PROCESS THEORY BASED ON BISIMULATION SEMANTICS

J.A. Bergstra
University of Amsterdam, Department of Computer Science
P.O. Box 19268, 1000 GG Amsterdam;
State University of Utrecht, Department of Philosophy,
P.O. Box 8810, 3508 TA Utrecht.

J.W. Klop
Centre for Mathematics and Computer Science,
P.O. Box 4079, 1009 AB Amsterdam;
Free University, Department of Mathematics and Computer Science,
De Boelelaan 1081, 1081 HV Amsterdam.

Note: Research partially supported by ESPRIT project 432, Meteor.
Chapter 1 of this paper is a modified version of 'Process algebra: specification and verification in bisimulation semantics', from CWI Monograph 4, Proc. of the CWI Symposium Mathematics and Computer Science II (eds. Hazewinkel, Lenstra, Meertens), North-Holland, Amsterdam 1986. Permission of the editors to include the present Chapter 1 here is gratefully acknowledged.

ABSTRACT
In this paper a process is viewed as a labeled graph modulo bisimulation equivalence. Three topics are covered: (i) specification of processes using finite systems of equations over the syntax of process algebra; (ii) inference systems which are complete for proving the equivalence of regular (finite state) processes; (iii) variations of the bisimulation model.

Introduction

We will discuss process theory on the basis of a given semantic concept. A process will be a rooted directed graph where arcs are labeled with actions. An example may clarify this matter (see Figure 1.1).

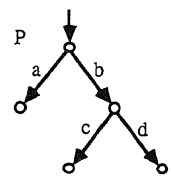


Figure 1.1

For instance the process P denotes a process that has two options for initial actions, a and b. After the a-step P will terminate, but after having done the b-step P has again two options, c and d.

Now obviously the concept of a process should be made independent of its incidental coding in a graph. So we must determine an appropriate equivalence relation on graphs. There are several possibilities for such equivalence relations. Relevant references are for instance: Brookes, Hoare & Roscoe [84], Hennessy [88], De Nicola & Hennessy [83], and Phillips [87]. However, bisimulation equivalence, as introduced in Park [81], stands out, in our view, as the most natural identification mechanism on process graphs discovered thus far.

Having thus established roughly the domain of processes as that of process graphs modulo

bisimulation, the next step is to incorporate the major discovery of Milner [80], namely that processes have an algebraic structure. Our paper has to balance between two opposite poles: (a) the syntax of process algebra and its axioms and proof rules, (b) the extremely rich world of process graphs and bisimulations. The main relations between (a) and (b) are as follows:

(1) Using the syntax of process algebra we may write down equational axioms and axiom schemes that 'specify' bisimulation semantics. These axioms capture the intended process semantics in algebraic terms.

Chapter 1 contains a survey of a possible syntax of process algebra (ACP, Algebra of Communicating Processes, and extensions) and its axioms and rules: see Table 22. In this setting one finds the concept of a process algebra, i.e. a model of (the axioms of) process algebra: an appropriate class of process graphs, together with a definition of the algebraic operators on these process graphs such that bisimulation equivalence becomes a congruence relation. The main model is $\mathcal{G}/\leftrightarrows_{\tau\delta}$, described in Chapter 1, Section 1.13.

(2) Equations over the syntax of process algebra having free variables ranging over processes can be solved in bisimulation semantics. In particular so-called systems of guarded recursion equations turn out to have unique solutions. These systems are used to specify processes.

Chapter 1 contains several examples of process specifications as well as a general theorem (1.14.2) that expresses the adequacy of finite guarded recursive equational specifications for the description of computable processes.

(3) Suppose that a particular class of process specifications in the sense of (2) is given. Then a major question is to decide whether or not two specifications specify the same process. This matter is undecidable in general, but in some cases positive results can be obtained.

Chapter 2 discusses the bisimulation equivalence problem for regular processes. For this case a complete inference system is presented.

(4) In the absence of the silent step τ, for each process algebra (based on graphs modulo bisimulation equivalence) one can define the corresponding algebra of processes modulo n with n a natural number. In this algebra processes are identified whenever the restrictions of their behaviour to the first n actions are bisimilar.

Complementary to this construction 'modulo n', there is the construction of projective limits of process algebras and processes. Equivalently, such a projective limit can be viewed as a topological completion in an appropriate topology. This leads to a topological view of process domains related to the work of De Bakker & Zucker [82a,b].

In Chapter 3 we study in detail the topological properties of process domains that result from general topological constructions on the basis of spaces with process graphs modulo bisimulation.

Contents

1. Specification and verification in bisimulation semantics

This chapter is a modified version of Bergstra & Klop [86c]. It serves as an introduction to both process algebra and bisimulation semantics. Sections 1-11 provide syntax and defining equations for our operator set of process algebra as well as several examples of process specifications including counters, bags, stacks and queues. Section 1.12 contains an extended example of a specification and verification in process algebra. To this end an alternating bit protocol is verified and specified in all detail. Sections 1.13, 1.14 introduce the bisimulation model and describe the expressive power of recursive specifications in the context of the bisimulation model.

1.1. Basic Process Algebra.

The kernel of all axiom systems for processes that we will consider, is Basic Process Algebra. The processes that we will consider are capable of performing atomic steps or actions a,b,c,..., with the idealization that these actions are events without positive duration in time; it takes only one moment to execute an action. The actions are combined into composite processes by the operations + and ·, with the interpretation that (a+b)·c is the process that first chooses between executing a or b and, second, performs the action c after which it is finished. (We will often suppress the dot and write (a+b)c.) These operations, 'alternative composition' and 'sequential composition' (or just sum and product), are the basic constructors of processes. Since time has a direction, multiplication is not commutative; but addition is, and in fact it is stipulated that the options (summands) possible at some stage of the process form a *set*. Formally, we will require that processes x,y,... satisfy the following axioms:

BPA

$x+y = y+x$
$(x+y)+z = x+(y+z)$
$x+x = x$
$(x+y)z = xz+yz$
$(xy)z = x(yz)$

Table 1

Thus far we used 'process algebra' in the generic sense of denoting the area of algebraic approaches to concurrency, but we will also adopt the following technical meaning for it: any model of these axioms will be a *process algebra*.. The simplest process algebra, then, is the term model of BPA (Basic Process Algebra), whose elements are BPA-expressions (built from the atoms a,b,c,... by means of the basic constructors) modulo the equality generated by the axioms. We will denote this structure with $\mathbb{A}_\omega$. This process algebra contains only finite processes; things get more lively if we admit recursion enabling us to define infinite processes. Even at this stage one can define, recursively, interesting processes; consider for instance the counter in Table 2.

COUNTER

$$X = (zero + up \cdot Y) \cdot X$$
$$Y = down + up \cdot Y \cdot Y$$

Table 2

Here 'zero' is the action that asserts that the counter has value 0, and 'up' and 'down' are the actions of incrementing, respectively decrementing, the counter by one unit. The process COUNTER is now represented by X; Y is an auxiliary process. COUNTER is a 'perpetual' process, that is, all its execution traces are infinite. Such a trace is e.g. zero·zero·up·down·zero·up·up·up· .... . A question of mathematical interest only is: can COUNTER be defined in a single equation, without auxiliary processes? The negative answer is an immediate consequence of the following fact:

1.1.1. THEOREM. *Let a system* $\{X_i = T(X_1,...,X_n) \mid i = 1,...,n\}$ *of guarded fixed point equations over* BPA *be given. Suppose the solutions* $\underline{X}_i$ *are all perpetual. Then they are regular.*

The solutions are in this case labeled transition graphs—modulo a certain equivalence relation which will be extensively discussed in the sequel. Two concepts in this statement need also an explanation: a fixed point equation (or recursion equation), like $X = (zero + up \cdot Y) \cdot X$ is *guarded* if every occurrence of a recursion variable in the right hand side is preceded ('guarded') by an occurrence of an action. For instance, the occurrence of X in the right-hand side of $X = (zero + up \cdot Y) \cdot X$ is guarded since, when this X is accessed, one has to pass either the guard zero or the guard up. A non-example: the equation $X = X + a \cdot X$ is not guarded. Furthermore, a process is *regular* if it has only finitely many 'states'; clearly, COUNTER is not regular since it has just as many states as there are natural numbers. Let us mention one other property of processes which have a finite recursive specification (by means of guarded recursion equations) in BPA: such processes are *uniformly finitely branching*. A process is finitely branching if in each of its states it can take steps (and thereby transform itself) to only finitely many subprocesses; for instance, the process defined by $X = (a+b+c)X$ has in each state branching degree 3. 'Uniformly' means that there is uniform bound on the branching degrees throughout the process.

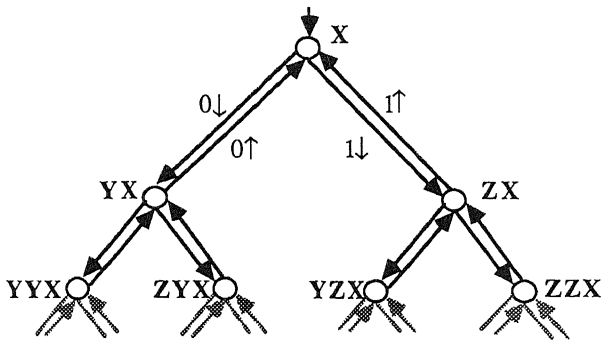
In fact, a more careful treatment is necessary to define concepts like 'branching degree' rigorously. For, clearly, the branching degree of a + a ought to be the same as that of the process 'a', since a + a = a. And the process X = aX will be the same as the process X = aaX; in turn these will be identified with the process X = aX + aaX. In the sequel we will extensively discuss the semantic criterion by means of which these processes are identified ('bisimilarity'). Milner [84] has found a simple axiom system (extending BPA) which is able to deal with recursion and which is complete for regular processes with respect to 'bisimilarity'. (See Section 2.3 in Chapter 2.)

Another non-trivial example is the following specification of the process behaviour of a Stack with data 0,1:

STACK	
	$X = 0\downarrow.YX + 1\downarrow.ZX$
	$Y = 0\uparrow + 0\downarrow.YY + 1\downarrow.ZY$
	$Z = 1\uparrow + 0\downarrow.YZ + 1\downarrow.ZZ$

Table 3

Here $0\downarrow$ and $0\uparrow$ are the actions 'push 0' and 'pop 0', respectively; likewise for 1. Now Stack is specified by the first recursion variable, X. Indeed, according to the first equation the process X is capable of performing either the action $0\downarrow$, after which the process is transformed into YX, or $1\downarrow$, after which the process is transformed into ZX. In the first case we have using the second equation $YX = (0\uparrow + 0\downarrow.YY + 1\downarrow.ZY)X = 0\uparrow\cdot X + 0\downarrow.YYX + 1\downarrow.ZYX$. This means that the process YX has three options; after performing the first one ($0\uparrow$) it behaves like the original X. Continuing in this manner we find a transition diagram or *process graph* as in Figure 1.2.



Stack

Figure 1.2

Before proceeding to the next section, let us assure the reader that the omission of the other distributive law, $z(x + y) = zx + zy$, is intentional. The reason will become clear after the introduction of 'deadlock'.

1.2. Deadlock. A vital element in the present set-up of process algebra is the process δ signifying 'deadlock'. The process ab performs its two steps and then stops, succesfully; but the process abδ deadlocks after the a- and b-action: it wants to do a proper action but it cannot. So δ is the acknowledgement of stagnation. With this in mind, the axioms to which δ is subject, should be clear:

DEADLOCK

$\delta + x = x$
$\delta \cdot x = \delta$

Table 4

(In fact, it can be argued that 'deadlock' is not the most appropriate name for the process constant δ. In the sequel we will encounter a process which can more rightfully claim this name: $\tau\delta$, where τ is the silent step. We will stick to the present terminology, however.)

The axiom system of BPA (Table 1) together with the present axioms for δ is called BPA_δ. We are now in a position to motivate the absence in BPA of the 'other' distributive law: $z(x+y) = zx+zy$. For, suppose it would be added. Then $ab = a(b + \delta) = ab + a\delta$. This means that a process with deadlock possibility is equal to one without, conflicting with our intention to model also deadlock behaviour of processes.

The essential role of the new process δ will only be fully appreciated after the introduction of communication, below.

1.3. The merge operator.

If x,y are processes, their 'parallel composition' $x \parallel y$ is the process that first chooses whether to do a step in x or in y, and proceeds as the parallel composition of the remainders of x,y. In other words, the steps of x,y are interleaved or *merged*. Using an auxiliary operator $\lfloor\!\lfloor$ (with the interpretation that $x \lfloor\!\lfloor y$ is like $x \parallel y$ but with the commitment of choosing the initial step from x) the operation $\parallel$ can be succinctly defined by the axioms:

MERGE

$x \parallel y$	$= x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$
$ax \lfloor\!\lfloor y$	$= a(x \parallel y)$
$a \lfloor\!\lfloor y$	$= ay$
$(x + y) \lfloor\!\lfloor z$	$= x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$

Table 5

The system of nine axioms consisting of BPA and the four axioms for merge will be called PA. Moreover, if the axioms for δ are added, the result will be PA_δ. The operators $\parallel$ and $\lfloor\!\lfloor$ will also be called *merge* and *left-merge* respectively.

The merge operator corresponds to what in the theory of formal languages is called *shuffle*. The shuffle of the words ab and cd is the set of words {abcd, acbd, cabd, acdb, cadb, cdab}. Merging the processes ab and cd yields the process

$$ab\|cd = ab\lfloor\!\lfloor cd + cd\lfloor\!\lfloor ab = a(b\|cd) + c(d\|ab) = a(b\lfloor\!\lfloor cd + cd\lfloor\!\lfloor b) + c(d\lfloor\!\lfloor ab + ab\lfloor\!\lfloor d) =$$
$$a(bcd + c(d\|b)) + c(dab + a(b\|d)) = a(bcd + c(db+bd)) + c(dab + a(bd+db)),$$

By means of these projections a distance between processes x,y can be defined: $d(x,y) = 2^{-n}$ where n is the least natural number such that $\pi_n(x) \neq \pi_n(y)$, and $d(x,y) = 0$ if there is no such n. If the term model $\mathbb{A}_\omega$ of BPA (or PA) as in Section 1.1 is equipped with this distance function, the result is an ultrametrical space $(\mathbb{A}_\omega, d)$. By metrical completion we obtain a model of BPA (resp. PA) in which all systems of guarded recursion equations have a unique solution. In fact, the guardedness condition is exactly what is needed to associate a contracting operator on the complete metrical space with a guarded recursion equation. (E.g. to the recursion equation $X = aX$ the contracting function $f(x) = ax$ is associated; indeed $d(f(x),f(y)) \leq d(x,y)/2$.) Banach's contraction theorem then proves the existence of a unique fixed point. This model construction has been employed in various settings by De Bakker & Zucker [82a,b], who also posed the question whether *unguarded* fixed point equations, such as $X = aX + X$ or $Y = (aY \parallel Y) + b$, always have a solution in the metric completion of $(\mathbb{A}_\omega, d)$ as well. This turns out to be the case:

1.4.1. THEOREM. *Let* q *be an arbitrary process in the metric completion of* $(\mathbb{A}_\omega, d)$ *and let* $X = s(X)$ *be a recursion equation in the signature of* PA.

Then the sequence q, s(q), s(s(q)), s(s(s(q))), ... *converges to a solution* $q^* = s(q^*)$.

For a proof see Bergstra & Klop [87]. In general, the fixed points $q^* = s(q^*)$ are not unique. The proof of 1.4.1 in Bergstra & Klop [87] is combinatorial in nature; it is not at all clear whether this convergence result can be obtained by the 'usual' convergence proof methods, such as invoking Banach's fixed point theorem or (in a complete partial order setting) the Knaster-Tarski fixed point theorem. In Kranakis [87] the present theorem is extended to the case where s(X) may contain parameters.

1.4.2. REMARK. An alternative way to obtain this model (the metric completion of $(\mathbb{A}_\omega, d)$) is as follows. Let $\mathbb{A}_n$ denote $\mathbb{A}_\omega$ modulo the equation $x = \pi_n(x)$; so $\mathbb{A}_n$ is the initial algebra of BPA $\cup \{x = \pi_n(x)\}$, containing only processes of depth at most n. Now the family of models and projections $(\mathbb{A}_n, \pi_n : A_{n+1} \to A_n \mid n \geq 0\}$ has a projective limit $\mathbb{A}^\infty$. This structure is isomorphic to the metric completion of $(\mathbb{A}_\omega, d)$. Therefore we will use $\mathbb{A}^\infty$ as an alternative notation for the metric completion of $(\mathbb{A}_\omega, d)$.

1.5. Communication.

So far, the parallel composition or merge ($\parallel$) did not involve communication in the process $x \parallel y$: one could say that x and y are 'freely' merged or interleaved. However, some actions in one process may need an action in another process for an actual execution, like the act of shaking hands requires simultaneous acts of two persons. In fact, 'handshaking' is the paradigm for the type of communication which we will introduce now. If $A = \{a,b,c,...,\delta\}$ is the action alphabet, let us adopt a binary communication function $| : A \times A \to A$ satisfying the axioms in Table 7.

COMMUNICATION FUNCTION

$a \mid b$	$= b \mid a$
$(a \mid b) \mid c$	$= a \mid (b \mid c)$
$\delta \mid a$	$= \delta$

Table 7

Here a,b vary over A, including δ. We can now specify *merge with communication* ; we use the same notation $\|$ as for the 'free' merge in Section 1.3 since in fact 'free' merge is an instance of merge with communication by choosing the communication function trivial, i.e. $a \mid b = \delta$ for all a,b $\in$ A. There are now two auxiliary operators, allowing a finite axiomatisation: left-merge ($\lfloor\!\lfloor$) as before and $\mid$ (*communication merge* or simply 'bar'), which is an extension of the communication function in Table 7 to all processes, not only the atoms. The axioms for $\|$ and its auxiliary operators are given in Table 8.

MERGE WITH COMMUNICATION

$x \,\|\| \, y$	$= x \lfloor\!\lfloor y + y \lfloor\!\lfloor x + x \mid y$
$ax \lfloor\!\lfloor y$	$= a(x \| y)$
$a \lfloor\!\lfloor y$	$= ay$
$(x + y) \lfloor\!\lfloor z$	$= x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$
$ax \mid b$	$= (a \mid b)x$
$a \mid bx$	$= (a \mid b)x$
$ax \mid by$	$= (a \mid b)(x \| y)$
$(x+y) \mid z$	$= x \mid z + y \mid z$
$x \mid (y+z)$	$= x \mid y + x \mid z$

Table 8

We also need the so-called *encapsulation* operators ∂_H (for every $H \subseteq A$) for removing unsuccessful attempts at communication:

ENCAPSULATION

$\partial_H(a)$	$= a$ if $a \notin H$
$\partial_H(a)$	$= \delta$ if $a \in H$
$\partial_H(x+y)$	$= \partial_H(x) + \partial_H(y)$
$\partial_H(xy)$	$= \partial_H(x) \cdot \partial_H(y)$

Table 9

These axioms express that ∂_H 'kills' all atoms mentioned in H, by replacing them with δ. The axioms for BPA, DEADLOCK together with the present ones in Tables 7-9 constitute the axiom system ACP (Algebra of Communicating Processes). Typically, a system of communicating processes $x_1,...,x_n$ is now represented in ACP by the expression $\partial_H(x_1 \| ... \| x_n)$. Prefixing the encapsulation operator says that the system $x_1,...,x_n$ is to be perceived as a separate unit with

respect to the communication actions mentioned in H; no communications between actions in H with an environment are expected or intended.

