

ON THE CONSISTENCY OF KOOMEN'S FAIR ABSTRACTION RULE *

J.C.M. BAETEN

Department of Computer Science, University of Amsterdam, 1098 SJ Amsterdam, The Netherlands

J.A. BERGSTRA

*Department of Computer Science, University of Amsterdam, 1098 SJ Amsterdam, The Netherlands,
and Department of Philosophy, State University of Utrecht, 3584 CS Utrecht, The Netherlands*

J.W. KLOP

*Department of Software Technology, Centre for Mathematics and Computer Science, 1009 AB
Amsterdam, The Netherlands*

Communicated by E. Engeler

Received June 1985

Revised August 1986

Abstract. 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.

Introduction

Process algebra is an algebraical theory of concurrency, i.e., a theory about concurrent, communicating processes. Almost anything can constitute a process: the execution of a program on a computer, or the execution of an algorithm by a person, but also a game of chess or the behavior of a vending machine.

The starting point for process algebra is the modular structure of concurrent processes at a given level of abstraction: we consider systems built up from certain basic processes by means of composition tools, including sequencing, alternative choice and parallel composition.

Process algebra tries to find laws or axioms for these composition operators, based on some a priori considerations of what features concurrent communicating processes should certainly have. Thus, we use the axiomatic method; after having established the axioms we can study different models of the theory, thus obtaining actual semantics.

* This work is sponsored in part by Esprit Project No. 432, An integrated Formal Approach to Industrial Software Development (Meteor).

Central to theories of concurrency is the solving of recursive equations (or equivalently, the finding of fixed points). In this paper, we investigate a model for the Algebra of Communicating Processes with abstraction (ACP_τ), in which all guarded recursive specifications have unique solutions. Moreover, we can describe fairness in this model. Also in our algebraic theory, we can discuss fairness through the use of Koomen's Fair Abstraction Rule (KFAR).

KFAR expresses the idea of fairness in process algebra, and is the translation in process algebra of an idea of C.J. Koomen of Philips Research (see [19]). KFAR was first formulated in [8], and its usefulness in protocol verification was demonstrated in [2, 3, 8, 9, 20, 26]. 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 [23]. Some other recent papers on fairness are [4, 5, 13, 14, 16, 17, 18, 22, 25].

When we use KFAR, all abstractions will be fair. Maybe this is too optimistic a model, and the theory should 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.

This paper 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, [11], or perhaps [1] (in Dutch). In this paper, we do the following things. In Section 1, we review the theory ACP_τ , and extra axioms and rules SC, PR and KFAR. In Section 2, we define and discuss labeled graphs, elements of the set \mathbb{G}_κ . In Section 3, we prove that if we divide out the equivalence relation $\simeq_{\tau\delta}$ (rooted $\tau\delta$ -bisimulation) on \mathbb{G}_κ , we obtain a model of $ACP_\tau+SC+PR+KFAR$, and we can even add some extra axioms (HA, ET, CA).

In Section 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}_κ for all finitely branching and bounded graphs. In Section 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}_κ for all guarded specifications.

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

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 [7]. In this section, we give a brief review of ACP_τ .

We start with an informal introduction to the composition operators used. A more elaborate and technical introduction can be found in [7].

Process algebra starts from a finite collection A of given objects, called atomic actions, atoms or steps. These actions are taken to be indivisible, usually have no duration and form the basic building blocks of our systems. The first two compositional operators we consider are “ \cdot ”, denoting sequential composition, and “ $+$ ” for alternative composition. If x and y are two processes, then $x \cdot y$ is the process that starts the execution of y after the completion of x , and $x + y$ is the process that chooses either x or y and executes the chosen process. Each time a choice is made, we choose from a set of alternatives. We do not specify whether the choice is made by the process itself or by the environment. Axioms A1–5 in Table 1 below give the laws that “ $+$ ” and “ \cdot ” obey. We leave out “ \cdot ” and brackets as in regular algebra, so $xy + z$ means $(x \cdot y) + z$.

Table 1.

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

On intuitive grounds $x(y+z)$ and $xy+xz$ present different mechanisms (the moment of choice is different), and therefore, an axiom $x(y+z) = xy+xz$ is not included.

We have a special constant “ δ ” denoting deadlock, the acknowledgement of a process that it cannot do anything anymore, the absence of an alternative. Axioms A6, 7 give the laws for “ δ ”.

Next, we have the parallel composition operator “ \parallel ”, called merge. The merge of processes x and y will interleave the actions of x and y , except for the communication actions. In $x \parallel y$, we can either do a step from x , or a step from y , or x and y both synchronously perform an action, which together make up a new action, the communication action. This trichotomy is expressed in axiom CM1. Here, we use two auxiliary operators: “ \ll ” (left-merge) and “ $|$ ” (communication merge). Thus $x \ll y$ is $x \parallel y$, but with the restriction that the first step comes from x , and $x|y$ is $x \parallel y$ with a communication step as the first step. Axioms CM2–9 give the laws for “ \ll ” and “ $|$ ”. On atomic actions, we assume the communication function given, obeying laws C1–3. Finally, we have on the left-hand side of Table 1 the laws for the encapsulation operator “ ∂_H ”. Here H is a set of atoms, and “ ∂_H ” blocks actions from H , renames them into δ . The operator “ ∂_H ” can be used to encapsulate a process, i.e., to block communications with the environment.

The right-hand side of Table 1 is devoted to laws for Milner’s silent step τ (see [20]). Laws T1–3 are Milner’s τ -laws, and TM1, 2 and TC1–4 describe the interaction of τ and merge. Finally, τ_I is the abstraction operator that renames atoms from I into τ .

1.1. Signature

S (sorts):	A	(a finite set of atomic actions),
	P	(the set of processes; $A \subseteq P$),
F (functions):	$+$: $P \times P \rightarrow P$	(alternative composition or sum),
	\cdot : $P \times P \rightarrow P$	(sequential composition or product),
	\parallel : $P \times P \rightarrow P$	(parallel composition or merge),
	\ll : $P \times P \rightarrow P$	(left-merge),
	$ $: $P \times P \rightarrow P$	(communication merge; $ $: $A \times A \rightarrow A$ is given),
	∂_H : $P \rightarrow P$	(encapsulation; $H \subseteq A$),
C (constants):	τ_I : $P \rightarrow P$	(abstraction; $I \subseteq A - \{\delta\}$),
	$\delta \in A$	(deadlock),
	$\tau \in P - A$	(silent or internal action).

1.2. Axioms

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

1.3. Digression

Let us consider for a moment the intuitive meaning of the silent step τ . A useful intuition is the following: suppose we have a machine executing a process, and we can only observe the machine starting and stopping, and the beginning of atomic actions. Then τ stands for *zero or more machinesteps*, i.e., the machine is running

for a certain period of time (which possibly has no duration), and we can observe no action beginning.

This intuition can help to understand the τ -laws T1–3:

T1: $a\tau = a$ for, in both cases, we see a beginning as soon as the machine starts, next the machine runs for a while, and then stops.

T2: $\tau x + x = \tau x$ for τ can also be zero machinesteps: when executing τx , the machine can start x right away; note that not $\tau x = x$ for, when executing τx , the machine can also run for a while before starting x .

T3: $a(\tau x + y) + ax = a(\tau x + y)$ for, when the machine executes $a(\tau x + y)$, we can see a begin and after some time the machine can start x (but not y).

Now in [27] the empty process ε is discussed in process algebra. The constant ε satisfies the laws $\varepsilon x = x\varepsilon = x$, and can therefore be considered to stand for *zero machinesteps*.

This led Koymans and Vrancken to consider a new constant η , standing for *one or more machinesteps*. We get the crucial equation

$$\tau = \eta + \varepsilon.$$

The hidden step η is the subject of current research by Baeten and Van Glabbeek. The only reference as yet is [1] (in Dutch)¹.

The constant η obeys τ -laws T1 and T3, but not law T2. Instead of T2, a different law can be chosen. We can define a hiding operator η_i that renames actions into η , and it seems that this form of hiding works very well for system verification. Abstracting to τ means that we abstract *further* than when we abstract to η ; it is possible to have a two-tiered abstraction: first to η , and then from η to τ .

Some nice properties of η are:

(1) We can take $\eta \in A$, i.e., all laws of ACP that hold for atomic actions also hold for η ;

(2) The set of finitely branching process graphs modulo (an appropriate notion of) bisimulation is a model for ACP with η ; this is not the case for τ , see Example 3.17.

1.4. Standard concurrency

Often we expand the system ACP_τ with the following axioms of Standard Concurrency (see Table 2). A proof that these axioms hold in the initial algebra of ACP_τ can be found in [7].

Table 2.

$(x \parallel y) \parallel z = x \parallel (y \parallel z)$	SC1
$(x \mid ay) \parallel z = x \mid (ay \parallel z)$	SC2
$x \mid y = y \mid x$	SC3
$x \parallel y = y \parallel x$	SC4
$x \mid (y \mid z) = (x \mid y) \mid z$	SC5
$x \parallel (y \parallel z) = (x \parallel y) \parallel z$	SC6

¹ Note added in proof: now there are references [28] and [29].

1.5. Projection

Reasoning about processes often uses a projection operator

$$\pi_n : P \rightarrow P \quad (n \geq 1),$$

which ‘cuts off’ 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.

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		

1.6. Koomen’s Fair Abstraction Rule

Koomen’s Fair Abstraction Rule (see [8]) 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 simplest form of the rule is KFAR₁:

if x and y are processes such that $x = i \cdot x + y$, and $i \in I$,
then $\tau_I(x) = \tau \cdot \tau_I(y)$.

In general, the algebraic formulation is parametrized by $k \geq 1$, indicating the length of an internal cycle.

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

This formulation is somewhat complicated. Therefore, we will write out in full the cases $k=1$ and $k=2$. First KFAR₁:

$$\frac{x = ix + y \quad (i \in I)}{\tau_I(x) = \tau \cdot \tau_I(y)} \text{KFAR}_1.$$

Then KFAR₂:

$$\frac{\begin{array}{l} x_0 = i_0 x_1 + y_0 \\ x_1 = i_1 x_0 + y_1 \end{array}}{\tau_I(x) = \tau \cdot \tau_I(y_0 + y_1)} \text{KFAR}_2.$$

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

$$\text{ACP}_\tau + \text{SC} + \text{PR} + \text{KFAR}_1 + \text{KFAR}_2 + \dots,$$

as defined in Sections 1.1–1.6.

1.7. Example

Suppose someone tosses a coin until heads comes up. He performs the process

$$P = \text{toss} \cdot (\text{tail} \cdot P + \text{heads}).$$

We define $I = \{\text{toss}, \text{tail}\}$. We write

$$P = \text{toss} \cdot Q + \delta, \quad Q = \text{tail} \cdot P + \text{heads},$$

so by applying KFAR_2 we get

$$\tau_I(P) = \tau \cdot \tau_I(\delta + \text{heads}) = \tau \cdot \text{heads},$$

so that eventually heads comes up.

1.8. Note

We finish this section by mentioning that in [26] a generalization of KFAR_k , called the Cluster Fair Abstraction Rule (CFAR), is introduced, by which clusters of internal steps can be handled that do not form a cycle.

2. Graphs

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

Definition 2.1. A *rooted directed multigraph* (which we will call *graph* for short) is a triple $\langle \text{NODES}, \text{EDGES}, \text{ROOT} \rangle$ with the following properties:

- (a) NODES is a set;
- (b) EDGES is a set; with each $e \in \text{EDGES}$ there is associated a pair $\langle s, t \rangle$ from NODES.

We say e goes from s to t , which we notate by

$$\textcircled{s} \xrightarrow{e} \textcircled{t} \quad \text{or} \quad \textcircled{s} \xrightarrow{e} \textcircled{s} \quad \text{if } s = t.$$

- (c) $\text{ROOT} \in \text{NODES}$.

Notation: $g = \langle \text{NODES}(g), \text{EDGES}(g), \text{ROOT}(g) \rangle$.

Definition 2.2. 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

$$\pi: \textcircled{S_0} \xrightarrow{e_0} \textcircled{S_1} \xrightarrow{e_1} \textcircled{S_2} \xrightarrow{e_2} \dots \xrightarrow{e_{k-1}} \textcircled{S_k}$$

or

$$\pi: \textcircled{S_0} \xrightarrow{e_0} \dots \xrightarrow{e_{k-1}} \textcircled{S_k} \xrightarrow{e_k} \dots$$

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 .

Remark 2.3. We will only consider graphs in which each node can be reached from the root.

Definition 2.4. 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 towards 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 in-degree of all other nodes is 1.
- (d) The *subgraph* $(g)_s$ of s is the graph with root s and with nodes and edges all those nodes and edges of g that can be reached from s .

Definition 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.

Definition 2.6. 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) $\mathbb{G}_\kappa^u(B, C) = \{g \in \mathbb{G}_\kappa(B, C) : g \text{ has acyclic root}\}$ is the set of *root-unwound* labeled graphs.

