadlock** or **failure**. It may be debatable what of these two it stands for exactly - however we want to short-circuit this debate since the rationale of δ is not so much its semantics as some particular phenomenon in the reality of concurrent systems, as its <u>algebraic function</u>. Indeed, δ will turn out to give a smooth theory and is instrumental in our axiomatization of concurrency.

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

Summarizing, we have the following signature for ACP:

$x+y$	alternative composition (sum)
$x \cdot y$	sequential composition (product)
$x \parallel y$	parallel composition (merge)
$x \between y$	left merge
$x \mid y$	communication merge
$\mid : A \times A \to A$	communication function
$\partial_H(x)$	encapsulation
δ	deadlock

* 1.1. **REMARK**. In the following sense ACP is an extension of PA:
Let the communication function be trivial, i.e.

$$a \mid b = \delta$$

for all $a, b \in A$. Then the models A_ω, $\mathbb{A}^\infty$, A^∞ for PA. (with signature $+, \cdot, \parallel, \between$) are just <u>reducts</u> of the models A_ω, $\mathbb{A}^\infty$, A^∞ for ACP which we will construct below and which have signature $+, \cdot, \parallel, \between, \mid, \partial_H, \delta$.

2. Process algebras for ACP.

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

A_ω,	the initial algebra of ACP
$\mathbb{A}^\infty$,	the process graph algebra of ACP
A^∞,	the standard model of ACP.

Here some confusion may arise as to which signature, that of PA or that of ACP, is meant when speaking about A_ω, A^∞, A^∞. This confusion is solved
(a) by the context,
(b) by mentioning the intended signature explicitly, as in

$$A_\omega(+,\cdot,\between,\lfloor\!\lfloor) \quad \text{vs.} \quad A_\omega(+,\cdot,\between,\lfloor\!\lfloor,|,\partial_H,\delta).$$

2.1. The initial algebra A_ω of ACP.

Before building A_ω, we have fixed the alphabet A, a communication function $|: A\times A \to A$, and a subset $H \subseteq A$ (hence an encapsulation operator ∂_H).

Now A_ω contains as elements: the closed process expressions (in the signature of ACP) modulo the equality given by ACP. By the following

2.1.1. FACT. (Normal form theorem)

For each closed term t there is a closed term t' not containing $\between$, $\lfloor\!\lfloor$, $|$, ∂_H such that

$$ACP \vdash t = t'.$$

we may think of elements of A_ω as built from A, $+$, $\cdot$ only. (just as in the case $A_\omega(+,\cdot,\between,\lfloor\!\lfloor)$), or as the finite process trees which we encountered in Ch. I.

2.1.2. EXAMPLE.

Let $A = \{a, b, c, c^0, d, \delta\}$.

Let $|: A\times A \to A$ be given by $c|c = c^0$, and all other communications equal δ (thus $a|b = c|c^0 = d|a = \dots = \delta$).

Further, let $H = \{c\}$. Then:

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

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

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

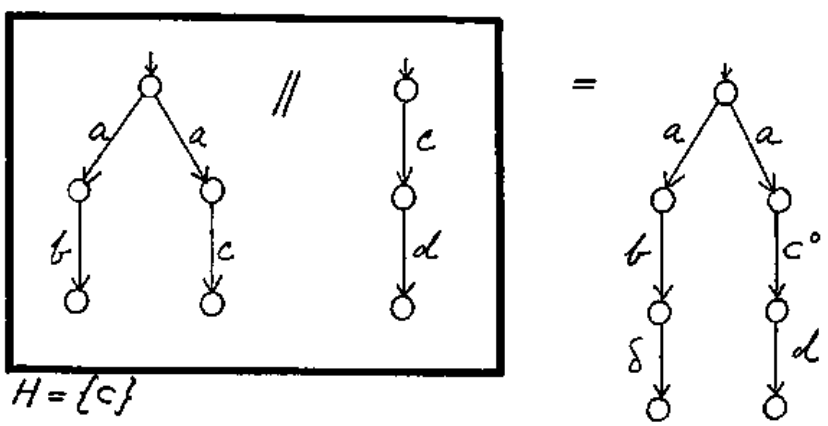
$\partial_{\{c\}}$ [a(bcd + c(d||b) + (b|c)d) + a(ccd+c(d||c)+(c|c)d) + c(d|| (ab+ac)) +

+ δ(d||b) + δ(d||c)] =

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

abδ + ac°d.

Once again, in a figure:



2.1.3. **EXAMPLE.** Let A and '|' be as in the previous Example. We want to give another example clarifying the role of ∂, the encapsulation operator.

Consider $c \,//\, c$.

(i) If $c \,//\, c$ is seen as a 'total' process that not needs to communicate further, i.e. if $c \,//\, c$ is in fact $\partial_{\{c\}}(c \,//\, c)$, the result is c°.

(Namely: $\partial_{\{c\}}(c \,//\, c) = \partial_{\{c\}}(c \lfloor\!\lfloor c + c \lfloor\!\lfloor c + c|c)$,
$\partial_{\{c\}}(cc + c^\circ) = \partial_{\{c\}}(c) \cdot \partial_{\{c\}}(c) + \partial_{\{c\}}(c^\circ) =$
$\delta\delta + c^\circ = c^\circ$.)

(ii) On the other hand, $c \,//\, c = cc + c^\circ$, and this process can communicate further; e.g. $(c \,//\, c) \,//\, c$ evaluates to something which after applying $\partial_{\{c\}}$ yields $c^\circ\delta$.

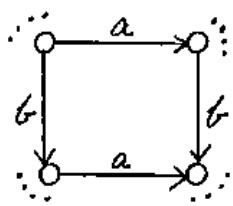
2.2. The process graph algebra $\mathbb{A}^\infty$ for ACP.