A useful theorem to break down such expressions is the *Expansion Theorem* (first formulated by Milner, for the case of CCS; see Milner [80]) which holds under the assumption of the *handshaking axiom* $x|y|z = \delta$. This axiom says that all communications are binary. (In fact we have to require associativity of '$\|$' first—see Table 10.)

1.5.1. EXPANSION THEOREM.

$$x_1\|...\|x_k = \Sigma_i\, x_i \mathbin{\llcorner} X_k^{\,i} + \Sigma_{i\neq j}\,(x_i|x_j) \mathbin{\llcorner} X_k^{\,i,j}$$

Here $X_k^{\,i}$ denotes the merge of $x_1,...,x_k$ except x_i, and $X_k^{\,i,j}$ denotes the same merge except x_i,x_j ($k \geq 3$). For instance, for $k = 3$:

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

In order to prove the Expansion Theorem, one first proves by simultaneous induction on term complexity that for all closed ACP-terms (i.e. ACP-terms without free variables) the following *axioms of standard concurrency* hold:

AXIOMS OF STANDARD CONCURRENCY

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

Table 10

The defining power of ACP is strictly greater than that of PA. The following is an example (from Bergstra & Klop [84b]) of a process U, recursively defined in ACP, but not definable in PA: let the alphabet be $\{a,b,c,d,\delta\}$ and let the communication function be given by $c|c = a$, $d|d = b$, and all other communications equal to δ. Let $H = \{c,d\}$. Now we recursively define the process U as in Table 11:

$U = \partial_H(dcY\|Z)$
$X = cXc + d$
$Y = dXY$
$Z = dXcZ$

Table 11

Then, we claim, $U = ba(ba^2)^2(ba^3)^2(ba^4)^2\ldots$. Indeed, using the axioms in ACP and putting

$$U_n = \partial_H (dc^nY \| Z)$$

for $n \geq 1$, a straightforward computation shows that

$$U_n = ba^nba^{n+1}U_{n+1}.$$

By Theorem 1.3.1, U is not definable in PA, since the one infinite trace of U is not eventually periodic.

We will often adopt a special format for the communication function, called *read-write communication*. Let a finite set D of *data* d and a set $\{1,\ldots,p\}$ of *ports* be given. Then the alphabet consists of *read* actions ri(d) and *write* actions wi(d), for $i = 1,\ldots,p$ and $d \in D$. The interpretation is: read datum d at port i, write datum d at port i respectively. Furthermore, the alphabet contains actions ci(d) for $i = 1,\ldots,p$ and $d \in D$, with interpretation: *communicate* d *at* i. These actions will be called *transactions* . The only non-trivial communications (i.e. not resulting in δ) are: $wi(d) \mid ri(d) = ci(d)$. Instead of wi(d) we will also use the notation si(d) (send d along i). Note that read-write communication satisfies the handshaking axiom: all communications are binary.

1.5.2. EXAMPLE. Using the present read-write communication format we can write the recursion equation for a Bag B_{12} (cf. Section 1.3) which reads data $d \in D$ at port 1 and writes them at port 2 as follows:

$$B_{12} = \Sigma_{d \in D}\, r1(d)(w2(d) \parallel B_{12}).$$

In order to illustrate the defining power of ACP, we will now give an infinite specification of the process behaviour of a queue with input port 1 and output port 2. Here D is a finite set of data (finite since otherwise the sums in the specification below would be infinite, and we do not consider infinite expressions), D^* is the set of finite sequences σ of elements from D; the empty sequence is λ. The sequence $\sigma^*\sigma'$ is the concatenation of sequences σ,σ'.

QUEUE

$Q = Q_\lambda$	$= \Sigma_{d \in D}\, r1(d).Q_d$
Q_{σ^*d}	$= s2(d).Q_\sigma + \Sigma_{e \in D}\, r1(e).Q_{e^*\sigma^*d}$ (for all $d \in D$ and $\sigma \in D^*$)

Table 12

Note that this infinite specification uses only the signature of BPA. We have the following remarkable fact:

1.5.2. THEOREM. *Using read-write communication, the process Queue cannot be specified in ACP by finitely many recursion equations.*

For the lengthy proof see Bergstra & Tiuryn [87]. It should be mentioned that the process Queue can be finitely specified in ACP if the read-write restriction is dropped and n-ary communications are allowed; in the next section it is shown how this can be done. In the sequel we will present some other finite specifications of Queue using features to be introduced later.

1.6. Renaming. A useful 'add-on' feature is formed by the renaming operators ρ_f, where f: A $\rightarrow$ A is a function keeping δ fixed. A renaming ρ_f replaces each action 'a' in a process by f(a). In fact, the encapsulation operators ∂_H are renaming operators; f maps H $\subseteq$ A to δ and fixes A - H pointwise. The following axioms, where 'id' is the identity function, are obvious:

RENAMING

$\rho_f(a)$	$= f(a)$
$\rho_f(x+y)$	$= \rho_f(x) + \rho_f(y)$
$\rho_f(xy)$	$= \rho_f(x)\cdot\rho_f(y)$
$\rho_{id}(x)$	$= x$
$(\rho_f \circ \rho_g)(x)$	$= \rho_{f\circ g}(x)$

Table 13

Again the defining power is enhanced by adding this feature. While Queue as in the previous section could not yet be finitely specified, it can now.

The actions are the r1(d), s2(d) as before; there are moreover 'auxiliary' actions r3(d), s3(d), c3(d) for each datum d. Communication is given by r3(d) | s3(d) = c3(d) and there are no other non-trivial communications. If we let $\rho_{c3\rightarrow s2}$ be the renaming c3(d) $\rightarrow$ s2(d) and $\rho_{s2\rightarrow s3}$: s2(d) $\rightarrow$ s3(d), then for H = {s3(d), r3(d) | d$\in$D} the following two guarded recursion equations give an elegant finite specification of Queue:

QUEUE, FINITE SPECIFICATION

$$Q = \Sigma_{d\in D}\, r1(d)\cdot(r_{c3\rightarrow s2} \circ \partial_H)(r_{s2\rightarrow s3}(Q) \parallel s2(d)\cdot Z)$$
$$Z = \Sigma_{d\in D}\, r3(d)\cdot Z$$

Table 14

(This specification was inspired by a similar specification in Hoare [84]. The present formulation is from Baeten & Bergstra [88].) The explanation that this is really Queue is as follows. We intend

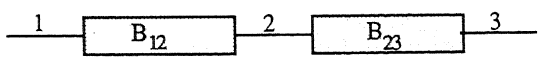
that Q processes data d in a queue-like manner, by performing 'input' actions r1(d) and 'output' actions s2(d). So $\rho_{s2\to s3}(Q)$ processes data in queue-like manner by performing input actions r1(d), output actions s3(d). First consider the parallel system $Q' = \partial_H(\rho_{s2\to s3}(Q) \parallel Z)$: since Z universally accepts s3(d) and transforms these into c3(d), this is just the queue with input r1(d), output c3(d). Now the process $Q^* = \partial_H(\rho_{s2\to s3}(Q) \parallel s2(d).Z)$ appearing in the recursion equation, is just like Q' but with the obligation to perform output action s2(d) *before all output actions* c3(d); this obligation is enforced since s2(d) must be passed before $\rho_{s2\to s3}(Q)$ and Z can communicate and thereby create the output actions c3(d). So $\rho_{c3\to s2}(Q^*) = Q_d$, the queue loaded with d, in the earlier notation used for the infinite specification of Queue (Table 10). But then $Q = \Sigma_{d\in D}\, r1(d).Q_d$ and this is exactly what we want.

In fact, the renamings used in this specification can be removed in favour of a more complicated communication format, as follows. Replace in the specification above $\rho_{s2\to s3}(Q)$ by $\partial_{s2}(Q \parallel V)$ where $V = \Sigma_d\, s2^*(d)\cdot V$ and $S2 = \{s2(d), s2^*(d) \mid d\in D\}$ with communications $s2(d)|s2^*(d) = s3(d)$ for all d. To remove the other renaming operator, put $P = \partial_H(\partial_{S2}(Q \parallel V) \parallel s2(d)\cdot Z)$, and replace $\rho_{c3\to s2}(P)$ by $\partial_{C3}(P \parallel W)$ where $W = \Sigma_d\, c3^*(d)\cdot W$ and $c3(d)|c3^*(d) = s2(d)$ for all d. However, though the renamings are removed in this way, the communication is no longer of the read-write format, or even in the hand shaking format, since we have ternary nontrivial communications $s2(d) = c3(d)|c3^*(d) = r3(d)|s3(d)|c3^*(d)$. As we already stated in the last theorem, this is unavoidable.

1.7. Abstraction.

A fundamental issue in the design and specification of hierarchical (or modularized) systems of communicating processes is *abstraction*. Without having an abstraction mechanism enabling us to abstract from the inner workings of modules to be composed to larger systems, specification of all but very small systems would be virtually impossible. We will now extend the axiom system ACP, obtained thus far, with such an abstraction mechanism.

Consider two Bags B_{12}, B_{23} (cf. Example 1.5.1) with action alphabets $\{r1(d), s2(d) \mid d\in D\}$ and $\{r2(d), s3(d) \mid d\in D\}$, respectively. That is, B_{12} is a bag-like channel reading data d at port 1, sending them to port 2; B_{23} reads data at 2 and sends them to 3. (That the channels are bags means that, unlike the case of a queue, the order of incoming data is lost in the transmission.) Suppose the bags are connected at port 2; so we adopt communications $s2(d) \mid r2(d) = c2(d)$ where c2(d) is the transaction of d at 2.



transparent Bag $\mathbb{B}_{13}$

Figure 1.4

The composite system $\mathbb{B}_{13} = \partial_H(B_{12} \parallel B_{23})$ where $H = \{s2(d), r2(d) \mid d \in D\}$, should, intuitively, be again a Bag between ports 1,3. However, from some (rather involved) calculations we learn that

$$\mathbb{B}_{13} = \Sigma_{d \in D}\, r1(d)\cdot((c2(d)\cdot s3(d)) \parallel \mathbb{B}_{13}).$$

So $\mathbb{B}_{13}$ is a 'transparent' Bag: the passage of d through 2 is visible as the transaction event c2(d). (Note that this terminology conflicts with the usual one in the area of computer networks, where a network is called transparent if the internal structure is *not* visible.)

How can we *abstract* from such internal events, if we are only interested in the external behaviour at 1,3? The first step to obtain such an abstraction is to remove the distinctive identity of the actions to be abstracted, that is, to rename them all into one designated action which we call, after Milner, τ: the *silent* action. This renaming is realised by the *abstraction operator* τ_I, parameterized by a set of actions $I \subseteq A$ and subject to the following axioms:

ABSTRACTION

$\tau_I(\tau)$	$= \tau$
$\tau_I(a)$	$= a$ if $a \notin I$
$\tau_I(a)$	$= \tau$ if $a \in I$
$\tau_I(x+y)$	$= \tau_I(x) + \tau_I(y)$
$\tau_I(xy)$	$= \tau_I(x)\cdot\tau_I(y)$

Table 15

The second step is to attempt to devise axioms for the silent step τ by means of which τ can be removed from expressions, as e.g. in the equation $a\tau b = ab$. However, it is not possible to remove *all* τ's in an expression if one is interested in a faithful description of deadlock behaviour of processes (at least in bisimulation semantics, the framework adopted in this paper). For, consider the process (expression) $a + \tau\delta$; this process can deadlock, namely if it chooses to perform the silent action. Now, if one would propose naively the equations $\tau x = x\tau = x$, then $a + \tau\delta = a + \delta = a$, and the latter process has no deadlock possibility. It turns out that one of the proposed equations, $x\tau = x$, can be safely adopted, but the other one is wrong. Fortunately, R. Milner has devised some simple axioms which give a complete description of the properties of the silent step (complete with respect to a certain semantical notion of process equivalence called $r\tau\delta$-bisimulation, which does respect deadlock behaviour; this notion is discussed below), as follows.

SILENT STEP

$x\tau = x$
$\tau x = \tau x + x$
$a(\tau x + y) = a(\tau x + y) + ax$

Table 16

To return to our example of the 'transparent' Bag $\mathbb{B}_{13}$, after abstraction of the set of transactions I = {c2(d) | d∈ D} the result is indeed an 'ordinary' Bag:

$$\begin{aligned}
&\tau_I(\mathbb{B}_{13}) = \\
&\tau_I(\Sigma_{d\in D}\, r1(d)(c2(d).s3(d) \parallel \mathbb{B}_{13})) = \qquad (*)\\
&\Sigma_{d\in D}\, r1(d)(\tau\cdot s3(d) \parallel \tau_I(\mathbb{B}_{13})) = \\
&\Sigma_{d\in D}\, (r1(d)\cdot \tau\cdot s3(d)) \between \tau_I(\mathbb{B}_{13}) = \\
&\Sigma_{d\in D}\, (r1(d).s3(d)) \between \tau_I(\mathbb{B}_{13}) = \\
&\Sigma_{d\in D}\, r1(d)(s3(d) \parallel \tau_I(\mathbb{B}_{13}))
\end{aligned}$$

from which it follows that $\tau_I(\mathbb{B}_{13}) = B_{13}$ (**), the Bag defined by

$$B_{13} = \Sigma_{d\in D}\, r1(d)(s3(d) \parallel B_{13}).$$

Here we were able to eliminate all silent actions, but this will not always be the case. For instance, 'chaining' two Stacks (see Figure 1.2) instead of Bags (Figure 1.3) yields a process with 'essential' τ-steps. Likewise for a Bag followed by a Stack. (Here 'essential' means: non-removable in bisimulation semantics.) In fact, the computation above is not as straightforward as was suggested: to justify the equations marked with (*) and (**) we need additional proof principles. As to (**), this equation is justified by the *Recursive Specification Principle* (RSP) stating that a *guarded system of recursion equations in which no abstraction operator τ_I appears, has a unique solution.*

1.8. Proof rules for recursive specifications. We have now presented a survey of ACP_τ; we refer to Bergstra & Klop [85] for an analysis of this proof system as well as a proof that (when the hand shaking axiom is adopted) the Expansion theorem carries over from ACP to ACP_τ unchanged. Note that ACP_τ (displayed in full in Section 1.11) is entirely equational. Without further proof rules it is not possible to deal (in an algebraical way) with infinite processes, obtained by recursive specifications, such as Bag; in the derivation above we tacitly used such proof rules and these will be made explicit below.

(i) RDP, the Recursive Definition Principle:
Every guarded and abstraction free recursive specification has a solution.
(ii) RSP, the Recursive Specification Principle:
Every guarded and abstraction free recursive specification has at most one solution.
(iii) AIP, the Approximation Induction Principle:
A process is determined by its finite projections.

In a more formal notation, AIP can be rendered as the infinitary rule

$$\frac{\forall n \quad \pi_n(x) = \pi_n(y)}{x = y}$$

As to (i), the restriction to guarded specifications is not very important (for an informal definition of 'guarded' see Section 1.1); in the process algebras that we have encountered and that satisfy RDP, also the same principle without the guardedness condition is true. More delicate is the situation in principle (ii): first, τ-*steps may not act as guards* : e.g. the recursion equation $X = \tau X + a$ has infinitely many solutions, namely $\tau(a + q)$ is a solution for arbitrary q; and second, the *recursion equations must not contain occurrences of abstraction operators* τ_I. That is, they are 'abstraction-free' (but there may be occurrences of τ in the equations). The latter restriction is in view of the fact that, surprisingly, the recursion equation $X = a \cdot \tau_{\{a\}}(X)$ possesses infinitely many solutions, even though it looks very guarded. (The solutions are: $a \cdot q$ where q satisfies $\tau_{\{a\}}(q) = q$.) That the presence of abstraction operators in recursive specifications causes trouble, was already noticed in Hoare [85].

As to (iii), we still have to define projections π_n in the presence of the τ-action. The extra clauses are:

PROJECTION, CONTINUED

$\pi_n(\tau)$	$= \tau$
$\pi_n(\tau x)$	$= \tau \cdot \pi_n(x)$

Table 17

So, τ-steps do not add to the depth; this is enforced by the τ-laws in Table 16 (since, e.g., $a\tau b = ab$ and $\tau a = \tau a + a$). Remarkably, there are infinitely many different terms t_n (that is, different in the term model of ACP_τ), built from τ and a single atom 'a', such that t_n has depth 1, i.e. $t = \pi_1(t)$. The t_n are inductively defined as follows:

$t_0 = a$, $t_1 = \tau a$, $t_2 = \tau$, $t_3 = \tau(a + \tau)$, $t_4 = a + \tau a$,
$t_{4k+i} = \tau \cdot t_{4k+i-1}$ for $i = 1,3$ and $k \geq 0$,
$t_{4k+i} = t_{4k+i-3} + t_{4k+i-5}$ for $i = 0,2$ and $k \geq 0$.

In fact, these are *all* terms (modulo provable equality in ACP_τ) with the properties as just stated. Furthermore, with respect to the "summand ordering" $\leq$ defined by $x \leq x + y$, the set of these term takes the form of the partial order in Figure 1.5, which has the same form (but for one point) as the Rieger-Nishimura lattice in intuitionistic propositional logic.

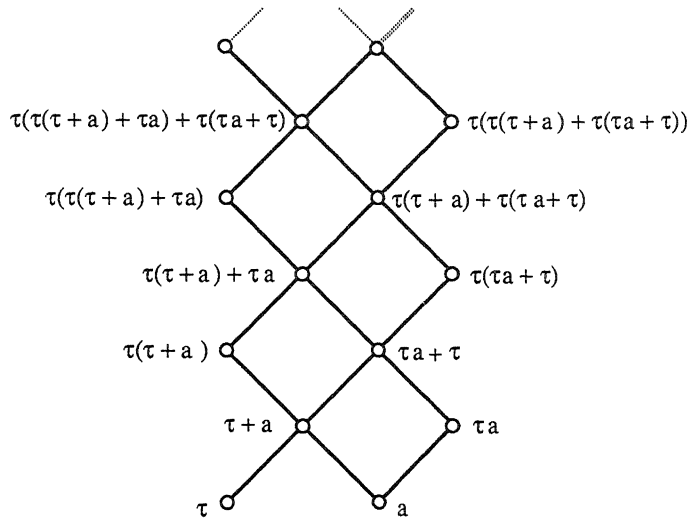


Figure 1.5

The unrestricted form of AIP as in (iii) will turn out to be too strong in some circumstances; it does not hold in one of the main models of ACP_τ, namely the graph model which is introduced in Section 1.13. Therefore we also introduce the following weaker form.

(iv) AIP^- (Weak Approximation Induction Principle):

Every process which has an abstraction-free guarded specification is determined by its finite projections .

Roughly, a process which can be specified without abstraction operators is one in which there are no infinite τ-traces (and which is definable). E.g. the process X_0 defined by the infinite specification $\{X_0 = bX_1, X_{n+1} = bX_{n+2} + a^n\}$, where a^n is a·a· ... ·a (n times), contains an infinite trace of b-actions; after abstraction with respect to b, the resulting process, $Y = \tau_{\{b\}}(X_0)$, has an infinite trace of τ-steps; and (at least in the main model of ACP_τ of Section 1.13) this Y is not definable without abstraction operators.

Even the Weak Approximation Induction Principle is rather strong. In fact a short argument shows the following:

1.8.1. THEOREM. $AIP^- \Rightarrow RSP$.

As a rule, we will be very careful in admitting abstraction operators in recursive specifications. Yet there are processes which can be elegantly specified by using abstraction inside recursion. The following curious specification of Queue is obtained in this manner. We want to specify Q_{12}, the queue from port 1 to 2, using an auxiliary port 3 and concatenating

auxiliary queues Q_{13}, Q_{32}; then we abstract from the internal transaction at port 3. Write, in an ad hoc notation, $Q_{12} = Q_{13}{*}Q_{32}$. Now Q_{13} can be similarly split up: $Q_{13} = Q_{12}{*}Q_{32}$. This gives rise to six similar equations: $Q_{ab} = Q_{ac}{*}Q_{cb}$ where $\{a,b,c\} = \{1,2,3\}$. (See Figure 1.6.)

