Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

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

On the consistency of Koomen's fair abstraction rule

Department of Computer Science · Report CS-R8511 · May

On the consistency of Koomen's Fair Abstraction Rule

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

We construct a graph model for ACP_τ, the algebra of communicating processes with silent steps, in which Koomen's Fair Abstraction Rule (KFAR) holds, and also versions of the Approximation Induction Principle (AIP) and the Recursive Definition & Specification Principles (RDP & RSP). We use this model to prove that in ACP_τ (but not in ACP !) each computably recursively definable process is finitely recursively definable.

1980 Mathematics Subject Classification: 68B10, 68C01, 68D25, 68F20.
1982 CR Categorie: F.1.1, F.1.2, F.3.2, F.4.3. 69F11, 69F12, 69F32, 69F43
Key Words & Phrases: process algebra, concurrency, Koomen's fair abstraction rule, recursion.
Note: This work is sponsored in part by Esprit project METEOR. This report will be submitted for publication elsewhere.

INTRODUCTION

This report is about process algebra, but it is not an introductory paper about process algebra; before reading this paper, the reader is advised to read some other papers on process algebra first, for example BERGSTRA & KLOP [10].

Koomen's Fair Abstraction Rule (KFAR) describes the idea of fairness in process algebra, and is the translation in process algebra of an idea of C.J. Koomen of Philips Research. KFAR was first formulated in BERGSTRA & KLOP [7], and its usefulness in protocol verification was demonstrated in BERGSTRA & KLOP [7, 8] and in BAETEN, BERGSTRA & KLOP [1, 2]. KFAR expresses the idea that, due to some fairness mechanism, abstraction from internal steps will yield an external step after finitely many repetitions; to be more precise, in the process $\tau_I(x)$, obtained from x by abstracting from steps in I, the steps in I will be fairly scheduled in such a way that eventually a step outside I is performed.

KFAR is the *algebraic* formulation of this idea, whereas the semantical implementation of fairness is already implicit in the notion of bisimulation on graphs, so is already implicit in the work of MILNER [18]. Some other recent papers on fairness are De BAKKER & ZUCKER [3, 4], COSTA & STIRLING [12], DARONDEAU [13], HENNESSY [14, 15, 16], MEYER [17] and PARROW [20].

When we use KFAR, all abstractions will be fair. Maybe this is a too optimistic model, and should the theory be able to describe situations where some abstractions are fair and others are not. Probably, an extension of the theory where this would be possible, will turn out to be rather complex.

In this paper, we do the following things. In §1, we review the theory ACP_τ, and extra axioms and rules SC, PR and KFAR. In §2, we define and discuss labeled graphs, elements of the set $\mathbb{G}_\kappa$. In §3, we prove that if we divide out the equivalence relation $\leftrightarrow_{r\tau\delta}$ (rooted $\tau\delta$-bisimulation) on $\mathbb{G}_\kappa$, we obtain a model of ACP_τ + SC + PR + KFAR, and we can even add some extra axioms (HA,ET, CA).

Report CS-R8511
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

In §4, we formulate the Approximation Induction Principle (AIP), which says that two processes are equal if all their projections are equal, and prove that AIP holds in $\mathbb{G}_\kappa$ for all finitely branching and bounded graphs. In §5, we look at recursive specifications, and formulate the Recursive Definition Principle (RDP) and the Recursive Specification Principle (RSP). Together, these principles say that a specification has a unique solution. We prove that RDP+RSP hold in $\mathbb{G}_\kappa$ for all guarded specifications.

In §6, we prove that every computable graph is recursively definable by a finite guarded specification, and we use this result in §7 to prove that any process, recursively definable by a computable guarded specification is already recursively definable by a finite guarded specification. In §8, we note that the abstraction operator is essential to prove these theorems.

Table of contents

§1. THE ALGEBRA OF COMMUNICATING PROCESSES WITH SILENT MOVES

The axiomatic framework in which we present this document is ACP_τ, the algebra of communicating processes with silent steps, as described in BERGSTRA & KLOP [6]. In this section, we give a brief review of ACP_τ.

1.1 Signature:

$\mathbb{S}$ (Sorts):	A	(a finite set of atomic actions)
	P	(the set of processes; $A \subseteq P$)
$\mathbb{F}$ (Functions):	$+ : P \times P \to P$	(alternative composition or sum)
	$\cdot : P \times P \to P$	(sequential composition or product)
	$\parallel : P \times P \to P$	(parallel composition or merge)
	$\lfloor\!\lfloor : P \times P \to P$	(left-merge)
	$\mid : P \times P \to P$	(communication merge; $\mid : A \times A \to A$ is given)
	$\partial_H : P \to P$	(encapsulation ; $H \subseteq A$)
	$\tau_I : P \to P$	(abstraction ; $I \subseteq A - \{\delta\}$)
$\mathbb{C}$ (constants):	$\delta \in A$	(deadlock)
	$\tau \in P - A$	(silent or internal action)

1.2 Axioms:

These are presented in table 1.
Here $a,b \in A$, $x,y,z \in P$, $H \subseteq A$ and $I \subseteq A - \{\delta\}$.

ACP$_\tau$

$x+y = y+x$	A1	$x\tau = x$	T1
$x+(y+z) = (x+y)+z$	A2	$\tau x + x = \tau x$	T2
$x+x = x$	A3	$a(\tau x+y) = a(\tau x+y)+ax$	T3
$(x+y)z = xz+yz$	A4		
$(xy)z = x(yz)$	A5		
$x+\delta = x$	A6		
$\delta x = \delta$	A7		
$a\vert b = b\vert a$	C1		
$(a\vert b)\vert c = a\vert(b\vert c)$	C2		
$\delta\vert a = \delta$	C3		
$x\Vert y = x⫽y+y⫽x+x\vert y$	CM1		
$a⫽x = ax$	CM2	$\tau⫽x = \tau x$	TM1
$(ax)⫽y = a(x\Vert y)$	CM3	$(\tau x)⫽y = \tau(x\Vert y)$	TM2
$(x+y)⫽z = x⫽z+y⫽z$	CM4	$\tau\vert x = \delta$	TC1
$(ax)\vert b = (a\vert b)x$	CM5	$x\vert\tau = \delta$	TC2
$a\vert(bx) = (a\vert b)x$	CM6	$(\tau x)\vert y = x\vert y$	TC3
$(ax)\vert(by) = (a\vert b)(x\Vert y)$	CM7	$x\vert(\tau y) = x\vert y$	TC4
$(x+y)\vert z = x\vert z+y\vert z$	CM8		
$x\vert(y+z) = x\vert y+x\vert z$	CM9		
		$\partial_H(\tau) = \tau$	DT
		$\tau_I(\tau) = \tau$	TI1
$\partial_H(a) = a$ if $a \notin H$	D1	$\tau_I(a) = a$ if $a \notin I$	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
$\partial_H(xy) = \partial_H(x)\cdot\partial_H(y)$	D4	$\tau_I(xy) = \tau_I(x)\cdot\tau_I(y)$	TI5

TABLE 1.

1.3 Standard concurrency

Often we expand the system ACP$_\tau$ with the following axioms of Standard Concurrency (see table 2). A proof that these axioms hold in the initial algebra of ACP$_\tau$ can be found in BERGSTRA & KLOP [6].

$(x⫽y)⫽z = x⫽(y\Vert z)$	SC 1
$(x\vert ay)⫽z = x\vert(ay⫽z)$	SC 2
$x\vert y = y\vert x$	SC 3
$x\Vert y = y\Vert x$	SC 4
$x\vert(y\vert z) = (x\vert y)\vert z$	SC 5
$x\Vert(y\Vert z) = (x\Vert y)\Vert z$	SC 6

TABLE 2.

1.4 Projection
Reasoning about processes often uses a projection operator

$$\pi_n : P \to P \quad (n \geqslant 1),$$

which "cuts of" processes at depth n (after doing n steps), but with the understanding that τ-steps are "transparent", i.e. a τ-step does not raise the depth.
Axioms for π_n are in table 3.

$\pi_n(a)=a$	PR1	$\pi_n(\tau)=\tau$	PRT1
$\pi_1(ax)=a$	PR2	$\pi_n(\tau x)=\tau\pi_n(x)$	PRT2
$\pi_{n+1}(ax)=a\pi_n(x)$	PR3		
$\pi_n(x+y)=\pi_n(x)+\pi_n(y)$	PR4		

TABLE 3.

1.5 Koomen's Fair Abstraction Rule
Koomen's Fair Abstraction Rule (see BERGSTRA & KLOP [7]) is a proof rule which is vital in algebraic computations for system verification, and expresses the fact that, due to some fairness mechanism, abstraction from 'internal' steps will yield an 'external' step after finitely many repetitions. The following algebraic formulation is parametrised by $k \geqslant 1$, indicating the length of an internal cycle.

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

In §3, we will find a model for the theory

$$\text{ACP}_\tau + \text{SC} + \text{PR} + \text{KFAR},$$

as defined in 1.1 -5.

§2. GRAPHS

In this section we will define the elements of the model that will be constructed in §3.

2.1 DEFINITION:
a *rooted directed multigraph* (which we will call *graph* for short) is a triple <NODES, EDGES, ROOT> with the following properties:

a. NODES is a set;
b. EDGES is a set; with each $e \in$ EDGES there is associated a pair $<s,t>$ from NODES.
   We say e *goes from* s *to* t, which we notate by

$$s \xrightarrow{e} t, \text{ or } s \circlearrowleft e \text{ if } s=t.$$

c. ROOT $\in$ NODES.

NOTATION: $g = <\text{NODES}(g), \text{EDGES}(g), \text{ROOT}(g)>$.

2.2 DEFINITIONS: Let g be a graph.
A *path* π in g is an alternating sequence of nodes and edges, such that each edge goes from the node before it to the node after it. We will only consider paths that are finite or have order type ω.
Thus, a path looks like

$$\pi: s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{k-1}} s_k$$

or

$$\pi: s_0 \xrightarrow{e_0} \cdots \xrightarrow{e_{k-1}} s_k \xrightarrow{e_k} \cdots$$

We say π *starts at* s_0 (in the pictured situations), and, if π is finite, that π *goes from* s_0 *to* s_k. If π goes from s_0 to s_0, π is a *cycle*, and any node in a cycle is called *cyclic*, a node not on any cycle is *acyclic*. If $s,t \in \text{NODES}(g)$, we say t *can be reached from* s if there is a finite path going from s to t.

2.3 *Note*: We will only consider graphs, in which each node can be reached from the root.

2.4 DEFINITIONS: Let g be a graph, $s \in \text{NODES}(g)$.

a. The *out-degree* of s is the cardinality of the set of edges starting at s; the *in-degree* of s is the cardinality of the set of edges going to s.
b. s is an *endnode* or *endpoint* of g if the out-degree of s is 0.
c. g is a *tree* if all nodes are acyclic, the in-degree of the root is 0 and the in-degree of all other nodes is 1.
d. the *subgraph* of s, $(g)_s$ is the graph with root s, and nodes and edges all those nodes and edges of g that can be reached from s.

2.5 Labeled graphs.
Let B,C be two sets, and κ an infinite cardinal number.
We define $\mathbb{G}_\kappa(B,C)$ (the set of labeled graphs) to be the set of all graphs such that:

1. each edge is labeled by an element of B;
2. each endnode is labeled by an element of C;
3. the out-degree of each node is less than κ.

Two elements of $\mathbb{G}_\kappa(B,C)$ are considered equal if they only differ in the names of nodes or edges.

2.6 DEFINITION: Let B,C,κ be given.

a. $\mathbb{G}_{\aleph_0}(B,C)$ is the set of *finitely branching* labeled graphs;
b. $\mathbb{T}_\kappa(B,C) = \{g \in \mathbb{G}_\kappa(B,C) : g \text{ is a tree}\}$ is the set of *labeled trees*;
c. $\mathbb{R}(B,C) = \{g \in \mathbb{G}_{\aleph_0}(B,C) : \text{NODES}(g) \cup \text{EDGES}(g) \text{ is finite}\}$ is the set of *finite* or *regular* labeled graphs;
d. $\mathbb{G}_\kappa{}^\rho(B,C) = \{g \in \mathbb{G}_\kappa(B,C) : g \text{ has acyclic root}\}$ is the set of *root-unwound* labeled graphs.

2.7 Root-unwinding.
The following definition is taken from BERGSTRA & KLOP [9], where most of the above terminology can also be found.

DEFINITION: let B, C, κ be given.

We define the *root-unwinding* map $\rho: \mathbb{G}_\kappa(B,C) \to \mathbb{G}_\kappa(B,C)$ as follows: let $g \in \mathbb{G}_\kappa(B,C)$.

a. NODES($\rho(g)$)=NODES(g)$\cup\{r\}$, where r is a 'fresh' node;
b. EDGES($\rho(g)$)=EDGES(g)$\cup\{ (r) \xrightarrow{e} (s) : (\text{ROOT}(g)) \xrightarrow{e} (s) \in$EDGES($g$)$\}$;
c. ROOT($\rho(g)$)=r
d. labeling is unchanged; if ROOT(g) has a label, r will get that label;
e. nodes and edges which cannot be reached from r are discarded.

2.8 Notes:
1. for all $g \in \mathbb{G}_\kappa(B,C)$, we have $\rho(g) \in \mathbb{G}_\kappa^\rho(B,C)$;
2. if $g \in \mathbb{G}_\kappa^\rho(B,C)$, then $g = \rho(g)$.

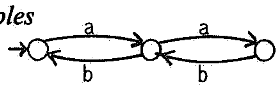
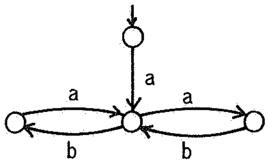
2.9 Examples
1. if g =  $,\rho(g)$ = 

fig. 1

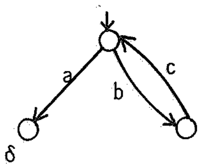
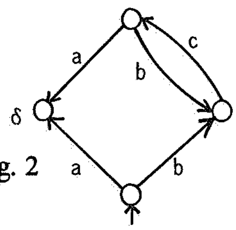
2. if g =  $,\rho(g)$ = 

fig. 2

(Note that when we picture graphs, we will not display names of nodes and edges, and only give their labels; we indicate the root by $\downarrow$).

§3. THE MODEL
We use the labeled graphs introduced in §2 to construct a model for ACP_τ.

3.1 DEFINITION: Let A be a given *finite* set of atoms, $\delta \in A$, $\tau \notin A$. Let a communication function $|: A \times A \to A$ be given, which is commutative and associative, such that $\delta | a = \delta$ for all $a \in A$.
We will use the symbol $\downarrow$ to denote successful termination (whereas δ denotes unsuccessful termination). Define the set of *process graphs* by:

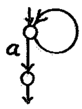
$$\mathbb{G}_\kappa = \mathbb{G}_\kappa(A_\tau - \{\delta\}, \{\delta, \downarrow\}) - \{\mathbb{O}\}.$$

Here κ is some infinite cardinal, $A_\tau = A \cup \{\tau\}$, and $\mathbb{O}$ is the graph $\circ$ (a single node labeled by $\downarrow$). Thus edges are labeled by elements of $A_\tau - \{\delta\}$, and endpoints by δ or $\downarrow$.

3.2 Next we will define an equivalence relation on $\mathbb{G}_\kappa$, which will say when two graphs denote the

same process. This is the notion of bisimulation (also see BERGSTRA & KLOP [6, 9, 10]).
First we define the label of a path in 3.3.