The following definition is taken from [10], where most of the above terminology can also be found.

Definition 2.7 (*root-unwinding*). Let B, C, κ be given. We define the *root-unwinding* map $\rho : \mathbb{G}_\kappa(B, C) \rightarrow \mathbb{G}_\kappa(B, C)$ as follows: Let $g \in \mathbb{G}_\kappa(B, C)$.

- (a) $\text{NODES}(\rho(g)) = \text{NODES}(g) \cup \{r\}$, where r is a ‘fresh’ node;

$$\text{EDGES}(\rho(g)) = \text{EDGES}(g) \cup \left\{ \begin{array}{c} \textcircled{r} \xrightarrow{e} \textcircled{s} : \textcircled{\text{ROOT}(g)} \xrightarrow{e} \textcircled{s} \in \text{EDGES}(g) \end{array} \right\};$$

- (c) $\text{ROOT}(\rho(g)) = r$;

- (d) labeling is unchanged; if $\text{ROOT}(g)$ has a label, r will get that label;
- (e) nodes and edges which cannot be reached from r are discarded.

Remark 2.8. (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 $g = \rho(g)$.

Example 2.9. (1) If g looks as shown in Fig. 1(a), then $\rho(g)$ is the graph shown in Fig. 1(b).

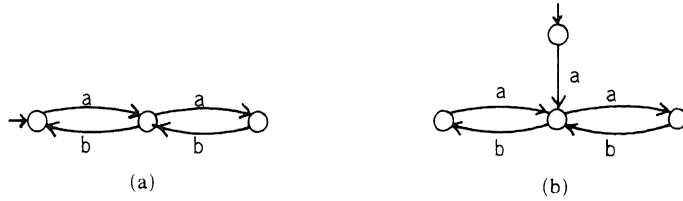


Fig. 1.

(2) If g is the graph shown in Fig. 2(a), then $\rho(g)$ looks as shown in Fig. 2(b). (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 “ $\rightarrow \circ$ ”.)

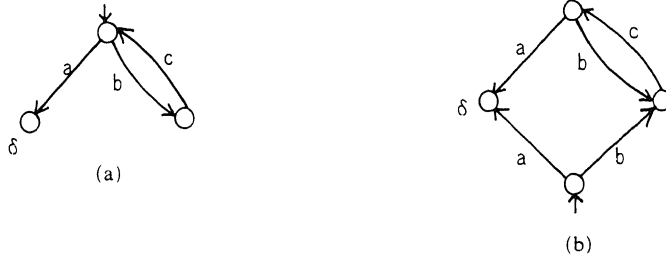


Fig. 2.

3. The model

We use the labeled graphs introduced in Section 2 to construct a model for ACP_τ .

Definition 3.1. Let A be a given *finite* set of atoms, $\delta \in A$, $\tau \notin A$. Let a communication function $| : A \times A \rightarrow A$ 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

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

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

3.1. Bisimulations

Next we will define an equivalence relation on \mathbb{G}_κ , which will say when two graphs denote the same process. This is the notion of bisimulation (also see [7, 10, 11]). First we define the label of a path in the following definition.

Definition 3.2. 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 $l_A(\pi) = \delta$.

Example 3.3. If $g = \begin{array}{c} \downarrow \\ \circ \\ \downarrow \\ \circ \\ \downarrow \end{array} \tau$ then g has paths with labels $\varepsilon, \downarrow, a, a\downarrow, \tau^n, \tau^\omega, \tau^n a, \tau^n a\downarrow$

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

We define three different bisimulations on \mathbb{G}_κ .

- (1) δ -bisimulation, $\rightleftharpoons_\delta$ is the simplest;
- (2) $\tau\delta$ -bisimulation, $\rightleftharpoons_{\tau\delta}$ is like $\rightleftharpoons_\delta$ but takes into account the special status of τ as a silent step;
- (3) *rooted* $\tau\delta$ -bisimulation, $\rightleftharpoons_{\tau\delta}^r$ is like $\rightleftharpoons_{\tau\delta}$ but also takes into account the special case when τ is an initial step.

For more information on bisimulations, see [23, 24]. (We use δ as a subscript, to distinguish the bisimulations introduced here from \rightleftharpoons , \rightleftharpoons_τ , and $\rightleftharpoons_{\tau\tau}$ defined in [10], where δ is absent.)

Definition 3.4. Let $g, h \in \mathbb{G}_\kappa$, $R \subseteq \text{NODES}(g) \times \text{NODES}(h)$.

- (1) R is a δ -bisimulation between g and h , $R: g \rightleftharpoons_\delta h$, if:
 - (i) $(\text{ROOT}(g), \text{ROOT}(h)) \in R$;
 - (ii) the domain of R is $\text{NODES}(g)$, the range is $\text{NODES}(h)$;
 - (iii) if $(p, q) \in R$ and $\begin{array}{c} \circ \\ \xrightarrow{l} \\ \circ \end{array} p \xrightarrow{l} p'$ is an edge in g with label $l \in A_\tau$, then there is a $q' \in \text{NODES}(h)$ and an edge $\begin{array}{c} \circ \\ \xrightarrow{l} \\ \circ \end{array} q \xrightarrow{l} q'$ in h with label l such that $(p', q') \in R$;
 - (iv) 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 ;
 - (v), (vi) as (iii), (iv) but with the roles of g and h reversed.
- (2) $g \rightleftharpoons_\delta h$ iff there is an $R: g \rightleftharpoons_\delta h$.
- (3) R is a $\tau\delta$ -bisimulation between g and h , $R: g \rightleftharpoons_{\tau\delta} h$, if:
 - (i), (ii) as in (1);
 - (iii)' if $(p, q) \in R$ and $\begin{array}{c} \circ \\ \xrightarrow{l} \\ \circ \end{array} p \xrightarrow{l} p'$ is an edge in g with A-label $l \in A \cup \{\varepsilon\}$, then there is a $q' \in \text{NODES}(h)$ and a path in h from q to q' with A-label l such that $(p', q') \in R$;

(iv)' 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 ;

(v)', (vi)' same as (iii)', (iv)' but with the roles of g and h reversed.

(4) $g \Leftrightarrow_{\tau\delta} h$ iff there is an $R: g \Leftrightarrow_{\tau\delta} h$.

(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 , $R: g_1 \Leftrightarrow_{\tau\delta} h_1$, if $R: g_1 \Leftrightarrow_{\tau\delta} h_1$ and, in addition, if $(p, q) \in R$, then $p = \text{ROOT}(g_1) \Leftrightarrow q = \text{ROOT}(h_1)$.

(6) $g \Leftrightarrow_{\tau\delta} h$ iff there is an $R: \rho(g) \Leftrightarrow_{\tau\delta} \rho(h)$.

Example 3.5. In Figs. 3-8 we show some examples of bisimulations.

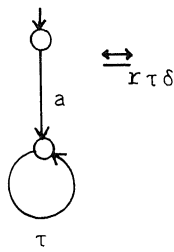


Fig. 3.

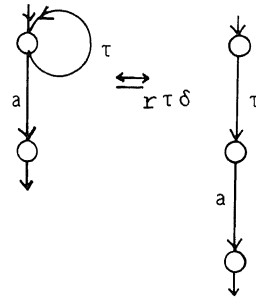


Fig. 4.

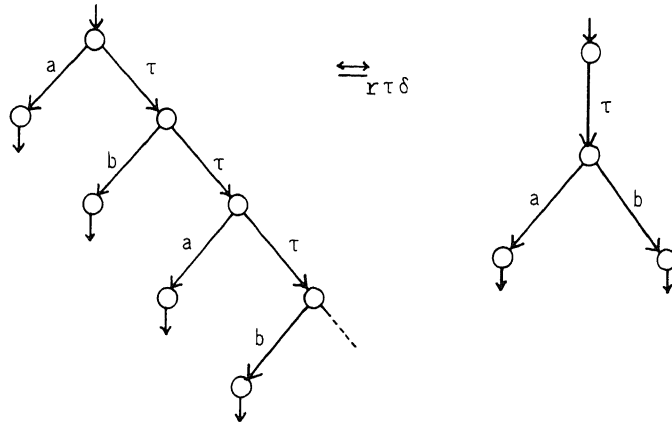


Fig. 5.

Lemma 3.6. (1) \Leftrightarrow_δ , $\Leftrightarrow_{\tau\delta}$ and $\Leftrightarrow_{\tau\delta}$ are equivalence relations on \mathbb{G}_κ .

(2) For all $g \in \mathbb{G}_\kappa$, $g \Leftrightarrow_\delta \rho(g)$, $g \Leftrightarrow_{\tau\delta} \rho(g)$ and $g \Leftrightarrow_{\tau\delta} \rho(g)$.

Proof. Easy. \square

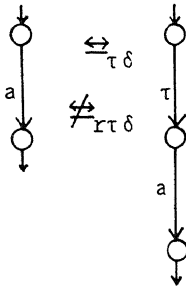


Fig. 6.

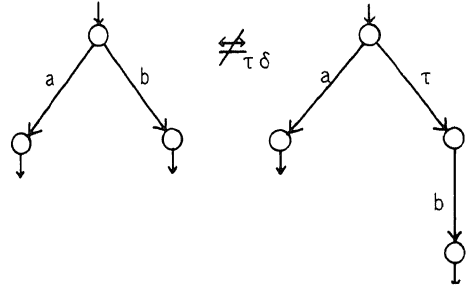


Fig. 7.

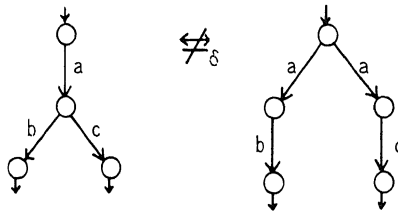


Fig. 8.

3.2. Operations and constants

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

Definition 3.7 (“+”). 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 $\rightarrow^{\circ\delta}$ (for $\circ \notin \mathbb{G}_\kappa$) and we delete this label. If both g and h are $\rightarrow^{\circ\delta}$ we put $g+h = \rightarrow^{\circ\delta}$.

Example 3.8. See Fig. 9.

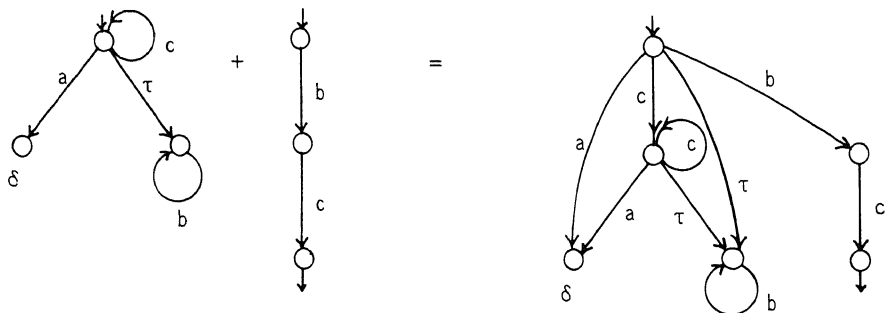


Fig. 9.

Definition 3.9 (“ \cdot ”). If $g, h \in \mathbb{G}_\kappa$, obtain $g \cdot h$ by identifying all \downarrow -endpoints of g with $\text{ROOT}(h)$ and removing the \downarrow -labels in g .

Example 3.10. See Fig. 10.

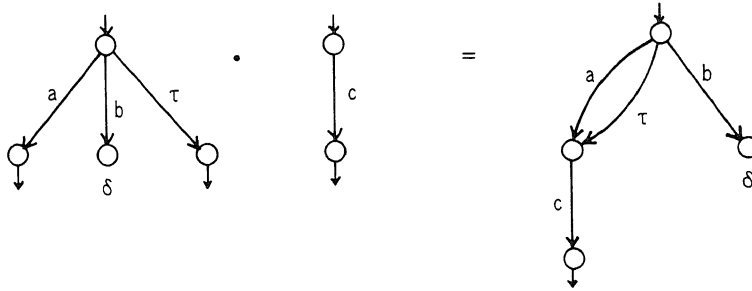


Fig. 10.

Definition 3.11 (“ \parallel ”). If $g, h \in \mathbb{G}_\kappa$, obtain $g \parallel h$ by taking the cartesian product graph of g and h (with as 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 $a \mid b = c \neq \delta$, a new edge $((p, q) \xrightarrow{c} (p', q'))$ with label c (a ‘diagonal’ edge).

In $g \parallel h$, 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 3.12. See Fig. 11 (assume $a \mid a = a \mid b = b \mid b = b \mid a = \delta$).