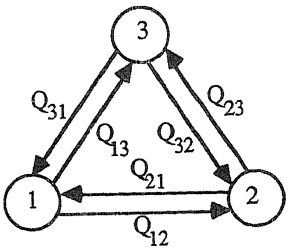


Figure 1.6

These six queues, which are merely renamings of each other, can now be specified in terms of each other as in the following table. One can prove that these recursion equations, though not abstraction-free, indeed have a unique solution.

QUEUE, FINITE SPECIFICATION WITH ABSTRACTION

$$
\begin{aligned}
Q_{12} &= \Sigma_{d\in D}\, r1(d)\cdot\tau_3\circ\partial_3(Q_{13} \parallel s2(d)\cdot Q_{32})\\
Q_{21} &= \Sigma_{d\in D}\, r2(d)\cdot\tau_3\circ\partial_3(Q_{23} \parallel s1(d)\cdot Q_{31})\\
Q_{23} &= \Sigma_{d\in D}\, r2(d)\cdot\tau_1\circ\partial_1(Q_{21} \parallel s3(d)\cdot Q_{13})\\
Q_{32} &= \Sigma_{d\in D}\, r3(d)\cdot\tau_1\circ\partial_1(Q_{31} \parallel s2(d)\cdot Q_{12})\\
Q_{31} &= \Sigma_{d\in D}\, r3(d)\cdot\tau_2\circ\partial_2(Q_{32} \parallel s1(d)\cdot Q_{21})\\
Q_{13} &= \Sigma_{d\in D}\, r1(d)\cdot\tau_2\circ\partial_2(Q_{12} \parallel s3(d)\cdot Q_{23})
\end{aligned}
$$

Table 18

Here the usual read-write notation is used: ri(d) means read d at i, si(d): send d at i, communications are ri(d)|si(d) = ci(d); further $\tau_i = \tau_{\{ci(d)|d\in D\}}$ and $\partial_i = \partial_{\{ri(d),si(d)\,|\,d\in D\}}$. This example shows that even with the restriction to read-write communication, ACP_τ is stronger than ACP.

1.9. Alphabet calculus. In computations with infinite processes one often needs information about the *alphabet* $\alpha(x)$ of a process x. E.g. if x is the process uniquely defined by the recursion equation $X = aX$, we have $\alpha(x) = \{a\}$. An example of the use of this alphabet information is given by the implication $\alpha(x)\cap H = \emptyset \Rightarrow \partial_H(x) = x$. For finite closed process expressions this fact can be proved with induction to the structure, but for infinite processes we have to require such a

property axiomatically. In fact, this example will be one of the 'conditional axioms' below (conditional, in contrast with the purely equational axioms we have introduced thus far). First we have to define the alphabet:

ALPHABET

$\alpha(\delta)$	$= \emptyset$
$\alpha(\tau)$	$= \emptyset$
$\alpha(a)$	$= \{a\}$
$\alpha(\tau x)$	$= \alpha(x)$
$\alpha(ax)$	$= \{a\} \cup \alpha(x)$
$\alpha(x+y)$	$= \alpha(x) \cup \alpha(y)$
$\alpha(x)$	$= \bigcup_{n \geq 1} \alpha(\pi_n(x))$
$\alpha(\partial_H(x))$	$= \alpha(x) - H$
$\alpha(\tau_I(x))$	$= \alpha(x) - I$

Table 19

To appreciate the non-triviality of the concept $\alpha(x)$, let us mention that a finite specification can be given of a process for which the alphabet is uncomputable (see Bergstra & Klop [84b] for an example).

Now the following conditional axioms will be adopted:

CONDITIONAL AXIOMS

$\alpha(x) \mid (\alpha(y) \cap H) \subseteq H$	$\Rightarrow$	$\partial_H(x \| y) = \partial_H(x \| \partial_H(y))$
$\alpha(x) \mid (\alpha(y) \cap I) = \emptyset$	$\Rightarrow$	$\tau_I(x \| y) = \tau_I(x \| \tau_I(y))$
$H = H_1 \cup H_2$	$\Rightarrow$	$\partial_H(x) = (\partial_{H1} \circ \partial_{H2})(x)$
$I = I_1 \cup I_2$	$\Rightarrow$	$\tau_I(x) = (\tau_{I1} \circ \tau_{I2})(x)$
$\alpha(x) \cap H = \emptyset$	$\Rightarrow$	$\partial_H(x) = x$
$\alpha(x) \cap I = \emptyset$	$\Rightarrow$	$\tau_I(x) = x$

Table 20

Using these axioms, one can derive for instance the following fact: if communication is of the read-write format and I is disjoint from the set of transactions (communication results) as well as disjoint from the set of communication actions, then the abstraction τ_I distributes over merges $x \| y$.

1.10. Koomen's Fair Abstraction Rule. Suppose the following statistical experiment is performed: somebody flips a coin, repeatedly, until head comes up. This process is described by the recursion equation $X = flip \cdot (tail \cdot X + head)$. Suppose further that the experiment takes place in a closed room, and all information to be obtained about the process in the room is that we can hear the experimenter shout joyfully: 'Head!'. That is, we observe the process $\tau_I(X)$ where $I = \{flip,$

tail}. Now, if the coin is 'fair', it is to be expected that sooner or later the action 'head' will be perceived. Hence, intuitively, $\tau_I(X) = \tau \cdot head$. (This vivid example is from Vaandrager [86].)

Koomen's Fair Abstraction Rule (KFAR) is an algebraic rule enabling us to arrive at such a conclusion formally. (For an extensive analysis of this rule see Bateen, Bergstra & Klop [87].) The simplest form is

$$\frac{x = ix + y \quad (i \in I)}{\tau_I(x) = \tau \cdot \tau_I(y)} \quad KFAR_1$$

So, $KFAR_1$ expresses the fact that the 'τ-loop' (originating from the i-loop) in $\tau_I(x)$ will not be taken infinitely often. In case this 'τ-loop' is of length 2, the same conclusion is expressed in the rule

$$\frac{x_1 = i_1x_2 + y_1,\ x_2 = i_2x_1 + y_2 \quad (i_1, i_2 \in I)}{\tau_I(x_1) = \tau \cdot \tau_I(y_1 + y_2)} \quad KFAR_2$$

and it is not hard to guess what the general formulation ($KFAR_n$, $n \geq 1$) will be. In fact, as observed in Vaandrager [86], $KFAR_n$ can already be derived from $KFAR_1$ (at least in the framework of $ACP_\tau^\#$, to be discussed below).

KFAR is of great help in protocol verifications. An example is given in Section 1.12, where KFAR is used to abstract from a cycle of internal steps which is due to a defective communication channel; the underlying fairness assumption is that this channel is not defective forever, but will function properly after an undetermined period of time. (Just as in the coin flipping experiment the wrong option, tail, is not chosen infinitely often.)

An interesting peculiarity of the present framework is the following. Call the process τ^ω (= $\tau \cdot \tau \cdot \tau \cdot ....$) *livelock*. Formally, this is the process $\tau_{\{i\}}(x)$ where x is uniquely defined by the recursion equation $X = i \cdot X$. Noting that $x = i \cdot x = i \cdot x + \delta$ and applying $KFAR_1$ we obtain $\tau^\omega = \tau_{\{i\}}(x) = \tau\delta$. In words: *livelock = deadlock*. There are other semantical frameworks for processes, also in the scope of process algebra but not in the scope of this paper, where this equality does not hold (see Bergstra, Klop & Olderog [86, 87]).

1.11. $ACP_\tau^\#$, a framework for process specification and verification.

We have now arrived at a framework which will be called $ACP_\tau^\#$, and which contains all the axioms and proof rules introduced so far. In Table 21 the list of all components of $ACP_\tau^\#$ is given; Table 22 contains the equational system ACP_τ. Note that for *specification* purposes one only needs ACP_τ or ACP_τ^+; for *verification* one will need $ACP_\tau^\#$ (an extensive example is given in Section 1.12). Also, it is important to notice that this framework resides entirely on the level of syntax and formal specifications and verification using that syntax—even though some proof rules are infinitary. No semantics for $ACP_\tau^\#$ has been provided yet; this will be done in Section 1.13. The

idea is that 'users' can stay in the realm of this formal system and execute algebraical manipulations, without the need for an excursion into the semantics. That this can be done is demonstrated by the verification of a simple protocol in the next section; at that point the semantics of $ACP_\tau^\#$ (in the form of some model) has, on purpose, not yet been provided. This does not mean that the semantics is unimportant; it does mean that the user need only be concerned with formula manipulation. The underlying semantics is of great interest for the theory, if only to guarantee the consistency of the formal system; but applications should not be burdened with it, in our intention.

$ACP_\tau^\#$

Basic Process Algebra	A1-5
Deadlock	A6,7
Communication function	C1-3
Merge with communication	CM1-9
Encapsulation	D1-4
Silent step	T1-3
Silent step: auxiliary axioms	TM1,2;TC1-4
Abstraction	DT; TI1-5
Renaming	RN
Projection	PR1-4
Hand shaking	HA
Standard concurrency	SC
Expansion theorem	ET
Alphabet calculus	CA
Recursive Definition Principle	RDP
Recursive Specification Principle	RSP
Weak Approximation Induction Principle	AIP^-
Koomen's Fair Abstraction Rule	KFAR

Table 21

The system up to the first double bar is ACP; up to the second double bar we have ACP_τ, and up to the third double bar, ACP_τ^+.

So $ACP_\tau^\#$ is a medium for formal process specifications and verifications; let us note that we also admit infinite specifications. As the system is meant to have practical applications, we will only encounter *computable* specifications. A finite specification (of which an expression is a particular case) is trivially computable; an infinite specification $\{E_n \mid n \geq 0\}$, where E_n is the recursion equation $X_n = T(X_1,...,X_{f(n)})$, is computable if after some coding, in which E_n is coded as a natural number e_n, the sequence $\{e_n \mid n \geq 0\}$ is computable. Here an important question arises: *is every computable specification provably equal to a finite specification* ? At present we are unable to answer this question; but we can state that the answer is affirmative *relative to certain models* of $ACP_\tau^\#$. Before we elaborate this, a verification of a simple protocol is demonstrated.

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		
$a \mid b = b \mid a$	C1		
$(a \mid b) \mid c = a \mid (b \mid c)$	C2		
$\delta \mid a = \delta$	C3		
$x \Vert y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x + x \mid y$	CM1		
$a \mathbin{\lfloor\!\lfloor} x = ax$	CM2	$\tau \mathbin{\lfloor\!\lfloor} x = \tau x$	TM1
$ax \mathbin{\lfloor\!\lfloor} y = a(x \Vert y)$	CM3	$\tau x \mathbin{\lfloor\!\lfloor} y = \tau(x \Vert y)$	TM2
$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$	CM4	$\tau \mid x = \delta$	TC1
$ax \mid b = (a \mid b)x$	CM5	$x \mid \tau = \delta$	TC2
$a \mid bx = (a \mid b)x$	CM6	$\tau x \mid y = x \mid y$	TC3
$ax \mid by = (a \mid b)(x \Vert y)$	CM7	$x \mid \tau y = x \mid y$	TC4
$(x + y) \mid z = x \mid z + y \mid z$	CM8		
$x \mid (y + z) = x \mid y + x \mid z$	CM9	$\partial_H(\tau) = \tau$	DT
		$\tau_I(\tau) = \tau$	TI1
$\partial_H(a) = a$ if $a \notin H$	D1	$\tau_I(a) = a$ if $a \notin I$	TI2
$\partial_H(a) = \delta$ if $a \in H$	D2	$\tau_I(a) = \tau$ if $a \in I$	TI3
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D3	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	TI4
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$	D4	$\tau_I(xy) = \tau_I(x) \cdot \tau_I(y)$	TI5

Table 22

1.12. An algebraic verification of the Alternating Bit Protocol.

In this section we will demonstrate a verification of a simple communication protocol, the Alternating Bit Protocol, in the framework of $ACP_\tau^\#$. (In fact, not all of $ACP_\tau^\#$ is needed.) This verification is from Bergstra & Klop [86a]; the present streamlined treatment was kindly made available to us by F.W. Vaandrager (CWI Amsterdam).

Let D be a finite set of data. Elements of D are to be transmitted by the ABP from port 1 to port 2. The ABP can be visualized as follows:

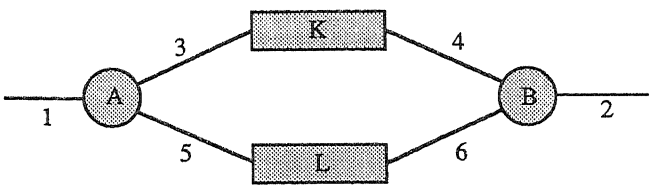


Figure 1.7

There are four components:
A: Reads a Message (RM) at 1. Thereafter it Sends a Frame (SF), consisting of the message and a control bit, into channel K until a correct Acknowledgement has been Received (RA) via channel L. The equations for A are as follows. We will always use the notations: datum $d \in D$, bit $b \in \{0,1\}$, frame $f \in D \times \{0,1\}$ (so a frame f is of the form db).

$$
\begin{aligned}
A &= RM^0 \\
RM^b &= \Sigma_d\, r1(d)\cdot SF^{db} \\
SF^{db} &= s3(db)\cdot RA^{db} \\
RA^{db} &= (r5(1-b) + r5(e))\cdot SF^{db} + r5(b)\cdot RM^{1-b}
\end{aligned}
$$

K: data transmission channel K communicates elements of $D \times \{0,1\}$, and may communicate these correctly or communicate an error value 'e'. K is supposed to be fair in the sense that it will not produce an infinite consecutive sequence of error outputs.

$$
\begin{aligned}
K &= \Sigma_f\, r3(f)\cdot K^f \\
K^f &= (\tau\cdot s4(e) + \tau\cdot s4(f))\cdot K
\end{aligned}
$$

The τ's in the second equation express that the choice whether or not a frame f is to be communicated correctly, cannot be influenced by one of the other components.
B: Receives a Frame (RF) via channel K. If the control bit of the frame is OK, then the Message is Sent (SM) at 2. B Sends back Acknowledgement (SA) via L.

$$
\begin{aligned}
B &= RF^0 \\
RF^b &= (\Sigma_d\, r4(d(1-b)) + r4(e))\cdot SA^{1-b} + \Sigma_d\, r4(db)\cdot SM^{db} \\
SA^b &= s6(b)\cdot RF^{1-b} \\
SM^{db} &= s2(d)\cdot SA^b
\end{aligned}
$$

L: the task of acknowledgement transmission channel L is to communicate boolean values from B to A. The channel L may yield error outputs but is also supposed to be fair.

$$L = \Sigma_b\, r6(b)\cdot L^b$$
$$L^b = (\tau\cdot s5(e) + \tau\cdot s5(b))\cdot L$$

Define $\mathbb{D} = D \cup (D \times \{0,1\}) \cup \{0,1\} \cup \{e\}$. $\mathbb{D}$ is the set of 'generalized' data (i.e. plain data, frames, bits, error) that occur as parameter of atomic actions. We use the notation: $g \in \mathbb{D}$. For $t \in \{1,2,...,6\}$ there are send, read, and communication actions:

$$A = \{st(g), rt(g), ct(g) \mid g \in \mathbb{D}, t \in \{1,2,...,6\}\}.$$

We define communication by $st(g) \mid rt(g) = ct(g)$ for $g \in \mathbb{D}$, $t \in \{1,2,...,6\}$ and all other communications give δ. Define the following two subsets of A:

$$H = \{st(g), rt(g) \mid t \in \{3,4,5,6\}, g \in \mathbb{D}\}$$
$$I = \{ct(g) \mid t \in \{3,4,5,6\}, g \in \mathbb{D}\}.$$

Now the ABP is described by $ABP = \tau_I \circ \partial_H(A \parallel K \parallel B \parallel L)$. The fact that this is a correct protocol is asserted by

1.12.1. THEOREM. $ACP_\tau^{\#} \vdash ABP = \Sigma_d\, r1(d)\cdot s2(d)\cdot ABP$.

(Actually, we need only the part of $ACP_\tau^{\#}$ consisting of ACP_τ+SC+RDP+RSP+CA+KFAR—see Tables 21, 22.)

PROOF. Let $I' = \{ct(g) \mid t \in \{3,4,5\}, f \in \mathbb{D}\}$. We will use [x] as a notation for $\tau_{I'} \circ \partial_H(x)$. Consider the following system of recursion equations:

(0) $X = X_1^0$
(1) $X_1^b = \Sigma_d\, r1(d)\cdot X_2^{db}$
(2) $X_2^{db} = \tau\cdot X_3^{db} + \tau\cdot X_4^{db}$
(3) $X_3^{db} = c6(1\text{-}b)\cdot X_2^{db}$
(4) $X_4^{db} = s2(d)\cdot X_5^{db}$
(5) $X_5^{db} = c6(b)\cdot X_6^{db}$
(6) $X_6^{db} = \tau\cdot X_5^{db} + \tau\cdot X_1^{1-b}$

We claim that $ACP_\tau^{\#} \vdash X = [A \parallel K \parallel B \parallel L]$. We prove this by showing that $[A \parallel K \parallel B \parallel L]$

satisfies the same recursion equations (0)-(6) as X does. In the computations below, the bold face part denotes the part of the expression currently being 'rewritten'.

(0) $[A \parallel K \parallel B \parallel L] = [RM^0 \parallel K \parallel RF^0 \parallel L]$

(1) $[\mathbf{RM^b} \parallel K \parallel RF^b \parallel L] =$
$\Sigma_d\, r1(d)\cdot[\mathbf{SF^{db}} \parallel \mathbf{K} \parallel RF^b \parallel L] =$
$\Sigma_d\, r1(d)\cdot\tau\cdot[RA^{db} \parallel K^{db} \parallel RF^b \parallel L] =$
$\Sigma_{d\in D}\, r1(d)\cdot[RA^{db} \parallel K^{db} \parallel RF^b \parallel L]$

(2) $[RA^{db} \parallel \mathbf{K^{db}} \parallel RF^b \parallel L] =$
$\tau\cdot[RA^{db} \parallel \mathbf{s4(e)\cdot K} \parallel \mathbf{RF^b} \parallel L] + \tau\cdot[RA^{db} \parallel \mathbf{s4(db)\cdot K} \parallel \mathbf{RF^b} \parallel L] =$
$\tau\cdot[RA^{db} \parallel K \parallel SA^{1-b} \parallel L] + \tau\cdot[RA^{db} \parallel K \parallel SM^{db} \parallel L]$

(3) $[RA^{db} \parallel K \parallel \mathbf{SA^{1-b}} \parallel \mathbf{L}] =$
$c6(1-b)\cdot[RA^{db} \parallel K \parallel RF^b \parallel \mathbf{L^{1-b}}] =$
$c6(1-b)\cdot(\tau\cdot[\mathbf{RA^{db}} \parallel K \parallel RF^b \parallel \mathbf{s5(e)\cdot L}] + \tau\cdot[\mathbf{RA^{db}} \parallel K \parallel RF^b \parallel \mathbf{s5(1-b)\cdot L}]) =$
$c6(1-b)\cdot\tau\cdot[\mathbf{SF^{db}} \parallel \mathbf{K} \parallel RF^b \parallel L] =$
$c6(1-b)\cdot\tau\cdot\tau\cdot[RA^{db} \parallel K^{db} \parallel RF^b \parallel L] =$
$c6(1-b)\cdot[RA^{db} \parallel K^{db} \parallel RF^b \parallel L].$

(4) $[RA^{db} \parallel K \parallel \mathbf{SM^{db}} \parallel L] =$
$s2(d)\cdot[RA^{db} \parallel K \parallel SA^b \parallel L].$

(5) $[RA^{db} \parallel K \parallel \mathbf{SA^b} \parallel \mathbf{L}] =$
$c6(b)\cdot[RA^{db} \parallel K \parallel RF^{1-b} \parallel L^b].$

(6) $[RA^{db} \parallel K \parallel RF^{1-b} \parallel \mathbf{L^b}] =$
$\tau\cdot[\mathbf{RA^{db}} \parallel K \parallel RF^{1-b} \parallel \mathbf{s5(e)\cdot L}] + \tau\cdot[\mathbf{RA^{db}} \parallel K \parallel RF^{1-b} \parallel \mathbf{s5(b)\cdot L}] =$
$\tau\cdot[SF^{db} \parallel K \parallel RF^{1-b} \parallel L] + \tau\cdot[RM^{1-b} \parallel K \parallel RF^{1-b} \parallel L].$

(7) $[\mathbf{SF^{db}} \parallel \mathbf{K} \parallel RF^{1-b} \parallel L] =$
$\tau\cdot[RA^{db} \parallel \mathbf{K^{db}} \parallel RF^{1-b} \parallel L] =$
$\tau\cdot(\tau\cdot[RA^{db} \parallel \mathbf{s4(e)\cdot K} \parallel \mathbf{RF^{1-b}} \parallel L] + \tau\cdot[RA^{db} \parallel \mathbf{s4(db)\cdot K} \parallel \mathbf{RF^{1-b}} \parallel L]) =$
$\tau\cdot[RA^{db} \parallel K \parallel SA^b \parallel L].$

Now substitute (7) in (6) and apply RSP + RDP. Using the conditional axioms (see Table 20, Section 1.9) we have $ABP = \tau_I(X) = \tau_I(X_1{}^0)$. Further, an application of $KFAR_2$ gives

$\tau_I(X_2{}^{db}) = \tau \cdot \tau_I(X_4{}^{db})$ and $\tau_I(X_5{}^{db}) = \tau \cdot \tau_I(X_1{}^{1\text{-}b})$.