3.3 DEFINITION: Let $g \in \mathbb{G}_\kappa$, and π a path in g.
1. The *label* of π, $l(\pi)$ is the word in $(A_\tau \cup \{\downarrow\})^*$ (possibly infinite) obtained by putting the labels in π after each other (possibly including an endpoint label).
2. The *A-label* of π, $l_A(\pi)$ is the word in $(A \cup \{\downarrow\})^*$ obtained by leaving out all τ's in $l(\pi)$, but with the exception that if $l(\pi)=\tau^\omega$ (an infinite sequence of τ's), then $l_A(\pi)=\delta$.

3.4 EXAMPLE if $g=$  τ, g has paths with

labels $\epsilon, \downarrow, a, a\downarrow, \tau^n, \tau^\omega, \tau^n a, \tau^n a\downarrow$ (for each $n \in \mathbb{N}$) and with A-labels $\epsilon, \downarrow, a, a\downarrow, \delta$ (ϵ is the empty word).

3.5 We define three different bisimulations on $\mathbb{G}_\kappa$.
1. *δ-bisimulation*, $\leftrightarrows_\delta$ is the simplest;
2. *$\tau\delta$-bisimulation*, $\leftrightarrows_{\tau\delta}$ is like $\leftrightarrows_\delta$ but takes into account the special status of τ as a silent step;
3. *rooted $\tau\delta$-* bisimulation, $\leftrightarrows_{r\tau\delta}$ is like $\leftrightarrows_{\tau\delta}$ but also takes into account the special case when τ is an initial step.
   For more information on bisimulations, see PARK [19], and MILNER [18]. (We use δ as a subscript, to distinguish the bisimulations introduced here from $\leftrightarrows$, $\leftrightarrows_\tau$ and $\leftrightarrows_{r\tau}$ defined in BERGSTRA & KLOP [9], where δ is absent).

3.6 DEFINITIONS: Let $g,h \in \mathbb{G}_\kappa$, $R \subseteq \text{NODES}(g) \times \text{NODES}(h)$.
3.6.1. R is a *δ-bisimulation* between g and h, $R: g \leftrightarrows_\delta h$, if
1. $(\text{ROOT}(g), \text{ROOT}(h)) \in R$;
2. the domain of R is NODES (g), the range is NODES (h);
3. if $(p,q) \in R$ and $p \xrightarrow{l} p'$ is an edge in g with label $l \in A_\tau$, then there is a $q' \in \text{NODES}(h)$ and an edge $q \xrightarrow{l} q'$ in h with label l such that $(p',q') \in R$;
4. if $(p,q) \in R$, and p is an endpoint in g with label $l \in \{\delta, \downarrow\}$, then q is an endpoint in h with label l;

5,6. as 3,4 but with the roles of g and h reversed.

3.6.2. $g \leftrightarrows_\delta h$ iff there is an $R: g \leftrightarrows_\delta h$.

3.6.3. R is a *$\tau\delta$-bisimulation* between g and h, $R: g \leftrightarrows_{\tau\delta} h$ if
1,2: as in 3.6.1;
3^*: if $(p,q) \in R$ and $p \xrightarrow{l} p'$ is an edge in g with A-label $l \in A \cup \{\epsilon\}$, then there is a $q' \in \text{NODES}(h)$ and a path in h from q to q' with A-label l such that $(p',q') \in R$;
4^*: if $(p,q) \in R$, and p is an endpoint in g with (A)-label $l \in \{\delta, \downarrow\}$, then there is a path in h starting at q with A-label l.
5^*,6^*: same as 3^*,4^* but with the roles of g and h reversed.

3.6.4. $g \leftrightarrows_{\tau\delta} h$ iff there is an $R: g \leftrightarrows_{\tau\delta} h$.

3.6.5 Let $g_1, h_1 \in \mathbb{G}_\kappa^\rho$ (so with acyclic root).
R is a *rooted $\tau\delta$-bisimulation* between g_1 and h_1, $R: g_1 \leftrightarrows_{r\tau\delta} h_1$, if $R: g_1 \leftrightarrows_{\tau\delta} h_1$ and in addition
7. if $(p,q) \in R$, then $p = \text{ROOT}(g_1) \Leftrightarrow q = \text{ROOT}(h_1)$.

3.6.6 $g \leftrightarrows_{r\tau\delta} h$ iff there is an $R: \rho(g) \leftrightarrows_{r\tau\delta} \rho(h)$.

3.7 EXAMPLES:

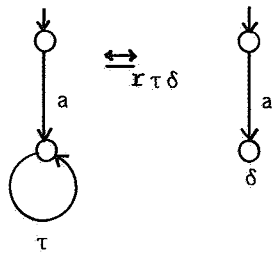


Fig.3

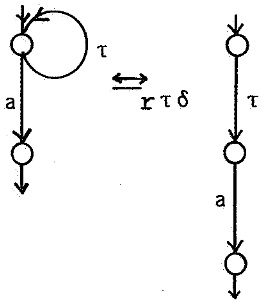


Fig.4

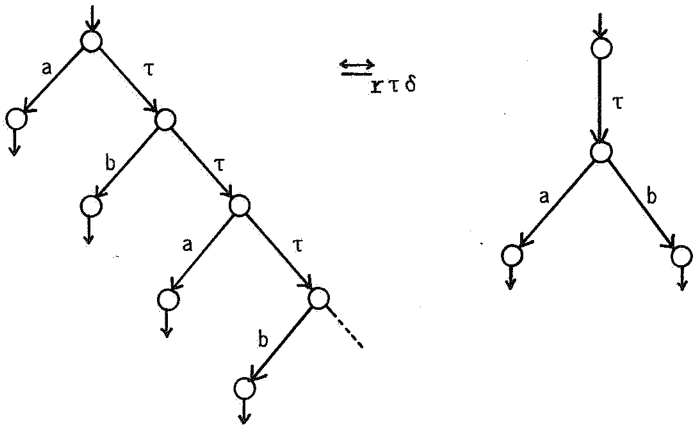


Fig.5

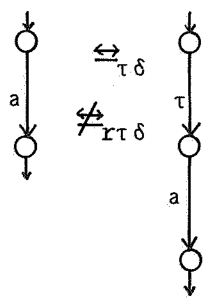


Fig.6

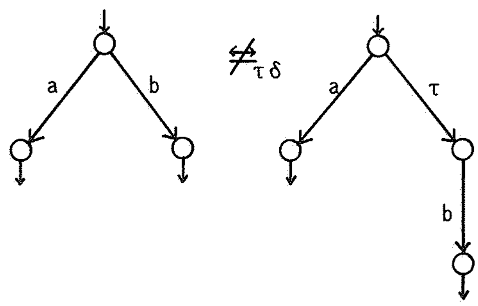


Fig.7

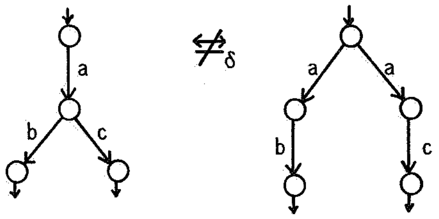


Fig.8

3.8 LEMMA:

1. $\underline{\leftrightarrow}_\delta$, $\underline{\leftrightarrow}_{\tau\delta}$ *and* $\underline{\leftrightarrow}_{r\tau\delta}$ *are equivalence relations on* $\mathbb{G}_\kappa$.
2. *for all* $g \in \mathbb{G}_\kappa$, $g \underline{\leftrightarrow}_\delta \rho(g)$, $g \underline{\leftrightarrow}_{\tau\delta} \rho(g)$ *and* $g \underline{\leftrightarrow}_{r\tau\delta} \rho(g)$.

PROOF: easy.

3.9 $\mathbb{G}_\kappa / \underline{\leftrightarrow}_{r\tau\delta}$ will be domain of our model. Next we need to define the operations of ACP_τ on $\mathbb{G}_\kappa / \underline{\leftrightarrow}_{r\tau\delta}$. Actually, we will define them on $\mathbb{G}_\kappa$, and leave it to the reader to check that $\underline{\leftrightarrow}_{r\tau\delta}$ is a congruence relation for all these operations.

3.9.1. $+$. If $g,h \in \mathbb{G}_\kappa$, obtain $g+h$ by identifying the roots of $\rho(g)$ and $\rho(h)$. If one root is an endpoint, it must be $\overset{\downarrow}{\circ}{}_\delta$ (for $\mathbf{O} \notin \mathbb{G}_\kappa$) and we delete this label. If both g and h are $\overset{\downarrow}{\circ}{}_\delta$, we put $g+h = \overset{\downarrow}{\circ}{}_\delta$.

EXAMPLE:

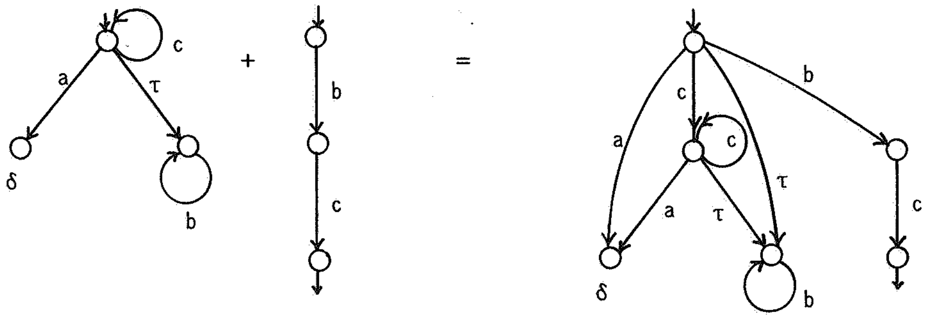


Fig.9

3.9.2. $\cdot$. If $g,h \in \mathbb{G}_\kappa$, obtain $g \cdot h$ by identifying all $\downarrow$-endpoints of g with ROOT(h), and removing the $\downarrow$-labels in g.

EXAMPLE:

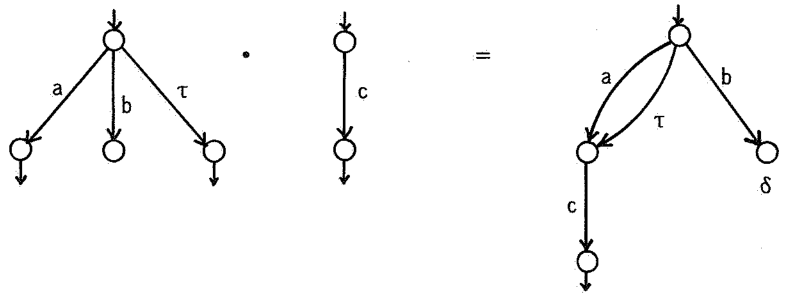


Fig.10

3.9.3. $\|$. If $g, h \in \mathbb{G}_K$, obtain $g \| h$ by taking the cartesian product graph of g and h (with root the pair of roots from g and h), and adding, for each edge $p \xrightarrow{a} p'$ in g with label a, and for each edge $q \xrightarrow{b} q'$ in h with label b, if $a|b = c \neq \delta$, a new edge $(p,q) \xrightarrow{c} (p',q')$ with label c (a 'diagonal' edge).

In $g \| h$, define the endpoint labeling as follows:

1. if in node (p,q), only one of the two components is an endpoint, drop its label;
2. if in node (p,q), both components are endpoints, give this endpoint label $\downarrow$ if both p and q have label $\downarrow$, and label δ otherwise.

EXAMPLE: (assume $a|a = a|b = b|b = b|a = \delta$)

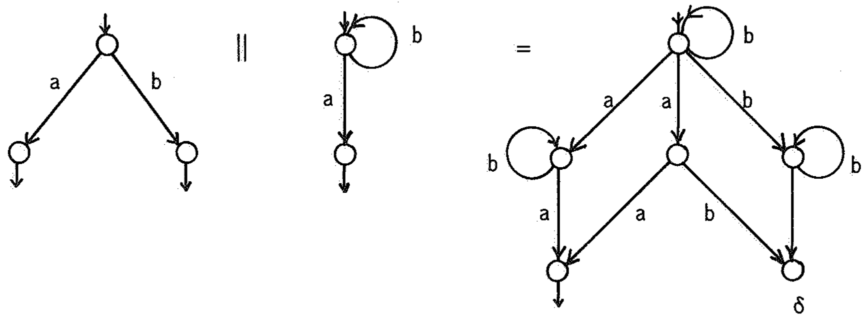


Fig.11

3.9.4. $\Lsh$. If $g, h \in \mathbb{G}_K$, $g \Lsh h$ is the maximal subgraph of $\rho(g) \| h$ in which each initial step is one from $\rho(g)$.

EXAMPLE:

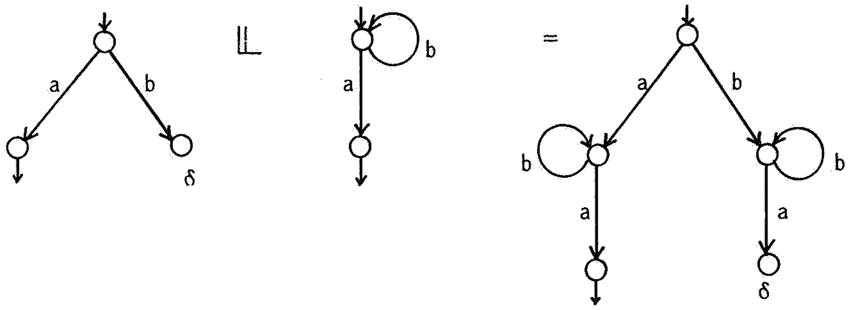


Fig.12

while

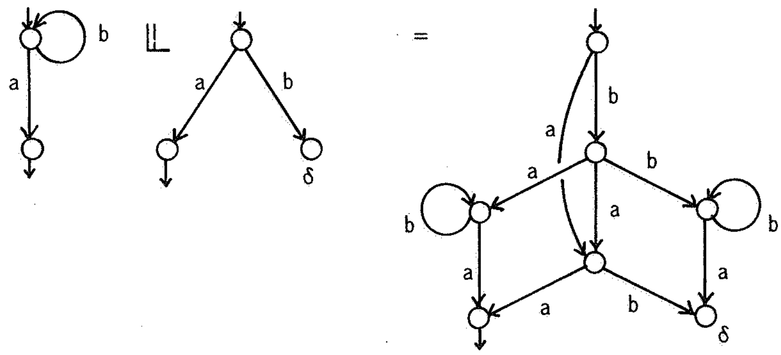


Fig.13

(we again assumed $a|a = a|b = b|b = b|a = \delta$).

3.9.5. $|$. If $g,h \in \mathbb{G}_\kappa$, $g|h$ is the sum of all the maximal subgraphs of $g \| h$ that start with a communication (diagonal) step and can be reached from the root by a path with A-label ϵ.