The definition of $\mathbb{A}^\infty(+, \cdot, \between, \llcorner, /, \partial_H, \delta)$ parallels that of $\mathbb{A}^\infty(+, \cdot, \between, \llcorner)$ for PA. In addition to the latter definition we must define the result of the operations $\between$, $\llcorner$, $/$ and ∂_H. (Also $\between$, $\llcorner$, because these are in ACP specified by different axioms than in PA.)

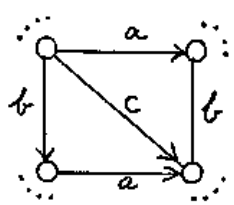
$g \between h$

If g, h are finitely branching process graphs then $g \between h$ is now the **cartesian product graph enriched with 'diagonal' edges** $\xrightarrow{a/b}$ in the following situation:

if



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



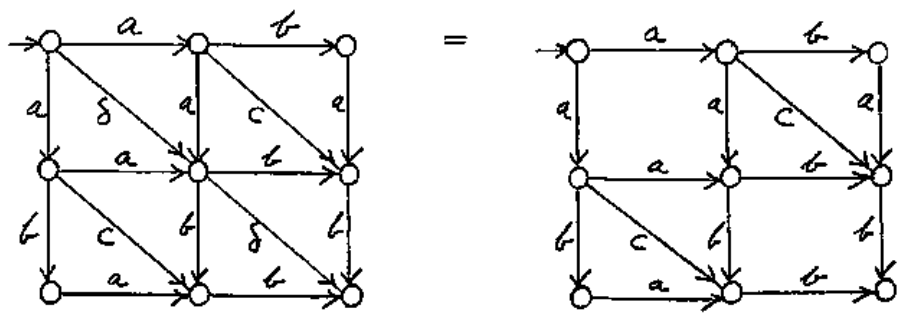
$g \llcorner h$, g/h yield results which can now be guessed. An example will suffice:

Let $A = \{a, b, c, \delta\}$, $a/b = c$ and all other communications equal δ.

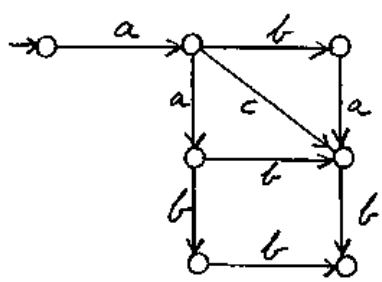
(So we have a 'communication table' like a/b is the only '*proper*' communication.

	a	b	c	δ
a	δ	c	δ	δ
b	c	δ	δ	δ
c	δ	δ	δ	δ
δ	δ	δ	δ	δ

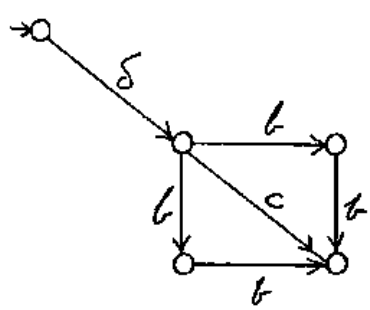
Then: $ab \parallel ab =$



$ab \between ab =$



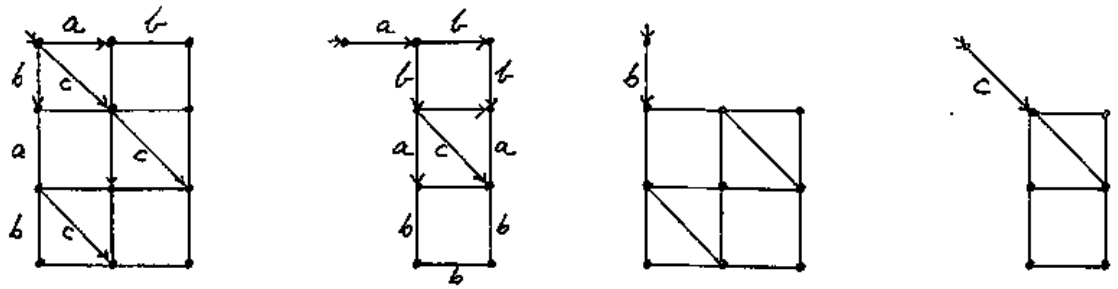
$ab / ab =$



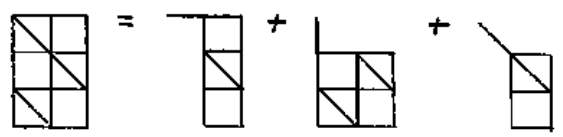
$= \delta$.

Another example :

$ab \parallel bab$ $\quad ab \between bab$ $\quad bab \between ab$ $\quad ab \mid bab.$



Note that indeed (modulo bisimulation):



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

2.2.1. **REMARK.** One thing was swept under the rug. The elements of A^∞ are finitely branching process graphs, modulo bisimulation as defined in Chapter I. However there the constant δ had not yet made its entrance.

So what is a bisimulation between graphs where some edges may be labelled with δ? The old concept is insufficient now since it would not satisfy the laws $x + \delta = x$ and $\delta x = \delta$.

We will choose the following solution:
Let g be a graph. Then the **δ-normal form** of g is the graph g' obtained by deleting all δ-steps which have a 'brother' step and creating for the remaining δ-steps if necessary separate end nodes. Afterwards disconnected pieces of the graph are removed.

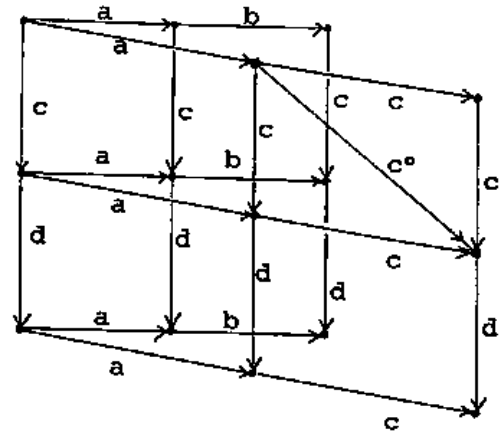
Now g and h are bisimilar if their δ-normal forms are bisimilar in the old sense.