Hence,

$$\tau_I(X_1{}^b) = \sum_d r1(d)\cdot\tau_I(X_2{}^{db}) = \sum_d r1(d)\cdot \tau_I(X_4{}^{db}) =$$
$$\sum_d r1(d)\cdot s2(d)\cdot \tau_I(X_5{}^{db}) = \sum_d r1(d)\cdot s2(d)\cdot\tau_I(X_1{}^{1\text{-}b})$$

and thus

$$\tau_I(X_1{}^0) = \sum_d r1(d)\cdot s2(d)\cdot\sum_{d'} r1(d')\cdot s2(d')\cdot \tau_I(X_1{}^0)$$
$$\tau_I(X_1{}^1) = \sum_d r1(d)\cdot s2(d)\cdot\sum_{d'} r1(d')\cdot s2(d')\cdot \tau_I(X_1{}^1).$$

Applying RDP + RSP gives $\tau_I(X_1{}^0) = \tau_I(X_1{}^1)$ and therefore

$$\tau_I(X_1{}^0) = \sum_d r1(d)\cdot s2(d)\cdot \tau_I(X_1{}^0),$$

which finishes the proof of the theorem. □

More complicated communication protocols have been verified in $ACP_\tau{}^\#$ by Vaandrager [86]: a Positive Acknowledgement with Retransmission protocol and a One Bit Sliding Window protocol. There the notion of *redundancy in a context* is used as a tool which facilitates the verifications. A related method, using a modular approach, is employed in Koymans & Mulder [86], where a version of the Alternating Bit Protocol called the Concurrent Alternating Bit Protocol is verified in $ACP_\tau{}^\#$. (In fact, also in the verifications in Vaandrager [86] and Koymans & Mulder [86] one only needs the part of $ACP_\tau{}^\#$ mentioned after Theorem 1.12.1.) Another verification of the Concurrent Alternating Bit Protocol is given in Van Glabbeek & Vaandrager [88].

1.13. Bisimulation semantics for $ACP_\tau{}^\#$: the model of countably branching graphs.

We will now give a short description of what we consider to be the 'main' model of $ACP_\tau{}^\#$. The basic building material consists of the domain $\mathcal{G}$ of *countably branching, labeled, rooted , connected, directed multigraphs*. (In the notation of Chapter 3, $\mathcal{G}$ will be $\mathcal{G}_{\alpha,\,\aleph 1}$, where α is the alphabet cardinality.) Such a graph, also called a *process graph*, consists of a possibly infinite set of nodes s with one distinguished node s_0, the root. The edges, also called transitions or steps, between the nodes are labeled with an element from the action alphabet; also δ and τ may be edge labels. We use the notation $s \rightarrow_a t$ for an a-transition from node s to node t; likewise $s \rightarrow_\tau t$ is a τ-transition and $s \rightarrow_\delta t$ is a δ-step. That the graph is connected means that every node must be accessible by finitely many steps from the root node. Examples of process graphs where already given in Figures 1-3. Regarding δ-steps in process graphs, we will suppose that all process graphs

are δ-normalised; the precise definition follows in Definition 1.13.3.

Corresponding to the operations $+, \cdot, \|, \lfloor\!\lfloor, |, \partial_H, \tau_I, \pi_n, \alpha$ in $ACP_\tau^\#$ we define operations in the domain $\mathcal{G}$ of process graphs. Precise definitions can be found in Baeten, Bergstra & Klop [87]; we will sketch some of them here. The sum g + h of two process graphs g, h is obtained by glueing together the roots of g and h; there is one caveat: if a root is cyclic (i.e. lying on a cycle of transitions leading back to the root), then the initial part of the graph has to be 'unwound' first so as to make the root acyclic. (In Chapter 2 we will be more precise about 'root-unwinding': see Definition 2.1.2 there.) The product g·h is obtained by appending copies of h to each terminal node of g; alternatively, one may first identify all terminal nodes of g and then append one copy of h to the unique terminal node if it exists. The merge $g \| h$ is obtained as a cartesian product of both graphs, with 'diagonal' edges for communications. (See Figure 1.8 for the merge of ab and cd, with communications b|c = g and a|d = f.) Definitions of the auxiliary operators $\lfloor\!\lfloor$, $|$ are somewhat more complicated and not discussed here. The encapsulation and abstraction operators are simply renamings, that replace the edge labels in H and I, respectively, by δ and τ, respectively. Definitions of the projection operators π_n and α should be clear from the axioms by which they are specified. As to the projection operators, it should be emphasized that τ-steps are transparent: they do not increase the depth.

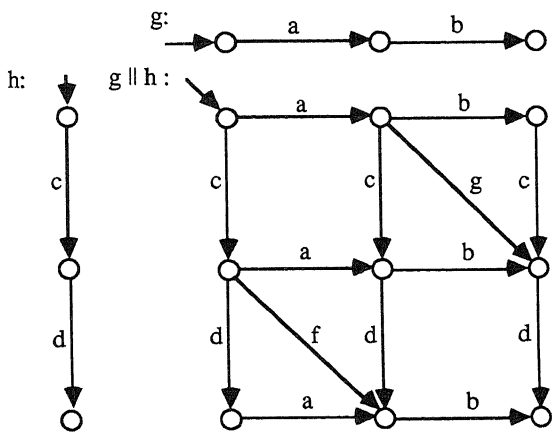


Figure 1.8

The domain $\mathcal{G}$ of process graphs equipped with the operations just introduced, is not yet a model of ACP_τ: for instance the axiom x + x = x does not hold. In order to obtain a model, we define an equivalence on the process graphs which is moreover a congruence with respect to the operations. This equivalence is called *bisimulation congruence* or *bisimilarity*. (The original notion is due to Park [81]; it was anticipated by Milner's observational equivalence, see Milner [80].)

1.13.1. DEFINITION. Let $g \in \mathfrak{G}$.

(i) Steps $s \rightarrow_u t$ and $s \rightarrow_v t'$ (where $u, v \in A \rightarrow \{\tau, \delta\}$; s, t, t' are nodes of g) are *brothers*. A step $t \rightarrow_v t'$ is a *son* of the step $s \rightarrow_u t$.

(ii) g is said to be δ-*normalised* if δ-steps have no brothers and no sons.

(iii) End points of δ-steps are *virtual* nodes; all other nodes in g are *proper*.

(iv) A node is a *deadlock node* if all outgoing traces have only edges with labels τ, δ and end all in δ. (See Figure 1.9.)

(v) Nodes from which only infinite τ-traces start, are *livelock* nodes.

(vi) A *deadlock-livelock node* is a node from which all outgoing traces have as labels only τ, δ and such that there is no succesfully terminating trace.

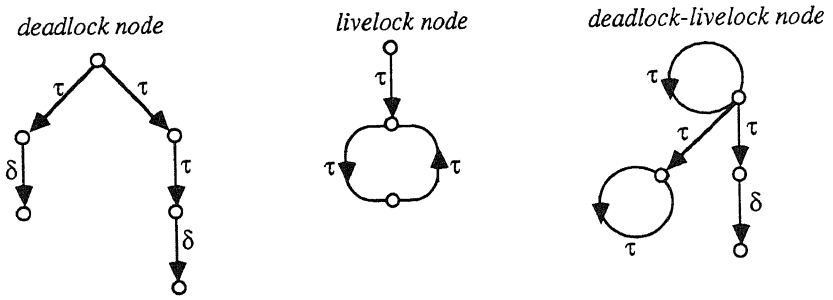


Figure 1.9

1.13.2. DEFINITION. A *path* π in g is a sequence

$$s_0 \rightarrow_{u0} s_1 \rightarrow_{u1} \cdots \rightarrow_{u(n-1)} s_n \quad (n \geq 0)$$

of proper nodes and labelled edges. The node s_0 is *begin*(π), the node s_n is *end*(π). The path π determines a sequence of labels $u_0u_1...u_{n-1}$ ($u_i \in A \cup \{\tau\}$); *val*(π) is this sequence with all τ's skipped. Note that *val*(π) $\in A^*$, the set of words over A, including the empty word λ.

1.13.3. DEFINITION. Let $g, h \in \mathfrak{G}$ be δ-normalised. Let R be a relation between the proper nodes of g,h. We say that R relates path π in g to path π' in h (notation π R π') if

begin(π) R *begin*(π')
end(π) R *end*(π')
val(π) = *val*(π').

(s R t means: s,t are related by R.) If π R π', we also say that π is *transfered* by R to π', and vice versa.

(ii) Relation R has the *transfer property* if:

- whenever π is a path in g and *begin*(π) R t, $t \in$ NODES(h), then π is transfered to some path

π' in h' with $begin(\pi') = t$;

- likewise with the role of g, h interchanged.

(Note that by definition the end points of π, π' are again related.)

1.13.4. DEFINITION. (i) Let g, h $\in \mathcal{G}$ be δ-normalised. Then $g \leftrightarrows_{r\tau\delta}^{R} h$ (g, h are $r\tau\delta$-bisimilar via R) if there is a relation between the proper nodes of g, h such that

(1) the roots of g, h are related,

(2) a root may only be related to a root,

(3) R has the transfer property,

(4) a deadlock-livelock node may only be related to a similar node.

(An equivalent definition is obtained by replacing (4) by:

(4') a node with possibly successful termination may only be related to a similar node. Here a node has 'possibly successful termination' if there is an outgoing trace ending succesfully.)

(ii) $g \leftrightarrows_{r\tau\delta} h$ if there is an R such that $g \leftrightarrows_{r\tau\delta}^{R} h$.

1.13.5. EXAMPLES. (i) Figure 1.10 contains an example of a bisimulation in which only proper atoms (no τ, δ) are involved: the cyclic process graph g is bisimilar to the infinite process graph h obtained by unwinding.

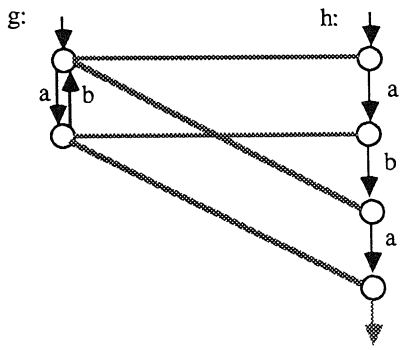


Figure 1.10

(ii) The two graphs in Figure 1.11 are bisimilar via the bisimulation relating nodes on the same level (i.e. joinable by a horizontal line).

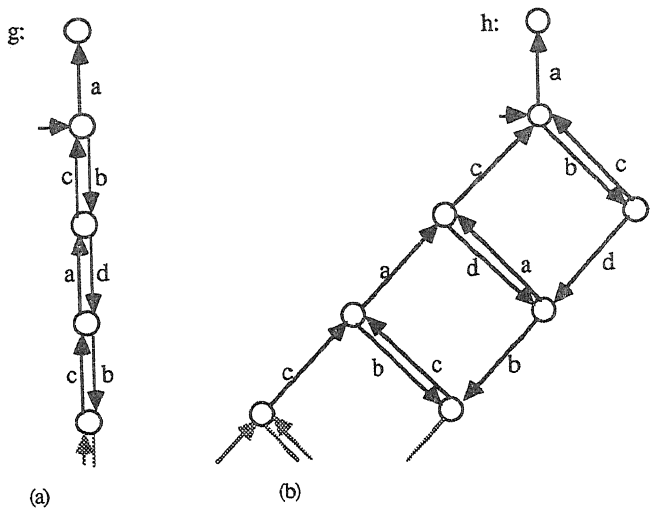
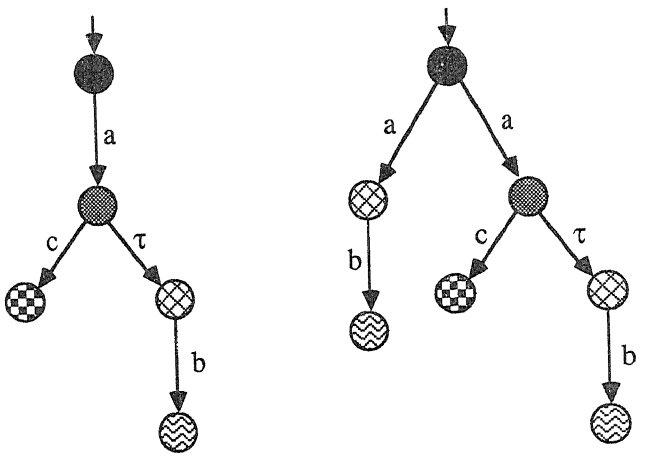


Figure 1. 11

(iii) Figure 1.12 demonstrates a bisimulation between process graphs involving τ-steps: nodes of the same 'color' are related.



Example of rτδ-bisimulation: nodes of the same colour are related

Figure 1.12

We now are in the fortunate position that rτδ-bisimilarity is not only an equivalence relation on the domain $\mathfrak{G}$ of process graphs, but even a congruence with respect to the operators on $\mathfrak{G}$. Thus we can take the quotient $\mathfrak{G}/\leftrightarrows_{\rho\tau\delta}$, notation: $\mathbb{G}$. The following theorem is from Baeten, Bergstra & Klop [87].

1.13.6. THEOREM. $\mathbb{G}$ *is a model of* $ACP_\tau^\#$.

Remarkably, this *graph model* (as we will call it henceforth) does not satisfy AIP, the unrestricted Approximation Induction Principle. A counterexample is given (in a self-explaining notation) by the two process graphs $g = \sum_{n\geq 1} a^n$ and $h = \sum_{n\geq 1} a^n + a^\omega$ (see Figure 1.13(a)); while g and h have the same finite projections $\pi_n(g) = \pi_n(h) = a + a^2 + a^3 + ... + a^n$, they are not ($r\tau\delta$-)bisimilar due to the presence of the infinite trace of a-steps in h. It might be thought that it would be helpful to restrict the domain $\mathbb{G}$ of process graphs to finitely branching graphs, in order to obtain a model which does satisfy AIP, but there are two reasons why this is not the case: (1) the finitely branching graph domain would not be closed under the operations, in particular the communication merge (|); (2) a similar counterexample can be obtained by considering the finitely branching graphs $g' = \tau_{\{t\}}(g'')$ where g'' is the process graph defined by $\{X_n = a^n + tX_{n+1} \mid n \geq 1\}$ and $h' = g' + a^\omega$. (See Figure 1.14(b).)

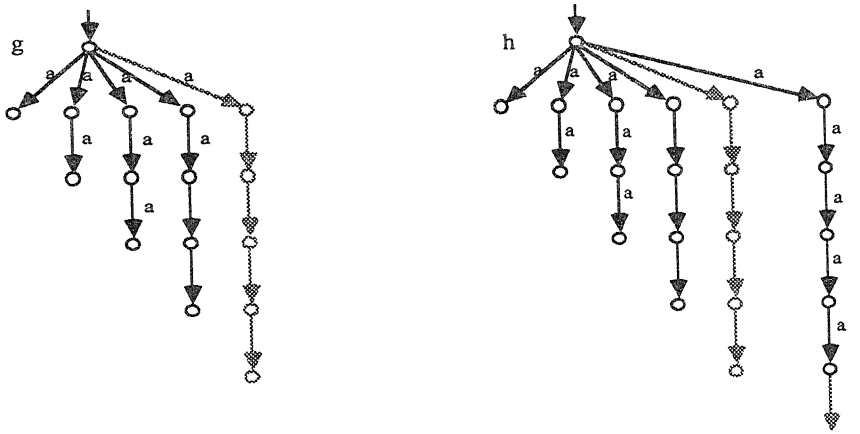


Figure 1.13

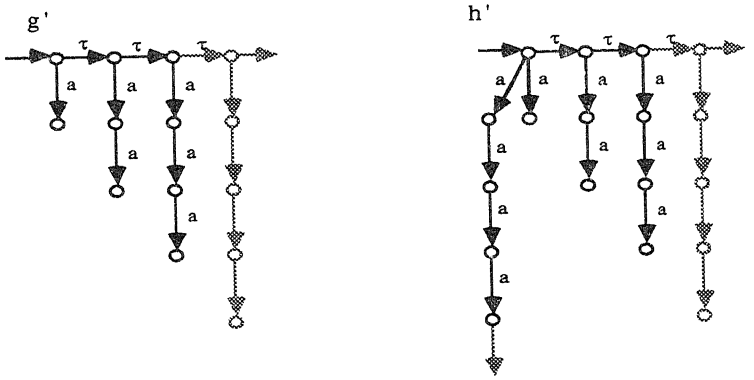


Figure 1.14

1.13.7. REMARK. It is not hard to see that the validity of AIP$^-$ in the model $\mathbb{G}$ is a direct consequence of the following general lemma about bisimulations. Here, for a graph g, $\pi_n(g)$ is the n-th projection of g, i.e. what remains of g after cutting off everything below depth n. Furthermore, $\leftrightarrows$ is the restriction of $\leftrightarrows_{rt\delta}$ to the case where no τ or δ is present. (For an explicit definition, see 2.1.4.1 in Chapter 2.)

1.13.7.1. LEMMA. *Let* g, h *be process graphs containing only proper steps (not* τ *or* δ*). Let* g *be finitely branching (*h *may be infinitely branching). Then:*

$$\forall n\ \pi_n(g) \leftrightarrows \pi_n(h) \ \Rightarrow\ g \leftrightarrows h.$$

PROOF. We may suppose that g, h are process *trees*. Suppose g is finitely branching. Define relations $\equiv_n$ ($n \geq 1$) and $\equiv$ between nodes s of g and t of h as follows: $s \equiv t$ iff $\forall n\ s \equiv_n t$ and $s \equiv_n t$ iff $\pi_n((g)_s \leftrightarrows \pi_n((h)t)$. Here $(g)_s$ is the subtree of g with root s. We will prove that $\equiv$ is a bisimulation.