EXAMPLE: if $b|a = a|b = c$, $a|a = b|b = \delta$, then

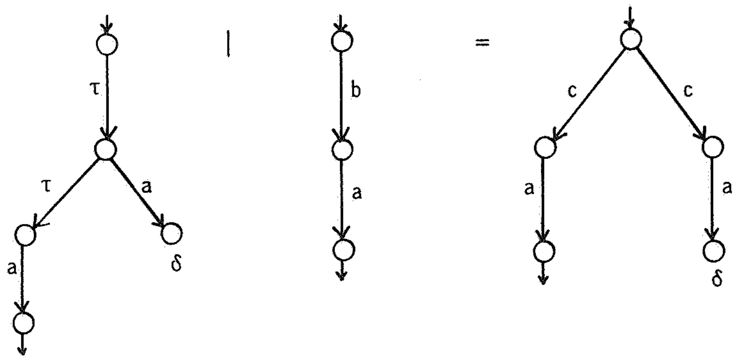


Fig.14

3.9.6. ∂_H. Let $H \subseteq A$ be given. If $g \in \mathbb{G}_\kappa$, obtain $\partial_H(g)$ by the following steps:

1. remove all edges with labels from H;
2. remove all parts of the graph that cannot be reached from the root;
3. label all unlabeled endpoints by δ.

EXAMPLE: if $a \in H$, then

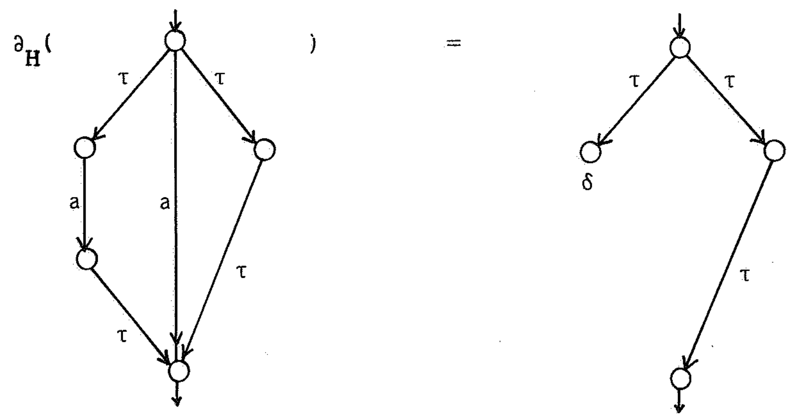


Fig.15

3.9.7. τ_I. Let $I \subseteq A - \{\delta\}$ be given. If $g \in \mathbb{G}_\kappa$, obtain $\tau_I(g)$ by changing all labels from I to τ.

3.9.8. π_n. Let $n \geq 1$ be given. If $g \in \mathbb{G}_\kappa$, obtain $\pi_n(g)$ as follows:

1. NODES $(\pi_n(g)) = \{s \in \text{NODES}(g): s$ can be reached from ROOT(g) by a path π with the length of $l_A(\pi)$ less than or equal to $n\}$;
2. EDGES $(\pi_n(g)) = \{e \in \text{EDGES}(g): e$ occurs in a path π from ROOT(g) with length $(l_A(\pi)) \leq n\}$;
3. ROOT $(\pi_n(g)) = \text{ROOT}(g)$;
4. all unlabeled endpoints in $\pi_n(g)$ get a label $\downarrow$;
5. if a δ-labeled endpoint cannot be reached by a path π with length $(l_A(\pi)) < n$, change the δ-label to a $\downarrow$-label;
6. all other labels remain unchanged.

EXAMPLE:

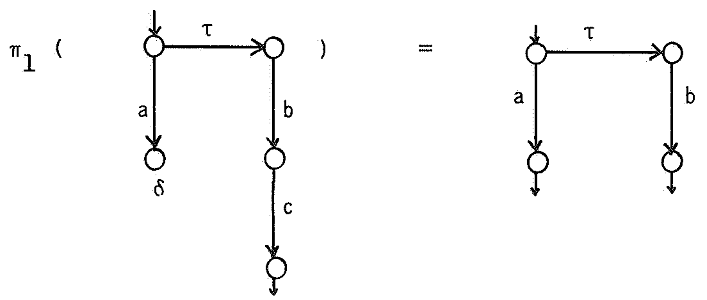
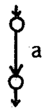
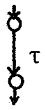


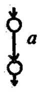
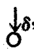
Fig.16

3.10 Finally we define an interpretation of the constants of ACP_τ into $\mathbb{G}_\kappa$.

1. If $a \in A - \{\delta\}$, its interpretation $[a]$ = 

2. $[\delta]$ = 
3. $[\tau]$ = 

3.11 **Theorem** *Let κ be a given infinite cardinal number.*

$(\mathbb{G}_\kappa, (+, \cdot, \|, \lfloor\!\lfloor, |, \partial_H, \tau_I, \pi_n), (\{$  $: a \in A - \{\delta\}\}$, , $))$

is a model of ACP$_\tau$ +SC+PR+KFAR.

The proof of this theorem is not very hard, but extremely tedious, which is why we will limit ourselves to some examples.
Also see Bergstra & Klop [6], 2.5 and Bergstra & Klop [10], 1.2.2, 2.1.2, 4.1.1, 4.2.1.

Examples: we denote bisimulations by linking related nodes by dotted lines.
1. A3: $a + a = a$.

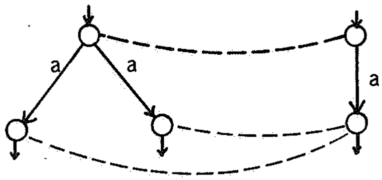


Fig.17

2. A4: $(a+b)c = ac + bc$.

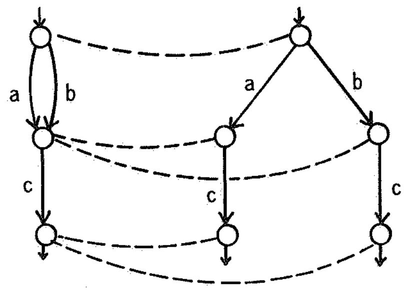


Fig.18

3. T1: $a\tau = a$

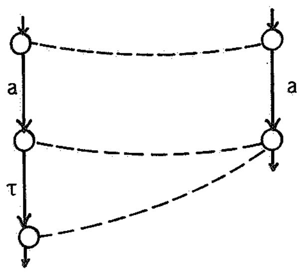


Fig.19

4. T2: $\tau a + a = \tau a$

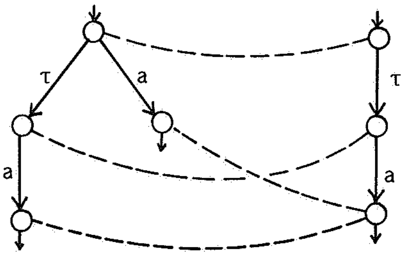


Fig.20

5. T3: $a(\tau b + c) = a(\tau b + c) + ab$

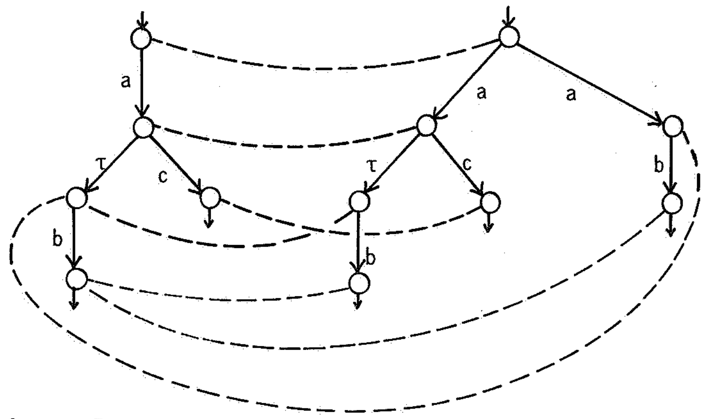


Fig.21

6. KFAR. Also see BERGSTRA & KLOP [9], 7.12, where a version of KFAR without δ is proved.
Let $k \geqslant 1$ be given, and suppose $i_0,..,i_{k-1} \in I$, $x_0,..,x_{k-1}$, $y_0,..,y_{k-1}$ are processes, and $x_n = i_n x_{n+1} + y_n$ for all $n \in \mathbb{Z}_k$. Now we need a result in our model from §5, which says that equations $x_n = i_n x_{n+1} + y_n$ have unique solutions in our model, i.e. there are unique $g_0,...,g_{k-1}, h_0,...,h_{k-1} \in \mathbb{G}_k$ (up to $\leftrightarrow_{r\tau\delta}$) such that $g_n \leftrightarrow_{r\tau\delta} i_n g_{n+1} + h_n$ holds for each $n \in \mathbb{Z}_k$.

6.1 Let us first consider the case $k = 1$, so we have

$$g \leftrightarrow_{r\tau\delta} ig + h$$

for some $i \in I$, $g, h \in \mathbb{G}_\kappa$.
case 1: $h = \delta$ (actually, we mean $h = $ δ).
Then $g \leftrightarrow_{r\tau\delta} ig$.
Notice that graph i satisfies this equation, so by the unicity of g we must have $g \leftrightarrow_{r\tau\delta}$ i. Then

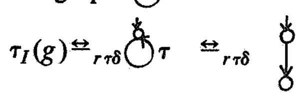


Fig.22

which is the desired result, because

$$\frac{x = ix = ix + \delta}{\tau_{\{i\}}(x) = \tau\delta}\ \mathrm{KFAR}_1.$$

case 2: h is not δ.

Then we obtain $g \leftrightarrow_{r\tau\delta}$

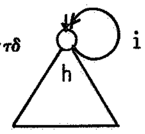


Fig.23

so $\tau_I(g) \leftrightarrow_{r\tau\delta}$

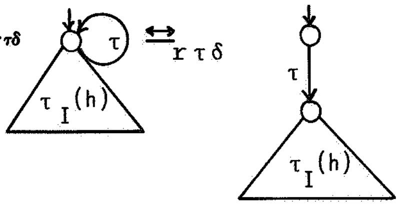


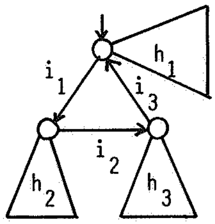
Fig.24

again the right result.

6.2 If $k > 1$, the proof works similarly. For instance, if $k = 3$, we have

$$g_1 \leftrightarrow_{r\tau\delta} i_1 g_2 + h_1$$
$$g_2 \leftrightarrow_{r\tau\delta} i_2 g_3 + h_2$$
$$g_3 \leftrightarrow_{r\tau\delta} i_3 g_1 + h_3$$

$(i_1, i_2, i_3, \in I)$, so
$g_1 \leftrightarrow_{r\tau\delta}$



whence $\tau_I(g_1) \leftrightarrow_{r\tau\delta}$

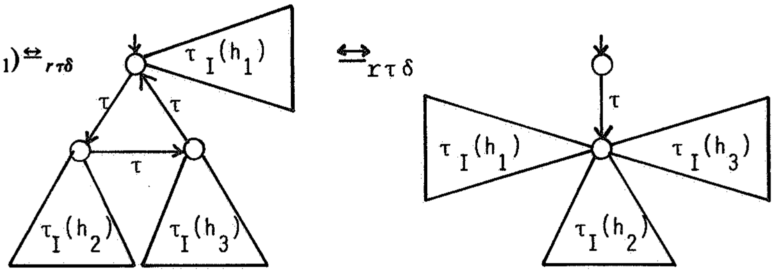


Fig.25

Fig.26

3.12 Handshaking
If we adopt the Handshaking Axiom (HA), namely

$$\boxed{\text{(HA)} \quad x|y|z = \delta}$$

for all processes x, y, z, which says that all communications are binary, then the following Expansion Theorem (ET) holds in the model $\mathbb{G}_\kappa / \leftrightarrow_{r\tau\delta}$. This is because $\mathbb{G}_\kappa / \leftrightarrow_{r\tau\delta}$ satisfies the Axioms of

Standard Concurrency of 1.3. A proof of this fact is given in BERGSTRA & TUCKER [11]. Let $x_1,...,x_n$ be given processes, and let $\bar{x}^i$ be the merge of all $x_1,...,x_n$ except x_i, $\bar{x}^{i,j}$ be the merge of all $x_1,...,x_n$ except x_i and x_j, then the Expansion Theorem is

$$\text{(ET)} \quad x_1 \| x_2 \| ... \| x_n = \sum_{1 \leq i \leq n} x_i \,⌊\!⌊\, \bar{x}^i + \sum_{1 \leq i < j \leq n} (x_i | x_j) \,⌊\!⌊\, \bar{x}^{i,j}$$

in words: if you merge a number of processes, you can start with an action from one of them or with a communication between two of them.

3.13 Alphabets

We can define, for each $g \in \mathbb{G}_k$, the *alphabet* of g, $\alpha(g)$, to be the set of all labels occurring in g except $\tau,\delta,\downarrow$. Note that here we will need the requirement of 2.3, that each node can be reached from the root. Then it is easy to see that if $g \leftrightarrow_{r\tau\delta} h$ (even if $g \leftrightarrow_{\tau\delta} h$), then $\alpha(g) = \alpha(h)$. With this definition, it is not hard to show that $\mathbb{G}_\kappa / \leftrightarrow_{r\tau\delta}$ satisfies the following Conditional Axioms (CA), first formulated in BAETEN, BERGSTRA & KLOP [2].

$\dfrac{\alpha(x) \mid (\alpha(y) \cap H) \subseteq H}{\partial_H(x \| y) = \partial_H(x \| \partial_H(y))}$	CA1	$\dfrac{\alpha(x) \mid (\alpha(y) \cap I) = \varnothing}{\tau_I(x \| y) = \tau_I(x \| \tau_I(y))}$	CA2
$\dfrac{\alpha(x) \cap H = \varnothing}{\partial_H(x) = x}$	CA3	$\dfrac{\alpha(x) \cap I = \varnothing}{\tau_I(x) = x}$	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
$\dfrac{H \cap I = \varnothing}{\tau_I \circ \partial_H(x) = \partial_H \circ \tau_I(x)}$	CA7		

TABLE 4.

§4 THE APPROXIMATION INDUCTION PRINCIPLE

The Approximation Induction Principle (AIP), expresses the idea that if two processes are equal to any depth, then they are equal, or, for processes x, y

$$\text{AIP} \quad \frac{\text{for all } n \quad \pi_n(x) = \pi_n(y)}{x = y}$$

We will prove in 4.3 that a restricted version of AIP holds in $\mathbb{G}_\kappa / \leftrightarrow_{r\tau\delta}$. In 4.7 we see that the unrestricted version does not hold. First a definition:

4.1 DEFINITION: let $g \in \mathbb{G}_\kappa$. Define the n^{th} *level* of g, $[g]_n$, by: $[g]_n = \{s \in \text{NODES}(g) : s \text{ can be reached from ROOT}(g) \text{ by a path } \pi \text{ with length } (l_A(\pi)) = n\}$.

We say $s \in \text{NODES}(g)$ is of *depth* n if $s \in [g]_n$. Note that the $[g]_n$ for different n need not be disjoint.

4.2 LEMMA: *let* $g,h \in \mathbb{G}_\kappa^\rho$ *and* $R : g \leftrightarrows_{r\tau\delta} h$. *If* $s \in$ NODES(g) *is of depth* n, *then there is a* $t \in$ NODES(h) *of depth* n *such that* $(s,t) \in R$.

PROOF: by definition of $\leftrightarrows_{r\tau\delta}$.

4.3 THEOREM: *let* $g,h \in \mathbb{G}_\kappa$ *and suppose that for each* n

i. $\pi_n(g) \leftrightarrows_{r\tau\delta} \pi_n(h)$
ii. either $[g]_n$ or $[h]_n$ is finite.

Then $g \leftrightarrows_{r\tau\delta} h$.