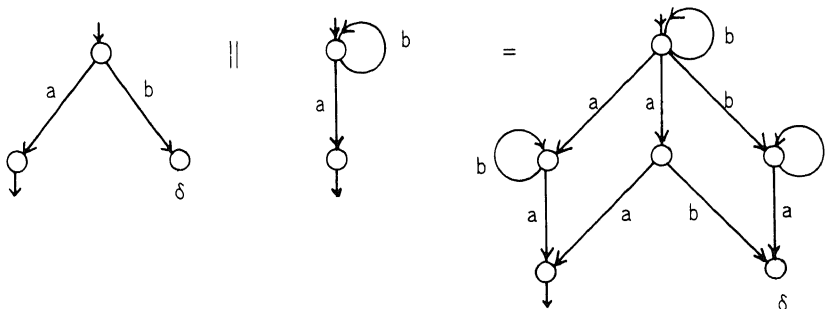


Fig. 11.

Definition 3.13 (“ \perp ”). If $g, h \in \mathbb{G}_\kappa$, $g \perp h$ is the maximal subgraph of $\rho(g) \parallel h$ in which each initial step is one from $\rho(g)$.

Example 3.14. See Fig. 12 which should be contrasted with Fig. 13 (we again assumed $a \mid a = a \mid b = b \mid b = b \mid a = \delta$).

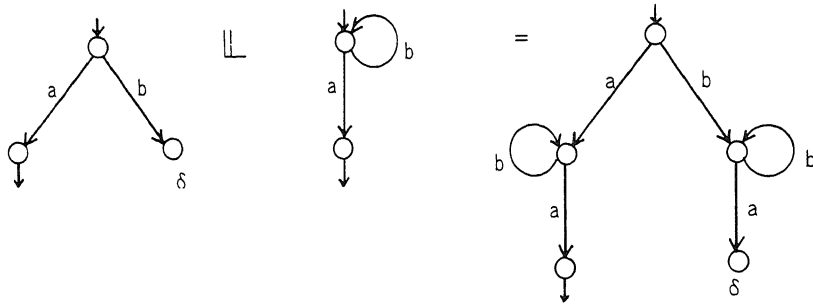


Fig. 12.

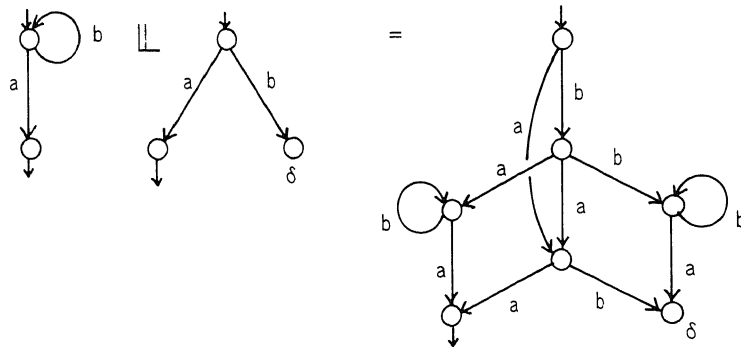


Fig. 13.

Definition 3.15 (“|”). If $g, h \in \mathbb{G}_\kappa$, $g|h$ is the sum of all the maximal subgraphs of $g \parallel h$ that start with a communication (diagonal) step and can be reached from the root by a path with A -label ε .

Example 3.16. If $b|a = a|b = c$, $a|a = b|b = \delta$, then the result is shown in Fig. 14. Notice that it is possible that the communication of two finitely branching graphs results in an infinitely branching graph that does not bisimulate with a finitely

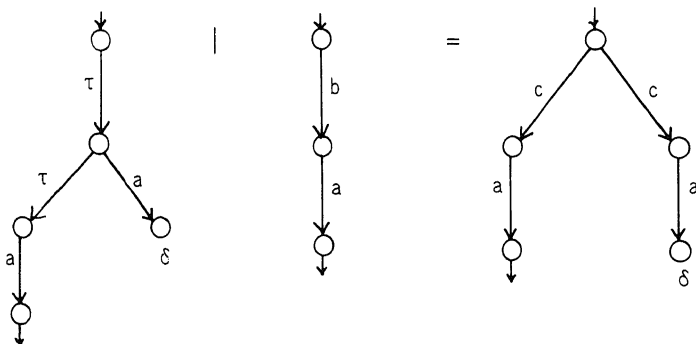


Fig. 14.

branching graph (see Example 3.17). This is the reason that \mathbb{G}_{\aleph_0} is not a model for ACP_τ (cf. remark (2) in Section 1.3).

Example 3.17. If $b|a = a|b = c$, and $a|a = b|b = \delta$, then we have Fig. 15.

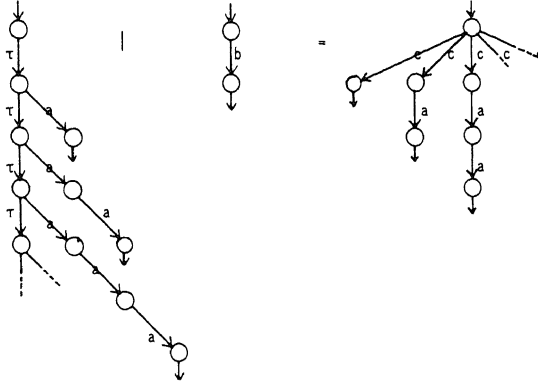


Fig. 15.

Without proof we mention the fact that if $g, h \in \mathbb{G}_\kappa$ for some $\kappa > \aleph_0$, then also $g|h \in \mathbb{G}_\kappa$

Definition 3.18 (“ ∂_H ”). Let $H \subset A$ 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 3.19. If $a \in H$, then we obtain Fig. 16.

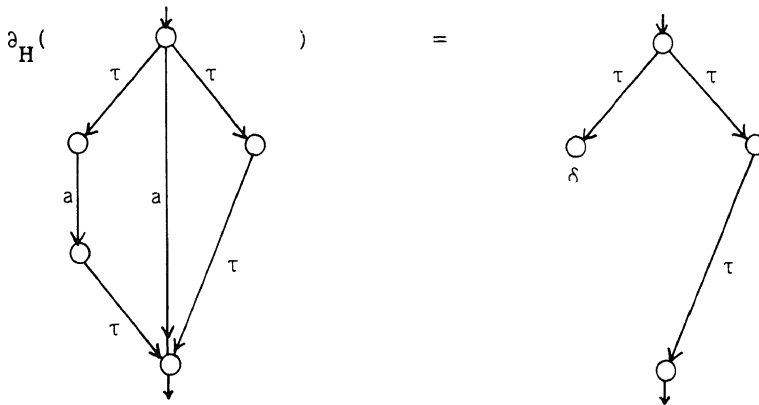


Fig. 16.

Definition 3.20 (“ 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 .

Definition 3.21 (“ π_n ”). Let $n \geq 1$ be given. If $g \in \mathbb{G}_\kappa$, obtain $\pi_n(g)$ as follows:

- (1) $\text{NODES}(\pi_n(g)) = \{s \in \text{NODES}(g) : s \text{ can be reached from } \text{ROOT}(g) \text{ by a path } \pi \text{ with the length of } l_A(\pi) \text{ less than or equal to } n\}$;
- (2) $\text{EDGES}(\pi_n(g)) = \{e \in \text{EDGES}(g) : e \text{ occurs in a path } \pi \text{ from } \text{ROOT}(g) \text{ with length } (l_A(\pi)) \leq n\}$;
- (3) $\text{ROOT}(\pi_n(g)) = \text{ROOT}(g)$;
- (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 3.22. See Fig. 17.

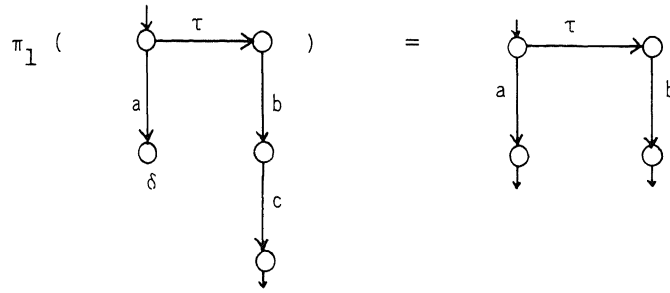


Fig. 17.

Definition 3.23. Finally we define an interpretation of the constants of ACP_τ into \mathbb{G}_κ .

- (1) If $a \in A - \{\delta\}$, its interpretation $[a] = \begin{array}{c} \downarrow \\ \circ \\ | \\ \circ \\ \downarrow \end{array} \begin{array}{l} \\ a \\ \\ \end{array}$
- (2) $[\delta] = \begin{array}{c} \downarrow \\ \circ \\ \downarrow \end{array}$
- (3) $[\tau] = \begin{array}{c} \downarrow \\ \circ \\ | \\ \circ \\ \downarrow \end{array} \begin{array}{l} \\ \tau \\ \\ \end{array}$

3.3. Main theorem

Theorem 3.24. Let κ be a given infinite cardinal number greater than \aleph_0 .

$$\left(\mathbb{G}_\kappa, (+, \cdot, \parallel, \perp, |, \partial_H, \tau_I, \pi_n), \left(\left\{ \begin{array}{c} \downarrow \\ \circ \\ | \\ \circ \\ \downarrow \end{array} \begin{array}{l} \\ a \\ \\ \end{array} : a \in A - \{\delta\} \right\}, \begin{array}{c} \downarrow \\ \circ \\ \downarrow \end{array}, \begin{array}{c} \downarrow \\ \circ \\ | \\ \circ \\ \downarrow \end{array} \right) \right)$$

is a model of $\text{ACP}_\tau + \text{SC} + \text{PR} + \text{KFAR}_1 + \text{KFAR}_2 + \dots$.

Proof. We have the restriction on κ because of the remarks in Example 3.16. The proof of the theorem is not very hard but extremely tedious, which is why we will limit ourselves to some examples and only consider the rules KFAR_κ in detail.

In [7, 11] the set of finite, acyclic process graphs modulo bisimulation is proven to be a model of ACP_τ . \square

3.3.1. Examples

In the following examples we shall denote bisimulations by linking related nodes by dotted lines.

Example 3.25 (A3: $a + a = a$). See Fig. 18.

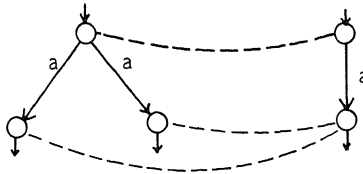


Fig. 18.

Example 3.26 (A4: $(a + b)c = ac + bc$). See Fig. 19.

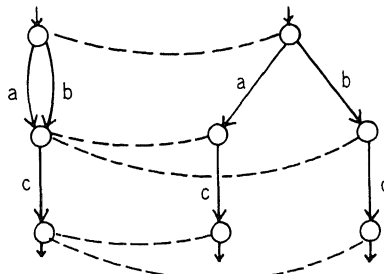


Fig. 19.

Example 3.27 (T1: $a\tau = a$). See Fig. 20.

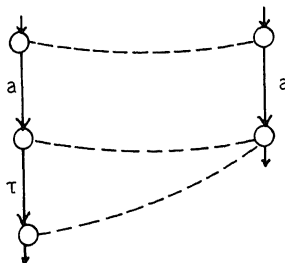


Fig. 20.

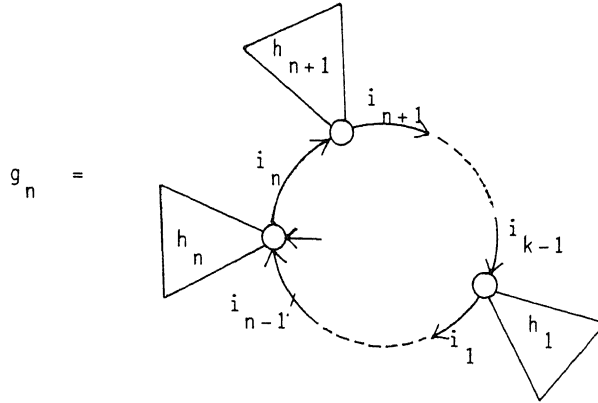


Fig. 23.

Fix $n < k$. Now we will define a rooted $\tau\delta$ -bisimulation

$$R: g'_n \rightleftharpoons_{\tau\delta} g_n,$$

thus finishing the proof of the claim. We put $(\text{ROOT}(g'_l), \text{ROOT}(h_l)) \in R$ for each $l \in \mathbb{Z}_k$. Let s be any other node in g'_n and let π be the path from $\text{ROOT}(g'_n)$ to s . Take a node s' in graph $i_n g'_{n+1} + h_n$ such that $(s, s') \in R_n$. If $s' \in \text{NODES}(h_n)$, define $(s, s') \in R$. If $s' = \text{ROOT}(g'_{n+1})$, define $(s, \text{ROOT}(h_{l+1})) \in R$. Otherwise, $s' \in \text{NODES}(g'_{n+1})$, and let π' be the path from $\text{ROOT}(g'_{n+1})$ to s' .