The effect of these definitions is :

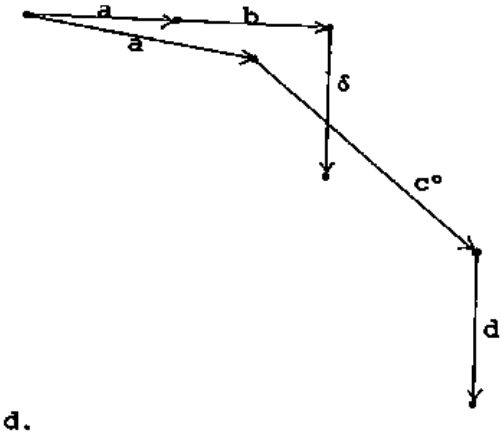
2.2.2. **FACT.** $A^{\infty} \models ACP$.

Using these graphs, we have an easy way to 'compute' the result of our earlier Example 2.1.2 (p.6) :

$(ab+ac) \between cd$ =



And now, after applying $\partial_{\{c\}}$, the remains are:



that is, $ab\delta + ac^{\circ}d$.

2.3. The standard model A^{∞} for ACP.

The construction of A^{∞} is entirely analogous to that for PA.

2.3.1. **EXAMPLE** of a computation in $A^{\infty}(+, \cdot, \between, \lfloor\!\lfloor, \lfloor, \partial_H, \delta$

Let $A = \{a, b, c, d, \delta\}$, $a/a = d$ the only proper communication.

Now let $p = (a, ab, aba, abab, \ldots)$ and
$q = (a, ac, aca, acac, \ldots)$

What is $\partial_{\{a\}}(p \| q)$? It is:

$\partial_{\{a\}}((a\|a)_1, (ab\|ac)_2, (aba\|aca)_3, \ldots)) =$

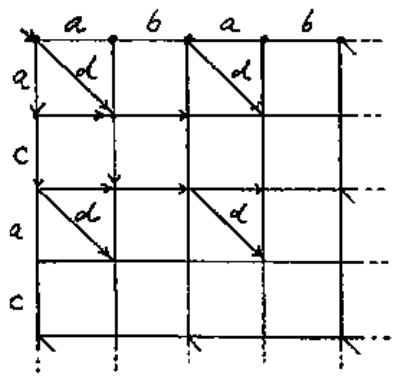
$(\partial_{\{a\}}(a\|a)_1, \partial_{\{a\}}(ab\|ac)_2, \ldots) =$

$(\partial_{\{a\}}(aa+d)_1, \partial_{\{a\}}(\ldots), \ldots) =$

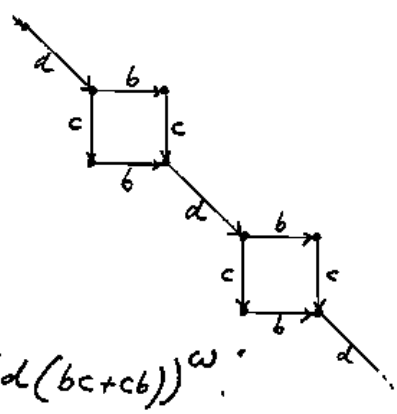
$(d, d(b+c), d(bc+cb), d(bc+cb)d, \ldots)$

The same in A^∞:

$\partial_{\{a\}}$



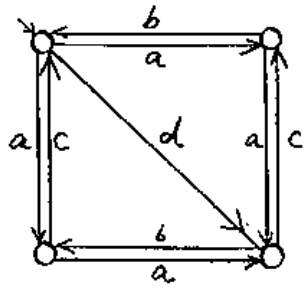
=



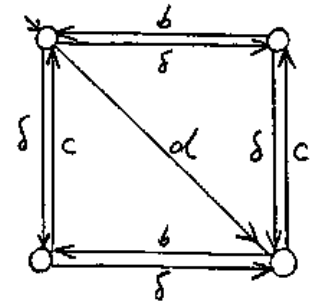
$= "(d(bc+cb))^\omega."$

Alternatively:

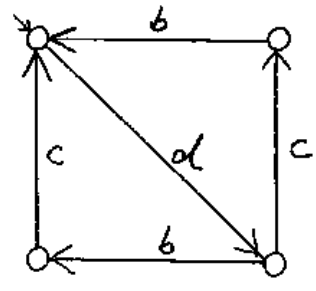
$\partial_{\{a\}}$



=



=



$= "(d(bc+cb))^\omega."$

2.4. Process algebras with standard concurrency and handshaking.

A useful intuition about communicating processes is to postulate that $\|$ is commutative and associative. This does not follow from the axioms of ACP; pathological process algebras with noncommutative and nonassociative $\|$ are possible. But in the process algebras A_ω, A^∞ and A^∞, $\|$ is indeed commutative and associative.

In fact, these algebras satisfy the following

axioms of standard concurrency:

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

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

Moreover, matters are greatly simplified by adopting the

handshaking axiom:

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

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

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

2.4.1. **FACT (Expansion Theorem)**

$$x_1 \parallel \dots \parallel x_k = \sum_i x_i \mathbin{\mathbb{L}} X_k^i + \sum_{i \neq j} (x_i / x_j) \mathbin{\mathbb{L}} X_k^{i,j}$$

Here X_k^i is obtained by merging $x_1, \dots, x_k$ except x_i, and $X_k^{i,j}$ is obtained by merging $x_1, \dots, x_k$ except x_i, x_j. $(k \geq 3)$.