PROOF: without loss of generality, we can suppose that g and h are root-unwound, so $g,h \in \mathbb{G}_\kappa^\rho$. Fix $n \in \mathbb{N}$, and let $s \in [g]_n$, $t \in [h]_n$.
Define

$$s \sim_m t \Leftrightarrow \text{there is an } R : \pi_{n+m}(g) \leftrightarrows_{r\tau\delta} \pi_{n+m}(h)$$

such that $R \cap ((g)_s \times (h)_t) : \pi_m((g)_s) \leftrightarrows_{\tau\delta} \pi_m((h)_t)$. (in words: there is an R, which is a rooted $\tau\delta$-bisimulation until depth $n+m$, and, restricted to the subgraphs of s and t, is a $\tau\delta$-bisimulation until depth m), and

$$s \sim t \Leftrightarrow \text{for all } m \in \mathbb{N}\ s \sim_m t.$$

We will show that $\sim$ is a rooted $\tau\delta$-bisimulation between g and h.
Note that by definition of $\sim$, and assumption i, we have

1. ROOT(g)$\sim$ ROOT(h), and
7. if $s \sim t$, then $s =$ ROOT(g) $\Leftrightarrow t =$ ROOT(h). By definition of $\leftrightarrows_{\tau\delta}$ and lemma 4.2 we also have
2. $dom(\sim)=$NODES(g) and $ran(\sim)=$NODES(h). It remains to verify $3^*,4^*,5^*,6^*$ of 3.6.3.

For 3^*, suppose $s \sim t$, and take n such that $s \in [g]_n$, $t \in [h]_n$. Let $s \xrightarrow{l} s^*$ be an edge in g with label l.
case 1: $l \neq \tau$, so $l = a \in A$. Then $s^* \in [g]_{n+1}$.
case 1.1: $[h]_{n+1}$ is finite.
Put $S_0 = \{t' \in [h]_{n+1}$: there is a path from t to t' with A-label $a\}$. Since $s \sim_1 t$, we know $S_0 \neq \varnothing$. Put $S_1 = \{t' \in S_0 : s^* \sim_1 t'\}$. Since $s \sim_2 t$, we have $S_1 \neq \varnothing$. In general, define $S_m = \{t' \in S_0 : s^* \sim_m t'\}$. We have $S_0 \supseteq S_1 \supseteq ... \supseteq S_m \supseteq ...$, and all the S_m are nonempty. Since $[h]_{n+1}$ is finite, we must have $\bigcap_{m \geqslant 0} S_m \neq \varnothing$. Take $t^* \in \bigcap_{m \geqslant 0} S_m$, then $s^* \sim t^*$. Since $t^* \in S_0$, 3^* is satisfied.
case 1.2: Otherwise. By assumption ii, $[g]_{n+1}$ is finite. Let, for each $s' \in [g]_{n+1}$,

$$H_{s'} = \{t' \in [h]_{n+1} : s' \sim t'\}.$$

Note that by lemma 4.2, $[h]_{n+1} = \bigcup_{s' \in [g]_{n+1}} H_{s'}$, and this is a finite union. Put $S_0 = \{t' \in [h]_{n+1}$: there is a path from t to t' with A-label $a\}$. If $S_0 \cap H_{s^*} \neq \varnothing$, we are done. Otherwise, we can find a sequence $<t_0, t_1, t_2 ...>$ in S_0 such that $s^* \sim_m t_m$ (since $s \sim_{m+1} t$). Since there are only finitely many $H_{s'}$, there is a s'' such that $t_m \in H_{s''}$ for infinitely many m. Pick $t^* \in H_{s''}$. We will prove $s^* \sim t^*$, and then we are done. So let $m \in \mathbb{N}$. Now $s^* \sim_m t_m$, $s'' \sim t^*$ and $s' \sim t_m$, so we can take $R_1, R_2, R_3 : \pi_{n+m+1}(g) \leftrightarrows_{r\tau\delta} \pi_{n+m+1}(h)$ such that

$$R_1 \cap ((g)_{s^*} \times (h)_{t_m}) : \pi_m((g)_{s^*}) \leftrightarrows_{\tau\delta} \pi_m((h)_{t_m}),$$

$$R_2 \cap ((g)_{s''} \times (h)_{t^*}) : \pi_m((g)_{s''}) \leftrightarrows_{\tau\delta} \pi_m((h)_{t^*}),$$

$$R_3 \cap ((g)_{s''} \times (h)_{t_m}) : \pi_m((g)_{s''}) \leftrightarrows_{\tau\delta} \pi_m((h)_{t_m}).$$

A picture might clarify the matter (Fig. 27).

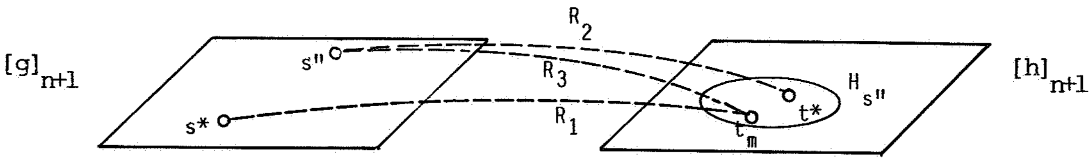


Fig.27

Now define $R \subseteq$ NODES(g)$\times$ NODES (h) by: $(p,q)\in R \Leftrightarrow$ there are $p' \in$NODES(g) and $q' \in$NODES(h) such that $(p,q')\in R_1$, $(p',q)\in R_2$ and $(p',q')\in R_3$. It follows that

$$R : \pi_{n+m+1}(g) \leftrightarrow_{r\tau\delta} \pi_{n+m+1}(h)$$

and $R \cap ((g)_s* \times (h)_t*) : \pi_m((g)_s*) \leftrightarrow_{\tau\delta} \pi_m((h)_t*)$,so $s^* \sim_m t^*$. Since m was chosen arbitrarily, we have shown $s^* \sim t^*$.

case 2: $l=\tau$. We reason as in case 1, but work in $[g]_n$ and $[h]_n$, since a τ-step does not increase depth, so $s^* \in [g]_n$, $t^* \in [h]_n$. In case 2.2, we observe

$$[h]_n \cap (h)_t = \bigcup_{s' \in [g]_n \cap (g)_s} H_{s'}.$$

Thus, we have verified 3^* of 3.6.3. For a verification of 4^*, suppose $s \sim t$, n is such that $s \in [g]_n, t \in [h]_n$ and s is an endpoint in g with label l. Since $s \sim_1 t$, there is an $R : \pi_{n+1}(g) \leftrightarrow_{r\tau\delta} \pi_{n+1}(h)$ with $(s,t)\in R$. s is also an endpoint in $\pi_{n+1}(g)$ with label l, so, since R is a $\tau\delta$-bisimulation, there must be a path in $\pi_{n+1}(h)$ starting at t with A-label l. Since $t \in [h]_n$, this path is also in h, and has the same A-label there.

Proofs for 5^*,6^* of 3.6.3 are like the proofs for 3^*,4^*, but with the roles of g and h reversed.

Thus, we have shown that $\sim$ is a rooted $\tau\delta$-bisimulation between g and h, which finishes the proof.

4.4 DEFINITION Let $g \in \mathbb{G}_\kappa$. We say that g is *bounded* if g has no path with label τ^ω. (A somewhat more restricted definition of boundedness is given in BERGSTRA & KLOP [5]).

4.5 LEMMA *If* $g \in \mathbb{G}_{\aleph_0}$ *(i.e. g is finitely branching), and g is bounded, then for each n,* $[g]_n$ *is finite.*

PROOF: By induction. For $n=0$, $[g]_0$ consists only of those nodes that can be reached from ROOT(g) by a path with all labels τ. The graph g' consisting of $[g]_0$ and these τ-paths cannot contain a cycle, for that would immediately give a path with lable τ^ω, contradicting the boundedness of g. Thus g' is acyclic, and by König's lemma it must be finite, for an infinite branch has label τ^ω. Then also $[g]_0 =$NODES(g') is finite.
For the induction step, suppose $[g]_n$ is finite. Put $B=\{s \in [g]_{n+1}$: there is a $t \in [g]_n$ and an edge $t \xrightarrow{a} s$, $a \in A\}$. Since each $t \in [g]_n$ can have only finitely many immediate successors in B, B must be finite. If $s \in [g]_{n+1} - B$, s can be reached from a member of B by a series of τ-steps, and the same argument as above shows that $[g]_{n+1}$ must be finite, which finishes the proof.

4.6 COROLLARY: *Let* $g,h \in \mathbb{G}_\kappa$. If *one of* g,h is finitely branching and bounded, then g,h satisfy AIP *(i.e. if for all n* $\pi_n(g) \leftrightarrow_{r\tau\delta} \pi_n(h)$, then $g \leftrightarrow_{r\tau\delta} h$).

PROOF: 4.3 + 4.5.

4.7 Counterexamples

Suppose a,b are two different atomic actions ($a,b \in A - \{\delta\}$). We will consider infinite sequences of a' s and b' s. Let E_a be the set of all such sequences that are eventually a (i.e. for each $s \in E_a$ there is an $n \in \mathbb{N}$ such that after the first n symbols, s consists only of a' s), and let E_b be the set of all sequences that are eventually b. Note that E_a and E_b are countable.

EXAMPLE 1: define $g = \sum_{s \in E_a} s$, $h = \sum_{s \in E_b} s$.

See Fig. 28.

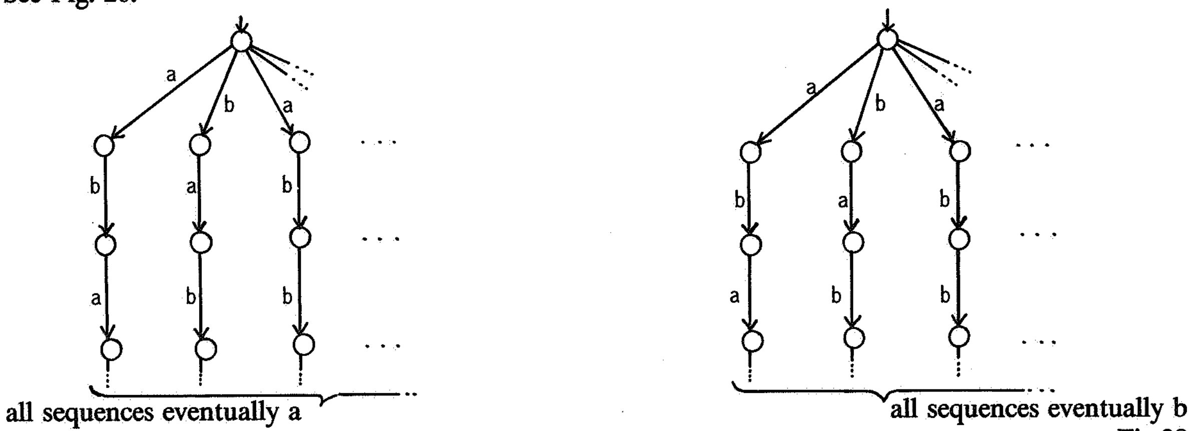


Fig.28

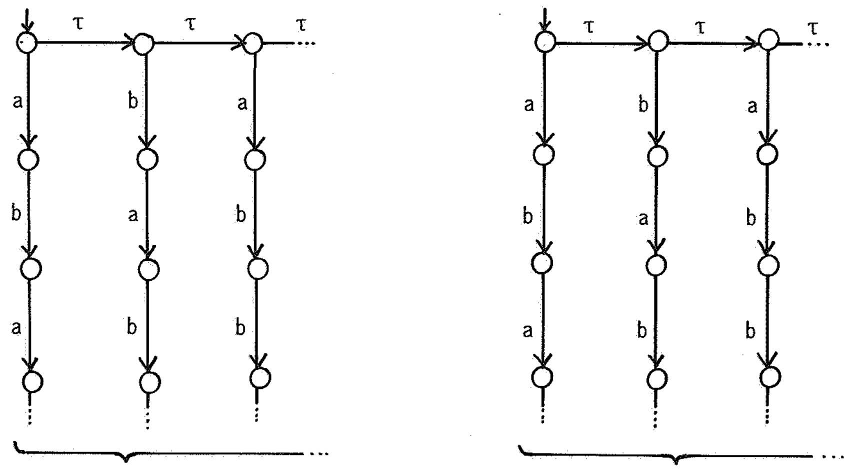
It is not hard to see that for each n

$$\pi_n(g) = \pi_n(h),$$

but not $g \leftrightarrow_{r\tau\delta} h$,

so g,h do not satisfy AIP. g and h are both bounded, but not finitely branching.

EXAMPLE 2: g' and h' are shown in Fig.29.



all sequences eventually a

all sequences eventually b

Fig.29

Again we have $\pi_n(g') \underline{\leftrightarrow}_{r\tau\delta} \pi_n(h')$ for each n (irrespective of how E_a and E_b are enumerated along the τ-path), but not $g' \underline{\leftrightarrow}_{r\tau\delta} h'$, so g',h' do not satisfy AIP. g' and h' are both finitely branching, but not bounded.

NOTE: although g and g' (and h and h') are certainly related, they do not $\tau\delta$-bisimulate. However, if we change g' so that each element of E_a occurs infinitely many times, we do have a $\tau\delta$-bisimulation (this is a sort of infinite version of KFAR).

4.8 *Note*: at this point, we cannot formulate the restricted version of AIP proved in 4.3 or 4.6 algebraically. We can do this in §5, after we have discussed RDP and RSP.

§5. The Recursive Definition Principle and the Recursive Specification Principle

In this section we will look at recursive specifications, which are sets of equations, and processes given by recursive specifications. The Recursive Definition Principle (RDP) states that certain specifications have a solution, while the Recursive Specification Principle (RSP) says that certain specifications have at most one solution. Specifications that satisfy both RDP and RSP have a unique solution.

5.1 DEFINITION: a *(recursive) specification* $E = \{E_j : j \in J\}$ is a set of equations in the language of ACP_τ with variables $\{X_j : j \in J\}$ (J is some set), such that equation E_j has the form

$$X_j = T_j$$

where T_j is a finite ACP_τ-term (with finitely many variables) and J contains a designated element j_0. If J is (partially) ordered, and has one minimal element, then j_0 is this minimal element.

5.2 EXAMPLE: Let E be

$$X_0 = X_1 \| X_2 + X_3 a$$

$$X_1 = \tau\partial_H(X_0 X_0)$$

$$X_2 = \tau X_2$$

$X_3 = \tau_I(aX_2 + X_3bX_1)$.

Here $J = \{0,1,2,3\}$, $j_0 = 0$, E_2 is equation $X_2 = \tau X_2$ and T_2 is term τX_2.

5.3 DEFINITIONS: Let J be a set, E a recursive specification indexed by J, and let $\{x_j : j \in J\}$ be processes. Put $x = x_{j_0}$, $\mathbb{X} = \{x_j : j \in J, j \neq j_0\}$
1. *x is a solution of E with parameters* $\mathbb{X}$, notation $E(x,\mathbb{X})$, if substituting the x_j for variables X_j in E gives only true statements about processes $\{x_j : j \in J\}$.
2. *x is a solution of E*, notation $E(x,-)$, if there are processes $\mathbb{X} = \{x_j : j \in J, j \neq j_0\}$ such that $E(x,\mathbb{X})$.
3. x is *(recursively) definable* if there is a specification E such that x is the unique solution of E.

5.4 The *Recursive Definition Principle* (RDP) for a recursive specification E is

$$\boxed{\text{(RDP)} \quad \exists x \quad E(x,-)}$$

i.e. there exists a solution for E. While it is probably true that RDP holds in general in the model $\mathbb{G}_\kappa / \leftrightarrows_{r\tau\delta}$, we will prove it only for a restricted class of specifications.