Since $l_A(\pi)$ must be equal to i_n followed by $l_A(\pi')$, we must have that $\text{length}(l_A(\pi')) = \text{length}(l_A(\pi)) - 1$. Now, repeat this procedure; so take node s'' in graph $i_{n+1} g'_{n+2} + h_{n+1}$ such that $(s', s'') \in R_{n+1}$. If $s'' \in \text{NODES}(h_{n+1})$, put $(s, s'') \in R$. If $s'' = \text{ROOT}(g'_{n+2})$, put $(s, \text{ROOT}(h_{l+2})) \in R$. Otherwise, $s'' \in \text{NODES}(g'_{n+2})$, but at a still shorter distance from the root.

Thus, every sequence s, s', s'', \dots must eventually 'surface', and to each $s \in \text{NODES}(g'_n)$ we will find an $s^* \in \text{NODES}(g_n)$ such that $(s, s^*) \in R$.

It is not hard to show that R is indeed a rooted $\tau\delta$ -bisimulation, so that the claim is proved. \square

(1) Let us now first consider the case $k = 1$, so we have

$$g \rightleftharpoons_{\tau\delta} ig + h$$

for some $i \in I, g, h \in \mathbb{G}_\kappa$.

Case 1: $h = \delta$ (actually, we mean $h = \rightarrow^{\circ\delta}$). Then $g \rightleftharpoons_{\tau\delta} ig$. We see by the claim that

$$g \rightleftharpoons_{\tau\delta} \begin{array}{c} \uparrow \\ \circ \\ \downarrow \end{array} i.$$

Then

$$\tau_I(g) \rightleftharpoons_{\tau\delta} \begin{array}{c} \uparrow \\ \circ \\ \downarrow \end{array} \tau \rightleftharpoons_{\tau\delta} \begin{array}{c} \uparrow \\ \circ \\ \downarrow \\ \delta \end{array} \tau$$

which is the desired result because

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

Case 2: h is not δ . Then we obtain that g is rooted $\tau\delta$ -bisimulated by the graph in Fig. 24(a), so $\tau_I(g)$ is rooted $\tau\delta$ -bisimulated by the graphs in Fig. 24(b): again the right result.

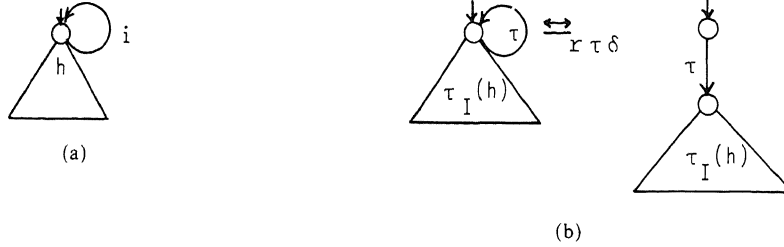


Fig. 24.

(2) If $k > 1$, the proof works similarly. (We remark that in [26] it has been shown that the rules KFAR_k , for $k > 1$, logically follow from KFAR_1 .) For instance, if $k = 3$, we have

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

($i_1, i_2, i_3, \in I$), so g_1 is rooted $\tau\delta$ -bisimulated by the the graph in Fig. 25, whence $\tau_I(g_1)$ is rooted $\tau\delta$ -bisimulated by the graphs in Fig. 26.

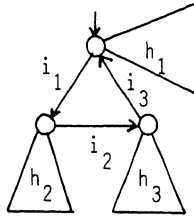


Fig. 25.

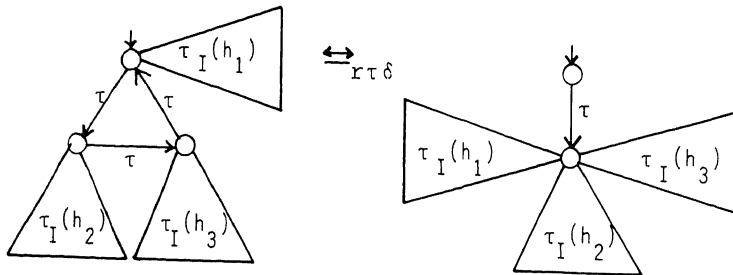


Fig. 26.

3.4. Handshaking

If we adopt the Handshaking Axiom (HA), namely

$$(HA) \quad x|y|z = \delta$$

for all processes x, y, z , which says that all communications are binary, then the following *Expansion Theorem* (ET) holds in the model $\mathbb{G}_\kappa / \Leftrightarrow_{\tau\tau\delta} (\kappa > \aleph_0)$. This is because $\mathbb{G}_\kappa / \Leftrightarrow_{\tau\tau\delta}$ satisfies the Axioms of Standard Concurrency of Section 1.3. A proof of this fact is given in [7]. The formulation of the Expansion Theorem is due to Bergstra and Tucker [12].

Theorem 3.31 (Expansion Theorem). *Let x_1, \dots, x_n be given processes, and let x^i be the merge of all x_1, \dots, x_n except x_i ; let $x^{i,j}$ be the merge of all x_1, \dots, x_n except x_i and x_j ; then the Expansion Theorem is*

$$(ET) \quad x_1 \parallel x_2 \parallel \dots \parallel x_n = \sum_{1 \leq i \leq n} x_i \parallel x^i + \sum_{1 \leq i < j \leq n} (x_i | x_j) \parallel x^{i,j}$$

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.

3.5. Alphabets

We can define, for each $g \in \mathbb{G}_\kappa$, the *alphabet* of g , $\alpha(g)$, to be the set of all labels occurring in g except τ, δ, \downarrow . Note that here we will need the requirement in Remark 2.3 that each node can be reached from the root. Then it is easy to see that if $g \Leftrightarrow_{\tau\tau\delta} h$ (even if $g \Leftrightarrow_{\tau\delta} h$), then $\alpha(g) = \alpha(h)$. With this definition, it is not hard to show that $\mathbb{G}_\kappa / \Leftrightarrow_{\tau\tau\delta} (\kappa > \aleph_0)$ satisfies the Conditional Axioms (CA), first formulated in [3], as shown in Table 4.

Table 4.

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

4. The approximation induction principle

The unrestricted 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 ,

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

We will prove in Theorem 4.3 that a restricted version of AIP, called AIP^- , holds in $\mathbb{G}_\kappa / \Leftrightarrow_{\tau\tau\delta}$ ($\kappa > \aleph_0$). In Section 4.1 we will see that the unrestricted version does not hold. First some definitions.

Definition 4.1. (i) Let $g \in \mathbb{G}_\kappa$. Define the n th level of g , $[g]_n$, by

$$[g]_n = \{s \in \text{NODES}(g) : s \text{ can be reached from } \text{ROOT}(g) \text{ by} \\ \text{a path } \pi \text{ with length } (l_A(\pi)) = n\}.$$

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. The $[g]_n$ are disjoint if g is a process *tree*.

(ii) Let $g, h \in \mathbb{G}_\kappa$. A relation R between nodes of g and nodes of h is called *history-preserving* if R only relates nodes with a common history; i.e., if, for $s \in \text{NODES}(g)$ and $t \in \text{NODES}(h)$, $R(s, t)$ holds, then there is a path π from $\text{ROOT}(g)$ to s and a path π' from $\text{ROOT}(h)$ to t such that $l_A(\pi) = l_A(\pi')$.

Note that a history-preserving relation relates only nodes of the same depth.

Lemma 4.2. *Let $g, h \in \mathbb{G}_\kappa$ and $g \Leftrightarrow_{\tau\tau\delta} h$. Then there is a history-preserving $\tau\tau\delta$ -bisimulation between g and h .*

Proof. Left to the reader (note that we build up such a bisimulation step by step from the root, using the definition of bisimulation). \square

Theorem 4.3. *Let $g, h \in \mathbb{G}_\kappa$ and suppose that for each n*

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

Then $g \Leftrightarrow_{\tau\tau\delta} h$.

Proof. Without loss of generality, we can suppose that g and h are completely unwound to process trees (the proof also works for general process graphs, but becomes harder to comprehend). All bisimulations appearing in this proof will be history-preserving.

Given is that $\pi_n(g) \Leftrightarrow_{\tau\delta} \pi_n(h)$ for each n ; we say that g and h $\tau\delta$ -bisimulate *until depth n* .

Suppose that R is a (history-preserving) $\tau\delta$ -bisimulation between g and h until depth $n+m$ that relates $s \in \text{NODES}(g)$ to $t \in \text{NODES}(h)$ at depth n . Then R induces a $\tau\delta$ -bisimulation between $(g)_s$ and $(h)_t$, until depth m . Thus, the given bisimulations between g and h until finite depth induce many bisimulations between subtrees of g and h , until finite depth. This leads to the following definitions. Fix $n \in \mathbb{N}$, and let $s \in [g]_n$, $t \in [h]_n$. Define

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

such that

$$R \cap ((g)_s \times (h)_t): \pi_m((g)_s) \Leftrightarrow_{\tau\delta} \pi_m((h)_t)$$

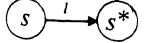
(in words: there is an R which is a rooted $\tau\delta$ -bisimulation until depth $n+m$ and, restricted to the subtrees of s and t , is a $\tau\delta$ -bisimulation until depth m ; if $m=0$, the second part boils down to $R(s, t)$), and

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

We will show that \sim is a rooted $\tau\delta$ -bisimulation between g and h . Note that \sim is history-preserving, so only relates nodes of the same depth. Let us first see how \sim works at a certain level n . Suppose $[g]_n$ is finite. Let us first consider a $t \in [h]_n$. Let S_m be the set of nodes in $[g]_n$ that are \sim_m related to t ; i.e., $s \in S_m$ iff $s \sim_m t$. We see $S_1 \supseteq S_2 \supseteq \dots \supseteq S_m \supseteq \dots$ and all S_m are nonempty. Therefore, since $[g]_n$ is finite, we get $\bigcap_{m \geq 1} S_m \neq \emptyset$.

Take s in this intersection; then we have $s \sim t$. Thus, for each $t \in [h]_n$, there is an $s \in [g]_n$ with $s \sim t$. Next, consider an $s \in [g]_n$. Let H_s be the set of nodes in $[h]_n$ that are \sim -related to s . Then $[h]_n$ is the union of these sets H_s , and this is a *finite* union. Note also that some H_s might be empty. Now we start the verification, that \sim is a bisimulation. First note that, by definition of \sim and assumption (i), we have (cf. Definition 3.4)

- (i) $\text{ROOT}(g) \sim \text{ROOT}(h)$, and
- (vii) if $s \sim t$, then $s = \text{ROOT}(g) \Leftrightarrow t = \text{ROOT}(h)$. Also it is not hard to see that
- (ii) $\text{dom}(\sim) = \text{NODES}(g)$ and $\text{ran}(\sim) = \text{NODES}(h)$. It remains to verify (iii)', (iv)', (v)', (vi)' of Definition 3.4(3).

For (iii)', suppose $s \sim t$ and take n such that $s \in [g]_n$, $t \in [h]_n$. Let  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. By the reasoning above, there is a node t^* in $[h]_{n+1}$ such that $s^* \sim t^*$. Since all bisimulations are history-preserving, there must be a path from t to t^* with A -label a .

Case 1.2: otherwise. By assumption (ii), $[g]_{n+1}$ is finite. If $H_{s^*} \neq \emptyset$, we are done. Otherwise, we can find a sequence $\langle t_0, t_1, t_2, \dots \rangle$ in $[g]_n$ such that $s^* \sim_m t_m$ (since $s \sim_{m+1} t$). Since there are only finitely many H_s , there is an 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 $R_1, R_2, R_3: \pi_{n+m+1}(g) \xrightarrow{\tau\delta} \pi_{n+m+1}(h)$ such that

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

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

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

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

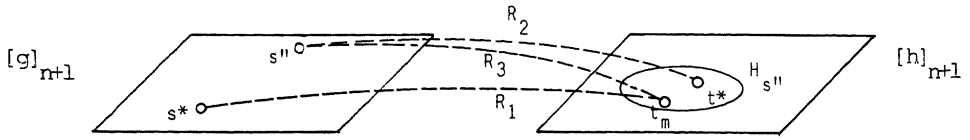


Fig. 27.

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

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

and

$$R \cap ((g)_{s^*} \times (h)_{t^*}): \pi_m((g)_{s^*}) \xrightarrow{\tau\delta} \pi_m((h)_{t^*}),$$

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 $s^* \in [g]_n$, $t^* \in [h]_n$. Also, it is useful to intersect the level $[g]_n$ with $(g)_s$ and the level $[h]_n$ with $(h)_t$. Thus, we have verified (iii)' of Definition 3.4(3).

For a verification of (iv)', 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 $R: \pi_{n+1}(g) \xrightarrow{\tau\delta} \pi_{n+1}(h)$ 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 (v)', (vi)' of Definition 3.4(3) are like the proofs for (iii)', (iv)', 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. \square

Definition 4.4. 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 [6].)

Lemma 4.5. *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 $\text{ROOT}(g)$ 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 label 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 $[g]_0 = \text{NODES}(g')$ is finite.