For the roots s_0, t_0 of g, h respectively we have indeed $s_0 \equiv t_0$; this is just the assumption $\forall n$ $\pi_n(g) \leftrightarrows \pi_n(h)$. Next we show the easy half of the bisimulation requirements: let $s \equiv t$ and $t \rightarrow_a t'$. We have to show that there is an s' such that $s \rightarrow_a s'$ and $s' \equiv t'$. By definition of $\equiv_n$, and because we have $\forall n\ s \equiv_n t$, for every n there must be a step $s \rightarrow_a s_n'$ such that $s_n' \equiv_n t'$. Since s has only finitely many successors (g is finitely branching), there must be an s' among the s_n' such that $s \rightarrow_a s'$ and $s' \equiv_n t'$ for infinitely many n. Since the relations $\equiv_n$ are decreasing ($\equiv_0 \supseteq \equiv_1 \supseteq \equiv_2 \supseteq$ ...) this means that $s' \equiv_n t'$ for *all* n, i.e. $s' \equiv_n t'$.

For the reverse bisimulation requirement, see Figure 1.15. Let $s \equiv t$ and $s \rightarrow_a s'$. To show that there is a t' such that $t \rightarrow_a t'$ and $s' \equiv t'$. We can find a-successors $t_1, t_2, \ldots, t_n, \ldots$ of t such that $s' \equiv_n t_n$. As was just proved, for every t_n there is an a-successor s_n of s with $s_n \equiv t_n$. Since s has only finitely many successors, the sequence $\{s_n\}_n$ is in fact finite. Hence there is an a-successor s* of s such that $s^* \equiv t_n$ for infinitely many n. So, $s' \equiv_n t_n \equiv s^*$ for infinitely many n. So $s' \equiv s^*$, and $s' \equiv t'$ where t' is one of the t_n with $t_n \equiv s^*$. □

The general case, where τ and δ may be present, follows by an entirely similar proof (see also Baeten, Bergstra & Klop [87]). Note however that $\pi_n(g)$ now is obtained by cutting away all steps that are reachable from the root only by passing n or more *proper* steps. (So $\pi_n(g)$ may contain infinite τ-paths.) Thus we have:

1.13.7.2. LEMMA. *Let* g, h *be process graphs. Let* g *have finite projections (i.e. every* $\pi_n(g)$ *is a finite graph.) Then:*

$$\forall n\ \pi_n(g) \leftrightarrows_{rt\delta} \pi_n(h) \ \Rightarrow\ g \leftrightarrows_{rt\delta} h.$$

Note that the assumption of finite projections is fulfilled for a graph which is defined by a system

of guarded recursion equations; hence AIP^- holds in $\mathbb{G}$.

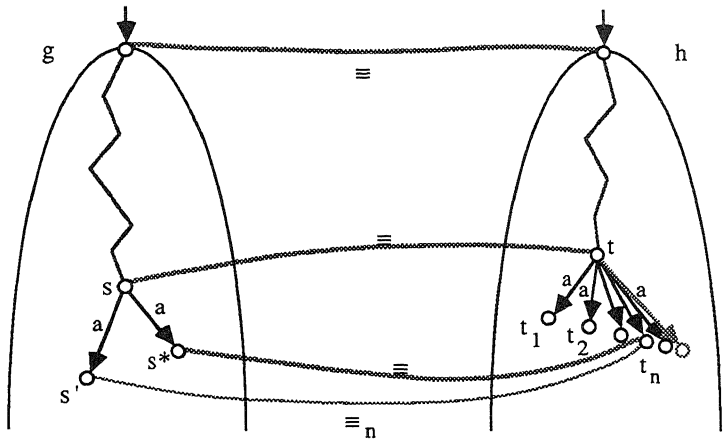


Figure 1.15

1.14. The expressive power of ACP_τ.

ACP_τ is a powerful specification mechanism; in a sense it is a universal specification mechanism: *every finitely branching, computable process can be finitely specified* in ACP_τ. We have to be more precise about the notion of 'computable process'. First, an intuitive explanation: suppose a finitely branching process graph g is actually given; the labels may include τ, and there may be even infinite τ-traces. That g is 'actually' given means that the process graph g must be 'computable': a finite recipe describes the graph, in the form of a coding of the nodes in natural numbers and recursive functions giving in-degree, out-degree, edge-labels. This notion of a computable process graph is rather obvious, and we will not give details of the definition here (these can be found in Baeten, Bergstra & Klop [87]).

Now even if g is an infinite process graph, it can be specified by an infinite computable specification, as follows. First rename all τ-edges in g to t-edges, for a 'fresh' atom t. Call the resulting process graph: g_t. Next assign to each node s of g_t a recursion variable X_s and write down the recursion equation for X_s according to the outgoing edges of node s. Let X_{s0} be the variable corresponding to the root s_0 of g_t. As g is computable, g_t is computable and the resulting 'direct' specification $E = \{X_s = T_s(\mathbf{X}) \mid s \in \text{NODES}(g_t)\}$ is evidently also computable (i.e.: the nodes can be numbered as s_n ($n \geq 0$), and after coding the sequence e_n of codes of equations E_n: $X_{sn} = T_{sn}(\mathbf{X})$ is a computable sequence). Now the specification which uniquely determines g, is simply: $\{Y = \tau_{\{t\}}(X_{s0})\} \cup E$. In fact all specifications below will have the form $\{X = \tau_I(X_0), X_n = T_n(\mathbf{X}) \mid n \geq 0\}$ where the guarded expressions $T_n(\mathbf{X})$ ($= T_n(X_{i1},...,X_{in})$) contain no abstraction operators τ_J. They may contain all other process operators. We will say that such specifications have *restricted abstraction.*

However, we want more than a computable specification with restricted abstraction: to describe process graph g we would like to find a *finite* specification with restricted abstraction for g. Indeed this is possible:

1.14.2. FINITE SPECIFICATION THEOREM. *Let the finitely branching and computable process graph* g *determine* $g^{\sim}$ *in the graph model* $\mathbb{G}$ *of* ACP_τ. *Then there is a finite specification with restricted abstraction* E *in* ACP_τ *such that* $[\![E]\!] = g^{\sim}$.

Here $[\![E]\!]$ is the semantics of E in the graph model. (The proof in Baeten, Bergstra & Klop [87] is by constructing a Turing machine in ACP_τ; the 'tape' is obtained by glueing together two stacks. A stack has a simple finite specification, already in BPA; see the example in Section 1.1.) A stronger fact would be the assertion that every computable specification with restricted abstraction in ACP_τ is provably equivalent (in $ACP_\tau^\#$) to a finite specification with restricted abstraction. At present we do not know whether this is true.

It should be noted that abstraction plays an essential role in this finite specification theorem. If $f: \mathbb{N} \to \{a,b\}$ is a sequence of a,b, let p_f be the process $f(0)\cdot f(1)\cdot f(2)\cdot\ldots$ (more precisely: the unique solution of the infinite specification $\{X_n = f(n)\cdot X_{n+1} \mid n \geq 0\}$). Now:

1.14.3. THEOREM. *There is a computable function* f *such that process* p_f *is not definable by a finite specification (in* ACP_τ*) without abstraction operator.*

A fortiori, p_f is not finitely definable in ACP. The proof in Baeten, Bergstra & Klop [87] is via a simple diagonalization argument.

1.14.4. REMARK. As we have seen, the graph model of $ACP_\tau^\#$ (Section 1.13) does not satisfy the unrestricted Approximation Induction Principle which states that every process is uniquely determined by its finite projections. It is natural to search for a model in which this principle does hold. However, Van Glabbeek [87] proves that such a model does not exist, if one wishes to adhere to the very natural assumption that composition of abstraction operators is commutative, and if one only allows models in which deadlock behaviour is respected (in which, therefore, the equation $\tau = \tau + \tau\delta$ does not hold). We will consider the following consequence of the axioms in Table 20: $\tau_{\{a\}} \circ \tau_{\{b\}} = \tau_{\{b\}} \circ \tau_{\{a\}}$ which we will denote by CA (commutativity of abstraction). Now Van Glabbeek [87] proves:

1.14.5. THEOREM. $ACP_\tau + KFAR_1 + RDP + RSP + CA + AIP \vdash \tau = \tau + \tau\delta$.

So, in every theory extending ACP_τ, the combination of features AIP, KFAR, CA, RDP+RSP is impossible. Among such theories are also theories where the equivalence on processes is much coarser, such as in Hoare's well-known failure model (see Hoare [85]).

2. Complete inference systems for regular processes

In the first chapter we have explained a proof system for specification of processes in bisimulation semantics (namely, in the graph model $\mathbb{G}$), which is 'complete' in the sense that every computable process in $\mathbb{G}$ can be finitely specified. In this chapter we will address the issue of completeness in the usual sense. In doing so, we restrict our attention to the submodel $\mathbb{R}$ of $\mathbb{G}$ consisting of processes having only finitely many 'states', i.e. to '*regular*'processes. Silent steps (τ-steps) are allowed in these processes. We will present a complete inference system for such processes; it is an improved version of the complete inference system in Bergstra & Klop [88].

To obtain the complete proof system we first explore various properties of bisimulations between process graphs with τ-steps ($r\tau$-bisimulation). This leads us to an analysis of $r\tau$-bisimulation which may be illuminating for its own sake. This part of the present chapter is taken from Bergstra & Klop [88]; Sections 2.1 and 2.2 are essentially 1.2-2.4 from Bergsta & Klop [88], with some modifications, and with some examples and proofs omitted.

In this chapter (and the next) we will not consider the process constant δ, deadlock. This is merely a matter of convenience, and in no way essential; all results can easily be adapted for the presence of δ. On the other hand, the presence of τ is very essential; without τ, complete proof systems for regular processes are relatively easy to find. Because δ is omitted from our considerations, we will refer to $r\tau\delta$-bisimulation (defined in Chapter 1, Definition 1.13.4) as $r\tau$-bisimulation.

2.1. Some properties of $r\tau$-bisimulation.

As in Chapter 1, $\mathcal{G}$ is the set of (at most) countably branching process graphs with edge labels from $A \cup \{\delta\} \cup \{\tau\}$. Here $A = \{a,b,c,...\}$ is the set of 'proper' atoms or actions. In the present chapter we will consider the set $\mathcal{R} \subseteq \mathcal{G}$ of finite process graphs in which no δ occurs; so the edge labels are from $A_\tau = A \cup \{\tau\}$. Notation: u,v,... vary over A_τ.

2.1.1. Root-unwinding

It will be convenient to have a canonical transformation of a process graph $g \in \mathcal{G}$ into an 'equivalent' root-acyclic one. (Here 'equivalent' is in a sense which will be explained below, in Proposition 2.1.4.3.)

2.1.2. DEFINITION. The map $\rho: \mathcal{G} \to \mathcal{G}$, *root-unwinding*, is defined as follows. Let $g \in \mathcal{G}$ have root r; then $\rho(g)$ is defined by the following clauses:

(i) $\text{NODES}(\rho(g)) = \text{NODES}(g) \cup \{r'\}$ where r' is a 'fresh' node;

(ii) the root of $\rho(g)$ is r';

(iii) $\text{EDGES}(\rho(g)) = \text{EDGES}(g) \cup \{r' \to_u s \mid r \to_u s \in \text{EDGES}(g)\}$;

(iv) nodes and edges which are inaccessible from the new root r' are discarded.

2.1.3. EXAMPLE. Figure 2.1 gives two examples of root-unwinding.

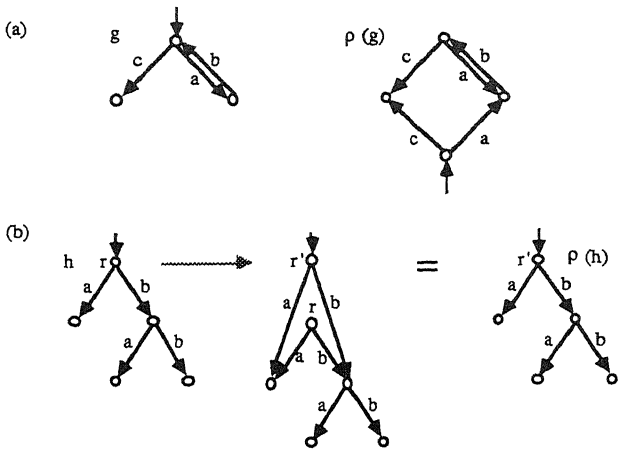


Figure 2.1

Observe that ρ is idempotent: $\rho^2(g) = \rho(g)$. Notation: $\mathfrak{G}^\rho$ is the set of all root-unwound graphs in $\mathfrak{G}$.

2.1.4. Bisimulations

In the previous chapter we have already defined rτδ-bisimulation; the concepts of 'ordinary' bisimulation $\leftrightarroweq$ (on $\mathfrak{G} \times \mathfrak{G}$), 'τ–bisimulation' $\leftrightarroweq_\tau$ (on $\mathfrak{G} \times \mathfrak{G}$) and 'rooted τ–bisimulation' $\leftrightarroweq_{r\tau}$ (on $\mathfrak{G}^\rho \times \mathfrak{G}^\rho$) are just restrictions of that of rτδ-bisimulation, but for the sake of clarity we give the successive definitions again, in a rephrased way which conforms more to the usual definition..

2.1.4.1. Bisimulation: $\leftrightarroweq$

Let g,h ∈ $\mathfrak{G}$. The relation R ⊆ NODES(g) × NODES(h) is a *bisimulation from* g *to* h, notation R: g $\leftrightarroweq$ h, if

(i) Domain(R) = NODES(g) and Range(R) = NODES(h)

(ii) (ROOT(g), ROOT(h)) ∈ R

(iii) if (s,t) ∈ R and $s \rightarrow_u s'$ ∈ EDGES(g) then there is an edge $t \rightarrow_u t'$ ∈ EDGES(h), such that (s',t') ∈ R.

(iv) if (s,t) ∈ R and $t \rightarrow_u t'$ ∈ EDGES(h) then there is an edge $s \rightarrow_u s'$ ∈ EDGES(g), such that (s',t') ∈ R.

Further, we write g $\leftrightarroweq$ h if ∃R R: g$\leftrightarroweq$ h. In this case g,h are called *bisimilar*.

2.1.4.2. EXAMPLE. See Figure 2.2 for a bisimulation between a graph and its root-unwinding; the shaded lines denote the bisimulation.

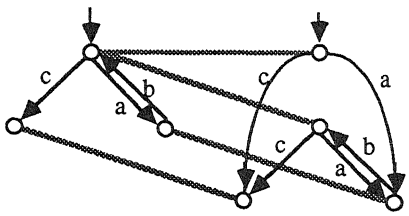


Figure 2.2

Bisimilar process graphs have the same sets of traces. The reverse, however, does not hold. We mention the following facts without proof:

2.1.4.3. PROPOSITION.

(i) *Let* $g \in \mathcal{G}$. *Then* $g \leftrightarrows \rho(g)$.

(ii) *The relation* $\leftrightarrows$ *(bisimilarity) is an equivalence relation on* $\mathcal{G}$.

(iii) *If* $g,h \in \mathcal{G}$, R: $g \leftrightarrows h$ *and for* $s \in$ NODES(g), $t \in$ NODES(h) *we have* $(s,t) \in R$, *then* R': $(g)_s \leftrightarrows (h)_t$, *where* R' *is the restriction of* R *to the nodes of* $(g)_s$ *and* $(h)_t$.

2.1.4.4. τ-Bisimulation: $\leftrightarrows_\tau$

An equivalent definition for ordinary bisimulation can be given as follows. Replace in the definition of 2.1.4.1. clauses (iii), (iv) by:

(iii)' if $(s,t) \in R$ and π: $s \twoheadrightarrow_w s'$ is a path in g (determining the 'word' $u_1u_2...u_k$ $(k \geq 0)$ of labels along the edges in π), then there is a path π': $t \twoheadrightarrow_{w'} t'$ in h such that $(s',t') \in R$ and such that $w \equiv w'$ (w,w' are identical).

(iv) likewise with the role of g, h interchanged.

The definition of $\leftrightarrows_\tau$ now parallels that for $\leftrightarrows$, with as only alteration that $w \equiv w'$ is replaced by $w \equiv_\tau w'$. Here $w \equiv_\tau w'$ (w,w' $\in A_\tau^*$ are equivalent modulo τ) if w,w' are identical after deletion of τ's. E.g. $\tau \equiv_\tau \tau\tau\tau \equiv_\tau \varepsilon$ (the empty word); $ab\tau\tau\tau c\tau \equiv_\tau \tau a\tau b\tau c$. Processes $g,h \in \mathcal{G}$ such that $g \leftrightarrows_\tau h$ are called τ-*bisimilar*.

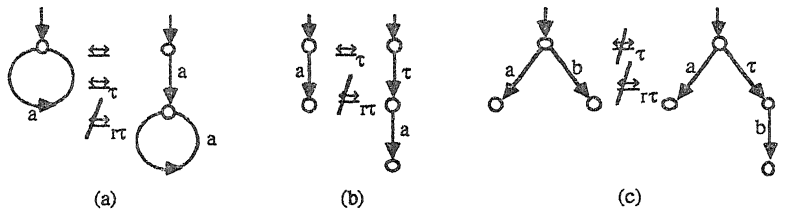
2.1.4.5. Rooted τ-bisimulation: $\leftrightarrows_{r\tau}$

Suppose $g,h \in \mathcal{G}^\rho$ and R: $g \leftrightarrows_\tau h$ in such a way that

$(s,t) \in R \Rightarrow s = \text{ROOT}(g)$ and $t = \text{ROOT}(h)$, or: $s \neq \text{ROOT}(g)$ and $t \neq \text{ROOT}(h)$.

(So a non-root cannot be related in the bisimulation to a root.) Then R is called a *rooted* τ-*bisimulation* between g,h and we write R: $g \leftrightarrows_{r\tau} h$ or $g \leftrightarrows_{r\tau}^R h$. Such g,h are called $r\tau$-*bisimilar* (via R). Note that $g \leftrightarrows h \Rightarrow g \leftrightarrows_\tau h$ and $g \leftrightarrows_{r\tau} h \Rightarrow g \leftrightarrows_\tau h$. As before, $\leftrightarrows_{r\tau}$ and $\leftrightarrows_\tau$ are equivalence relations on $\mathcal{G}^\rho$ and $\mathcal{G}$, respectively. Also $\leftrightarrows_\tau$, $\leftrightarrows_{r\tau}$ are invariant under ρ.

2.1.4.6. EXAMPLES.



(a) (b) (c)

Figure 2.3

Some further obvious facts are:

2.1.4.7. PROPOSITION. (i) *Let* g,h ∈ $\mathbb{G}$ *be* τ*-bisimilar via* R. *Let* (s,t) ∈ R. *Then* $(g)_s$ *and* $(h)_t$ *are* τ*-bisimilar (via the appropriate restriction of* R*). (The nodes* s,t *are called in this case* τ*-bisimilar.)*
(ii) *Let* g,h ∈ $\mathbb{G}^\rho$ *and* $g \triangleq_{r\tau}^{R} h$. *Let* (s,t) ∈ R. *Then* $(g)_s \triangleq_\tau (h)_t$ *(in general not* rτ*-bisimilar).* □

2.1.4.8. PROPOSITION. *Let* g,h ∈ $\mathbb{G}$ *and suppose R:* $g \triangleq h$ *as well as R':* $g \triangleq h$. *Then* R∪R': $g \triangleq h$. *Similar for* $\triangleq_\tau$ *and* $\triangleq_{r\tau}$. □

(Note that the *intersection* of bisimulations R, R' need not be a bisimulation.)