5.5 The *Recursive Specification Principle* (RSP) for a recursive specification E is

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

It is obvious that RSP does not hold for every specification E (every process is a solution of the trivial specification $X_0 = X_0$).
In the sequel, we will formulate a condition of guardedness, such that RSP holds for all guarded specifications in $\mathbb{G}_\kappa / \leftrightarrows_{r\tau\delta}$. However, we run into big problems when we want to formulate guardedness for specifications containing abstraction operators τ_I. As a hint to the problems involved, consider the specification

$$\begin{cases} X_0 = a\tau_{\{b\}}(X_1) \\ X_1 = b\tau_{\{a\}}(X_0). \end{cases}$$

This specification certainly looks guarded, but has infinitely many solutions in $\mathbb{G}_\kappa / \leftrightarrows_{r\tau\delta}$, so does not satisfy RSP. Because of these problems, we will formulate guardedness and the following theorems *only* for specifications that contain no abstraction.

5.6 DEFINITION: Let T be an open ACP $_\tau$-term without an abstraction operator τ_I. An occurrence of a variable X in T is *guarded* if T has a subterm of the form aM, with $a \in A$ (so $a \neq \tau$), and this X occurs in M. Otherwise, the occurrence is *unguarded.*

5.7 EXAMPLES: let T be the term

$$aX_0 + \tau X_1 + a \mathbin{\|\!\!\_} X_2 + X_3 \| aX_4.$$

In T, X_0 and X_4 occur guarded, and X_1,X_2,X_3 unguarded.

5.8 DEFINITION: Let $E = \{E_j : j \in J\}$ be a specification without an abstraction operator τ_I, and let $i,j \in J$. We define
$X_i \xrightarrow{u} X_j \Leftrightarrow X_j$ occurs unguarded in T_i, and we call E *guarded* if relation $\xrightarrow{u}$ is well-founded (i.e. there is no infinite sequence $(X_{j_1} \xrightarrow{u} X_{j_2} \xrightarrow{u} X_{j_3} \xrightarrow{u} \ldots)$.

Next we start the proof of RDP and RSP in $\mathbb{G}_\kappa / \underline{\leftrightarrow}_{r\tau\delta}$.

5.9 DEFINITION: Let $E=\{E_j : j \in J\}$ be a specification, and let $j \in J$. An *expansion* of X_j is an open ACP$_\tau$-term obtained by a series of substitutions of T_i for occurrences of X_i in E_j. For a more precise definition, see BAETEN, BERGSTRA & KLOP [2], 2.7.

5.10 LEMMA: *Let E be a guarded recursive specification in which no abstraction operator τ_I occurs, and let $j \in J$ (the index set of E). Then X_j has an expansion in which all occurrences of variables are guarded.*

PROOF: Essentially, this is lemma 2.14 in BAETEN, BERGSTRA & KLOP [2]. We build up such an expansion in the following way. If in T_j, all occurrences of variables are guarded, we are done. Otherwise, substitute T_i for all unguarded X_i in T_j, and repeat this process. This must stop after finitely many steps, for otherwise we obtain by König's lemma an infinite sequence $X_j \xrightarrow{u} X_i \xrightarrow{u} \ldots$, which contradicts the well-foundedness of $\xrightarrow{u}$.

5.11 THEOREM: *Let E be a guarded recursive specification in which no abstraction operator occurs. Then, in the model $\mathbb{G}_\kappa / \underline{\leftrightarrow}_{r\tau\delta}$, E has a solution which is finitely branching and bounded.*

PROOF: We will construct a solution g in stages g_n, for $n \in \mathbb{N}$. For $n=1$, let T^1 be an expansion of X_{j_0} in which all variables are guarded (T^1 exists by 5.10). Then it is easy to see that $\pi_1(T^1)$ does not contain any variables, so is a finite closed ACP$_\tau$-term. Let g_1 be the canonical graph of $\pi_1(T^1)$. By canonical, we mean that we do not use any ACP$_\tau$-equations in constructing g_1, but only the operations defined in 3.9 (we can replace all variables occurring in T^1 by δ, since they do not matter anyway). Note that g_1 is *finite*. Now suppose g_n is constructed, and is the canonical graph of $\pi_n(T^n)$, with T^n an expansion of X_{j_0} such that $\pi_n(T^n)$ does not contain any variables. Now, if X_i is a variable occurring in T^n, expand X_i to a term S_i in which all variables occur guarded (S_i exists by 5.10). T^{n+1} is the result of substituting the S_i for each X_i occurring in T^n. Then T^{n+1} is an expansion of X_{j_0}, and $\pi_{n+1}(T^{n+1})$ does not contain any variables, so is a finite closed ACP$_\tau$-term. g_{n+1} is the canonical graph of $\pi_{n+1}(T^{n+1})$. Note that g_{n+1} is finite, and $\pi_n(g_{n+1}) = g_n$ (=,not just $\underline{\leftrightarrow}_{r\tau\delta}$!).
Now we define $g = \bigcup_{n=1}^{\infty} g_n$ (leaving out all $\downarrow$-labels in non-endpoints). Note that for each n, $\pi_n(g)=g_n$, and that g is finitely branching and bounded. It remains to be shown that g is a solution of E.
The same way we constructed $g=g_{j_0}$, we can construct graphs g_j for each $j \in J$. We will show that the graphs $\{g_j : j \in J\}$ satisfy all equations of E. Let $i_0 \in J$, and let equation E_{i_0} be

$$X_{i_0} = T_{i_0}(X_{i_1},\ldots,X_{i_m}),$$

where $X_{i_1},\ldots,X_{i_m}$ are the variables occurring in T_{i_0}. We have to show

$$g_{i_0} \underline{\leftrightarrow}_{r\tau\delta} T_{i_0}(g_{i_1},\ldots,g_{i_m}).$$

We do this by AIP (4.6 applies since g_{i_0} is finitely branching and bounded). So fix $n \in \mathbb{N}$. Let, for $0 \leqslant k \leqslant m$, $T^n_{i_k}$ be an expansion of X_{i_k} such that $\pi_n(T^n_{i_k})$ contains no variables and $\pi_n(g_{i_k})$ is its canonical graph. Then

$$\pi_n(T_{i_0}(g_{i_1},\ldots,g_{i_m})) =$$

$$= \pi_n(T_{i_0}(\pi_n(g_{i_1}),\ldots,\pi_n(g_{i_m}))) \quad \text{(use definition 3.9.8)} =$$

$$= \pi_n(T_{i_0}(\pi_n(T^n_{i_1}),\ldots,\pi_n(T^n_{i_m}))) \quad \text{(by assumption)} =$$

$= \pi_n(T_{i_0}(T^n_{i_1},...,T^n_{i_m}))$ (again by 3.9.8) $=$

$= \pi_n(T^n_{i_0})$ (by construction of $T^n_{i_0}$) $=$

$= \pi_n(g_{i_0})$ (by assumption).

This finishes the proof.

5.12 THEOREM: *Let E be a guarded recursive specification in which no abstraction operator occurs. Then, in the model $\mathbb{G}_\kappa / \leftrightarrows_{r\tau\delta}$, E has a unique solution.*

PROOF: By 5.11, E has a solution g which is finitely branching and bounded. Let h be any other solution of E. We will show $g \leftrightarrows_{r\tau\delta} h$ by AIP. So let $n \in \mathbb{N}$, and let T^n be an expansion of X_{j_0} so that $\pi_n(g) = \pi_n(T^n)$. On the other hand, if $h = h_{j_0}$ solves E with parameters $\{h_j : j \in J, j \neq j_0\}$, and T_{j_0} has variables $X_{j_1},...,X_{j_m}$,
then

$h \leftrightarrows_{r\tau\delta} T_{j_0}(h_{j_1},...,h_{j_m})$(for h is a solution)

$\leftrightarrows_{r\tau\delta} T_{j_0}(T_{j_1}(\vec{h}),...,T_{j_m}(\vec{h}))$

(for the same reason, for some sequences $\vec{h}$ from $\{h_j : j \in J\}$)

$\leftrightarrows_{r\tau\delta} \cdots$

$\cdots$

$\leftrightarrows_{r\tau\delta} T^n(\vec{h})$ (for some sequence $\vec{h}$),

whence $\pi_n(h) \leftrightarrows_{r\tau\delta} \pi_n(T^n(\vec{h})) = \pi_n(T^n(\vec{X})) = \pi_n(T^n)$.
Now we can give the following algebraical formulation of AIP, which holds in the model $\mathbb{G}_\kappa / \leftrightarrows_{r\tau\delta}$.

5.13 THEOREM: $\mathbb{G}_\kappa / \leftrightarrows_{r\tau\delta}$ *satisfies the following principle, which we will call* AIP:

(AIP)	for all n $\pi_n(x) = \pi_n(y)$ x is specifiable by a guarded E without τ_I
	$x = y$

PROOF: If x is the solution of a guarded recursive specification in which no abstraction operator occurs, in the model it is the equivalence class of a finitely branching and bounded graph, by 5.11 and 5.12, which satisfies AIP by 4.6.

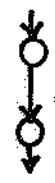
It is a drawback of the previous theorems that we cannot use abstractions in our specifications. We can partially remedy this deficiency, however, by introducing a *hiding* operator t_I. This we do in 5.14.

5.14 DEFINITION: We define an auxiliary theory ACP^t_τ as follows:
1. ACP^t_τ extends ACP_τ;
2. ACP^t_τ has a new atom $t \in A$ with $t|a = \delta$ for all $a \in A$.
3. ACP^t_τ has a new operator t_I (where $I \subseteq A_\tau - \{\delta\}$) defined by the four equations in table 5.

$t_I(a) = a$	if $a \notin I$
$t_I(a) = t$	if $a \in I$
$t_I(x+y) = t_I(x) + t_I(y)$	
$t_I(xy) = t_I(x) \cdot t_I(y)$	

TABLE 5.

(here $a \in A_\tau$, so $a = \tau$ or $a = t$ is possible, and x,y are processes over ACP_τ^t; compare 2.10 in BAETEN, BERGSTRA & KLOP [2]).

5.15 DEFINITION: we extend $\mathbb{G}_\kappa$ with a new element  t

(t a new label) and we define t_I on $\mathbb{G}_\kappa$ by stipulating that $t_I(g)$ is the graph g with all labels from I changed to t.

5.16 *Note*: theorem 5.12 still holds for specifications E in which a hiding operator t_I occurs. This is not hard to see.

5.17 COROLLARY: *$\mathbb{G}_\kappa / \leftrightarrow_{r\tau\delta}$ satisfies the following principles, which we will call* RDP *and* RSP:

$$\boxed{(\mathrm{RDP}) \quad \frac{E \text{ guarded, no } \tau_I.}{\exists x\ E(x,-)}}$$

$$\boxed{(\mathrm{RSP}) \quad \frac{E(x,-) \quad E(y,-) \quad E \text{ guarded, no } \tau_I.}{x = y}}$$

§6 COMPUTABLE GRAPHS

In this paragraph, we look at computable graphs. We will prove that every computable finitely branching graph is definable by a finite guarded specification in the language of ACP_τ. We will prove this result via a number of intermediate results. First we define what we mean by a computable graph. In a computable graph, one must know at every point how many possibilities there are to proceed, and the label of each of those possibilities. Therefore, we need two computable functions od (for out-degree) and lb (for label). Since these must be number-theoretic functions, we need some coding of graphs. We do this by numbering the edges starting from each node. It also follows that we have to restrict ourselves to finitely branching graphs (although countably branching graphs could possibly also be considered).

6.1 DEFINITION: Let $g \in \mathbb{G}_{\aleph_0}$ (so g is finitely branching). A *coding* of g consists of the following:
1. if $s \in$ NODES(g), and the out-degree of s is n, then the outgoing edges are named $0,1,\ldots,n-1$.
2. this leads to the following naming of nodes: a sequence $\sigma \in \omega^*$ names the node reached by following the path from ROOT(g) with edge-names in σ.

6.2 EXAMPLE: let g be the graph of Fig.30 with indicated coding.

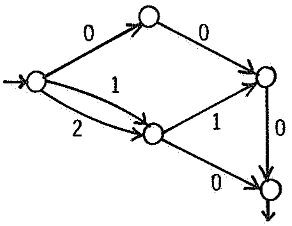


Fig.30

ROOT (g) has name ϵ and the endpoint of g has names 000, 10, 110, 20 and 210.

6.3 *Note*: $g \in \mathbb{G}_{\aleph_0}$ is a tree $\Leftrightarrow$ each node has exactly one name.

6.4 DEFINITION: Let $g \in \mathbb{G}_{\aleph_0}$ be coded. We define two partial functions:

$$od : \omega^* \longrightarrow \omega$$

$$lb : \omega^* \longrightarrow A \cup \{\delta, \downarrow\},$$

as follows:

1. $od(\sigma) =$ the out-degree of the node named by σ, if σ names a node;
2. $od(\sigma)$ is undefined otherwise.
3. $lb(\sigma * n) =$ the label of edge n starting at node σ if σ names a node and $n < od(\sigma)$ (here $\sigma * n$ is sequence σ followed by number n);
4. $lb(\sigma * 0) =$ the label of endnode σ if σ names a node and $od(\sigma)=0$;
5. $lb(\sigma)$ is undefined otherwise.

6.5 DEFINITION: $g \in \mathbb{G}_{\aleph_0}$ is *computable* if there is a coding of g such that functions od and lb are computable (since the set A is assumed to be finite, a coding of $A \cup \{\delta, \downarrow\}$ into ω is not important).

Now we start the proof of the main theorem of this paragraph. The first step towards proving it will be to show that every computable function can be represented by a finite guarded specification. First we say what we mean by a representation.

6.6 Let D be a finite set of data. We suppose we have a number of *communication channels* $0,1,...,k$ $(k \geqslant 1)$, of which channel 0 is the *input channel* and channel 1 the *output channel*. Any other channel is an *internal channel*. Furthermore, we suppose our set of atoms A contains elements

1. $s_i(d) =$ *send d along channel i* $(d \in D, i \leqslant k)$;
2. $r_i(d) =$ *receive d along channel i* $(d \in D, i \leqslant k)$;
3. $c_i(d) =$ *communicate d along channel i* $(d \in D, i \leqslant k)$.

On these elements, we define the communication function by

$$s_i(d) | r_i(d) = c_i(d)$$

and all other communications give δ.

Now suppose $f : D^* \longrightarrow D^*$ is a partial function. We say process $\hat{f}$ *represents* f iff for any $\sigma, \rho \in D^*$ $f(\sigma) = \rho \Leftrightarrow$ inputting sequence σ along channel 0 will be followed by outputting sequence ρ along channel 1; and $f(\sigma)$ is undefined $\Leftrightarrow$ inputting sequence σ along channel 0 will be followed by deadlock. To be more precise, suppose a sequence $\sigma = d_1....d_n$ is given, and we have a marker *eos* indicating the end of a sequence.