For the induction step, suppose $[g]_n$ is finite. Put

$$B = \left\{ s \in [g]_{n+1} : \text{there is a } t \in [g]_n \text{ and an edge } \textcircled{t} \xrightarrow{a} \textcircled{s}, a \in A \right\}.$$

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. \square

Corollary 4.6. *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 , $\pi_n(g) \Leftrightarrow_{\tau\tau\delta} \pi_n(h)$, then $g \Leftrightarrow_{\tau\tau\delta} h$).*

Proof. Combine Theorem 4.3 and Lemma 4.5. \square

4.1. Counterexamples

Suppose a is an atomic action different from δ .

Example 4.7. Define $g = \sum_{n \geq 1} a^n$, $h = g + a^\omega$. See Fig. 28.

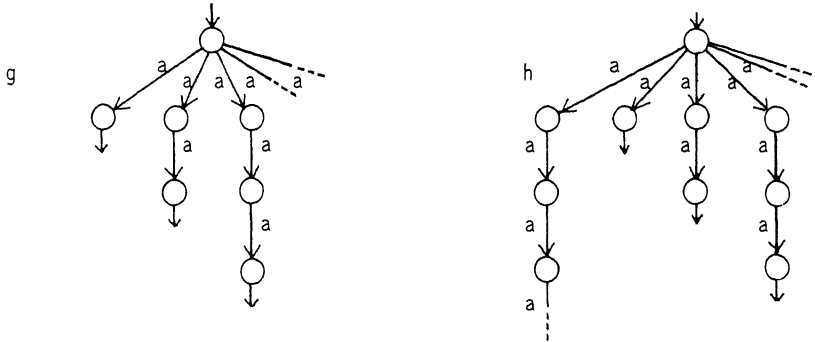


Fig. 28.

It is not hard to see that, for each n , $\pi_n(g) \Leftrightarrow_{\tau\tau\delta} \pi_n(h)$, but not $g \Leftrightarrow_{\tau\tau\delta} h$ so g, h do not satisfy (AIP). g and h are both bounded, but not finitely branching.

Example 4.8. g' and h' are shown in Fig. 29. Again we have $\pi_n(g') \Leftrightarrow_{\tau\tau\delta} \pi_n(h')$ for each n (using the second τ -law T2), but not $g' \Leftrightarrow_{\tau\tau\delta} h'$, so g', h' do not satisfy (AIP). g' and h' are both finitely branching, but not bounded.

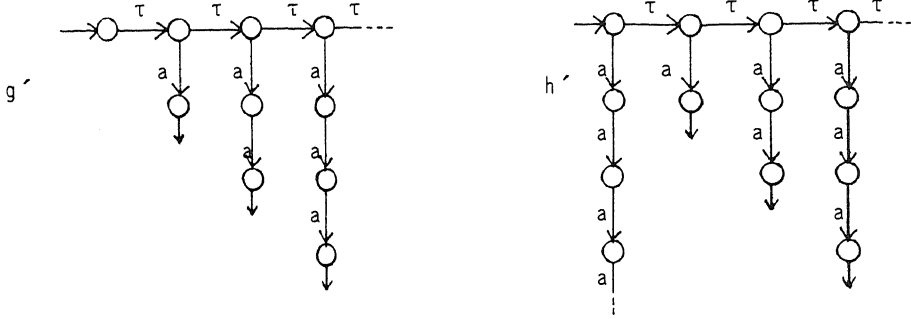


Fig. 29.

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 branch occurs infinitely many times, we do have a $\tau\delta$ -bisimulation (this is a sort of infinite version of KFAR).

Remark 4.9. At this point, we cannot formulate the restricted version of (AIP) proved in Theorem 4.3 or Corollary 4.6 algebraically. We will be able to do this in Section 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.

Definition 5.1. 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.

Example 5.2. Let E be

$$\begin{aligned} X_0 &= X_1 \parallel X_2 + X_3 a, & X_1 &= \tau \partial_H (X_0 X_0), \\ X_2 &= \tau X_2, & X_3 &= \tau_l (a X_2 + X_3 b X_1). \end{aligned}$$

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 .

Definition 5.3. 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 .

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

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

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

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

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

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 $\mathbb{G}_\kappa / \Leftrightarrow_{\text{rr}\delta}$ ($\kappa > \aleph_0$). 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 $\mathbb{G}_\kappa / \Leftrightarrow_{\text{rr}\delta}$, so does not satisfy RSP. (If p is any process not containing an a or b , then $a \cdot p$ is a solution for X_0 , and $b \cdot p$ is a solution for X_1 .) Because of these problems, we will formulate guardedness and the following theorems *only* for specifications that contain no abstraction.

Definition 5.6. 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*.

Examples 5.7. Let T be the term

$$aX_0 + \tau X_1 + a \parallel X_2 + X_3 \parallel aX_4.$$

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

Definition 5.8. Let $E = \{E_j : j \in J\}$ be a specification without an abstraction operator τ_i , and let $i, j \in J$. We define $X_i \rightarrow^u X_j \Leftrightarrow X_j$ occurs unguarded in T_i and we call E *guarded* if relation \rightarrow^u is well-founded (i.e., there is no infinite sequence

$$X_{j_1} \rightarrow^u X_{j_2} \rightarrow^u X_{j_3} \rightarrow^u \dots).$$

Next we start the proof of RDP and RSP in $\mathbb{G}_\kappa / \cong_{\tau, \delta}$ ($\kappa > \aleph_0$).

Definition 5.9. 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 . To be more precise, we use:

- (1) *substitution*: if we obtain t by substituting T_i for an occurrence of X_i in s , then t is an expansion of s ;
- (2) *reflexivity*: t is an expansion of t ;
- (3) *transitivity*: if t is an expansion of s and u is an expansion of t , then u is an expansion of s . For more details, see [3, Section 2.7].

Lemma 5.10. *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 [3, Lemma 2.14]. 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 $X_j \rightarrow^u X_i \rightarrow^u \dots$, which contradicts the well-foundedness of \rightarrow^u . \square

Theorem 5.11. *Let E be a guarded recursive specification in which no abstraction operator occurs. Then, in the model $\mathbb{G}_\kappa / \cong_{\tau, \delta}$ ($\kappa > \aleph_0$), 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 Lemma 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 Section 3.2 (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 Lemma 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_τ -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 $\Leftrightarrow_{\text{rr}\delta}$!). 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

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

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

$$g_{i_0} \Leftrightarrow_{\text{rr}\delta} T_{i_0}(g_{i_1}, \dots, g_{i_m}).$$

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

$$\begin{aligned} & \pi_n(T_{i_0}(g_{i_1}, \dots, g_{i_m})) \\ &= \pi_n(T_{i_0}(\pi_n(g_{i_1}), \dots, \pi_n(g_{i_m}))) \quad (\text{use Definition 3.21}) \\ &= \pi_n(T_{i_0}(\pi_n(T_{i_1}^n), \dots, \pi_n(T_{i_m}^n))) \quad (\text{by assumption}) \\ &= \pi_n(T_{i_0}(T_{i_1}^n, \dots, T_{i_m}^n)) \quad (\text{again by Definition 3.21}) \\ &= \pi_n(T_{i_0}^n) \quad (\text{by construction of } T_{i_0}^n) \\ &= \pi_n(g_{i_0}) \quad (\text{by assumption}). \end{aligned}$$

This finishes the proof. \square

Theorem 5.12. *Let E be a guarded recursive specification in which no abstraction operator occurs. Then, in the model $\mathbb{G}_\kappa / \Leftrightarrow_{\text{rr}\delta}$, E has a unique solution ($\kappa > \aleph_0$).*

Proof. By Theorem 5.11, E has a solution g which is finitely branching and bounded. Let h be any other solution of E . We will show $g \Leftrightarrow_{\text{rr}\delta} h$ 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}, \dots, X_{j_m} , then

$$\begin{aligned} h & \Leftrightarrow_{\text{rr}\delta} T_{j_0}(h_{j_1}, \dots, h_{j_m}) \quad (\text{for } h \text{ is a solution}) \\ & \Leftrightarrow_{\text{rr}\delta} T_{j_0}(T_{j_1}(\mathbf{h}), \dots, T_{j_m}(\mathbf{h})) \end{aligned}$$

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

$$\begin{array}{c} \vdots \\ \Leftrightarrow_{\tau\delta} T^n(\mathbf{h}) \quad (\text{for some sequence } \mathbf{h}), \end{array}$$

whence $\pi_n(\mathbf{h}) \Leftrightarrow_{\tau\delta} \pi_n(T^n(\mathbf{h})) = \pi_n(T^n(X)) = \pi_n(T^n)$. \square

Note that Theorem 5.12 does not suffice to conclude that the equation $x = ix + y$, occurring in Example 3.30, has a unique solution x for each given process y . In this paper, we do not consider equations with parameters at all. We refer to [21] for a discussion on solving equations with parameters.

Now we can give the following algebraical formulation of AIP, which holds in the model $\mathbb{G}_\kappa / \Leftrightarrow_{\tau\delta} (\kappa > \aleph_0)$.

Theorem 5.13. $\mathbb{G}_\kappa / \Leftrightarrow_{\tau\delta} (\kappa > \aleph_0)$ satisfies the following principle, which we will call AIP⁻:

$\text{for all } n \pi_n(x) = \pi_n(y)$ $\text{(AIP}^-\text{)} \frac{x \text{ is specifiable by a guarded } E \text{ without } \tau_I}{x = y}$
--

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 Theorems 5.11 and 5.12, which satisfies AIP by Corollary 4.6. \square

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 Definition 5.14. We also remark that, in [15], another formulation of AIP⁻ appears, which is a little less restrictive and which we can also use in the presence of an abstraction operator.

Definition 5.14. We define an auxiliary theory ACP_τ^t as follows:

- (1) ACP_τ^t extends ACP_τ ;
- (2) ACP_τ^t has a new atom $t \in A$ with $t|a = \delta$ for all $a \in A$;
- (3) ACP_τ^t has a new operator t_I (where $I \subseteq A_\tau - \{\delta\}$) defined by the four equations in 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 [3, Section 2.10].)

Definition 5.15. We extend \mathbb{G}_κ with a new element

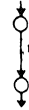


Table 5.

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

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

Remark 5.16. Theorem 5.12 still holds for specifications E in which a hiding operator t_I occurs. This is not hard to see.

Corollary 5.17. $\mathbb{G}_\kappa / \cong_{\tau, \delta}$ ($\kappa > \aleph_0$) satisfies the following principles, which we will call RDP and RSP:

$$\text{(RDP)} \frac{E \text{ guarded, no } \tau_I}{\exists x: E(x, -)}$$

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

6. Computable graphs

In the previous sections, we have defined a model for ACP_τ , in which a number of desirable principles hold (KFAR, RSP, RDP, AIP⁻). In the rest of the paper we show that this model is not too big: every computable graph is the solution of a finite recursive specification. Thus, the graph models are the ‘natural’ models of ACP_τ .

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).

In order to show that every computable graph is the solution of a finite recursive specification, we first show in Theorem 6.7 that every partial computable function on natural numbers can be represented as the solution of a finite recursive specification. In the proof of Theorem 6.7, we use the principles RDP, RSP, AIP⁻ and KFAR₁. In Theorem 6.8 and Corollary 6.9, we then prove that the theorem holds for every binary branching graph. In Lemma 6.10, we show that it is sufficient to look at binary branching graphs. The proof of Lemma 6.10 takes place in the graph model, and this is the only place in this section where the proof is not algebraical. To turn the proof of Lemma 6.10 into an algebraical proof, it will be necessary to formulate an extended version of KFAR, more extended even than the rule CFAR mentioned in Section 1.8. When such a proof is found, however, we will have shown that every process that is the unique solution of a computable recursive specification also is the unique solution of a finite recursive specification (after abstraction), independent of a model. In the present text, we only obtain this result (in Section 8) relative to the graph model.

6.1. Definitions

Definition 6.1. Let $g \in \mathbb{G}_{\mathbb{N}_0}$ (so g is finitely branching). A *coding* of g consists of the following:

- (1) If $s \in \text{NODES}(g)$ and the out-degree of s is n , then the outgoing edges are named $0, 1, \dots, n-1$.
- (2) This leads to the following naming of nodes: a sequence $\sigma \in \omega^*$ names the node reached by following the path from $\text{ROOT}(g)$ with edge-names in σ .

Example 6.2. Let g be the graph of Fig. 30 with indicated coding. $\text{ROOT}(g)$ has name ε and the endpoint of g has names $000, 10, 110, 20$ and 210 .

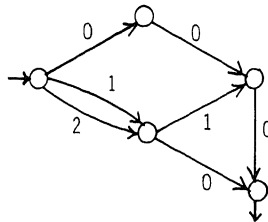


Fig. 30.

Remark 6.3. $g \in \mathbb{G}_{\mathbb{N}_0}$ is a tree \Leftrightarrow each node has exactly one name.