2.4.1.1. **EXAMPLE.** We prove the Expansion Theorem for $k=3$:

$$x_1 \parallel x_2 \parallel x_3 =$$

$$(x_1 \parallel x_2) \parallel x_3 = (x_1 \parallel x_2) \mathbin{\mathbb{L}} x_3 + \underline{x_3 \mathbin{\mathbb{L}} (x_1 \parallel x_2)} + (x_1 \parallel x_2) / x_3$$

$$(x_1 \mathbin{\mathbb{L}} x_2 + x_2 \mathbin{\mathbb{L}} x_1 + x_1 / x_2) \mathbin{\mathbb{L}} x_3 + \underline{x_3 \mathbin{\mathbb{L}} (x_1 \parallel x_2)} + (x_1 \parallel x_2) / x_3$$

$$(x_1 \mathbin{\mathbb{L}} x_2) \mathbin{\mathbb{L}} x_3 + (x_2 \mathbin{\mathbb{L}} x_1) \mathbin{\mathbb{L}} x_3 + \underline{(x_1 / x_2) \mathbin{\mathbb{L}} x_3} + \underline{\quad} + \underline{\quad} =$$

$$\underline{x_1 \mathbin{\mathbb{L}} (x_2 \parallel x_3)} + \underline{x_2 \mathbin{\mathbb{L}} (x_1 \parallel x_3)} + \underline{(x_1 / x_2) \mathbin{\mathbb{L}} x_3} + \underline{x_3 \mathbin{\mathbb{L}} (x_1 \parallel x_2)} +$$

$$(x_1 \mathbin{\mathbb{L}} x_2 + x_2 \mathbin{\mathbb{L}} x_1 + x_1 / x_2) / x_3 =$$

$$- + - + - + - + \quad x_3 / (x_1 \mathbin{\mathbb{L}} x_2) + x_3 / (x_2 \mathbin{\mathbb{L}} x_1) + \underbrace{x_1 / x_2 / x_3}_{=\delta}$$

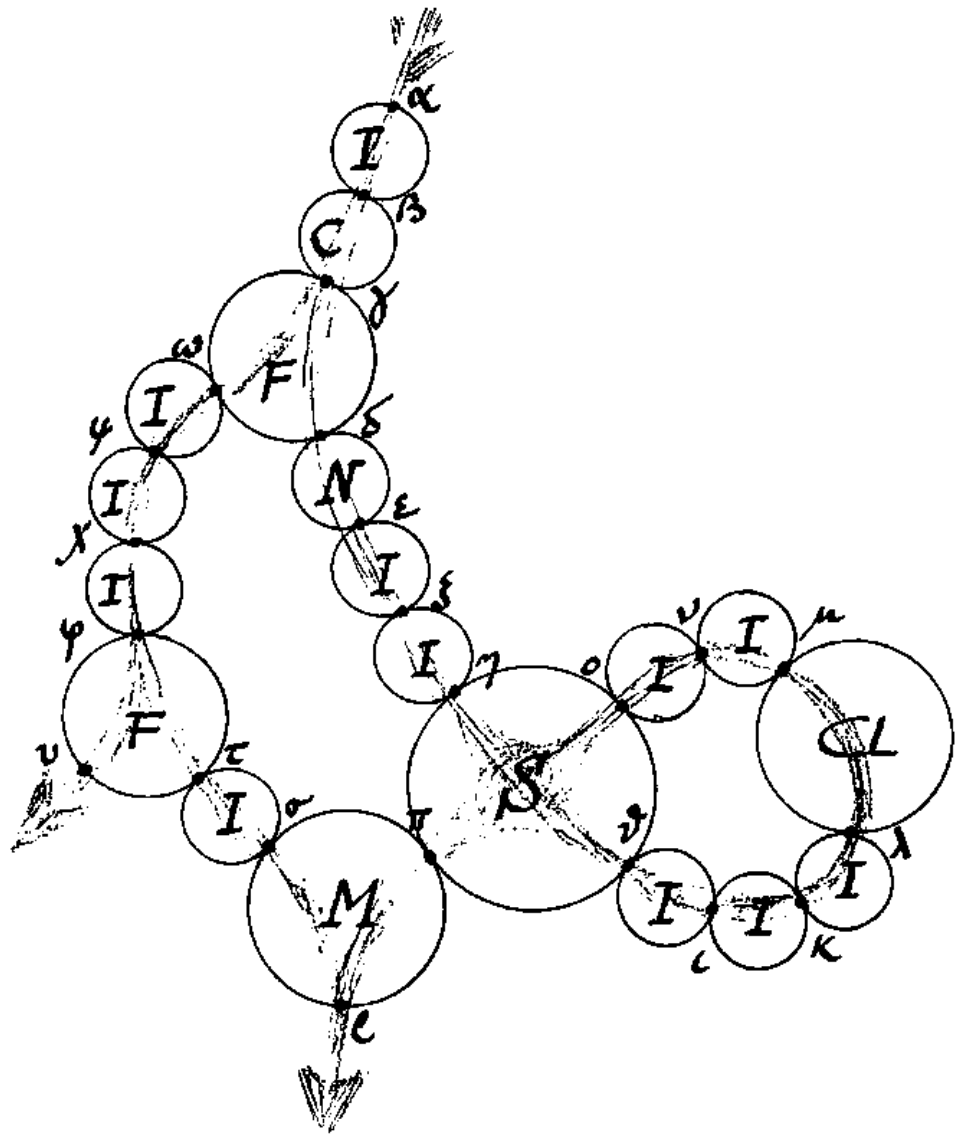
$$= x_1 \mathbin{\mathbb{L}} (x_2 \parallel x_3) + x_2 \mathbin{\mathbb{L}} (x_1 \parallel x_3) + x_3 \mathbin{\mathbb{L}} (x_1 \parallel x_2) +$$

$$+ (x_2 / x_3) \mathbin{\mathbb{L}} x_1 + (x_3 / x_1) \mathbin{\mathbb{L}} x_2 + (x_1 / x_2) \mathbin{\mathbb{L}} x_3 .$$

2.5. Networks of processes communicating by handshaking.

As a first step towards giving a process algebra for the operational semantics of data flow networks, we will consider networks of processes.