We define the *sender* $\mathbb{S}_\sigma = s_0(d_1)s_0(d_2)\cdot\ldots\cdot s_0(d_n)\cdot s_0(eos)$ and the *receiver* $\mathbb{R}$ by the following finite guarded specification (which has a unique solution in $\mathbb{G}_\kappa / \underline{\leftrightarrow}_{r\tau\delta}$ by theorem 5.12):

$$\boxed{\mathbb{R} = \sum_{d\in D} r_1(d)\cdot\mathbb{R} + r_1(eos)}$$

Then, we will hide unsuccessful communications:

$$H' = \{s_i(d), r_i(d) |\ d \in D \cup \{eos\},\ i = 0,1\},$$

and now we can give the formal definition: process $\hat{f}$ *represents* function f iff for any $\sigma,\rho \in D^*$, say $\sigma = d_1,\ldots d_n$, $\rho = e_1 \ldots e_m$ (with $n,m \geqslant 0$):

1. $f(\sigma) = \rho \Leftrightarrow \partial_{H'}(\mathbb{S}_\sigma \| \hat{f} \| \mathbb{R}) =$

$$c_0(d_1)\cdot c_0(d_2)\cdot\ldots\cdot c_0(d_n)\cdot c_0(eos)\cdot c_1(e_1)\cdot\ldots\cdot c_1(e_m)\cdot c_1(eos),$$

2. $f(\sigma)$ is undefined $\Leftrightarrow \partial_{H'}(\mathbb{S}_\sigma \| \hat{f} \| \mathbb{R}) =$

$$c_0(d_1)\cdot c_0(d_2)\cdot\ldots\cdot c_0(d_n)\cdot c_0(eos)\cdot\delta.$$

6.7 THEOREM: *Let $f : \omega^* \longrightarrow \omega^*$ be a partial computable function. Then f can be represented by a process, defined using a finite guarded recursive specification.*

PROOF: Let f be given. It is well-known that f can be represented by a Turing Machine over a finite alphabet D with finitely many states $0,\ldots,k$, $(k \geqslant 1)$ of which 0 is the starting state and k the ending state. In turn, we will simulate this Turing Machine by a finite specification

$$x = t_I \circ \partial_H\ (C \| S_2 \| S_3), \text{ namely } \hat{f} = \tau_{\{t\}}(x).$$

Here C is a finite control and S_2 and S_3 are stacks. We have the following picture (Fig.31)

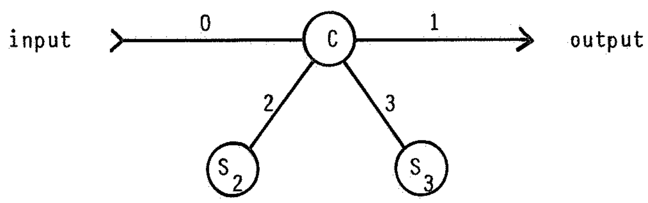


Fig.31

The specifications of S_2 and S_3 are

$$\boxed{\begin{aligned} S_i &= \sum_{d\in D\cup\{eos\}} r_i(d)T_i^d S_i + r_i(stop) \\ T_i^d &= s_i(d) + \sum_{e\in D\cup\{eos\}} r_i(e)T_i^e T_i^d \\ &\qquad\text{(for each } d \in D \cup \{eos\}) \end{aligned}} \quad (i = 2,3)$$

(see e.g. BERGSTRA & KLOP [10]), (the extra atom *stop* is needed for successful termination). C is specified using variables $C_0, C_1, \ldots, C_k, C_{k+1}, C_{k+2}$ (think of these C_i as the "states" of C, and $C_0,\ldots,C_k$ correspond to the states of the Turing Machine). The specification of C consists of three parts:

1. input, 2. calculation, 3. output.

Part 1. **input.**

$$\boxed{\begin{aligned} C &= r_0(eos)s_2(eos)s_3(eos)C_{k+2} + \sum_{d \in D} r_0(d)s_2(eos)s_2(d)C_{k+1} \\ C_{k+1} &= r_0(eos)s_3(eos)C_{k+2} + \sum_{d \in D} r_0(d)s_2(d)C_{k+1} \\ C_{k+2} &= r_2(eos)s_2(eos)C_0 + \sum_{d \in D} r_2(d)s_3(d)C_{k+2} \end{aligned}}$$

When C_0 is reached, the input sits in S_3 in the right order, and ends with a *eos* (end-of-stack).

Part 2. **calculation**

This specification will have one equation for each Turing Machine instruction in the Turing Machine representation of f.

a) for each TM instruction $i\ d\ e\ R\ m$ $(i<k, m \leqslant k; d,e \in D)$ (meaning that if in state i, the head reads d, it is changed to e, the head moves right and goes into state m), we have an equation

$$\boxed{C_i = r_3(d)s_2(e)C_m}$$

b) for each TM instruction $i\ d\ e\ L\ m$ $(i<k, m \leqslant k; d,e \in D)$ (the head moves left instead of right), we have an equation

$$\boxed{C_i = r_3(d)s_3(e)\sum_{f \in D} r_2(f)s_3(f)C_m}$$

Figs. 32 and 33 might clarify the matter: if the Turing Machine is in the position of Fig.32, control and stacks are as in Fig.33.

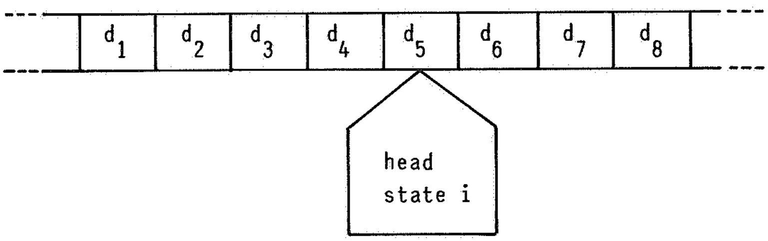


Fig.32

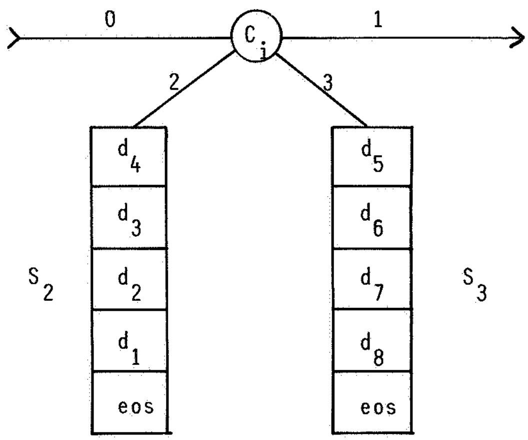


Fig.33

Part 3. **output**

When state C_k is reached, the output sits in S_3 in the right order, and S_2 is empty, so we put

$$\boxed{C_k = r_3(eos)r_2(eos)s_3(stop)s_2(stop)s_1(eos) + \sum_{d \in D} r_3(d)s_1(d)C_k}$$

This completes the specification of C.

Next we hide all unsuccessful communications by encapsulation: we define

$$H = \{s_i(d), r_i(d) : d \in D \cup \{eos, stop\}, i=2,3\}$$

and we hide all internal communications by abstraction: we define $I = \{c_i(d) : d \in D \cup \{eos, stop\}, i=2,3\}$, and consider $\hat{f} = \tau_{\{t\}}(x)$, where x is the unique solution of specification $X = t_I \circ \partial_H(C \| S_2 \| S_3)$. Informally, we will write

$$\hat{f} = \tau_I \circ \partial_H(C \| S_2 \| S_3).$$

Now we want to show that $\hat{f}$ indeed represents f, so let $\sigma \in D^*$ be given (instead of working with f we work with its Turing Machine representation). Let $H' = \{s_i(d), r_i(d) : d \in D \cup \{eos\}, i=0,1\}$ as in 6.6 and consider

$$\partial_{H'}(\mathbb{S}_\sigma \| \hat{f} \| \mathbb{R}).$$

Let $\sigma = d_1..d_n$ and let S_i^ρ denote stack S_i with contents $\rho \in D^*$ followed by eos.

Then $\partial_{H'}(\mathbb{S}_\sigma \| \hat{f} \| \mathbb{R}) =$

$$\partial_{H'}(\mathbb{S}_\sigma \underline{\|} (\hat{f} \| \mathbb{R})) + \partial_{H'}(\hat{f} \underline{\|} (\mathbb{S}_\sigma \| \mathbb{R})) + \partial_{H'}(\mathbb{R} \underline{\|} (\hat{f} \| \mathbb{S}_\sigma)) +$$

$$+ \partial_{H'}((\mathbb{S}_\sigma | \hat{f}) \underline{\|} \mathbb{R}) + \partial_{H'}((\mathbb{S}_\sigma | \mathbb{R}) \underline{\|} \hat{f}) + \partial_{H'}((\hat{f} | \mathbb{R}) \underline{\|} \mathbb{S}_\sigma)$$

(by the expansion theorem of 3.12) $=$

$$= \delta + \delta + \delta + c_0(d_1)\partial_{H'}(\mathbb{S}_{d_2 \cdots d_n} \| \tau_I \circ \partial_H((s_2(eos)s_2(d_1)C_{k+1}) \| S_2 \| S_3) \| \mathbb{R}) + \delta + \delta =$$

$$= c_0(d_1)\partial_{H'}(\mathbb{S}_{d_2 \cdots d_n} \| \tau_I(c_2(eos)c_2(d) \cdot \partial_H(C_{k+1} \| S_2^{d_1} \| S_3)) \| \mathbb{R}) =$$

$$= c_0(d_1)\tau \cdot \partial_{H'}(\mathbb{S}_{d_2 \cdots d_n} \| \tau_I \circ \partial_H(C_{k+1} \| S_2^{d_1} \| S_3) \| \mathbb{R}) = \ldots. =$$

$$= c_0(d_1)c_0(d_2)\ldots c_0(d_n) \cdot \partial_{H'}(s_0(eos) \| \tau_I \circ \partial_H(C_{k+1} \| S_2^{d_n \ldots d_1} \| S_3) \| \mathbb{R}) =$$

$$= c_0(d_1)\ldots.c_0(d_n)c_0(eos) \cdot \partial_{H'}(\tau_I \circ \partial_H(C_{k+2} \| S_2^{d_n \ldots d_1} \| S_3^{\varnothing}) \| \mathbb{R}) =$$

$$= c_0(d_1)....c_0(d_n)c_0(eos)\cdot\partial_{H'}(\tau_I(c_2(d_n)c_3(d_n)\cdot\partial_H(C_{k+2}\|S_2^{d_{n-1}..d_1}\|S_3^{d_n}))\|\mathbb{R}) =$$

$$=... = c_0(d_1)...c_0(d_n)c_0(eos)\cdot\partial_{H'}(\tau_I\circ\partial_H(C_0\|S_3^{\varnothing}\|S_3^{\sigma})\|\mathbb{R}).$$

So we have reached the calculation part of the specification. Now we have two cases, according to whether or not $f(\sigma)$ is defined.
case 1: $f(\sigma)$ is defined, say $f(\sigma)=\rho$.
We claim that then

$$\tau_I\circ\partial_H(C_0\|S_2^{\varnothing}\|S_3^{\sigma}) = \tau\tau_I\circ\partial_H(C_k\|S_2^{\varnothing}\|S_3^{\rho}).$$

This can be seen if we look at figs 32 and 33: each position of the Turing Machine is mirrored by a position of the specification: thus position

$$\boxed{\tau\cdot\tau_I\circ\partial_H(C_i\|S_2^{\sigma'}\|S_3^{d*\sigma''})}$$

$(i<k;\sigma',\sigma''\in D^*, d\in D)$ corresponds to the Turing Machine in state i with tape contents the reverse of σ' followed by d followed by σ'' and head pointing at position d. Thus all we have to show is that the TM instructions "do the correct thing".

a) suppose there is a TM instruction $i\ d\ e\ R\ m$.
Then $\tau\cdot\tau_I\circ\partial_H\ (C_i\|S_2^{\sigma'}\|S_3^{d*\sigma''}) =$

$$= \tau\cdot\tau_I(c_3(d)\cdot\partial_H((s_2(e)C_m)\|S_2^{\sigma'}\|S_3^{\sigma''}) =$$

$$= \tau\cdot\tau\cdot\tau_I(c_2(e)\cdot\partial_H(C_m\|S_2^{e*\sigma'}\|S_3^{\sigma''}) =$$

$$= \tau\cdot\tau_I\circ\partial_H(C_m\|S_2^{e*\sigma'}\|S_3^{\sigma''}).$$

b) suppose there is a TM instruction $i\ d\ e\ L\ m$
Then $\tau\cdot\tau_I\circ\partial_H(C_i\|S_2^{f*\sigma'}\|S_3^{d*\sigma''}) =$

$$= \tau\cdot\tau_I(c_3(d)\cdot\partial_H((s_3(e)\sum_{f\in D}r_2(f)s_3(f)C_m)\|S_2^{f*\sigma'}\|S_3^{\sigma''}) =$$

$$= \tau\cdot\tau_I(c_3(e)\cdot\partial_H((\sum_{f\in D}r_2(f)s_3(f)C_m)\|S_2^{f*\sigma'}\|S_3^{e*\sigma''}) =$$

$$= \tau\cdot\tau_I(c_2(f)\cdot\partial_H((s_3(f)C_m)\|S_2^{\sigma'}\|S_3^{e*\sigma''}) =$$

$$= \tau\cdot\tau_I(c_3(f)\cdot\partial_H(C_m\|S_2^{\sigma'}\|S_3^{f*e*\sigma''}) =$$

$$= \tau\cdot\tau_I\circ\partial_H(C_m\|S_2^{\sigma'}\|S_3^{f*e*\sigma''}).$$

Thus, since the Turing Machine terminates on input σ, with ρ on the tape, in state k, with the head pointing at the first symbol of ρ, we must have that

$$\tau_I\circ\partial_H(C_0\|S_2^{\varnothing}\|S_3^{\sigma}) = \tau\cdot\tau_I\circ\partial_H(C_k\|S_2^{\varnothing}\|S_3^{\rho}).$$

Then we can finish the calculation (let $\rho=e_1...e_m$)

$$\partial_{H'}([\tau\tau_I\circ\partial_H(C_k\|S_2^{\varnothing}\|S_3^{\rho})]\|\mathbb{R}) =$$

$$= \tau\cdot\partial_{H'}(\tau_I\circ\partial_H(C_k\|S_2^{\varnothing}\|S_3^{\rho})\|\mathbb{R}) =$$

$$= \tau\cdot c_1(e_1)\cdot\partial_{H'}(\tau_I\circ\partial_H(C_k\|S_2^{\varnothing}\|S_3^{e_2...e_m})\|\mathbb{R}) = ... =$$

$$= \tau c_1(e_1)...c_1(e_m)\partial_{H'}(\tau_I \circ \partial_H(C_k \| S_2^{\emptyset} \| S_3^{\emptyset}) \| \mathbb{R}) =$$

$$= \tau c_1(e_1)...c_1(e_m)\partial_{H'}(\tau_I(c_3(eos)c_3(eos)\partial_H([s_3(stop)s_2(stop)s_1(eos)] \| S_2 \| S_3)) \| \mathbb{R}) =$$

$$= \tau c_1(e_1)...c_1(e_m)\partial_{H'}(\tau \cdot \tau_I(c_3(stop)c_2(stop)s_1(eos)) \| \mathbb{R}) =$$

$$= \tau c_1(e_1)...c_1(e_m)\tau\partial_{H'}(s_1(eos) \| \mathbb{R}) =$$

$$= \tau c_1(e_1)...c_1(e_m)c_1(eos),$$

which finishes the proof of case 1.