2.1.4.9. DEFINITION. (i) A τ*-cycle* in a process graph g is a cycle

$$\pi: s_0 \to_\tau s_1 \to_\tau \dots \to_\tau s_k \equiv s_0 \quad (k \geq 1).$$

(ii) A τ*–loop* is a τ-cycle of length 1:

$$\pi: s_0 \to_\tau s_0.$$

2.1.4.10. PROPOSITION. *Let* g ∈ $\mathbb{G}$ *contain a* τ*–cycle passing through the nodes* s,t. *Then* s,t *are* τ*-bisimilar (i.e.* $(g)_s \triangleq_\tau (g)_t$*).*

PROOF. (See Figure 2.4, next page.) Note that every point in g accessible form s is accessible from t and vice versa. Hence the node sets of $(g)_s$ and $(g)_t$ coincide. Now let Id be the identity relation on NODES($(g)_s$). Then it is easy to verify that Id ∪ {(s,t)} is a τ-bisimulation from $(g)_s$ to $(g)_t$. □

2.1.4.11. PROPOSITION. (i) *Let* g ∈ $\mathbb{G}$ *contain* τ*-bisimilar nodes* s,t. *Let* g* *be the result of adding a* τ*-edge from* s *to* t. *Then* g *and* g* *are* τ*-bisimilar.*
(ii) *Let* g ∈ $\mathbb{G}^\rho$ *contain non-root nodes* s,t *which are* τ*-bisimilar. Then* $g \triangleq_{r\tau} g^*$.

PROOF. (i) Let Id be the identity relation on NODES(g) (= NODES(g*)). Then Id ∪ {(s,t)} is a τ-bisimulation from g to g* as required. (ii) Similar. □

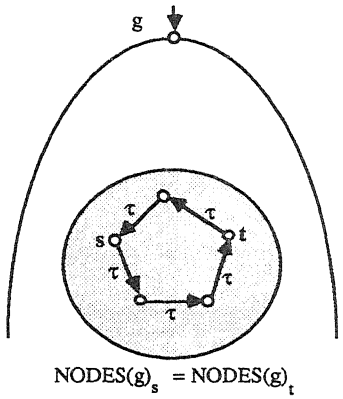


Figure 2.4

This proposition says that adding τ-steps between τ-bisimilar nodes in a graph g does not change the "τ-bisimilarity character" of g (and for the same reason, of any node q, or better, subgraph $(g)_q$ of g). Here the τ-bisimilarity character of g is the class of all $g' \in \mathcal{G}$ which are τ-bisimilar with g. In particular, the τ-bisimilarity character is not disturbed by appending τ-loops to nodes of g. Vice versa, removing τ-loops also does not change the τ-bisimilarity character.

2.1.4.12. EXAMPLE.

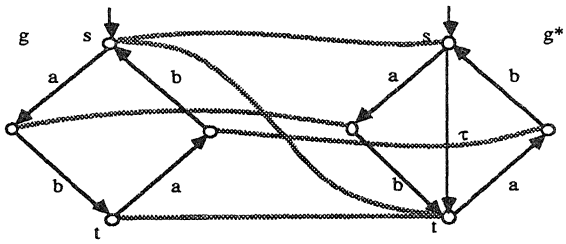


Figure 2.5

Just as all τ-loops can be removed from g without changing τ-bisimilarity (which follows from the previous proposition, by taking s = t), it is possible to remove all τ-cycles from g. We need a definition first:

2.1.4.13. DEFINITION. Let $g \in \mathcal{G}$ contain nodes s,t. Then $g_{id(s,t)}$ is the process graph resulting from the identification of s and t, in the obvious sense.

2.1.4.14. EXAMPLE. Let g be as in Figure 2.5. Then $g_{id(s,t)}$ is:

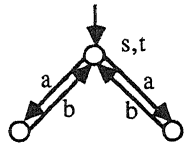


Figure 2.6

2.1.4.15. PROPOSITION. (i) *Let* $g \in \mathcal{G}$ *and suppose* $s,t \in$ NODES(g) *are* τ*-bisimilar. Then* g *and* $g_{id(s,t)}$ *are* τ*-bisimilar.*
(ii) *Let* $g \in \mathcal{G}^\rho$ *and suppose the non-root nodes* $s,t \in$ NODES(g) *are* τ*-bisimilar. Then* $g \leftrightarroweq_{r\tau} g_{id(s,t)}$.

PROOF. Obvious. □

2.1.4.16. COROLLARY. (i) *Every* $g \in \mathcal{G}$ *is* τ*-bisimilar with some* $g' \in \mathcal{G}$ *without* τ*-cycles.*
(ii) *Every* $g \in \mathcal{G}^\rho$ *is* $r\tau$*-bisimilar with some* $g' \in \mathcal{G}^\rho$ *without* τ*-cycles.*
(iii) *Every* $g \in \mathcal{R}$ *is* τ*-bisimilar to some* $g' \in \mathcal{R}$ *without infinite* τ*-paths.*

PROOF. Follows from considering Figure 2.7. □

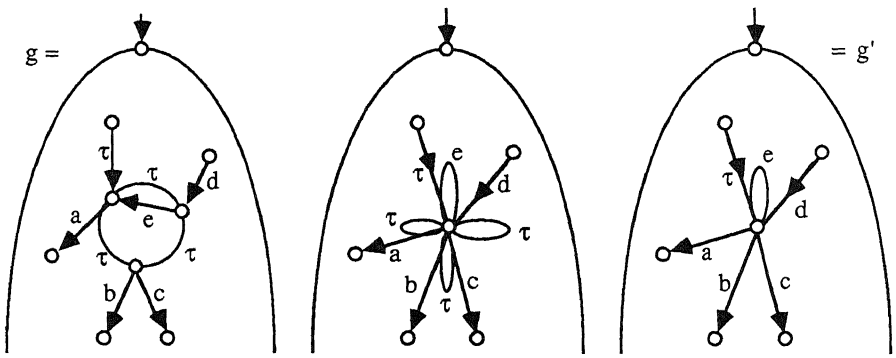


Figure 2.7

We conclude this section with an observation illuminating the difference between $\leftrightarroweq_\tau$ and $\leftrightarroweq_{r\tau}$. The easy proof is left to the reader (or, see Bergstra & Klop [88]).

2.1.4.17. REMARK. Let $g,h \in \mathcal{G}$ and let τg, τh be the result of prefixing a τ-step. Then:
$g \leftrightarroweq_\tau h \Leftrightarrow \tau g \leftrightarroweq_{r\tau} \tau h$.

2.2. An analysis of rτ-bisimulation.

The main result of this section is that an $r\tau$-bisimulation R between $g,h \in \mathcal{R}$ can be analysed into

more simple parts:

$$\begin{array}{ccc} g & \leftrightarrows_{r\tau} & h \\ \downarrow & & \downarrow \\ \Delta(g) & & \Delta(h) \\ \downarrow & & \downarrow \\ E(\Delta(g)) & \leftrightarrows & E(\Delta(h)) \end{array}$$

(Corollary 2.2.4). I.e. $g \leftrightarrows_{r\tau} h$ iff g,h after 'preprocessing' (by means of some simple operations Δ, E: $\mathfrak{G} \to \mathfrak{G}$), are bisimilar in the ordinary sense where τ does not play its special role. This analysis is the basis for the completeness theorem in the sequel where axioms are given describing rτ-bisimulation.

2.2.1. The operation Δ

First we need some terminology: if $g \in \mathfrak{G}$, then an *arc* in g is a part of the form (a) in Figure 2.9 (here $u \in A_\tau$). In case n = m = 0, the arc is a *double edge* as in (b). Other special cases are in Figure 2.9(c), (d): these are called Δ-*arcs*. It is not required that the three nodes displayed in (a)–(d) are indeed pairwise different. The u-step between nodes s,t is called the *primary* edge of the arc.

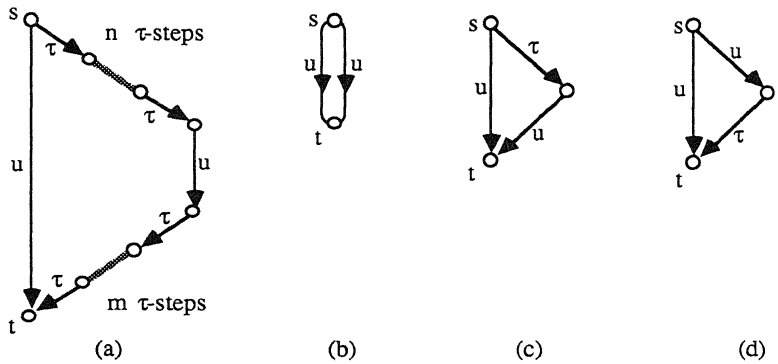


Figure 2.9

Now the operation Δ: $\mathfrak{G} \to \mathfrak{G}$ is defined as follows: whenever $g \in \mathfrak{G}$ contains a path $s_1 \to_\tau s_2 \to_u s_3$ (where s_1,s_2,s_3 need not be pairwise different), an edge $s_1 \to_u s_3$ is added if not yet present. Likewise for every path $s_1 \to_u s_2 \to_\tau s_3$. $\Delta(g)$ is the result of this completion of g with edges as indicated.

Further, we say that $g \in \mathfrak{G}$ is Δ-*saturated* if $\Delta(g) = g$.

2.2.2. EXAMPLE.

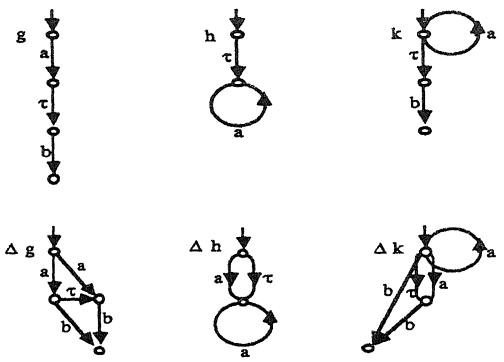


Figure 2.10

2.2.3. PROPOSITION. (i) $\Delta(g) \leftrightarrows_{\tau} g$ *if* $g \in \mathfrak{G}$; (ii) $\Delta(g) \leftrightarrows_{r\tau} g$ *if* $g \in \mathfrak{G}^{\rho}$.

PROOF. The identity relation R gives a (r)τ-bisimulation. □

2.2.4. The operation E

Call a node of $g \in \mathfrak{G}^{\rho}$ *internal* if it is not the root, and an edge of g *internal* if it is between internal nodes. Further, call an internal τ-step $s \rightarrow_{\tau} t$ in $g \in \mathfrak{G}^{\rho}$ an ε-*step* if s,t are τ-bisimilar. Finally, consider the set of internal nodes of $g \in \mathfrak{G}^{\rho}$ and the equivalence relation on this set given by τ-bisimilarity. We will call the equivalence classes: *clusters*. So ε-steps always occur 'inside' a cluster (see Figure 2.11).

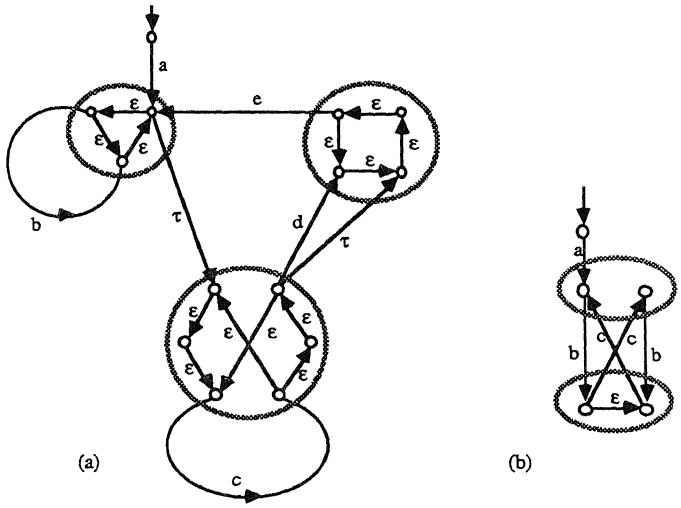


Figure 2.11

2.2.4.1. NOTATION. If s,s' are in the same cluster we write also s ~ s'.

The concept of clusters of nodes makes the structure of a process graph more perspicuous. In particular, Δ-saturated process graphs g have a local structure as indicated in Figure 2.12:

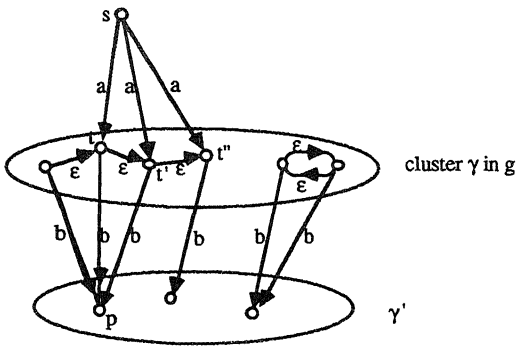


Figure 2.12

Namely, if γ is a cluster in g and $s \to_a t$ is an 'incoming' edge, then the endpoint t is carried in the direction of the ε-steps, thus providing arrows $s \to_a t'$, $s \to_a t''$. Vice versa, if $t' \to_b p$ is an outgoing edge, the starting point t' is carried backwards along ε-paths. This is a simple consequence of Δ-saturation and in fact it does not depend on the particular nature of ε-steps. Moreover, and this does depend on the definition of cluster in terms of $\leftrightarrows_\tau$, if γ has an outgoing edge $\to_b$ to some cluster γ', then from every point in γ there is an edge $\to_b$ to γ'. We will need this last fact so let us prove it:

2.2.4.2. PROPOSITION. *Let* $g \in \mathcal{GP}$ *be* Δ*-saturated. Let* $s \to_u t$ *be an edge of* g *and let* $s' \sim s$. *Then* g *contains an edge* $s' \to_u t'$ *for some* $t' \sim t$.

PROOF. Consider an rτ-bisimulation R of g with itself relating s to s'. (R can be taken to be the union of the identity relation on g and a τ-bisimulation from $(g)_s$ to $(g)_{s'}$.) Now by definition of τ-bisimulation, given the edge $s \to_u t$ and $s \sim s'$ there is a path π: $s' \twoheadrightarrow t'$ with label $\tau^n u \tau^m$ in g for some $n,m \geq 0$ and some t' with $t' \sim t$. By virtue of Δ-saturation, we now have an edge $s' \to_u t'$. □

Now we would like, in order to obtain the 'structure theorem' 2.2.4.7 concerning (r)τ-bisimulation as well as the completeness result in Section 2.3, to omit all ε-steps in a Δ-saturated graph g, resulting in a graph g' which is still rτ-bisimilar to g. Here the need for Δ-saturation comes in, for omitting ε-steps could make a non-Δ-saturated graph g disconnected, as in Example 2.2.2: there the τ-step in g (which clearly is an ε-step) cannot be removed, but it can in Δ(g).

2.2.4.3. DEFINITION. E is the operation from $\mathbb{G}^{\rho}$ to $\mathbb{G}^{\rho}$ which removes in $g \in \mathbb{G}^{\rho}$ all ε-steps (as well as parts of g which become disconnected in that process). If $g = E(g)$, g is called *prenormal*.

The straightforward proofs of the next two propositions are omitted (they can be found in Bergstra & Klop [88]).

2.2.4.4. PROPOSITION. E *preserves* Δ*-saturation.*

2.2.4.5. PROPOSITION. (i) *If* $g \in \mathbb{G}^{\rho}$ *is* Δ*-saturated, then* $g \leftrightarrows_{r\tau} E(g)$.
(ii) *For* $g \in \mathbb{G}^{\rho}$: $E(\Delta(g)) \leftrightarrows_{r\tau} g$.

Now we arrive at a key lemma:

2.2.4.6. LEMMA. *Let* $g,h \in \mathbb{G}^{\rho}$ *be* Δ*-saturated and prenormal. Then:*

$$g \leftrightarrows_{r\tau} h \Rightarrow g \leftrightarrows h.$$

PROOF. (1) Let R be an $r\tau$-bisimulation between g,h. Then there is no τ-step in g which is "contracted" by R in h, as in Figure 2.13 (and likewise with g,h interchanged):

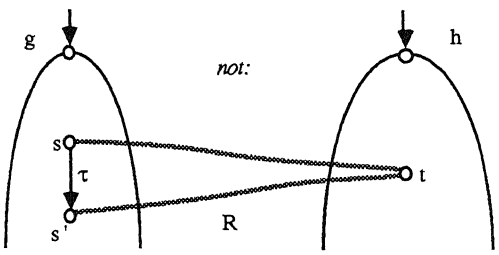


Figure 2.13

Namely, if s = r, the root of g, then this claim follows by definition of $\leftrightarrows_{r\tau}$. Otherwise, $s \rightarrow_{\tau} s'$ is an internal step ($s' \neq r$ since $g \in \mathbb{G}^{\rho}$) and now by Proposition 2.1.4.7(ii):

$$(g)_s \leftrightarrows_{\tau} (h)_t \leftrightarrows_{\tau} (g)_{s'}.$$

That is: $s \rightarrow_{\tau} s'$ is an ε-step. But then g is not prenormal.
(2) Let $s \rightarrow_u s'$ ($u \in A_{\tau}$) be a step in g (see Figure 2.14, next page). By definition of the $r\tau$-bisimulation R, there is given a t such that $(s,t) \in R$, a path $t \twoheadrightarrow t'$ with label $\tau^n u \tau^m$, for some t' such that $(s',t') \in R$. By Δ-saturation of h, there is now a step $t \rightarrow_u t'$. (1) and (2) together imply that the $r\tau$-bisimulation R is in fact an ordinary bisimulation. □

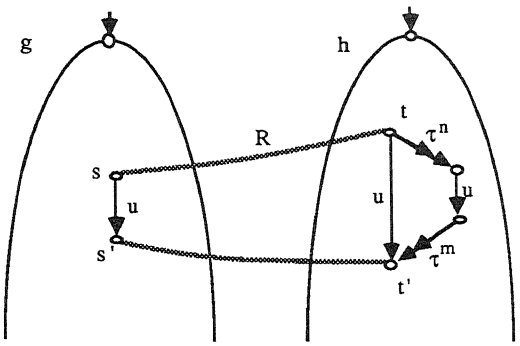


Figure 2.14

2.2.4.7. COROLLARY. *Let* g,h ∈ $\mathcal{G}^\rho$ *and let* g $\leftrightarrows_{r\tau}$ h. *Then* E(Δ(g)) $\leftrightarrows$ E(Δ(h)).

PROOF. By Proposition 2.2.3, Δg $\leftrightarrows_{r\tau}$ Δh. By Proposition 2.2.4.5, E(Δg) $\leftrightarrows_{r\tau}$ E(Δh). By Proposition 2.2.4.4, E(Δg) and E(Δh) are Δ-saturated. Hence by Lemma 2.2.4.6 these two graphs are bisimilar in the ordinary sense. □

2.3. Complete inference systems for rτ-bisimulation.

(This section will be slightly informal and gloss over some details; for these we refer to Bergstra & Klop [88].) A corollary of the preceding section (Corollaries 2.1.4.16 and 2.2.4.7) is that an rτ-bisimulation between two graphs g, h ∈ $\mathcal{R}^\rho$ can be analyzed in the following parts. (See Figure 2.15.)