Definition 6.4. Let $g \in \mathbb{G}_{\mathbb{N}_0}$ be coded. We define two partial functions

$$\text{od} : \omega^* \rightarrow \omega, \quad \text{lb} : \omega^* \rightarrow A \cup \{\delta, \downarrow\},$$

as follows:

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

Definition 6.5. $g \in \mathbb{G}_{\infty_0}$ 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, coding of $A \cup \{\delta, \downarrow\}$ into ω is not important).

6.2. Results

Now we start the proof of the main theorem of this section. 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.

Definition 6.6. Let D be a finite set of data. We suppose we have a number of *communication channels* $0, 1, \dots, k$ ($k \geq 1$), 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* ($d \in D, i \leq k$);
- (2) $r_i(d)$ = *receive d along channel i* ($d \in D, i \leq k$);
- (3) $c_i(d)$ = *communicate d along channel i* ($d \in D, i \leq k$).

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 $f: D^* \rightarrow D^*$ 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 $\sigma = d_1 \dots d_n$ is given, and we have a marker 'eos' indicating the end of a sequence.

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

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

Then, we will hide unsuccessful communications:

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

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

$$(1) \quad f(\sigma) = \rho \Leftrightarrow \partial_{H'}(\mathbb{S}_\sigma \parallel \hat{f} \parallel \mathbb{R}) = c_0(d_1) \cdot c_0(d_2) \cdot \dots \cdot c_0(d_n) \\ \cdot c_0(\text{eos}) \cdot c_1(e_1) \cdot \dots \cdot c_1(e_m) \cdot c_1(\text{eos}),$$

$$(2) \quad f(\sigma) \text{ is undefined} \Leftrightarrow \partial_{H'}(\mathbb{S}_\sigma \parallel \hat{f} \parallel \mathbb{R}) \\ = c_0(d_1) \cdot c_0(d_2) \cdot \dots \cdot c_0(d_n) \cdot c_0(\text{eos}) \cdot \delta.$$

Theorem 6.7. *Let $f: \omega^* \rightarrow \omega^*$ 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 $0, \dots, k$ ($k \geq 1$) of which 0 is the starting state and k the ending state. In turn, we will simulate this Turing machine by a finite specification

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

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

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

(see, e.g., [11]), (the extra atom 'stop' is needed for successful termination). C is specified using variables $C_0, C_1, \dots, C_k, C_{k+1}, C_{k+2}$ (think of these C_i as the 'states' of C , and C_0, \dots, C_k correspond to the states of the Turing machine). The

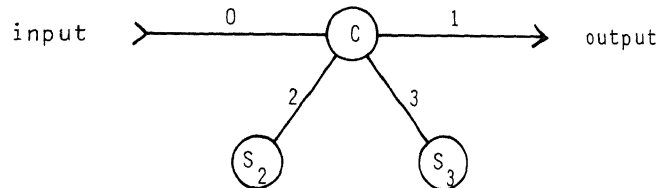


Fig. 31.

specification of C consists of three parts:

- (1) input,
- (2) calculation,
- (3) output.

Part 1: input

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

When C_0 is reached, input sits in S_3 in the right order, and ends with an '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 $i d e R m$ ($i < k, m \leq k; d, e \in D$) (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

$$C_i = r_3(d)s_2(e)C_m;$$

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

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

Figures 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.

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

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

This completes the specification of C .

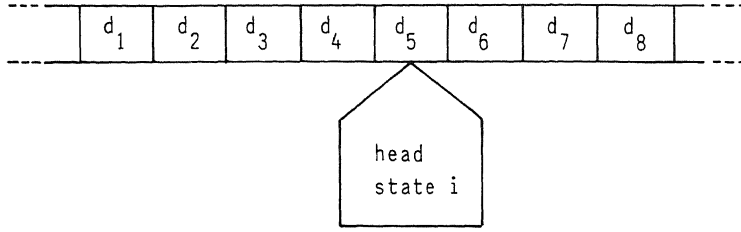


Fig. 32.

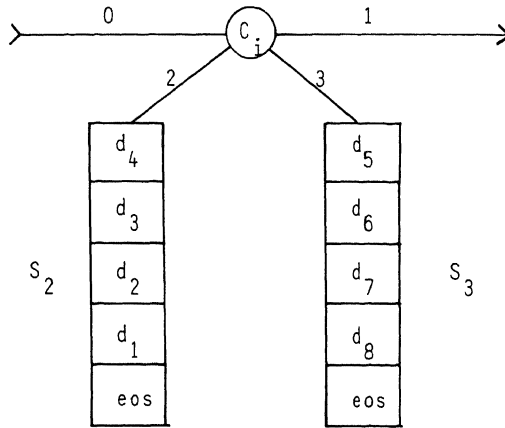


Fig. 33.

Next we hide all unsuccessful communications by encapsulation: we define

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

and we hide all internal communications by abstraction: we define

$$I = \{c_i(d) : d \in D \cup \{\text{eos}, \text{stop}\}, i = 2, 3\},$$

and consider $\hat{f} = \tau_{\{i\}}(x)$, where x is the unique solution of specification $X = \tau_I \circ \partial_H(C \parallel S_2 \parallel S_3)$. Informally, we will write

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

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 $H' = \{s_i(d), r_i(d) : d \in D \cup \{\text{eos}\}, i = 0, 1\}$ as in Definition 6.6 and consider

$$\partial_{H'}(S_\sigma \parallel \hat{f} \parallel \mathbb{R}).$$

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

Then

$$\begin{aligned} \partial_{H'}(\mathbb{S}_\sigma \parallel \hat{f} \parallel \mathbb{R}) &= \partial_{H'}(\mathbb{S}_\sigma \parallel (\hat{f} \parallel \mathbb{R})) + \partial_{H'}(\hat{f} \parallel (\mathbb{S}_\sigma \parallel \mathbb{R})) + \partial_{H'}(\mathbb{R} \parallel (\hat{f} \parallel \mathbb{S}_\sigma)) \\ &\quad + \partial_{H'}((\mathbb{S}_\sigma \mid \hat{f}) \parallel \mathbb{R}) + \partial_{H'}((\mathbb{S}_\sigma \mid \mathbb{R}) \parallel \hat{f}) + \partial_{H'}((\hat{f} \mid \mathbb{R}) \parallel \mathbb{S}_\sigma) \end{aligned}$$

(by Expansion Theorem 3.31)

$$\begin{aligned} &= \delta + \delta + \delta + c_0(d_1) \partial_{H'} \\ &\quad (\mathbb{S}_{d_2 \dots d_n} \parallel \tau_I \circ \partial_H((s_2(\text{eos})s_2(d_1)C_{k+1}) \parallel S_2 \parallel S_3) \parallel \mathbb{R}) + \delta + \delta \\ &= c_0(d_1) \partial_{H'}(\mathbb{S}_{d_2 \dots d_n} \parallel \tau_I(c_2(\text{eos})c_2(d) \cdot \partial_H(C_{k+1} \parallel S_2^{d_1} \parallel S_3)) \parallel \mathbb{R}) \\ &= c_0(d_1) \tau \cdot \partial_{H'}(\mathbb{S}_{d_2 \dots d_n} \parallel \tau_I \circ \partial_H(C_{k+1} \parallel S_2^{d_1} \parallel S_3) \parallel \mathbb{R}) \\ &\quad \vdots \\ &= c_0(d_1)c_0(d_2) \dots c_0(d_n) \\ &\quad \cdot \partial_{H'}(s_0(\text{eos}) \parallel \tau_I \circ \partial_H(C_{k+1} \parallel S_2^{d_1 \dots d_1} \parallel S_3) \parallel \mathbb{R}) \\ &= c_0(d_1) \dots c_0(d_n)c_0(\text{eos}) \cdot \partial_{H'}(\tau_I \circ \partial_H(C_{k+2} \parallel S_2^{d_1 \dots d_1} \parallel S_3^\theta) \parallel \mathbb{R}) \\ &= c_0(d_1) \dots c_0(d_n)c_0(\text{eos}) \cdot \partial_{H'}(\tau_I(c_2(d_n)c_3(d_n) \\ &\quad \cdot \partial_H(C_{k+2} \parallel S_2^{d_1 \dots d_1} \parallel S_3^{d_n})) \parallel \mathbb{R}) \\ &\quad \vdots \\ &= c_0(d_1) \dots c_0(d_n)c_0(\text{eos}) \cdot \partial_{H'}(\tau_I \circ \partial_H(C_0 \parallel S_3^\theta \parallel S_3^\sigma) \parallel \mathbb{R}). \end{aligned}$$

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

$$\tau_I \circ \partial_H(C_0 \parallel S_2^\theta \parallel S_3^\sigma) = \tau \tau_I \circ \partial_H(C_k \parallel S_2^\theta \parallel S_3^\rho).$$

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

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

($i < k$; $\sigma', \sigma'' \in D^*$, $d \in D$) corresponds to the Turing machine in state i with as 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

$$\begin{aligned} \tau \cdot \tau_I \circ \partial_H(C_i \parallel S_2^{\sigma'} \parallel S_3^{d^* \sigma''}) &= \tau \cdot \tau_I(c_3(d) \cdot \partial_H((s_2(e)C_m) \parallel S_2^{\sigma'} \parallel S_3^{\sigma''})) \\ &= \tau \cdot \tau \cdot \tau_I(c_2(e) \cdot \partial_H(C_m \parallel S_2^{e^* \sigma'} \parallel S_3^{\sigma''})) \\ &= \tau \cdot \tau_I \circ \partial_H(C_m \parallel S_2^{e^* \sigma'} \parallel S_3^{\sigma''}). \end{aligned}$$

(b) Suppose there a TM instruction $i d e L m$. Then

$$\begin{aligned}
\tau \cdot \tau_I \circ \partial_H(C_i \| S_2^{f^* \sigma'} \| S_3^{d^* \sigma''}) &= \tau \cdot \tau_I \left(c_3(d) \right. \\
&\quad \left. \cdot \partial_H \left(\left(s_3(e) \sum_{f \in D} r_2(f) s_3(f) C_m \right) \| S_2^{f^* \sigma'} \| S_3^{\sigma''} \right) \right) \\
&= \tau \cdot \tau_I \left(c_3(e) \right. \\
&\quad \left. \cdot \partial_H \left(\left(\sum_{f \in D} r_2(f) s_3(f) C_m \right) \| S_2^{f^* \sigma'} \| S_3^{e^* \sigma''} \right) \right) \\
&= \tau \cdot \tau_I(c_2(f)) \cdot \partial_H((s_3(f) C_m) \| S_2^{\sigma'} \| S_3^{e^* \sigma''}) \\
&= \tau \cdot \tau_I(c_3(f)) \cdot \partial_H(C_m \| S_2^{\sigma'} \| S_3^{f^* e^* \sigma''}) \\
&= \tau \cdot \tau_I \circ \partial_H(C_m \| S_2^{\sigma'} \| S_3^{f^* e^* \sigma''}).
\end{aligned}$$

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

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

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

$$\begin{aligned}
&\partial_H'([\tau \tau_I \circ \partial_H(C_k \| S_2^\theta \| S_3^\rho)] \| \mathbb{R}) \\
&= \tau \cdot \partial_H'(\tau_I \circ \partial_H(C_k \| S_2^\theta \| S_3^\rho) \| \mathbb{R}) \\
&= \tau \cdot c_1(e_1) \cdot \partial_H'(\tau_I \circ \partial_H(C_k \| S_2^\theta \| S_3^{e_1 \dots e_m}) \| \mathbb{R}) \\
&\quad \vdots \\
&= \tau c_1(e_1) \dots c_1(e_m) \partial_H'(\tau_I \circ \partial_H(C_k \| S_2^\theta \| S_3^\theta) \| \mathbb{R}) \\
&= \tau c_1(e_1) \dots c_1(e_m) \\
&\quad \cdot \partial_H'(\tau_I(c_3(\text{eos}) c_3(\text{eos}) \partial_H([s_3(\text{stop}) s_2(\text{stop}) s_1(\text{eos})] \| S_2 \| S_3)) \| \mathbb{R}) \\
&= \tau c_1(e_1) \dots c_1(e_m) \partial_H'(\tau \cdot \tau_I(c_3(\text{stop}) c_2(\text{stop}) s_1(\text{eos})) \| \mathbb{R}) \\
&= \tau c_1(e_1) \dots c_1(e_m) \tau \partial_H'(s_1(\text{eos})) \| \mathbb{R}) \\
&= \tau c_1(e_1) \dots c_1(e_m) c_1(\text{eos}),
\end{aligned}$$

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

$$\tau_I \circ \partial_H(C_0 \| S_2^\theta \| S_3^\sigma)$$

will do an infinite number of internal steps (steps from I). We will prove the following claim, which will finish the proof of Case 2.

Claim. $\tau_I \circ \partial_H(C_0 \| S_2^\theta \| S_3^\sigma) = \tau \delta$.