Consider the following network.



Through these 'nodes' $I_{\alpha\beta}$, $C_{\beta\gamma}$, $F_{\gamma\omega\delta}$, ... we imagine a flow of data; for simplicity we work in this example with the data domain $\{0, 1\}$.

The 'nodes' are, intuitively, devices having **ports** $\alpha, \beta, \gamma, \ldots$. What happens inside the nodes with the data is invisible, but at the ports we can 'see' the **event of passing a datum**. (In fact, the word 'passing' is misleading since it suggests a <u>direction</u> of flow

which, interestingly, disappears at the level of our analysis of data flow.)

The event of passing a 0 at port α will be denoted by α_0, passing a 1 is α_1, etc. So a part of the "alphabet" (the collection of events or actions) which is relevant for the network example above, is:

$$\{\alpha_0, \alpha_1, \beta_0, \beta_1, \ldots, \omega_0, \omega_1\} = X$$

Now let us specify how the nodes $I_{\alpha,\beta}$, $C_{\beta\gamma}$, ... function. These are conceived as processes over the action set X, more precise: as elements of the process algebra X^∞.

In the example we have chosen:

$$I_{\alpha\beta} = (\alpha_0\beta_0 + \alpha_1\beta_1) I_{\alpha\beta} \quad \text{(Likewise } I_{\varepsilon\xi} \text{ etc.)}$$

This means that $I_{\alpha\beta}$ is the **Identity** 'processor' between α and β: it relays 0, 1 unchanged and restores itself thereafter.

$$C_{\beta\gamma} = (\beta_0\gamma_0\gamma_0 + \beta_1\gamma_1\gamma_1) C_{\beta\gamma}$$

$C_{\beta\gamma}$ is the **Copy** processor which transforms an incoming 0 in an outgoing 00, likewise 1 is output as 11.

$$F_{\gamma\omega\delta} = (\gamma_0(\omega_0 \parallel \delta_0) + \gamma_1(\omega_1 \parallel \delta_1)) F_{\gamma\omega\delta}$$

$F_{\gamma\omega\delta}$ is the **Fan-out** processor: a 0 incoming at γ is relayed to both ω an δ, in either order (i.e. $\omega_0 \parallel \delta_0$, or $\omega_0\delta_0 + \delta_0\omega_0$).

$$N_{\delta\varepsilon} = (\delta_0\varepsilon_1 + \delta_1\varepsilon_0) N_{\delta\varepsilon}$$

$N_{\delta\varepsilon}$ is **Negation**.

$$CL_{\lambda\mu} = \left(\sum_{i,j} \lambda_i \lambda_j \mu_i \mu_j\right) CL_{\lambda\mu} .$$

$CL_{\lambda\mu}$ is the processor **Clustering** which waits till a sequence of two data was 'received' before it outputs the same sequence, and restores itself.

$$M_{\sigma\pi\varrho} = \sum_i (\sigma_i + \pi_i)\varrho_i M_{\sigma\pi\varrho} .$$

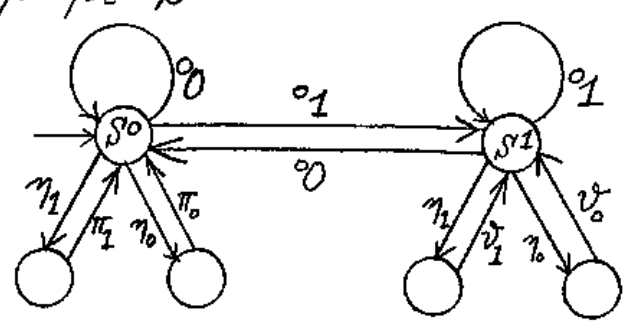
$M_{\sigma\pi\varrho}$ is the processor **Merge** which relays the signals in order of entrance at σ or π.

The remaining node S' cannot be described in one recursion equation (or can it?) It is a **Switch** and functions as follows:

the incoming 0's and 1's at η are relayed to π or ϑ, depending from the input at o. Namely: S' transports the i incoming at η to π as long as the last value observed along o was 0 and turns to ϑ if a value 1 is observed at o; after the next 0 at o the switch is back in its original position. There are two states: S'^0 and S'^1.

$$\begin{cases} S'^0 = \sum_i (\eta_i \cdot \pi_i \cdot S'^0 + o_0 \cdot S'^0 + o_1 \cdot S'^1) \\ S'^1 = \sum_i (\eta_i \cdot \vartheta_i \cdot S'^1 + o_0 \cdot S'^0 + o_1 \cdot S'^1) \end{cases}$$

That these recursion equations define processes in A^∞ (or in A^∞) for $A = \{\alpha_0, \ldots, \omega_1\}$ is not hard to see; it is a simple matter to associate process graphs to them. E.g. for S'^0:



Now each node in the example network is defined; and it is 'somehow clear' that this network has an intuitive semantics. That is, one can envisage the existence of some process (though not the madly complicated process itself) which governs the flow of the data 0,1 through the network. The question is what this process (the operational semantics of the network) is exactly. Before we can answer this question, we have to do something more. What we have not yet specified, is how the nodes $I_{\alpha\beta}$, $C_{\beta\gamma}$, ... communicate. The simplest way to arrange the "pattern of communication" is as follows.