case 2 $f(\sigma)$ is undefined.
In this case, the Turing Machine calculation does not terminate, state k will never be reached, and process

$$\tau_I \circ \partial_H \ (C_0 \| S_2^{\emptyset} \| S_3^{\sigma})$$

will do an infinite number of internal steps (steps from I). We will prove the following
Claim: $\tau_I \circ \partial_H(C_0 \| S_2^{\emptyset} \| S_3^{\sigma}) = \tau\delta$, which will finish the proof of case 2.
To prove this, we put $y = \partial_H(C_0 \| S_2^{\emptyset} \| S_3^{\sigma})$ and consider $x = t_I(y)$. Since the Turing Machine does not terminate, it will keep doing instructions

a) $i \ d \ e \ R \ m$
or b) $i \ d \ e \ L \ m.$
$(i,m<k \ ; \ d,e \in D)$.
A general step of type a) looks like:

$$t_I \circ \partial_H \ (C_i \| S_2^{\sigma'} \| S_3^{d*\sigma''}) =$$

$$t_I(c_3(d)c_2(e)\partial_H(C_m \| S_2^{e*\sigma'} \| S_3^{\sigma''}) =$$

$$tt \ t_I \circ \partial_H \ (C_m \| S_2^{e*\sigma'} \| S_3^{\sigma''}),$$

and a general step of type b) looks like:

$$t_I \circ \partial_H \ (C_i \| S_2^{f*\sigma'} \| S_3^{d*\sigma''}) =$$

$$t_I(c_3(d)c_3(e)c_2(f)c_3(f)\partial_H(C_m \| S_2^{\sigma'} \| S_3^{f*e*\sigma''})) =$$

$$tttt_I \circ \partial_H \ (C_m \| S_2^{\sigma'} \| S_3^{f*e*\sigma''}).$$

Thus, process $t_I(y) = t_I \circ \partial_H \ (C_0 \| S_2^{\emptyset} \| S_3^{\sigma})$
has states of the form

$$t_I \circ \partial_H(C_i \| S_2^{\sigma'} \| S_3^{\sigma''})$$

and will do 2 or 4 t-steps to go from one such state to the next. From this, we conclude that for each n

$$\pi_n(t_I(y)) = t^n.$$

Now consider specification

$$X = tX.$$

This is a finite guarded specification with no abstraction operator, so it has a unique solution by RDP+RSP, to which AIP applies.
We call this process t^{ω}. It is easy to see that $\pi_n(t^{\omega}) = t^n$ for each n, so applying AIP (thm. 5.13) we obtain

$$t_I(y) = t^{\omega},$$

so

$$t_I(y) = t \cdot t_I(y),$$

because $t_I(y)$ will satisfy the specification of t^ω. From this last equation, it follows by $KFAR_1$ that $\tau_I(y) = \tau_{\{t\}} \circ t_I(y) = \tau \cdot \tau_{\{t\}}(\delta) = \tau\delta$, which proves the claim, and at the same time ends the proof of theorem 6.7.

Thus, every computable function can be represented using a finite guarded specification. We want to prove that every computable graph is definable using a finite guarded specification, but we will first prove this with two extra restrictions: the graph must be bounded and binary (i.e. an element of $\mathbb{G}_3$).

6.8 THEOREM: *Let $g \in \mathbb{G}_3$ be computable and bounded. Then $g = \tau_{\{t\}}(h)$, with h the solution of a finite guarded recursive specification.*

PROOF: Code g such that functions od and lb, defined in 6.4, are computable. Let $\hat{od}$ and $\hat{lb}$ be process representations of od, lb (defined in 6.7).

First we will give an infinitary specification of g.
We have a state X_σ for each name σ of a node which is not a $\downarrow$-endpoint (so our index set is the set of all $\sigma \in \{0,1\}^*$ with $od(\sigma)>0$ or $lb(\sigma * 0) = \delta$, with designated element ϵ, a name of the root).
We have 7 cases:

1. $od(\sigma) = 0$, so $lb(\sigma * 0) = \delta$.
   Then $X_\sigma = \delta$.
2. $od(\sigma) = 1$, and $od(\sigma * 0)>0$ or $lb(\sigma * 0 * 0) = \delta$.
   Then $X_\sigma = lb(\sigma * 0) X_{\sigma * 0}$.
3. $od(\sigma) = 1$, and $lb(\sigma * 0 * 0) = \downarrow$.
   Then $X_\sigma = lb(\sigma * 0)$.
4. $od(\sigma) = 2$, both ($od(\sigma * 0)>0$ or $lb(\sigma * 0 * 0) = \delta$) and ($od(\sigma * 1)>0$ or $lb(\sigma * 1 * 0) = \delta$).
   Then $X_\sigma = lb(\sigma * 0) X_{\sigma * 0} + lb(\sigma * 1) X_{\sigma * 1}$.
5. $od(\sigma) = 2$, and ($od(\sigma * 0)>0$ or $lb(\sigma * 0 * 0) = \delta$) but $lb(\sigma * 1 * 0) = \downarrow$.
   Then $X_\sigma = lb(\sigma * 0) X_{\sigma * 0} + lb(\sigma * 1)$
6. $od(\sigma) = 2$, and $lb(\sigma * 0 * 0) = \downarrow$ but ($od(\sigma * 1)>0$ or $lb(\sigma * 1 * 0) = \delta$).
   Then $X_\sigma = lb(\sigma * 0) + lb(\sigma * 1) X_{\sigma * 1}$.
7. $od(\sigma) = 2$, and $lb(\sigma * 0 * 0) = lb(\sigma * 1 * 0) = \downarrow$.
   Then $X_\sigma = lb(\sigma * 0) + lb(\sigma * 1)$.

It is not hard to see that g is indeed the solution of this specification, with parameters which we will call x_σ (we have guardedness since g is bounded). Now we want to give a finite specification for g.
We will describe three parts:
Part 1: the transition from X_σ to $X_{\sigma * i}$ ($i = 0,1$), execution of steps;
Part 2: the history, saved in a stack;
Part 3: the calculation, containing $\hat{od}$ and $\hat{lb}$.

We have the following configuration (Fig.34):

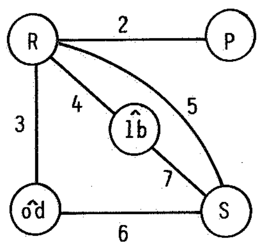


Fig.34

We have channels 2,3,4,5,6,7 (all internal) and we extend the alphabet A_τ by:
1. $\{s_2(d), r_2(d), c_2(d) : d \in A_\tau^2 \cup A \cup \{\tau, \downarrow\} \cup \{0,1\}\}$
2. $\{s_3(d), r_3(d), c_3(d) : d \in \{start, stop, 0,1,2\}\}$
3. $\{s_4(d), r_4(d), c_4(d) : d \in \{start, stop\} \cup A \cup \{\tau, \downarrow\}\}$
4. $\{s_5(d), r_5(d), c_5(d) : d \in \{stop, 0, 1, eos\}\}$
5. $\{s_6(d), r_6(d), c_6(d) : d \in \{0,1,eos\}\}$
6. $\{s_7(d), r_7(d), c_7(d) : d \in \{0,1,eos\}\}$.

Part 1. description of P

P has states P, P_a for $a \in A_\tau$ and $P_{<a,b>}$ for $a,b \in A_\tau - \{\delta\}$, with the following specification:

$$\begin{aligned}
P &= \sum_{a,b \in A_\tau - \{\delta\}} r_2(<a,b>)P_{<a,b>} + \sum_{a \in A_\tau} r_2(a)P_a + r_2(\downarrow). \\
P_{<a,b>} &= as_2(0)P + bs_2(1)P \\
P_a &= as_2(0)P \qquad \text{(if } a \neq \delta) \\
P_\delta &= \delta
\end{aligned}$$

Part 2: description of S

S is a stack that keeps track of the history up to the point reached, and has states S, T_0, T_1, with the following specification: $(k = 5,6,7)$

$$\begin{aligned}
S &= (s_k(eos) + r_k(0)T_0 + r_k(1)T_1)S + r_5(stop) \\
T_i &= s_k(i) + \sum_{j=0,1} r_k(j)T_jT_i + r_5(stop) \qquad (i = 0,1)
\end{aligned}$$

Part 3. description of $\hat{od}$, $\hat{lb}$, R

We assume $\hat{od}$ and $\hat{lb}$ are specifications as given in 6.7, that work as follows:
$\hat{od}$ has input channel 6 and output channel 3;
$\hat{lb}$ has input channel 7 and output channel 4;
upon receiving a signal *start* from R, they will read the contents σ of stack S, return those data to the stack, calculate $od(\sigma)$ respectively $lb(\sigma)$ and sent the result to R.
Thus, after abstraction from channels 5 and 6, we have (let S contain σ):

$$\hat{od} = r_3(start)s_3(od(\sigma))\hat{od} + r_3(stop)$$

$$\hat{lb} = r_4(start)s_4(lb(\sigma))\hat{lb} + r_4(stop)$$

R is the finite control, and is given by the following equation:

$$R = s_3(start)\ [r_3(0)s_5(0)s_4(start) \sum_{l=\delta,\downarrow} r_4(l)s_2(l)s_3(stop)s_4(stop)s_5(stop)] +$$
$$+ [r_3(1)s_5(0)s_4(start) \sum_{l\in A_\tau-\{\delta\}} r_4(l)s_2(l)r_2(0) +$$
$$+ r_3(2)s_5(0)s_4(start) \sum_{l\in A_\tau-\{\delta\}} r_4(l)r_5(0)s_5(1)s_4(start)$$
$$\sum_{l'\in A_\tau-\{\delta\}} r_4(l')r_5(1)s_2(<l,l'>) \sum_{i=0,1} r_2(i)s_5(i)]\ R$$

Next we do encapsulation:
$H = \{r_i(d), s_i(d) : i=2,..,7\ ;\ d$ from appropriate sets$\}$ and abstraction:
$I = \{c_i(d) : i=2,..,7\ ;\ d$ from appropriate sets$\}$. Now let S^σ denote stack S with contents σ, then we can define processes $\{y_\sigma : \sigma$ a node-name$\}$ by the following equation

$$Y'_\sigma = t_I \circ \partial_H (P \| S^\sigma \| R \| \hat{od} \| \hat{lb})\ ,\ y_\sigma = \tau_{\{t\}}(y'_\sigma)$$

(this equation indeed defines a process, since all equations for $P, S, R, \hat{od}, \hat{lb}$ are guarded).

Claim: $y_\sigma = \tau x_\sigma$.

PROOF: We show processes y_σ satisfy the 7 defining equations for x_σ, multiplied by τ.

1. $od(\sigma) = 0$, so $lb(\sigma * 0) = \delta$.
   Then $y_\sigma = \tau_I \circ \partial_H\ (P \| S^\sigma \| R \| \hat{od} \| \hat{lb}) =$
   $$= \tau_I(c_3(start)c_3(0)c_5(0)c_4(start)c_4(\delta)c_2(\delta)$$
   $$c_3(stop)c_4(stop)c_5(stop)\delta) = \tau\delta.$$
2. $od(\sigma) = 1$, and $(od(\sigma * 0) > 0$ or $lb(\sigma * 0 * 0) = \delta)$.
   Then $y_\sigma = \tau_I \circ \partial_H\ (P \| S^\sigma \| R \| \hat{od} \| \hat{lb}) =$
   $$= \tau_I(c_3(start)c_3(1)c_5(0)c_4(start)c_4(lb(\sigma * 0))c_2(lb(\sigma * 0))$$
   $$\partial_H(P_{lb(\sigma * 0)} \| S^{\sigma * 0} \| r_2(0)R \| \hat{od} \| \hat{lb})) =$$
   $$= \tau \cdot \tau_I(lb(\sigma * 0)c_2(0) \cdot \partial_H(P \| S^{\sigma * 0} \| R \| \hat{od} \| \hat{lb})) =$$
   $$= \tau lb(\sigma * 0)\tau_I \circ \partial_H\ (P \| S^{\sigma * 0} \| R \| \hat{od} \| \hat{lb}) =$$

$= \tau lb(\sigma * 0) y_{\sigma * 0}.$

3. $od(\sigma) = 1$, and $lb(\sigma * 0 * 0) = \downarrow$.
Then $y_\sigma = \tau_I \circ \partial_H (P \| S^\sigma \| R \| \hat{od} \| \hat{lb}) =$

$$= \tau lb(\sigma * 0) y_{\sigma * 0} = \tau lb(\sigma * 0) \tau_I \circ \partial_H (P \| S^{\sigma * 0} \| R \| \hat{od} \| \hat{lb}) =$$

$$= \tau lb(\sigma * 0) \tau_I (c_3(start) c_3(0) c_5(0) c_4(start) c_4(\downarrow) c_2(\downarrow)$$

$$c_3(stop) c_4(stop) c_5(stop)) =$$

$$= \tau lb(\sigma * 0) \tau = \tau lb(\sigma * 0).$$

4. $od(\sigma) = 2$, both ($od(\sigma * 0) > 0$ or $lb(\sigma * 0 * 0) = \delta$) and ($od(\sigma * 1) > 0$ or $lb(\sigma * 1 * 0) = \delta$).
Then $y_\sigma = \tau_I \circ \partial_H (P \| S^\sigma \| R \| \hat{od} \| \hat{lb}) =$

$$= \tau_I (c_3(start) c_3(2) c_5(0) c_4(start) c_4(lb(\sigma * 0)) c_5(0) c_5(1)$$

$$c_4(start) c_4(lb(\sigma * 1)) c_5(1) c_2(< lb(\sigma * 0), lb(\sigma * 1) >)$$

$$\partial_H (P_{< lb(\sigma * 0), lb(\sigma * 1) >} \| S^\sigma \| (\sum_{i=0,1} r_2(i) s_5(i) R) \| \hat{od} \| \hat{lb})) =$$

$$= \tau \cdot \tau_I (lb(\sigma * 0) c_2(0) c_5(0) \partial_H (P \| S^{\sigma * 0} \| R \| \hat{od} \| \hat{lb}) +$$

$$+ lb(\sigma * 1) c_2(1) c_5(1) \partial_H (P \| S^{\sigma * 1} \| R \| \hat{od} \| \hat{lb})) =$$

$$= \tau (lb(\sigma * 0) y_{\sigma * 0} + lb(\sigma * 1) y_{\sigma * 1}).$$

5. $od(\sigma) = 2$ and ($od(\sigma * 0) > 0$ or $lb(\sigma * 0 * 0) = \delta$)
but $lb(\sigma * 1 * 0) = \downarrow$

Then $y_\sigma = \tau (lb(\sigma * 0) y_{\sigma * 0} + lb(\sigma * 1) y_{\sigma * 1}) =$

$$= \tau (lb(\sigma * 0) y_{\sigma * 0} + lb(\sigma * 1) \tau) =$$

$$= \tau (lb(\sigma * 0) y_{\sigma * 0} + lb(\sigma * 1)).$$

6 and 7. likewise.
Now we will give a finite guarded recursive specification with a unique solution h, so that $g = \tau_{\{t\}}(h)$. We have three cases (X is the designated element).

case 1: $od(\epsilon) = 0$. The root has out-degree 0, so since graph is not in $\mathbb{G}_\kappa$, we have $g = \downarrow\delta$, and we can define

$$X = \delta.$$

case 2: $od(\epsilon) = 1$. Suppose $lb(0) = a$. Then

$$X = a t_I \circ \partial_H (P \| T_0 \| R \| \hat{od} \| \hat{lb}). = a Y_0$$

case 3: $od(\epsilon) = 2$. Suppose $lb(0) = a$ and $lb(1) = b$. Then

$$X = a t_I \circ \partial_H (P \| T_0 \| R \| \hat{od} \| \hat{lb}) + b t_I \circ \partial_H (P \| T_1 \| R \| \hat{od} \| \hat{lb}).$$

We see that this is a finite guarded specification. Moreover, since $y_\sigma = \tau x_\sigma$, it is clear that $\tau_{\{t\}}(h)$ satisfies the equation for X_ϵ, whence $g \leftrightarrow_{r\tau\delta} \tau_{\{t\}}(h)$. This finishes the proof of theorem 6.8.