To prove this, we put $y = \partial_H(C_0 \parallel S_2^0 \parallel S_3^\sigma)$ 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* ($i, m < k$; $e \in D$).

A general step of type (a) looks like:

$$\begin{aligned} t_I \circ \partial_H(C_i \parallel S_2^{\sigma'} \parallel S_3^{d^* \sigma''}) &= t_I(c_3(d)c_2(e)\partial_H(C_m \parallel S_2^{e^* \sigma'} \parallel S_3^{\sigma''})) \\ &= ttt_I \circ \partial_H(C_m \parallel S_2^{e^* \sigma'} \parallel S_3^{\sigma''}), \end{aligned}$$

and a general step of type (b) looks like:

$$\begin{aligned} t_I \circ \partial_H(C_i \parallel S_2^{f^* \sigma'} \parallel S_3^{d^* \sigma''}) &= t_I(c_3(d)c_3(e)c_2(f)c_3(f)\partial_H(C_m \parallel S_2^{\sigma'} \parallel S_3^{f^* e^* \sigma''})) \\ &= ttttt_I \circ \partial_H(C_m \parallel S_2^{\sigma'} \parallel S_3^{f^* e^* \sigma''}). \end{aligned}$$

Thus, process $t_I(y) = t_I \circ \partial_H(C_0 \parallel S_2^0 \parallel S_3^\sigma)$ has states of the form

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

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⁻ (Theorem 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 KFAR₁, 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. \square

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).

Theorem 6.8. *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 Definition 6.4, are computable. Let ‘ $\widehat{\text{od}}$ ’ and ‘ $\widehat{\text{lb}}$ ’ be process representations of od, lb (defined in the proof of Theorem 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 $\text{od}(\sigma) > 0$ or $\text{lb}(\sigma^*0) = \delta$, with designated element ϵ , a name of the root). We have seven cases:

- (1) $\text{od}(\sigma) = 0$, so $\text{lb}(\sigma^*0) = \delta$. Then $X_\sigma = \delta$.
- (2) $\text{od}(\sigma) = 1$, and $\text{od}(\sigma^*0) > 0$ or $\text{lb}(\sigma^*0^*0) = \delta$. Then $X_\sigma = \text{lb}(\sigma^*0)X_{\sigma^*0}$.

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

It is not hard to see that g is need 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:

- (1) the transition from X_σ to X_{σ^*i} ($i = 0, 1$), execution of steps,
- (2) the history, saved in a stack
- (3) the calculation, containing $\widehat{\text{od}}$ and $\widehat{\text{lb}}$.

We have the configuration shown in 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) $\{s_3(d), r_3(d), c_3(d) : d \in \{\text{start}, \text{stop}, 0, 1, 2\}\}$,
- (3) $\{s_4(d), r_4(d), c_4(d) : d \in \{\text{start}, \text{stop}\} \cup A \cup \{\tau, \downarrow\}\}$,
- (4) $\{s_5(d), r_5(d), c_5(d) : d \in \{\text{stop}, 0, 1, \text{eos}\}\}$,
- (5) $\{s_6(d), r_6(d), c_6(d) : d \in \{0, 1, \text{eos}\}\}$,
- (6) $\{s_7(d), r_7(d), c_7(d) : d \in \{0, 1, \text{eos}\}\}$.

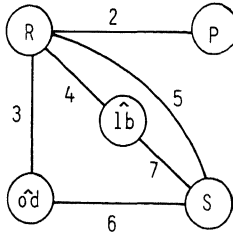


Fig. 34.

Part 1: description of P

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

$$\begin{aligned}
 P &= \sum_{a,b \in A_\tau - \{\delta\}} r_2(\langle a, b \rangle) P_{\langle a,b \rangle} + \sum_{a \in A_\tau} r_2(a) P_a + r_2(\downarrow); \\
 P_{\langle a,b \rangle} &= a s_2(0) P + b s_2(1) P, \\
 P_a &= a s_2(0) P, \quad (\text{if } a \neq \delta) \\
 P_\delta &= \delta.
 \end{aligned}$$

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 ($k = 5, 6, 7$):

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

Part 3: description of $\widehat{\text{od}}, \widehat{\text{lb}}, R$

We assume $\widehat{\text{od}}$ and $\widehat{\text{lb}}$ are specifications as given in the proof of Theorem 6.7 that work as follows:

- $\widehat{\text{od}}$ has input channel 6 and output channel 3;
- $\widehat{\text{lb}}$ 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 $\text{od}(\sigma)$ respectively $\text{lb}(\sigma)$ and send the result to R . Thus, after abstraction from channels 5 and 6, we have (let S contain σ):

$$\begin{aligned} \widehat{\text{od}} &= r_3(\text{start})s_3(\text{od}(\sigma))\widehat{\text{od}} + r_3(\text{stop}), \\ \widehat{\text{lb}} &= r_4(\text{start})s_4(\text{lb}(\sigma))\widehat{\text{lb}} + r_4(\text{stop}). \end{aligned}$$

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

$$\begin{aligned} R &= s_3(\text{start}) \left[r_3(0)s_5(0)s_4(\text{start}) \sum_{l \in \delta, \downarrow} r_4(l)s_2(l)s_3(\text{stop})s_4(\text{stop})s_5(\text{stop}) \right] \\ &\quad + \left[r_3(1)s_5(0)s_4(\text{start}) \sum_{l \in A_r - \{\delta\}} r_4(l)s_2(l)r_2(0) \right. \\ &\quad \quad \left. + r_3(2)s_5(0)s_4(\text{start}) \sum_{l \in A_r - \{\delta\}} r_4(l)r_5(0)s_5(1)s_4(\text{start}) \right. \\ &\quad \quad \left. \sum_{l' \in A_r - \{\delta\}} r_4(l')r_5(1)s_2(\langle l, l' \rangle) \sum_{i=0,1} r_2(i)s_5(i) \right] R. \end{aligned}$$

Next we do encapsulation:

$$H = \{r_i(d), s_i(d) : i = 2, \dots, 7; d \text{ from appropriate sets}\}$$

and abstraction:

$$I = \{c_i(d) : i = 2, \dots, 7; d \text{ from appropriate sets}\}.$$

Now, let S^σ denote stack S with contents σ ; then we can define processes $\{y_\sigma : \sigma \text{ a node-name}\}$ by the following equation:

$$Y'_\sigma = \iota_I \circ \partial_H(P \parallel S^\sigma \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}}), \quad y_\sigma = \tau_{\{\iota\}}(Y'_\sigma)$$

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

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

Proof. We show processes y_σ satisfy the seven defining equations for x_σ , multiplied by τ .

(1) $\text{od}(\sigma) = 0$, so $\text{lb}(\sigma^*0) = \delta$. Then

$$\begin{aligned} y_\sigma &= \tau_I \circ \partial_H(P \parallel S^\sigma \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}}) \\ &= \tau_I(c_3(\text{start})c_3(0)c_5(0)c_4(\text{start})c_4(\delta)c_2(\delta)c_3(\text{stop})c_4(\text{stop})c_5(\text{stop})\delta) = \tau\delta. \end{aligned}$$

(2) $\text{od}(\sigma) = 1$ and $(\text{od}(\sigma^*0) > 0 \text{ or } \text{lb}(\sigma^*0^*0) = \delta)$. Then

$$\begin{aligned} y_\sigma &= \tau_I \circ \partial_H(P \parallel S^\sigma \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}}) \\ &= \tau_I(c_3(\text{start})c_3(1)c_5(0)c_4(\text{start})c_4(\text{lb}(\sigma^*0)) \\ &\quad \cdot c_2(\text{lb}(\sigma^*0))\partial_H(P_{\text{lb}(\sigma^*0)} \parallel S^{\sigma^*0} \parallel r_2(0)R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}})) \\ &= \tau \cdot \tau_I(\text{lb}(\sigma^*0)c_2(0) \cdot \partial_H(P \parallel S^{\sigma^*0} \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}})) \\ &= \tau \text{lb}(\sigma^*0)\tau_I \circ \partial_H(P \parallel S^{\sigma^*0} \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}}) = \tau \text{lb}(\sigma^*0)y_{\sigma^*0}. \end{aligned}$$

(3) $\text{od}(\sigma) = 1$ and $\text{lb}(\sigma^*0^*0) = \downarrow$. Then

$$\begin{aligned} y_\sigma &= \tau_I \circ \partial_H(P \parallel S^\sigma \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}}) \\ &= \tau \text{lb}(\sigma^*0)y_{\sigma^*0} = \tau \text{lb}(\sigma^*0)\tau_I \circ \partial_H(P \parallel S^{\sigma^*0} \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}}) \\ &= \tau \text{lb}(\sigma^*0)\tau_I(c_3(\text{start})c_3(0)c_5(0) \\ &\quad \cdot c_4(\text{start})c_4(\downarrow)c_2(\downarrow)c_3(\text{stop})c_4(\text{stop})c_5(\text{stop})) \\ &= \tau \text{lb}(\sigma^*0)\tau = \tau \text{lb}(\sigma^*0). \end{aligned}$$

(4) $\text{od}(\sigma) = 2$, both $(\text{od}(\sigma^*0) > 0 \text{ or } \text{lb}(\sigma^*0^*0) = \delta)$ and $(\text{od}(\sigma^*1) > 0 \text{ or } \text{lb}(\sigma^*1^*0) = \delta)$. Then

$$\begin{aligned} y_\sigma &= \tau_I \circ \partial_H(P \parallel S^\sigma \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}}) \\ &= \tau_I \left(c_3(\text{start})c_3(2)c_5(0)c_4(\text{start})c_4(\text{lb}(\sigma^*0))c_5(0)c_5(1) \right. \\ &\quad \left. c_4(\text{start})c_4(\text{lb}(\sigma^*1))c_5(1)c_2(\langle \text{lb}(\sigma^*0), \text{lb}(\sigma^*1) \rangle) \right. \\ &\quad \left. \partial_H \left(P_{\langle \text{lb}(\sigma^*0), \text{lb}(\sigma^*1) \rangle} \parallel S^\sigma \parallel \left(\sum_{i=0,1} r_2(i)s_5(i)R \right) \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}} \right) \right) \end{aligned}$$

$$\begin{aligned}
&= \tau \cdot \tau_I(\text{lb}(\sigma^*0)c_2(0)c_3(0)\partial_H(P \parallel S^{\sigma^*0} \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}})) \\
&\quad + \text{lb}(\sigma^*1)c_2(1)c_3(1)\partial_H(P \parallel S^{\sigma^*1} \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}})) \\
&= \tau(\text{lb}(\sigma^*0)y_{\sigma^*0} + \text{lb}(\sigma^*1)y_{\sigma^*1}).
\end{aligned}$$

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

$$\begin{aligned}
y_\sigma &= \tau(\text{lb}(\sigma^*0)y_{\sigma^*0} + \text{lb}(\sigma^*1)y_{\sigma^*1}) = \tau(\text{lb}(\sigma^*0)y_{\sigma^*0} + \text{lb}(\sigma^*1)\tau) \\
&= \tau(\text{lb}(\sigma^*0)y_{\sigma^*0} + \text{lb}(\sigma^*1)).
\end{aligned}$$

(6) and (7): likewise. \square (of Claim)

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: $\text{od}(\varepsilon) = 0$. The root has out-degree 0, so since graph $\rightarrow \circ \rightarrow$ is not in \mathbb{G}_κ , we have $g = \rightarrow \circ \delta$, and we can define $X = \delta$.

Case 2: $\text{od}(\varepsilon) = 1$. Suppose $\text{lb}(0) = a$. Then

$$X = a t_I \circ \partial_H(P \parallel T_0 \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}}) = a Y_0.$$

Case 3: $\text{od}(\varepsilon) = 2$. Suppose $\text{lb}(0) = a$ and $\text{lb}(1) = b$. Then

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

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 $g \Leftrightarrow_{\text{rr}\delta} \tau_{\{t\}}(h)$. This finishes the proof of Theorem 6.8. \square

Corollary 6.9. *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 Theorem 6.8 there is a specification E with unique solution k such that $h \Leftrightarrow_{\text{rr}\delta} \tau_{\{t\}}(k)$. It easily follows that

$$g \Leftrightarrow_{\text{rr}\delta} \tau_{\{t'\}}(h) \Leftrightarrow_{\text{rr}\delta} \tau_{\{t,t'\}}(k). \quad \square$$

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.