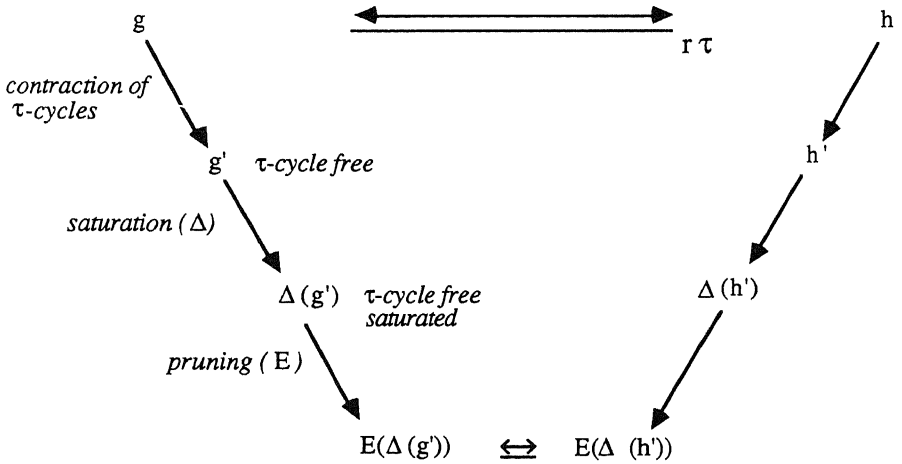


Figure 2.15

So, in order to have a complete proof system for $\leftrightarrows_{r\tau}$, it suffices to have:

I. A complete proof system for $\leftrightarrows$;
II. Proof rules which make '*contraction* of τ-cycles provable';
III. Likewise for *saturation* (the operation Δ);
IV. Likewise for *pruning* (the operation E).

First we have to explain the syntax used for regular processes, and the interpretation of expressions in that syntax into the semantic domain $\mathbb{R} = \mathcal{R}/\leftrightarrows_{rt}$. Precise syntax definitions can be found in Bergstra & Klop [88]; here we will be more informal and give some suggestive examples instead.

Our syntactic expressions, denoting regular processes in $\mathbb{R}$, will be either of the form t where t is a closed BPA-term (see Table 1) or recursive expressions $\langle X_1 \mid E\rangle$ where $E = \{X_i = t_i(X_1,...,X_n) \mid i = 1,...,n\}$. Here $t_i(X)$ $(= t_i(X_1,...,X_n))$ is a BPA-term possibly involving formal recursion variables from $\{X_1,...,X_n\}$. Moreover, the $t_i(X)$ are 'simple' terms, defined as follows.*)

2.3.1. DEFINITION. (i) Every $u \in A_\tau$ is a simple term.
(ii) Let X be a recursion variable and let $u \in A_\tau$. Then uX is a simple term.
(iii) Let t, t' be simple terms. Then t + t' is a simple term.

So, $aX + \tau Y + c$ is a simple term, but $abX + c$, $b(aX + c)$ and $aXY + bYY$ are not.

The *semantics* of an expression $\langle X \mid E\rangle$ in $\mathbb{R}$ is obvious: to $\langle X \mid E\rangle$ there corresponds in an immediate way, suggested by the next example, a process graph in $\mathcal{R}$, call it $g_{\langle X|E\rangle}$; now the semantics $[\![\langle X \mid E\rangle]\!]_{\mathbb{R}}$ is $g_{\langle X|E\rangle} / \leftrightarrows_{rt}$.

2.3.2. EXAMPLE. The semantics of $\tau\cdot\langle X \mid X = \tau Y + aY, Y = \tau X + b\rangle$ is the graph g in Figure 2.19(a), modulo $\leftrightarrows_{rt}$. The semantics of $t\cdot\langle X \mid X = aX + b\rangle$ is $h/\leftrightarrows_{rt}$, h as in Figure 2.16(b). Actually, $g \leftrightarrows_{rt} h$; we will return to this example and show that the two expressions just mentioned are provably equal.

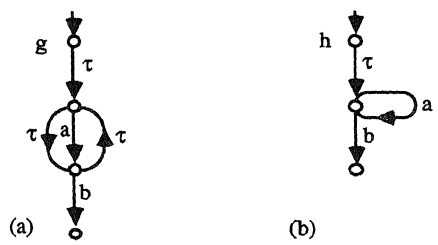


Figure 2.16

(*) (Actually, we have to be slightly more liberal w.r.t. the form of the t_i in $E = \{X_i = t_i(X_1,...,X_n) \mid i = 1,...,n\}$. In fact, we will allow substitutions for variables in the t_i, in order to have equalities like e.g. $\langle X \mid X = t(X,Y), Y = s(X,Y)\rangle = \langle X \mid X = t(X,Y), Y = s(t(X,Y),Y)\rangle$. For a more precise treatment see Bergstra & Klop [88].)

I. Having thus established our syntax and semantics, we turn to the question of finding a complete proof system for the easier case of bisimulation, $\leftrightarrows$. This question was solved in Milner [84b], using the syntax of μ-expressions. Milner's complete proof system 'M' for regular processes is given in Table 23.

M

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

Table 23

2.3.3. EXAMPLE. Consider the μ-expressions $\mu X.\ aX$ and $\mu Y.\ (aY + a\mu X.aX)$, denoting the graphs g, h (modulo $\leftrightarrows$) in Figure 2.17. Since $g \leftrightarrows h$, we must be able to prove equality between the two μ-expressions. Indeed: abbreviate $\mu X.\ aX$ by L, and the other μ-expression by R. Then, in M, one proves: $L = aL = aL + aL$ and $R = aR + aL$. Hence L, R are solutions of the same guarded recursion equation $X = aX + aL$. Therefore $L = R$.

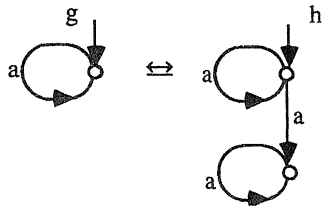


Figure 2.17

In the present framework we have the equivalent proof system BPA_{rec} (equivalent, modulo some inessential details, discussed in Bergstra & Klop [88]) as in Table 24 below. Here $E = \{X_i = T_i(X_1,...,X_n) \mid i = 1,...,n\}$. The rules R1,2 correspond to $\mu 1,2$ in Table 23. In particular, R1 implies the following axiom (which is equivalent to R1):

$$\langle X_1 \mid E\rangle = T_1(\langle X_1 \mid E\rangle, ..., \langle X_n \mid E\rangle)$$

and this axiom corresponds exactly to $\mu 1$.

The axiom $\mu 3$ in M has no counterpart in BPA_{rec}. The axioms A4, A5 come in here since multiplication is general (i.e. not merely 'prefix-multiplication').

Rule R3 states that 'conversions' in the right-hand sides of the equations in $\langle X \mid E\rangle$ are allowed.

BPA_{rec}

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$\dfrac{x_i = \langle X_i \mid E\rangle,\ i=1,\ldots,n}{x_1 = T_1(x_1,\ldots,x_n)}$	R1
$\dfrac{x_i = T_i(x_1,\ldots,x_n),\ i = 1,\ldots,n}{x_1 = \langle X_1 \mid E\rangle}$ $T_i(X_1,\ldots,X_n)$ is guarded	R2
$\dfrac{E = E'}{\langle X_1 \mid E\rangle = \langle X_1 \mid E'\rangle}$	R3

Table 24

II. Next, we discuss the problem of making the contraction of τ-cycles provable. Of course, we start with adopting the τ-laws T1-3 as in Table 16 or 22. Now, for instance, we want to be able to prove

$$\langle X \mid X = \tau X + a\rangle = \langle X \mid X = \tau a\rangle \ (= \tau a)$$

in view of the $r\tau$-bisimilarity of the corresponding process graphs. Note that a proof rule like $x = \tau x + a \Rightarrow x = \tau a$ would not do the job; while it is true that τa is a solution of the equation $X = \tau X + a$ (since $\tau a = \tau(\tau a) + a$, using the τ-laws), it is unfortunately the case that also $\tau(a + q)$ for arbitrary q is a solution:

$$\tau(a + q) = \tau(a + q) + (a + q) = \tau(a + q) + (a + q) + a = \tau(a + q) + a = \tau(\tau(a + q)) + a.$$

The solution is the use of the abstraction operator τ_I (renaming every $i \in I$ into τ, see Table 15 or 22), and the proof rule KFAR (see Chapter 1, Section 1.10), enabling us to conclude from $x = ix + a$ that $\tau_{\{i\}}(x) = \tau a$. This is an instance of the proof rule $KFAR_1$:

$$\frac{x = ix + y}{\tau_{\{i\}}(x) = \tau \cdot \tau_{\{i\}}(y)}$$

which in turn can be derived from $KFAR_2$:

$$\frac{x = iy + z, \qquad y = jx + z'}{\tau_{\{i,j\}}(x) = \tau\cdot\tau_{\{i,j\}}(z + z')}$$

The desired equation can now be proved as follows. Put $x = \langle X \mid X = iX + a\rangle$, so $x = ix + a$. By $KFAR_1$: $\tau_{\{i\}}(x) = \tau\cdot\tau_{\{i\}}(a) = \tau a$. Furthermore, $\tau_{\{i\}}(x) = \tau_{\{i\}}(\langle X \mid X = iX + a\rangle) = \langle X \mid X = \tau X + a\rangle$, which proves the result.

Using $KFAR_2$ we can "contract in a provable way" every τ-cycle in (the graph corresponding to) a system $\langle X \mid E\rangle$. That $KFAR_2$ already suffices, and that one does not need $KFAR_n$ for $n > 2$, is demonstrated in Example 2.3.7 below.

III. Making the operation Δ (saturation) provable is no problem at all: here the τ-laws T1-3 suffice. We will not prove this here (see Bergstra & Klop [88]), but refer to the examples below.

IV. More consideration is required to see that also 'pruning' of internal ε-steps (by means of the operation E) is provable. Suppose g is a saturated, τ-cycle free graph $\in \mathcal{R}^\rho$. Then in order to execute operation E, we can successively remove the ε-steps. In each such removal the node set of g is not affected, since ε-steps are internal and g is saturated; furthermore, the "τ-bisimilarity character" of all nodes in g remains invariant. Hence also the cluster structure of the initial g remains invariant. At the end of the pruning operation, i.e. in E(g), each cluster still is a $\leftrightarrows_\tau$-equivalence class. Moreover, by similar arguments as used in the proof of Corollary 2.2.4.7 one proves:

2.3.4. PROPOSITION. *Let* $h_1, h_2 \in \mathcal{R}$ *be saturated,* τ*-cycle free, and suppose all* ε*-steps in* h_1, h_2 *(i.e.* τ*-steps between* τ*-bisimilar nodes) have been removed (including possible ones to or from the root). Then:*

$$h_1 \leftrightarrows_\tau h_2 \;\Rightarrow\; h_1 \leftrightarrows h_2.$$

Using this proposition we observe that in E(g) with g as above (saturated, τ-cycle free, $\in \mathcal{R}^\rho$) every cluster only contains nodes s, t which are bisimilar in the ordinary sense ($(g)_s \leftrightarrows (g)_t$). Here $(g)_s$, $(g)_t$ are h_1, h_2 from Proposition 2.3.4.

Now our way to make the transformation from g to E(g) provable, is to *start* with E(g) and then add ε-edges to arrive at g. Using the observation just made this is easy, and instead of a proof we just give an example.

2.3.5. EXAMPLE. Let E(g) and g be as in Figure 2.17.

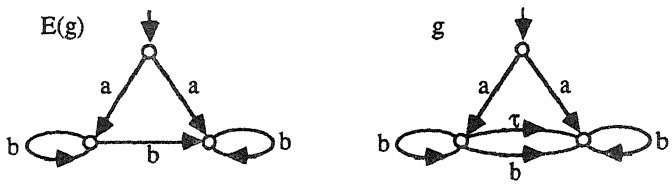


Figure 2.17

The corresponding expressions <X | E>, <X' | E'> (written as systems of equations where the first recursion variable is the 'designated' one) are:

E: $X = aY + aZ$, $Y = bY + bZ$, $Z = bZ$

E': $X' = aY' + aZ'$, $Y' = bY' + \tau Z' + bZ'$, $Z' = bZ'$

We now prove <X | E> = <X' | E'> as follows; here we use some of the proof rules from the proof system below in Table 25.

Abbreviating $\underline{X} = <\ |\ E>$, $\underline{Y} = <Y\ |\ E>$, $\underline{Z} = <Z\ |\ E>$ and similarly for $\underline{X}'$, $\underline{Y}'$, $\underline{Z}'$, we have:

$$\underline{X} = a\underline{Y} + a\underline{Z}$$
$$\underline{Y} = b\underline{Y} + b\underline{Z}$$
$$\underline{Z} = b\underline{Z}.$$

Now consider the expression $\underline{Y}^* = \tau\underline{Y}$. Then $\vdash \underline{Y}^* = \underline{Y} + \tau\underline{Y} = \underline{Y} + \tau\underline{Z} = b\underline{Y} + b\underline{Z} + \tau\underline{Z} = b\tau\underline{Y} + b\underline{Z} + \tau\underline{Z} = b\underline{Y}^* + b\underline{Z} + \tau\underline{Z}$. Here we used that $\vdash \underline{Y} = \underline{Z}$, which follows from the fact that the graphs corresponding to Y, Z are bisimilar as stated in Proposition 2.3.4, and from the fact that the proof system is complete for ordinary bisimulation. Therefore:

$$\underline{X} = a\underline{Y} + a\underline{Z} = a\tau\underline{Y} + a\underline{Z} = a\underline{Y}^* + a\underline{Z}$$
$$\underline{Y}^* = b\underline{Y}^* + b\underline{Z} + \tau\underline{Z}$$
$$\underline{Z} = b\underline{Z}.$$

Hence $(\underline{X}, \underline{Y}^*, \underline{Z})$ satisfies E'. Hence $\vdash \underline{X} = \underline{X}'$. *)

The general case, where g and E(g) differ by more than one ε-step, is only notationally more complicated and left to the reader.

Table 25 presents the complete inference system $BPA_{\tau,rec}$.

*) We use here that E' is a *guarded* system of equations, which enables us to use rule R2 in Table 25. Actually, E' is only 'essentially' guarded; the τ occurring in E' is not a guard (a guard must be a *proper* atom), but substituting bZ' for Z' we arrive at a guarded system. It is not hard to prove that indeed, in general, the system E' corresponding to $\Delta(g')$ as in Figure 2.15 is essentially guarded (i.e. that $\Delta(g')$ is τ-cycle free).

$BPA_{\tau,rec}$

$x+y = y+x$	A1
$(x+y)+z = x+(y+z)$	A2
$x+x = x$	A3
$(x+y)z = xz+yz$	A4
$(xy)z = x(yz)$	A5
$x\tau = x$	T1
$\tau x + x = \tau x$	T2
$a(\tau x + y) = a(\tau x + y) + ax$	T3
$\tau_I(X) = X$	TI0
$\tau_I(\tau) = \tau$	TI1
$\tau_I(a) = \tau$ if $a \in I$	TI2
$\tau_I(a) = a$ if $a \notin I$	TI3
$\tau_I(x+y) = \tau_I(x) + \tau_I(y)$	TI4
$\tau_I(\tau y) = \tau \cdot \tau_I(y)$	TI5'
$\tau_I(ay) = \tau_I(a) \cdot \tau_I(y)$	TI5''
$\tau_I(\langle X_1 \mid E\rangle) = \langle X_1 \mid \tau_I(E)\rangle$	TI6
$\dfrac{x_i = \langle X_i \mid E\rangle,\ i = 1,\ldots,n}{x_1 = T_1(x_1,\ldots,x_n)}$	R1
$\dfrac{x_i = T_i(x_1,\ldots,x_n),\ i = 1,\ldots,n}{x_1 = \langle X_1 \mid E\rangle}$ $T_i(X_1,\ldots,X_n)$ is guarded	R2
$\dfrac{E = E'}{\langle X_1 \mid E\rangle = \langle X_1 \mid E'\rangle}$	R3
$\dfrac{x = iy + z, \quad y = jx + z'}{\tau_{\{i,j\}}(x) = \tau\cdot\tau_{\{i,j\}}(z+z')}$	$KFAR_2$

Table 25

A very elegant alternative complete proof system, employing the formalism of μ-expressions, is given in Milner [88]. It consists of the proof system in Table 23, extended with the τ-laws (T1-3) and the following two axioms, which play the role of $KFAR_2$:

$$\mu X(\tau X + E) = \mu X(\tau E)$$
$$\mu X(\tau(X + E) + F) = \mu X(\tau X + E + F).$$

Here E, F are arbitrary expressions.

We conclude this chapter with some examples showing the use of the proof system $BPA_{\tau,rec}$.

2.3.6. EXAMPLE. We resume the question in Example 2.3.2, to prove $\tau\cdot\langle X \mid X = \tau Y + aY,\ Y = \tau X + b\rangle = \tau\cdot\langle X \mid X = aX + b\rangle$. (See Figure 2.16.) Abbreviate:

$$\underline{X} = \langle X \mid E\rangle = \langle X \mid X = \tau Y + aY,\ Y = \tau X + b\rangle,\ \underline{Y} = \langle Y \mid E\rangle,$$
$$\underline{X}^i = \langle X \mid E^i\rangle = \langle X \mid X = iY + aY,\ Y = iX + b\rangle,\ \underline{Y}^i = \langle Y \mid E^i\rangle.$$

So we have $\underline{X}^i = i\underline{Y}^i + a\underline{Y}^i$ and $\underline{Y}^i = i\underline{X}^i + b$. Hence by $KFAR_2$:

$$\tau_{\{i\}}(\underline{X}^i) = \underline{X} = \tau\cdot\tau_{\{i\}}(a\underline{Y}^i + b)$$

which yields $\underline{X} = \tau\cdot(a\underline{Y} + b)$. Likewise $\underline{Y} = \tau(a\underline{Y} + b)$. Therefore $\underline{X} = \underline{Y}$, and so $\underline{X} = \tau(a\underline{X} + b)$. Thus

$$\underline{X} = \langle U \mid U = \tau(aU + b)\rangle = \langle U \mid U = \tau V,\ V = aU + b\rangle =$$
$$\langle U \mid U = \tau V,\ V = a\tau V + b\rangle = \langle U \mid U = \tau V,\ V = aV + b\rangle.$$

Now abbreviate: $\underline{U} = \langle U \mid U = \tau V,\ V = aV + b\rangle$, $\underline{V} = \langle V \mid U = \tau V,\ V = aV + b\rangle =$
$\langle V \mid V = aV + b\rangle$. So we have proved $\tau\underline{X} = \tau\underline{U} = \tau\tau\underline{V} = \tau\langle V \mid V = aV + b\rangle$, which was our goal.

2.3.7. EXAMPLE. We want to prove that the expressions corresponding to the graphs in Figure 2.18 are equal.

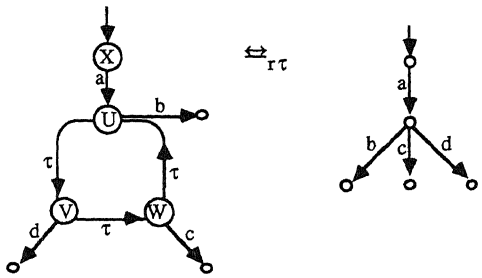


Figure 2.18

So, to prove:

$$\underline{X} = \langle X \mid E\rangle =$$
$$\langle X \mid X = aU,\ U = \tau V + b,\ V = \tau W + d,\ W = \tau U + c\rangle = a(b + c + d).$$

Now

$$\langle X \mid E\rangle = \langle X \mid X = aU,\ U = \tau V + b,\ V = \tau(\tau U + c) + d,\ W = \tau U + c\rangle =$$
$$\langle X \mid X = aU,\ U = \tau V + b,\ V = \tau(\tau U + c) + \tau U + d,\ W = \tau U + c\rangle =$$
$$\langle X \mid X = aU,\ U = \tau V + b,\ V = \tau W + \tau U + d,\ W = \tau U + c\rangle = \langle X \mid F\rangle.$$

The last system corresponds to the graph in Figure 2.19(a).