6.9 COROLLARY: *Let $g \in \mathbb{G}_3$ be computable. Then $g = \tau_I(k)$, where k is recursively definable by a finite guarded specification.*

PROOF: Put $h = t'_{\{\tau\}}(g)$, the graph with all τ-labels replaced by t'-labels, where t' is some new atom. Since h is computable, binary but also bounded, by 6.8 there is a specification E with unique solution k such that $h \stackrel{\leftrightarrow}{=}_{r\tau\delta} \tau_{\{t\}}(k)$.
It follows easily that

$$g \stackrel{\leftrightarrow}{=}_{r\tau\delta} \tau_{\{t'\}}(h) \stackrel{\leftrightarrow}{=}_{r\tau\delta} \tau_{\{t,t'\}}(k).$$

Thus, we removed the restriction, that g must be bounded. Next, we will remove the restriction that g must be binary. First we need a lemma.

6.10 LEMMA: *Let $g \in \mathbb{G}_{\aleph_0}$. Then $g \stackrel{\leftrightarrow}{=}_{r\tau\delta} h$, for some $h \in \mathbb{G}_{\aleph_0}$ of which all non-root nodes have out-degree 0 or 2. If moreover g is computable, h is also computable.*

PROOF: We can assume that g is root-unwound (so $g \in \mathbb{G}^{\varrho}_{\aleph_0}$), and coded (see 6.1). We define h as follows:

1. $\mathrm{NODES}(h) = \{\langle s,n\rangle : s \in \mathrm{NODES}(g),\ s \neq \mathrm{ROOT}(g),\ n < \text{out-degree}(s)\}$
   $\cup \{\langle s,0\rangle : s \in \mathrm{NODES}(g)$, and $s = \mathrm{ROOT}(g)$ or $\text{out-degree}(s) = 0\}$.
2. $\mathrm{EDGES}(h) = \{\ \langle s,0\rangle \xrightarrow[n]{l} \langle t,0\rangle : s \xrightarrow[n]{l} t \in \mathrm{EDGES}(g), s = \mathrm{ROOT}(g)$
   ($n < od(s)$ the name of the edge, l a label)$\}$ $\cup$
   $\cup \{\ \langle s,n\rangle \xrightarrow[0]{l} \langle t,0\rangle : s \xrightarrow[n]{l} t \in \mathrm{EDGES}(g), s \neq \mathrm{ROOT}(g)$
   (n,l as above)$\}$ $\cup$
   $\cup \{\ \langle s,n\rangle \xrightarrow[1]{\tau} \langle s,n+1\rangle : s \in \mathrm{NODES}(g), s \neq \mathrm{ROOT}(g)$
   $(n+1) < od(s)$, l a label$\}$ $\cup$
   $\cup \{\ \langle s,n\rangle \xrightarrow[1]{\tau} \langle s,0\rangle : s \in \mathrm{NODES}(g), s \neq \mathrm{ROOT}(g)$,
   $(n+1) = od(s)$, l a label$\}$.
3. $\mathrm{ROOT}(h) = \langle \mathrm{ROOT}(g),0\rangle$
4. the endpoint label of $\langle s,0\rangle \in \mathrm{NODES}(h)$ is the endpoint label of $s \in \mathrm{NODES}(g)$.

An example might clarify the matter (Fig.35):

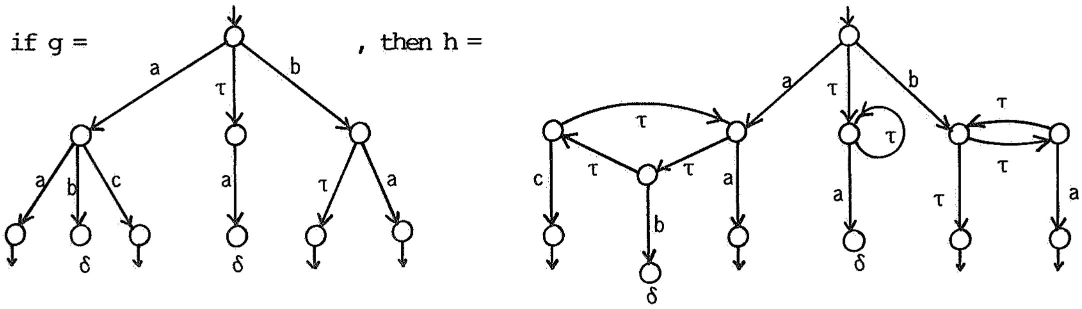


Fig.35

It is obvious that h is root-unwound, that all non-root nodes have out-degree 2 or 0, and that if g is computable, then so is h. Now we can define $R \subseteq \mathrm{NODES}(g) \times \mathrm{NODES}(h)$ as follows: R relates all nodes $s \in \mathrm{NODES}(g)$ with all $\langle s,n\rangle \in \mathrm{NODES}(h)$ ($n < od(s)$ or $n = 0 = od(s)$).
It is easy to prove that $R : g \stackrel{\leftrightarrow}{=}_{r\tau\delta} h$:

1. if $(s) \xrightarrow[n]{l} (t)$ is an edge in g with label l ($n < od(s)$) and $R(s, \langle s,k \rangle)$, then

1.1 if $k \leqslant n$, take path

$$(\langle s,k \rangle) \xrightarrow[1]{\tau} (\langle s,k+1 \rangle) \xrightarrow[1]{\tau} \dots \xrightarrow[1]{\tau} (\langle s,n \rangle) \xrightarrow[0]{l} (\langle t,0 \rangle)$$

in h with A-label l and $R(t, \langle t,0 \rangle)$;

1.2 if $k > n$, take path

$$(\langle s,k \rangle) \xrightarrow[1]{\tau} \dots \xrightarrow[1]{\tau} (\langle s,od(s)-1 \rangle) \xrightarrow[1]{\tau} (\langle s,0 \rangle) \xrightarrow[1]{\tau} \dots \xrightarrow[1]{\tau} (\langle s,n \rangle) \xrightarrow[0]{l} (\langle t,0 \rangle)$$

in h with A-label l and $R(t, \langle t,0 \rangle)$.

2. Conversely, for each edge $(\langle s,n \rangle) \xrightarrow[0]{l} (\langle t,0 \rangle)$ in h we have $(s) \xrightarrow[n]{l} (t)$ in g.
3. Endpoints and root are all right, since nothing is changed there.

6.11 **Theorem**: *Let g be a computable graph. Then $g = \tau_{\{t\}}(h)$, where h is recursively definable by a finite guarded specification.*

Proof: By 6.10, we can assume that all non-root nodes of g have out-degree 2 or 0. Put $h = t_{\{\tau\}}(g)$, and code h such that functions od, lb for h are computable, with process representations $\hat{od}$, $\hat{lb}$.
Let the root have out-degree $n_0 > 0$ (if $n_0 = 0, h = \delta$). For all non-root nodes, we will use the specifications for P, S, R given in 6.8, with the only difference that the first element of stack S can be any number up to n_0.
Then h is given by the following specification E:

$X = \sum_{i < n_0} lb(i) \cdot t_I \circ \partial_H (P \Vert T_i \Vert R \Vert \hat{od} \Vert \hat{lb})$.
$P, S, T_i, R, \hat{od}, \hat{lb}, H, I$ given in 6.8

We see that E is finite and guarded, and that h is a solution of E, using 6.8 and 6.9.

6.12 *Note*: When we want to translate the trick of 6.10 in the graph-model to the theory of ACP$_\tau$, we use KFAR. The details of this translation are not clear, however.

§7. Computably recursively definable processes

In §6, we looked at computable graphs. In this paragraph, we discuss computable recursive specifications, and show that any process, recursively definable by a computable specification is already definable by a finite specification. First a remark about coding:

7.1 *Coding*: it is not hard to give a computable injective coding function with computable inverse from all finite ACP$_\tau$-terms to natural numbers, so we will not mention this function in the following.

7.2 **Definition**: Let $E = \{E_n : n < \omega\}$ be a specification. E is *computable* if the function

$$f : n \mapsto T_n$$

is computable (T_n is the right-hand side of the equation for X_n).

7.3 LEMMA: *Let E be a computable guarded recursive specification, in which no abstraction operator occurs. Then, for each $n<\omega$, we can computably find an expansion of T_n in which each occurring variable is guarded.*

PROOF: In a finite ACP$_\tau$-term, it is easy to compute which variables are guarded, and which are not, using definition 5.5. Therefore, we can compute a guarded expansion of each T_n as in the proof of lemma 5.10.

7.4 LEMMA: *Let E be a computable guarded recursive specification, in which no abstraction operator occurs. Then E has a computable solution in $\mathbb{G}_{\aleph_0}$.*

PROOF: First, note that all graph operations defined in 3.9 are computable, so that if graphs g,h are computable (as defined in 6.5), then so are graphs $g + h, g \cdot h, g \| h, g \mathbin{\|\!\_} h, g | h, \partial_H(g), \tau_I(g), \pi_n(g)$ and $t_I(g)$ (defined in 5.15). Thus, we see that the canonical graph of each finite ACP$_\tau$-term is computable, so we obtain from the proof of 5.11 and lemma 7.3 that each computable guarded specification without abstraction has a computable solution.

7.5 COROLLARY: *if x is a process such that $x = \tau_I(y)$, where y is the solution of a computable guarded specification without abstraction, then also $x = \tau_{I'}(z)$, where z is the solution of a finite guarded specification without abstraction.*

PROOF: 6.11 plus 7.4.

§8. THE ROLE OF ABSTRACTION

In this last section, we show that the abstraction operator τ_I plays an essential role in the previous sections. In particular, we show that theorem 7.5 does not hold if we cannot use abstraction. Our conclusion is, that the defining power of theory ACP$_\tau$ is much greater than the defining power of theory ACP (where ACP is the theory given by the left-hand column of table 1 on page 3).

8.1 Let the set of atoms A contain two elements a,b different from δ. Let a function

$$f : \omega \longrightarrow \{a,b\}$$

be given. We define a recursive specification $E^f = \{E_n^f : n<\omega\}$ by:

$$E_n^f = f(n)E_{n+1}^f.$$

It is obvious that E^f is a guarded specification without abstraction, which is computable if f is computable. E^f has a unique solution by RDP + RSP, which we call x^f ($x^f = f(0)f(1)f(2)....$). By theorem 7.5, each x^f for computable f is the abstraction of a process, definable by a finite guarded specification without abstraction.

8.2 THEOREM: *there exists a computable function*

$$f : \omega \longrightarrow \{a,b\}$$

such that process x^f (defined in 8.1) is not recursively definable by a finite guarded specification in which no abstraction operator occurs.

PROOF: We can enumerate all finite guarded specifications without abstraction in a list

$<E_n : n<\omega>$. By 5.11, we can, for each $n<\omega$, construct a graph $g_n \in \mathbb{G}_{\aleph_0}$, of which all levels are finite, such that g_n is a solution of E_n. By 7.4, each g_n is computable. Now, to each specification E_n ($n<\omega$) we assign a function $f_n : \omega \longrightarrow \{a,b\}$ in the following way:

- $f_n(k) = a$ if all edges in g_n starting from a node at depth k have label a;
- $f_n(k) = b$ otherwise.

Since all g_n have all levels finite, it follows that all f_n are computable functions. Now, it follows immediately that if E_n defines a process x^f, it must be x^{f_n}. Thus, the set of all processes x^f recursively definable by a finite guarded specification without abstraction is included in $\{x^{f_n} : n<\omega\}$. Now we define a computable function

$$f : \omega \longrightarrow \{a,b\}$$

by $f(n) = a$ if $f_n(n) = b$

and $f(n) = b$ if $f_n(n) = a$.

f is not among $\{f_n : n<\omega\}$, so process x^f is not recursively definable by a finite guarded specification without abstraction.

REFERENCES:

[1] BAETEN, J.C.M., J.A. BERGSTRA & J.W. KLOP, *Syntax and defining equations for an interrupt mechanism in process algebra,* report CS-R8503, Centrum voor Wiskunde en Informatica, Amsterdam 1985.

[2] BAETEN, J.C.M., J.A. BERGSTRA & J.W. KLOP, *Conditional axioms and α / β calculus in process algebra,* report CS-R 8502, Centrum voor Wiskunde en Informatica, Amsterdam 1985.

[3] DE BAKKER, J.W & J.I. ZUCKER, *Compactness in semantics for merge and fair merge,* report IW 238/83, Mathematisch Centrum, Amsterdam 1983.

[4] DE BAKKER, J.W. & J.I. ZUCKER, *Processes and a fair semantics for the ADA rendez-vous,* in: 10th ICALP, ed. J. Díaz, Springer LNCS 154, Barcelona 1983, pp. 52-66.

[5] BERGSTRA, J.A. & J.W. KLOP, *An abstraction mechanism for process algebras,* report IW 231/83, Mathematisch Centrum, Amsterdam 1983.

[6] BERGSTRA, J.A. & J.W. KLOP, *Algebra of communicating processes with abstraction,* to appear in Theor. Comp. Sci.

[7] BERGSTRA, J.A. & J.W. KLOP, *Verification of an alternating bit protocol by means of process algebra,* report CS-R 8404, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[8] BERGSTRA, J.A. & J.W. KLOP, *Fair FIFO queues satisfy an algebraic criterion for protocol correctness,* report CS-R 8405, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[9] BERGSTRA, J.A. & J.W. KLOP, *A complete inference system for regular processes with silent moves,* report CS-R 8420, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[10] BERGSTRA, J.A. & J.W. KLOP, *Algebra of communicating processes,* to appear in: Proceedings of the CWI Symposium Mathematics and Computer Science, eds. J.W. de Bakker, M. Hazewinkel & J.K. Lenstra, Amsterdam 1985.

[11] BERGSTRA, J.A. & J.V. TUCKER, *Top-down design and the algebra of communicating processes,* to appear in Science of Comp. Prog.

[12] COSTA, G. & C. STIRLING, *A fair calculus of communicating systems,* report CSR-137-83, University of Edinburgh, 1983.

[13] DARONDEAU, P., *Infinitary languages and fully abstract models for fair asynchrony,* CNRS-IRISA, Rennes 1984.

[14] HENNESSY, M., *Modelling fair processes,* Proc. 16th ACM symposium on theory of computing, Washington D.C. 1984.

[15] HENNESSY, M., *Axiomatising finite delay operators,* Acta Informatica 21, 1984, pp. 61-88.
[16] HENNESSY, M., *An algebraic theory of fair asynchronous communicating processes,* to appear.
[17] MEYER, J.-J.CH., *Fixed points and arbritrary and fair merge of a fairly simple class of processes,* report IR-89, Vrije Universiteit, Amsterdam 1984.
[18] MILNER, R., *A calculus of communicating systems,* Springer LNCS 92, 1980.
[19] PARK, D.M.R., *Concurrency and automata on infinite sequences,* Proc. 5th GI Conference, Springer LNCS 104, 1981.
[20] PARROW, J., *Infinitary process algebra,* to appear.