Lemma 6.10. *Let $g \in \mathbb{G}_{\aleph_0}$. Then $g \Leftrightarrow_{\text{rr}\delta} h$, 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 $g \in \mathbb{G}_{\aleph_0}^0$), and coded (see Definition

6.1). We define h as follows:

- (1) $\text{NODES}(h) = \{ \langle s, n \rangle : s \in \text{NODES}(g), s \neq \text{ROOT}(g), n < \text{out-degree}(s) \}$
 $\cup \{ \langle s, 0 \rangle : s \in \text{NODES}(g), \text{ and } s = \text{ROOT}(g) \text{ or } \text{out-degree}(s) = 0 \}$.
- (2) $\text{EDGES}(h) = \left\{ \begin{array}{l} \langle \langle s, 0 \rangle \rangle \xrightarrow{l}_n \langle \langle t, 0 \rangle \rangle : \langle s \rangle \xrightarrow{l}_n \langle t \rangle \in \text{EDGES}(g), s = \text{ROOT}(g) \\ (n < \text{od}(s) \text{ the name of the edge, } l \text{ a label}) \end{array} \right\}$
 $\cup \left\{ \langle \langle s, n \rangle \rangle \xrightarrow{l}_0 \langle \langle t, 0 \rangle \rangle : \langle s \rangle \xrightarrow{l}_n \langle t \rangle \in \text{EDGES}(g), s \neq \text{ROOT}(g) (n, l \text{ as above}) \right\}$
 $\cup \left\{ \langle \langle s, n \rangle \rangle \xrightarrow{\tau}_1 \langle \langle s, n+1 \rangle \rangle : s \in \text{NODES}(g), s \neq \text{ROOT}(g) [(n+1) < \text{od}(s), l \text{ a label}] \right\}$
 $\cup \left\{ \langle \langle s, n \rangle \rangle \xrightarrow{\tau}_1 \langle \langle s, 0 \rangle \rangle : s \in \text{NODES}(g), s \neq \text{ROOT}(g) [(n+1) = \text{od}(s), l \text{ a label}] \right\}$.
- (3) $\text{ROOT}(h) = \langle \text{ROOT}(g), 0 \rangle$.
- (4) The endpoint label of $\langle s, 0 \rangle \in \text{NODES}(h)$ is the endpoint label of $s \in \text{NODES}(g)$.
 An example might clarify the matter (Fig. 35).

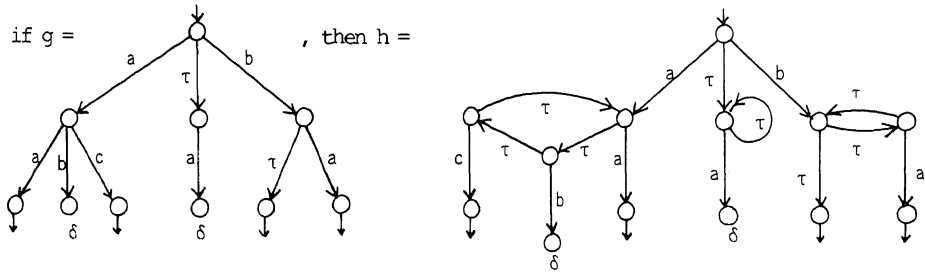


Fig. 35.

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 $R \subseteq \text{NODES}(g) \times \text{NODES}(h)$ as follows: R relates all nodes $s \in \text{NODES}(g)$ with all $\langle s, n \rangle \in \text{NODES}(h)$ ($n < \text{od}(s)$ or $n = 0 = \text{od}(s)$).

It is easy to prove that $R : g \xrightarrow{\tau, \delta} h$:

- (1) If $\langle s \rangle \xrightarrow{l}_n \langle t \rangle$ is an edge in g with label l ($n < \text{od}(s)$) and $R(s, \langle s, k \rangle)$, then
 (1.1) if $k \leq n$, take path

$$\langle \langle s, k \rangle \rangle \xrightarrow{\tau}_1 \langle \langle s, k+1 \rangle \rangle \xrightarrow{\tau}_1 \cdots \xrightarrow{\tau}_1 \langle \langle s, n \rangle \rangle \xrightarrow{l}_0 \langle \langle t, 0 \rangle \rangle$$

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

- (1.2) if $k > n$, take path

$$\langle \langle s, k \rangle \rangle \xrightarrow{\tau}_1 \cdots \xrightarrow{\tau}_1 \langle \langle s, \text{od}(s)-1 \rangle \rangle \xrightarrow{\tau}_1 \langle \langle s, 0 \rangle \rangle \xrightarrow{l}_0 \cdots \xrightarrow{\tau}_1 \langle \langle s, n \rangle \rangle \xrightarrow{l}_0 \langle \langle t, 0 \rangle \rangle$$

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

- (2) Conversely, for each edge $\langle s, n \rangle \xrightarrow{t}_0 \langle t, 0 \rangle$ in h we have $\textcircled{s} \xrightarrow{t}_n \textcircled{t}$ in g .
- (3) Endpoints and root are alright since nothing is changed there. \square

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

Proof. By Lemma 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 $\widehat{\text{od}}$, $\widehat{\text{lb}}$. 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 the proof of Theorem 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 :

$$X = \sum_{i < n_0} \text{lb}(i) \cdot t_i \circ \partial_H(P \parallel T_i \parallel R \parallel \widehat{\text{od}} \parallel \widehat{\text{lb}}),$$

$P, S, T_i, R, \widehat{\text{od}}, \widehat{\text{lb}}, H, I$ given in the proof of Theorem 6.8.

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

Remark 6.12. When we want to translate the trick in the proof of Lemma 6.10 in the graph-model to the theory of ACP_τ , we have to use an extended version of KFAR. The details of this translation are not clear, however.

7. Computably recursively definable processes

In Section 6, we looked at computable graphs. In this section, we will 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.

Remark 7.1 (coding). It is not hard to give a computable injective coding function with computable inverse from all finite ACP_τ -terms to natural numbers, so we will not mention this function in the following.

Definition 7.2. Let $E = \{E_n : n < \omega\}$ be a specification. E is *computable* if the function $f : n \rightarrow T_n$ is computable (T_n is the right-hand side of the equation for X_n).

Lemma 7.3. *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_τ -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. \square

Lemma 7.4. *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 Section 3.2 are computable, so that if graphs g, h are computable (as defined in Definition 6.5), then so are graphs $g+h, g \cdot h, g \parallel h, g \perp\!\!\!\perp h, g \mid h, \partial_H(g), \tau_I(g), \pi_n(g)$ and $t_I(g)$ (defined in Definition 5.15). Thus, we see that the canonical graph of each finite ACP_τ -term is computable, so we obtain from the proof of Theorem 5.11 and Lemma 7.3 that each computable guarded specification without abstraction has a computable solution. \square

Corollary 7.5. *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. Combine Theorem 6.11 and Lemma 7.4. \square

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_τ is much greater than the defining power of theory ACP (where ACP is the theory given by the left-hand column of Table 1).

Definition 8.1. Let the set of atoms A contain two elements a, b different from δ . Let a function $f: \omega \rightarrow \{a, b\}$ 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 ($x^f = f(0)f(1)f(2)\dots$). By Theorem 7.5, each x^f for computable f is the abstraction of a process, definable by a finite guarded specification without abstraction.

Theorem 8.2. *There exists a computable function $f: \omega \rightarrow \{a, b\}$ such that process x^f*

(defined in Definition 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 $\langle E_n : n < \omega \rangle$. By Theorem 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 Lemma 7.4, each g_n is computable. Now, to each specification E_n ($n < \omega$) we assign a function $f_n : \omega \rightarrow \{a, b\}$ 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 $f : \omega \rightarrow \{a, b\}$ by

$$f(n) = \begin{cases} a & \text{if } f_n(n) = b, \\ b & \text{if } f_n(n) = a. \end{cases}$$

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

References

- [1] J.C.M. Baeten, *Procesalgebra* (Kluwer Programmatuurkunde, Deventer, The Netherlands, 1986) (in Dutch).
- [2] J.C.M. Baeten, J.A. Bergstra and J.W. Klop, Syntax and defining equations for an interrupt mechanism in process algebra, *Fund. Inform.* **IX** (1986) 127–168.
- [3] J.C.M. Baeten, J.A. Bergstra and J.W. Klop, Conditional axioms and α/β calculus in process algebra, in: M. Wirsing, ed., *IFIP Conf. on Formal Description of Programming Concepts*, Eberup 1986 (North-Holland, Amsterdam, 1987).
- [4] J.W. de Bakker and J.I. Zucker, Compactness in semantics for merge and fair merge, in: E. Clarke and D. Kozen, eds., *Proc. Workshop Logics of Programs*, Lecture Notes in Computer Science **164** (Springer, Berlin, 1983) 18–33.
- [5] J.W. de Bakker and J.I. Zucker, Processes and a fair semantics for the ADA rendez-vous, in: J. Díaz, ed., *Proc. 10th ICALP*, Lecture Notes in Computer Science **154** (Springer, Berlin, 1983) 52–66.
- [6] J.A. Bergstra and J.W. Klop, An abstraction mechanism for process algebras, Report IW-231/83, Mathematisch Centrum, Amsterdam, 1983.
- [7] J.A. Bergstra and J.W. Klop, Algebra of communicating processes with abstraction, *Theoret. Comput. Sci.* **37**(1) (1985) 77–121.
- [8] J.A. Bergstra and J.W. Klop, Verification of an alternating bit protocol by means of process algebra, in: W. Bibel and K.P. Jantke, eds., *Mathematical Methods of Specification and Synthesis of Software Systems '85*, Mathematical Research **31** (Akademie-Verlag, Berlin, 1986) 9–23.
- [9] J.A. Bergstra and J.W. Klop, Fair FIFO queues satisfy an algebraic criterion for protocol correctness, Report CS-R 8405, Centrum voor Wiskunde en Informatica, Amsterdam, 1984.
- [10] J.A. Bergstra and J.W. Klop, A complete inference system for regular processes with silent moves, Report CS-R 8420, Centrum voor Wiskunde en Informatica, Amsterdam, 1984 (to appear in: *Proc. Logic Colloquium*, Hull, 1986).

- [11] J.A. Bergstra and J.W. Klop, Algebra of communicating processes, in: J.W. de Bakker, M. Hazewinkel and J.K. Lenstra, eds., *Proc. CWI Symp. on Mathematics and Computer Science* (North-Holland, Amsterdam, 1986) 89-138.
- [12] J.A. Bergstra and J.V. Tucker, Top-down design and the algebra of communicating processes, *Sci. Comput. Programming* **5**(2) (1985) 171-199.
- [13] G. Costa and C. Stirling, A fair calculus of communicating systems, Report CSR-137-83, University of Edinburgh, 1983.
- [14] P. Darondeau, Infinitary languages and fully abstract models for fair asynchrony, Report CNRS-IRISA, Rennes, 1984.
- [15] R.J. Van Glabbeek, Bounded nondeterminism and the Approximation Induction Principle in process algebra, in: F.J. Brandenburg, G. Vidal-Naquet and M. Wirsing, eds., *Proc. STACS*, Lecture Notes in Computer Science **247** (Springer, Berlin, 1987) 336-347.
- [16] M. Hennessy, Modelling fair processes, *Proc. 16th ACM Symp. on Theory of Computing*, Washington, DC, 1984.
- [17] M. Hennessy, Axiomatising finite delay operators, *Acta Inform.* **21** (1984) 61-88.
- [18] M. Hennessy, An algebraic theory of fair asynchronous communicating processes, in: W. Brauer, ed., *Proc. 12th ICALP*, Lecture Notes in Computer Science **194** (Springer, Berlin, 1985) 260-269.
- [19] C.J. Koomen, Algebraic specification and verification of communication protocols, *Sci. Comput. Programming* **5** (1985) 1-36.
- [20] C.P.J. Koymans and J.C. Mulder, A modular approach to protocol verification using process algebra, Report LGPS 6, State University of Utrecht, The Netherlands, 1986.
- [21] E. Kranakis, Fixed-point equations with parameters in the projective model, Report CS-R8606, Centrum voor Wiskunde en Informatica, Amsterdam, 1986.
- [22] J.-J.Ch. Meyer, Merging regular processes by means of fixed-point theory, *Theoret. Comput. Sci.* **45** (1986) 193-260.
- [23] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science **92** (Springer, Berlin, 1980).
- [24] D.M.R. Park, Concurrency and automata on infinite sequences, in: P. Denssen, ed., *Proc. 5th GI Conf.*, Lecture Notes in Computer Science **104** (Springer, Berlin, 1981) 167-183.
- [25] J. Parrow, *Fairness properties in process algebra*, Ph.D. Thesis, Department of Computer Science, Uppsala University, 1985.
- [26] F.W. Vaandrager, Verification of two communication protocols by means of process algebra, Report CS-R8608, Centrum voor Wiskunde en Informatica, Amsterdam, 1986.
- [27] J.L.M. Vrancken, The algebra of communicating processes with empty process, Report FVI 86-01, University of Amsterdam, 1986.
- [28] J.C.M. Baeten and R.J. van Glabbeek, Another look at abstraction in process algebras, Report CS-R8701, Centre for Mathematics and Computer Science, Amsterdam, 1987 (to appear in: *Proc. 14th ICALP*, Karlsruhe, 1987).
- [29] J.C.M. Baeten and R.J. van Glabbeek, Abstraction and empty process in process algebra, Report CS-R8721, Centre for Mathematics and Computer Science, Amsterdam, 1987.