For **all the ports except the 'external ones' α, ν, ϱ** we form **subatomic events** or **communication events**, by priming the events $\beta_0, \beta_1, \gamma_0, \gamma_1, \ldots$
So we have the set Y of

Communication events:
$$Y = \{\beta_0', \beta_1', \gamma_0', \gamma_1', \ldots\}$$

Further we stipulate that

the only proper communications are:
$$a'/a' = a \quad \text{for all } a' \in Y.$$

Thus $\beta_0'/\beta_0' = \beta_0$, $\beta_1'/\beta_1' = \beta_1$, etc.

Next we attach primes to the internal port names $\beta, \gamma, \ldots$ in the definitions above of $I_{\alpha\beta}$, $C_{\beta\gamma}$, ...
Let A be the alphabet $X \cup Y$.
Then we can assign as the operational semantics of the example network: the process in A^∞ (or $\mathbb{A}^\infty$)

$$\partial_Y (I_{\alpha\beta} \| C_{\beta\gamma} \| F_{\gamma\mu\delta} \| N_{\delta\varepsilon} \| \ldots \| S^0 \| M_{\sigma\pi\varrho}).$$

And this process is hereby precisely defined and can in principle be computed to any depth. (In fact, for this example, it is given by a finite process graph.)

2.5.1. **EX.** Consider the network

where $p = I_{\alpha\beta} = \sum_i \alpha_i \beta_i' I_{\alpha\beta}$ $(i=0,1)$

$q = I_{\beta\gamma} = \sum_i \beta_i' \gamma_i I_{\beta\gamma}$ $(i=0,1)$

and the only proper communication is $\beta_i' | \beta_i' = \beta_i$.
Then one computes for $r = \partial_{\{\beta_0', \beta_1'\}}(p \| q)$:

$$\begin{cases} r = \sum_i \alpha_i \beta_i r_i & \text{where} \\ r_i = \gamma_i r + \sum_j \alpha_j \gamma_i \beta_j r_i . \end{cases}$$

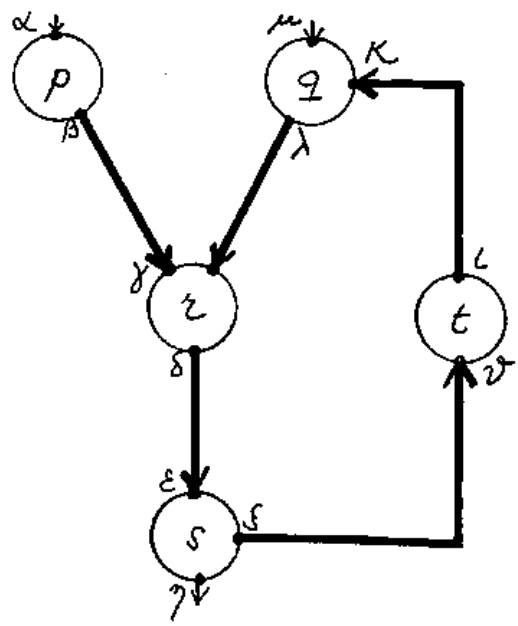
Note that r is a <u>buffer with capacity 2</u>.
Strictly speaking, it is a "transparant" buffer, since one can see the transition γ_i of the datum i. This transition step is an "internal" step. A theory for hiding internal steps can be given which abstracts from "most" internal steps — but in general not from all internal steps. In these notes we will not cover this theory; we refer to: BERGSTRA - KLOP, "An abstraction mechanism for process algebras", Mathematical Centre Report IW 231/83.
Even though we will not treat abstraction theory here, the following point seems worth making. This point is that **encapsulation** and **hiding internal steps** are very different operations. A pictorial way to express this difference is as follows.

- **Encapsulation** $\partial_H(p)$ is like putting p in a <u>glass box</u>. The encapsulated process cannot communicate with the environment; but the internal steps in p are all visible.

- **Hiding internal steps** in a process q is like putting an 'almost' black box around q. Only the external events and possibly some internal steps are now visible. (With the abstraction mechanisms now around it is in general not possible to hide <u>all</u> internal steps.)

2.6. Pure data flow.

We will now extend the considerations in 2.5 about networks of processes which communicate by hand shaking, to data flow networks where the 'contact' between the nodes is not so immediately, but where nodes are connected via **channels**; cf. the heavy lines in the following figure:



An important observation is that e.g. β and γ denote <u>different</u> ports, since, although passing the value d at β will eventually evoke that d is passed at γ as well, there may be a long temporal interval in between these events β_d and γ_d. In fact, the semantics of "$\beta \longrightarrow \gamma$" is not obvious. There are several possibilities. Probably the first

possibility which comes to mind, is that the channel ⟶ is a **queue** where the first incoming messages are the first to be output; here the order of the messages (data) is preserved.

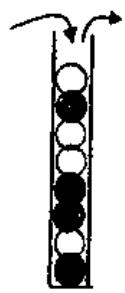
Another possibility is that the channel ⟶ is a **bag**. It is like queue but the order of the messages is lost during the passage through the channel – messages may 'overtake' each other.

A third type of channel is the **stack**. Further, a channel may be a **bounded queue, bag, stack.**

queue

bag

stack



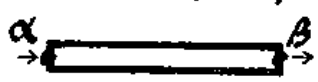
Now an important realization is that channels and nodes are in fact the same type of entities, viz. processes. Indeed, certain channels, 'buffers with bounded capacity,' were already encountered in Section 2.5 as sequences of Ⓘ.

So we could end this Section with giving recursion equations for the processes alias channels Queue, bag, Stack and referring to the previous Section 2.5.

The problem, however, is that these recursion equation

are more complicated than those for the simple nodes in the example network of section 2.5. In all cases these nodes could be given as finite process graphs (such as the one for Switch on p. 44) This is not so for QUEUE, BAG and STACK. Yet two of these processes can be described by recursion equations over ACP.:

BAG $\mathcal{B}$ with data domain $\{0,1\}$, input port α, output port β



$$\mathcal{B} = \sum_{i \in \{0,1\}} \alpha_i (\beta_i \parallel \mathcal{B})$$

STACK $\mathcal{S}$ with data domain $\{0,1\}$ input port α, output port β

$$\begin{cases} \mathcal{S} = \sum_{i \in \{0,1\}} \alpha_i \cdot T_i \cdot \mathcal{S} \\ T_i = \beta_i + \sum_{j \in \{0,1\}} \alpha_j \cdot T_j \cdot T_i \end{cases}$$

We will discuss these equations in the next Chapter.

With the recognition that channels are processes too, and with the recursive definition as processes of some of the main types of channels, the problem of assigning a process to a data flow network as its operational semantics is reduced to the same problem

as we discussed in Section 2.5.

* 2.6.1. For **QUEUE** the situation is essentially more complicated. If one admits <u>infinite systems of equations</u>, QUEUE can be defined in ACP as the first component of the solution of such an infinite system of equations.

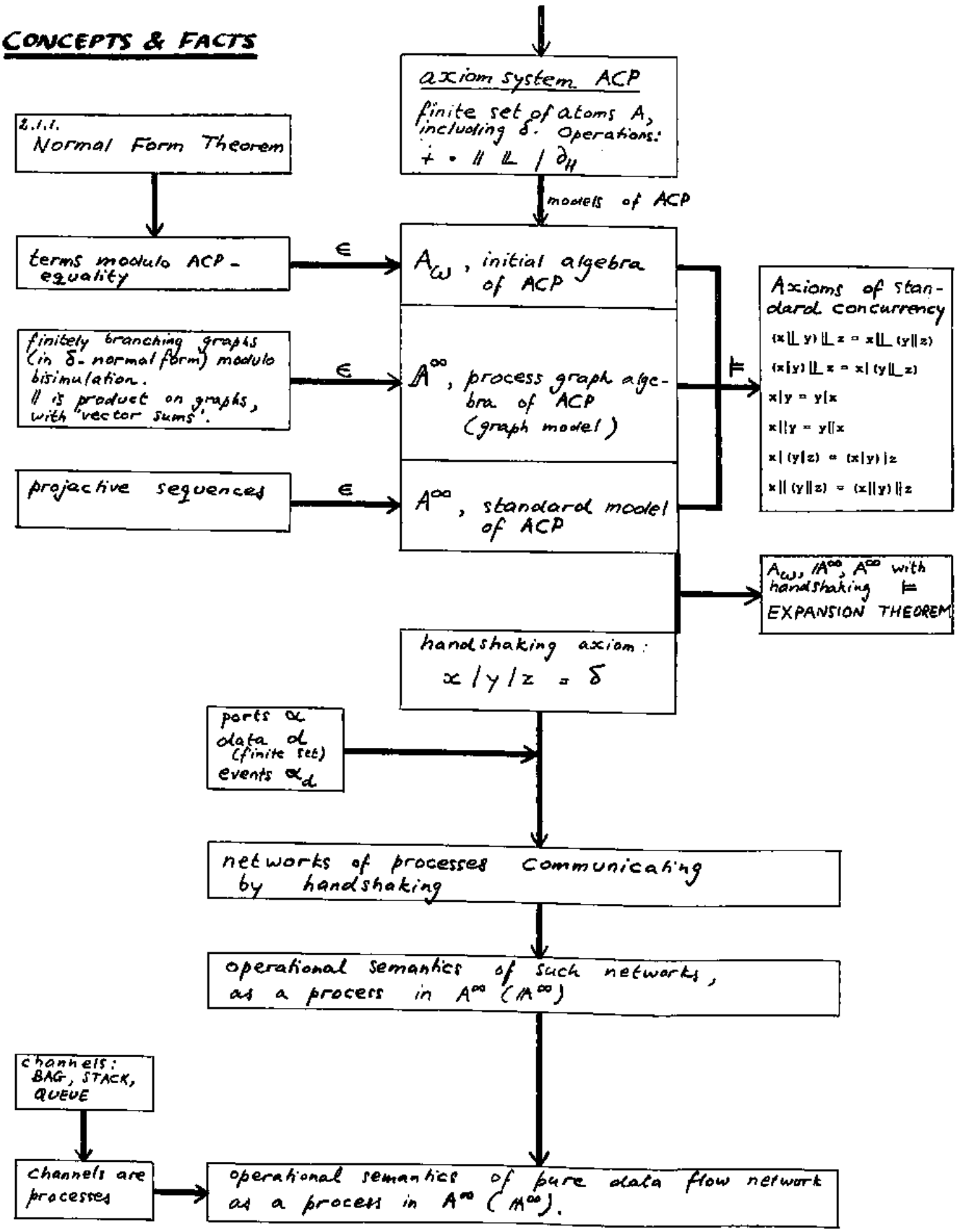
One can prove (BERGSTRA - TIURYN , "Process algebra semantics for queues", Math. Centre Report IW 2.... /83) that **QUEUE cannot be defined recursively by means of a finite system of recursive equations over ACP.**

If one allows extensions of the ACP formalism, there are two ways of specifying QUEUE. The first method is via **auxiliary operators** $\wedge$ and $\triangle$ that can be defined by finitely many equations. Then QUEUE can be recursively defined over the ACP-signature extended with $\wedge$, $\triangle$. This was done in BERGSTRA - KLOP, "A process algebra for the operational semantics of static data flow networks", Math. Centre Report IW 222/83.

The second method is via the "**π - construction**" which uses **process graphs defined by means of abstract data types.** See BERGSTRA - KLOP "An algebraic specification method for processes over a finite action set", Math. Centre Report IW 232/83.

SUMMARY OF CHAPTER II, PROCESS ALGEBRA WITH COMMUNICATION

CONCEPTS & FACTS



III. RECURSIVELY DEFINED PROCESSES.

In the previous chapters we have used, occasionally, some processes which were defined as the solutions of recursion equations.

Examples: (i) $X = p.X$; this process X can be 'developed' to $p.p.p.\ldots$, notation p^ω.
(ii) $X = p \between X$; this is the process $p \| p \| p \| \ldots$, notation $p^{\overline{\omega}}$.
(iii) **Switch**, on p. 44, the solution of a system of two recursion equations.

We will now look more closely and systematically at this 'specification' method for processes. This will give us some criteria as to which processes in A^∞ can be defined recursively. It will also give us some other process algebras.

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

1. Linear terms and guarded terms.

Let $X_1,\ldots,X_n$ be variables ranging over processes. Given a (restricted) signature of operators from $+, \cdot, \|, \between, |, \partial_H, \delta$ two kinds of terms containing variables $X_1,\ldots,X_n$ are of particular importance:

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

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

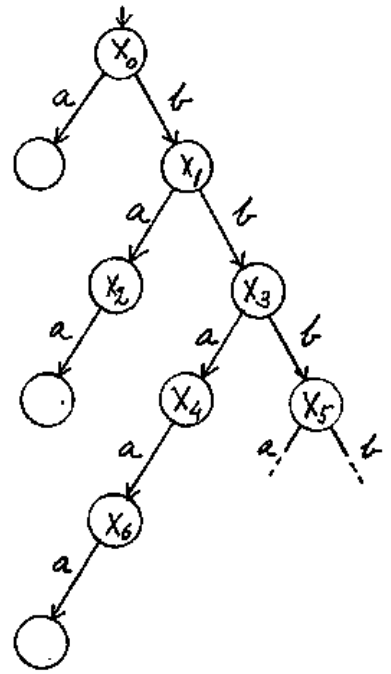
(ii) *Guarded terms*. The *unguarded* terms are inductively defined as follows:
- X_i is unguarded,
- if T is unguarded then so are $T + T'$, $T \cdot T'$, $\partial_H(T)$, $T \| T'$, $T \between T'$, $T | T'$ (for every T').

A term T is guarded if it is not unguarded.

1.1. Note that we introduced "formal" variables $X_1, \ldots, X_n$; they are meant as the 'unknowns' in recursion equations. They should not be confused with the metavariables $x_1, \ldots, x_n$ which occur in the axioms of PA and ACP.

2. Consider an element from A^∞, i.e. a finitely branching process graph modulo bisimulation. Let g be a representation of this element:

$g =$



Then g gives rise to an infinite system E of equations:

$$E \begin{cases} X_0 = a + bX_1 \\ X_1 = aX_2 + bX_3 \\ X_2 = a \\ \vdots \end{cases}$$

and since all the RHS's of these equations are guarded, it is easy to see that this infinite system E determines a unique solution $\underline{X}_0$ in A^∞. In fact, it is in this sense that A^ω is a subalgebra of A^∞, as was already asserted in the Summary of Chapter I, p. 27.

Note also that E contains only linear equations.

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

* 2.1. **Open question.** From a theoretical point of view it is interesting to consider also unguarded E. For PA one can prove the existence of solutions : see Ch. 1, p. 24.
Does this existence theorem generalize to ACP ?

3. Regular processes.

We remember that an element $\in A^\infty$ has a canonical process graph, with the subprocesses as nonterminal nodes (and '0' as terminal node).
Now we define:

> $p \in A^\infty$ is regular
> $\Updownarrow$ def.
> Sub(p) is finite

Further: $r(A^\infty)$ is the collection of the regular processes in A^∞.

This gives us immediately the following

3.1. **FACT.** (i) *If* p *is regular then there is a finite process graph* g *with* $[\![g]\!] = p$, *and conversely.*
(ii) *The class of regular processes is closed under the operations* $+, \cdot, \|, \between, \lfloor, \partial_H$. *Hence* $r(A^\infty)$ *is a subalgebra of* A^∞.
(iii) $r(A^\infty)$ *contains exactly the solutions of finite systems of guarded linear equations.*

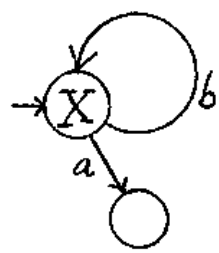
Here $\|g\|$ is the process in A^∞ associated (as explained above) to the process graph g.
Let us state the situation once more:

p is regular
⇕
$Sub(p)$ is finite
⇕
G_p is finite
⇕
p is the first component of the solution vector of a finite, guarded, linear system of equations.

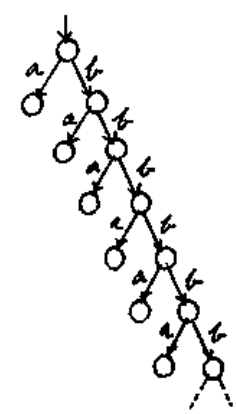
Moreover, we have a new process algebra, whose position relative to the previous ones is as follows:

$$A = A_1 \subseteq A_2 \subseteq \cdots \subseteq A_\omega \subseteq \mathcal{Z}(A^\infty) \subseteq \mathbb{A}^\infty \subseteq A^\infty.$$

3.2. **EXAMPLE**. (i) $X = a + bX$.
Let $\underline{X}$ be the solution. Then $\underline{X}$ has the canonical process graph $G_{\underline{X}}$:



Expanded in tree representation:
Note that $Sub(p) = \{p\}$.
p as a projective sequence is:
$(a+b,\ a+b(a+b),\ a+b(a+b(a+b)), \ldots)$



p is regular.