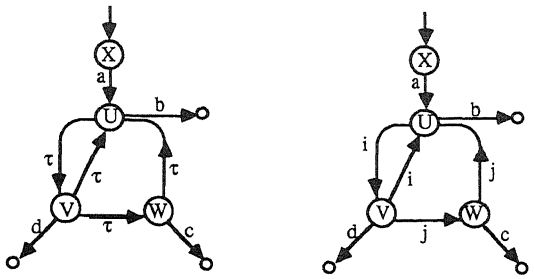


Figure 2.19

We now introduce:

$$X^{i,j} = \langle X \mid F^{i,j} \rangle = \langle X \mid X = aU,\ U = iV + b,\ V = jW + iU + d,\ W = jU + c \rangle.$$

So:

$$\underline{X}^{i,j} = a\underline{U}^{i,j},\ \underline{U}^{i,j} = i\underline{V}^{i,j} + b,\ \underline{V}^{i,j} = j\underline{W}^{i,j} + i\underline{U}^{i,j} + d,\ \underline{W}^{i,j} = j\underline{U}^{i,j} + c.$$

Now we apply $KFAR_2$ on the "i-cycle"; that is, from $\underline{U}^{i,j} = i\underline{V}^{i,j} + b$, $\underline{V}^{i,j} = j\underline{W}^{i,j} + i\underline{U}^{i,j} + d$ it follows that $\tau_{\{i\}}(\underline{U}^{i,j}) = \tau \cdot \tau_{\{i\}}(j\underline{W}^{i,j} + d + b)$ (*). Since

$$\tau_{\{i\}}(\underline{X}^{i,j}) = \underline{X}^{j} = \langle X \mid F^{j} \rangle =$$
$$\langle X \mid X = aU,\ U = \tau V + b,\ V = jW + \tau U + d,\ W = jU + c \rangle,$$

we now have: $\underline{X}^{j} = a\underline{U}^{j}$, $\underline{U}^{j} = \tau(j\underline{W}^{j} + d + b)$ (by (*)), $\underline{W}^{j} = j\underline{U}^{j} + c$. Here $\underline{U}^{j} = \langle U \mid F^{j} \rangle$ and $\underline{W}^{j} = \langle W \mid F^{j} \rangle$. Therefore

$$\underline{X}^{j} = \langle X \mid X = aU,\ U = \tau(jW + d + b),\ W = jU + c \rangle \text{ and}$$
$$\underline{X} = \tau_{\{j\}}(\underline{X}^{j}) = \langle X \mid X = aU,\ U = \tau(\tau W + d + b),\ W = \tau U + c \rangle =$$
$$\langle X \mid X = aU,\ U = \tau V,\ V = \tau W + d + b,\ W = \tau U + c \rangle =$$
$$\langle X \mid X = aV,\ V = \tau W + d + b,\ W = \tau V + c \rangle.$$

Here the last two recursion expressions correspond to the graphs in Figure 2.19(a,b) respectively.

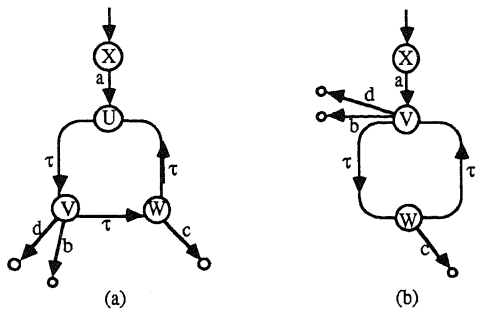


Figure 2.19

The remaining τ-cycle of two steps can now be contracted, as in the previous example, by one more application of $KFAR_2$. The result is: $\underline{X} = a(b + c + d)$.

3. A comparison of process models related to bisimulation semantics

In this chapter we compare the class of graph models as defined in Chapter 1 with a class of 'projective' models as well as with a class of metric models as in the work of De Bakker & Zucker [82a,b]. In doing so, we will restrict ourselves to the simple case of pure interleaving, without communication; that is, we will consider only models of the axiom system PA, in Tables 1 and 5 of Chapter 1. The alphabet involved will be A = {a,b,c,...}; it does not contain τ nor δ. Hence, the notion of bisimulation that is employed is $\leftrightarroweq$, defined in 2.1.4.1. Two parameters will play an important role in this chapter: the cardinality α of alphabet A, and the branching degree β of process graphs.

3.1. DEFINITION. (i) Process graphs without cycles and without 'shared subgraphs' are process *trees*. (In Milner [80] these are called 'synchronisation trees'.) More precisely: a process graph is a process tree if every node has exactly one incoming arrow where the small root arrow also counts as an arrow. A process graph is finite if it contains finitely many edges and nodes.

(ii) If g is a process graph, and $s \in$ NODES(g) is a node of g, then the *branching degree* of s is the number of arrows leaving s. The branching degree of g is the maximum of the branching degrees of the nodes in g.

(iii) $\mathcal{G}_{\alpha,\beta}$ is the set of process graphs 'over' an alphabet of cardinality α and with branching degree $< \beta$. Here $\alpha \geq 1$ and $\beta \geq \aleph_0$. (The bound β on the branching degree must be infinite since otherwise the process graph domains below would not be closed under '+', as defined in 1.13.) On $\mathcal{G}_{\alpha,\beta}$ we define operations $+, \cdot, \|, \lfloor\!\lfloor, (\,)_n$ $(n \geq 1)$; see Section 1.13 with the understanding that merge $\|$ is now simply the cartesian product graph, without 'diagonal' edges representing communications as in Chapter 1. Furthermore, in this chapter we employ the alternative notation $(\,)_n$ instead of $\pi_n(\,)$.This *projection* $(g)_n$ $(n \geq 1)$ is defined for trees g: it is the tree obtained by cutting away all nodes reachable from the root by a path of length $> n$. The corresponding edges are also left away. If g is not a tree, then $(g)_n$ is defined as $(g')_n$ where g' is the tree obtained by unwinding g.

As in Chapter 1, it turns out that bisimilarity $\leftrightarroweq$ is a congruence on $\mathcal{G}_{\alpha,\beta}$ with respect to the operations just defined. Hence we can take the quotient

$$\mathbb{G}_{\alpha,\beta} = \mathcal{G}_{\alpha,\beta}/\leftrightarroweq.$$

The quotient structures are models of PA, i.e. process algebras for PA. Using the usual distance function d, defined by

$$d(x,y) = \begin{cases} 2^{-m} & \text{if } \exists n\ (x)_n \neq (y)_n;\ m = \min\{n \mid (x)_n \neq (y)_n\} \\ 0 & \text{otherwise, i.e. } \forall n\ (x)_n = (y)_n \end{cases}$$

we have that $\mathbb{G}_{\alpha,\beta}$ is a pseudo-metric space but not yet a metric space. (For instance, in $\mathbb{G}_{1,\aleph 1}$ the elements determined by the process graphs $\Sigma_{n\geq 1}\, a^n$ and $\Sigma_{n\geq 1}\, a^n + a^\omega$ in Figure 1.13 are different but have distance 0.) It becomes a metric space after dividing out the congruence induced by the *Approximation Induction Principle* (AIP), discussed also in Chapter 1:

$$\frac{\forall n\ \ (x)_n = (y)_n}{x = y}$$

The result of 'dividing out' AIP is

$$\mathbb{G}^\circ_{\alpha,\beta} = \mathbb{G}_{\alpha,\beta}/\mathrm{AIP}.$$

The $\mathbb{G}^\circ_{\alpha,\beta}$ have been defined as a 'double quotient' by first dividing out $\leftrightarroweq$ and next AIP. The same result can be obtained by defining a suitable equivalence relation at once; this is done in Golson & Rounds [83] where 'weak equivalence' is divided out. In Milner [80], p.42 this notion is called 'observation equivalence'. It is defined as follows:

3.2. DEFINITION. (i) If $s \in$ NODES(g), then $(g)_s$ is the subgraph of g with root s, and nodes: all nodes in g reachable from s, and edges as induced by g.
(Warning: the notation $(g)_s$ should not be confused with $(g)_n$ for the n-th projection of g.)
(ii) On a process graph domain $\mathbb{G}_{\alpha,\beta}$ we define transition relations $\rightarrow_a$ for each atom a: if $s \rightarrow_a t$ is a step (edge) in $g \in \mathbb{G}_{\alpha,\beta}$, then $(g)_s \rightarrow_a (g)_t$.
(Note the difference in notation: open arrows stand for transitions between process graphs, normal arrows denote steps between nodes in one process graph.)

3.3. DEFINITION. On $\mathbb{G}_{\alpha,\beta}$ we define equivalences $\equiv_n$ for each $n \geq 0$:
(i) $g \equiv_0 h$ for all g,h;
(ii) $g \equiv_{n+1} h$ if
  (1) whenever $g \rightarrow_a g'$ there is a transition $h \rightarrow_a h'$ with $g' \equiv_n h'$;
  (2) as (1) with the roles of g,h interchanged.
Furthermore, $g \equiv h$ if $g \equiv_n h$ for all $n \geq 0$.

An alternative, equivalent definition is:

3.4. DEFINITION. Let $g,h \in \mathcal{G}_{\alpha,\beta}$ be process graphs. Then $g \equiv_n h$ if $(g)_n \triangleq (h)_n$ $(n \geq 1)$. Furthermore, $g \equiv h$ if $g \equiv_n h$ for all $n \geq 1$.

The proof that these definitions are indeed equivalent is left to the reader. We also omit the routine proof of the next proposition, where $\cong$ denotes *isometry*.

3.5. PROPOSITION. $\mathbb{G}^\circ_{\alpha,\beta} \cong \mathcal{G}_{\alpha,\beta}/\equiv$. □

3.5.1. REMARK. For *finitely branching* graphs (i.e. $\beta = \aleph_0$) and arbitrary alphabet, we have in fact

$$\mathcal{G}_{\alpha,\aleph 0}/\equiv \; = \; \mathcal{G}_{\alpha,\aleph 0}/\triangleq.$$

That is, weak equivalence (or observational equivalence) coincides with bisimulation equivalence. In fact, the proof follows from Lemma 1.13.7.1. We give an alternative proof for the present simpler case here: Suppose g,h are finitely branching process graphs and suppose $g \equiv h$, or equivalently: $\forall n\ (g)_n \triangleq (h)_n$. Now consider

$$B_n = \{R \mid R \text{ is a bisimulation from } (g)_n \text{ to } (h)_n\},$$
$$B = \bigcup_{n\geq 1} B_n.$$

This collection of 'partial' bisimulations between g,h is ordered by set-theoretic inclusion ($\subseteq$). In fact, $B' = B \cup \{(s_0,t_0)\}$ where s_0,t_0 are the roots of g,h respectively, is a tree w.r.t. $\subseteq$. Because g,h are finitely branching, this tree is also finitely branching: there are only finitely many extensions of a bisimulation between $(g)_n$, $(h)_n$ to a bisimulation between $(g)_{n+1}$, $(h)_{n+1}$. Moreover, because $\forall n\ (g)_n \triangleq (h)_n$, the tree B' has infinitely many nodes. Therefore, by König's Lemma, B' has an infinite branch. This infinite branch is a chain of partial bisimulations R_i $(i\geq 1)$:

$$R_1 \subseteq R_2 \subseteq \ldots \subseteq R_n \subseteq \ldots$$

such that R_i is a bisimulation from $(g)_i$ to $(h)_i$. Now $R = \bigcup_{n\geq 1} R_i$ is a bisimulation from g to h.

The structures $\mathbb{G}^\circ_{\alpha,\beta}$ are also process algebras for PA. While all of the $\mathbb{G}^\circ_{\alpha,\beta}$ are metric spaces, they are not all *complete*. An example is given in Golson & Rounds [83]: $\mathbb{G}^\circ_{1,\aleph 0}$ is incomplete. (Consider the approximations of $\sum_{n\geq 1} a^n$.) Another example is as follows.

3.6. EXAMPLE. $\mathbb{G}^\circ_{\aleph\omega,\aleph\omega}$ is an incomplete metric space.

PROOF (sketch). The alphabet is $\{a_i \mid i < \aleph_\omega\}$. Define a sequence of process graphs g_n $(n \geq 1)$ by

$$g_n = \sum_{i1<\aleph 1} \sum_{i2<\aleph 2} \cdots \sum_{in<\aleph n} a_{i1}a_{i2}\ldots a_{in}.$$

Let brd(g) be the branching degree of process graph g, defined as follows: if s is a node of g, then brd(s) is the (cardinal) number of arrows leaving s; furthermore, brd(g) is the cardinal sum of the brd(s), $s \in$ NODES(g). We claim:

(i) $brd(g_n) = \aleph_n$ for g_n as defined above,

(ii) $brd((g)_n) \leq brd(g)$ for all $g \in \mathcal{G}_{\alpha,\beta}$,

(iii) $h \triangleq g_n \Rightarrow brd(h) \geq brd(g_n)$ for g_n as defined above.

Claim (ii) is trivial; the inductive proofs of the other two claims are left to the reader. Using these claims, one shows immediately that there is no limit $g/\equiv$ for the sequence of elements $g_n/\equiv$ in $\mathbb{G}^\circ_{\aleph\omega,\aleph\omega}$ as this would require a process graph g with branching degree at least $\Sigma_{n<\omega}\, \aleph_n = \aleph_\omega$. □

We will now define *projective models* $\mathbb{A}^\infty_{\alpha,\beta}$ of PA for arbitrary $\alpha \geq 1$ and $\beta \geq \aleph_0$. These will all be complete metric spaces. Furthermore, modulo isometry $\mathbb{A}^\infty_{\alpha,\beta}$ is an extension of $\mathbb{G}^\circ_{\alpha,\beta}$, so the projective model can be considered as the metric completion of $\mathbb{G}^\circ_{\alpha,\beta}$. (In case $\mathbb{G}^\circ_{\alpha,\beta}$ is also complete, it is of course isometric to the projective model.) The projective models defined below differ from the ones in Kranakis [86,87]; there an element of a projective sequence is a sequence of *terms* (modulo derivable equality), below it is a sequence of finitely deep *process graphs* (modulo bisimilarity).

3.7. DEFINITION. (i) $\mathcal{G}^n_{\alpha,\beta} = \{g \in \mathcal{G}_{\alpha,\beta} \mid g = (g)_n\}$.

(ii) $\mathbb{G}^n_{\alpha,\beta} = \mathcal{G}^n_{\alpha,\beta}/\triangleq$.

(iii) Let $g_i \in \mathbb{G}^n_{\alpha,\beta}$ $(i \geq 1)$. Then the sequence $(g_1, g_2, ...)$ is *projective* if for all i: $g_i = (g_{i+1})_i$.

(iv) $\mathbb{A}^\infty_{\alpha,\beta}$ is the projective limit of the $\mathbb{G}^n_{\alpha,\beta}$ $(n \geq 1)$; the elements of $\mathbb{A}^\infty_{\alpha,\beta}$ are the projective sequences. The operators $+, \cdot, \|, \mathbb{L}$ are defined as follows: if $\gamma = (g_1, g_2, ...)$ and $\gamma' = (g_1', g_2', ...)$ then $\gamma \cdot \gamma' = ((g_1 \cdot g_1')_1, (g_2 \cdot g_2')_2, ...)$ and likewise for the other operators.

3.8. THEOREM. $\mathbb{A}^\infty_{\alpha,\beta}$ *is a complete metric space.*

PROOF (sketch). Consider a converging sequence $\gamma_i = (g_{i1}, g_{i2}, ...)$, $i \geq 1$. For growing i and fixed k, the sequence g_{ik} will eventually be constant, say after N(k) steps. We may suppose that N is a monotonic function. Now $\gamma = (g_{N(1),1}, g_{N(2),2}, ...)$ is the required limit. □

Van Glabbeek (personal communication) remarked that for finite α, there is no need to consider uncountably branching process graphs, see statement (i) in Corollary 3.12. His observation can be generalized to infinite α. First some notation.

3.9. NOTATION. Let α be a cardinal number (finite or infinite). Then $\alpha^* = \Sigma_{n<\omega}\, \alpha_n$, where $\alpha_0 =$

α, $\alpha_{n+1} = 2^{\alpha_n}$. For finite α, we have $\alpha^* = \aleph_0$. For $\alpha = \aleph_0$, the numbers α_n are known as the beth-numbers $\beth_n$ and $\alpha^* = \beth_\omega$. The cardinality of a set X is card(X). If κ is a cardinal, then κ^+ denotes the least cardinal larger than κ.

3.10. PROPOSITION. (i) *For infinite* α: $\mathrm{card}(\mathbb{G}^n_{\alpha,\alpha^*}) = \alpha_n$.

(ii) $\mathrm{card}(\bigcup_{n\geq 1} \mathbb{G}^n_{\alpha,\alpha^*}) = \alpha^*$.

(iii) *For any* α,κ: $\mathbb{G}^n_{\alpha,\alpha^*} \cong \mathbb{G}^n_{\alpha,\alpha^*+\kappa}$.

PROOF. (i) Induction on n. For n = 1 the statement is clear, since the process graphs $g_I = \sum_{a\in I} a$ for arbitrary non-empty $I \subseteq A$ are mutually non-bisimilar, and since every process graph in $\mathbb{G}^1_{\alpha,\alpha^*}$ is bisimilar with some g_I. Suppose the statement has been proved for n. Let $\mathbb{X}^n \subseteq \mathbb{G}^n_{\alpha,\alpha^*}$ be a set of representatives of the α_n bisimulation equivalence classes of $\mathbb{G}^{n+1}_{\alpha,\alpha^*}$, so $\mathrm{card}(\mathbb{X}^n) = \alpha_n$. Now every element of $\mathbb{G}^{n+1}_{\alpha,\alpha^*}$ is bisimilar to one of the process graphs

$$g_{h,I,f} = h + \sum_{a\in I} \sum_{x\in f(I)} ax$$

where $h \in \mathbb{X}^n$, $I \subseteq A$ (possibly empty) and $f: I \to \wp(\mathbb{X}^n)$. Moreover, for different triples h,I,f the corresponding $g_{h,I,f}$ are not bisimilar. Hence $\mathrm{card}(\mathbb{G}^n_{\alpha,\alpha^*}) = \alpha_n \cdot \alpha_1 \cdot \alpha_{n+1} = \alpha_{n+1}$. Here the factor α_n stems from the variation in h, α_1 from the variation in I while for each I the choice of f contributes a factor $(2^{\mathrm{card}(\mathbb{X}^n)})^{\mathrm{card}(I)} = 2^{\alpha_n} = \alpha_{n+1}$.

Part (ii) is by definition; (iii) is left to the reader. □

3.11. THEOREM. $\mathbb{A}^\infty_{\alpha,\alpha^*} \cong \mathbb{A}^\infty_{\alpha,\alpha^*+\kappa}$ *for any cardinal* κ.

PROOF. The isometry follows at once from Proposition 3.10(iii). □

3.12. COROLLARY.

(i) *For finite* α: $\mathbb{A}^\infty_{\alpha,\aleph_0} \cong \mathbb{A}^\infty_{\alpha,\aleph_0+\kappa}$ *for any cardinal* κ.

(ii) *For countably infinite alphabet:* $\mathbb{A}^\infty_{\aleph_0,\beth_\omega} \cong \mathbb{A}^\infty_{\aleph_0,\beth_{\omega+\kappa}}$ *for any cardinal* κ. □

We will now turn our attention to the models $\mathbb{G}^\circ_{\alpha,\beta}$ in order to compare them with the projective models.

3.13. PROPOSITION. *If* β *is sufficiently large,* $\mathbb{G}^\circ_{\alpha,\beta}$ *is complete.*

PROOF. We will try to prove that $\mathbb{G}^\circ_{\alpha,\beta}$ is isometric to $\mathbb{A}^\infty_{\alpha,\beta}$ and deduce from that attempt a requirement on β.

We will drop the subscripts α,β. So let us try to establish an isometry φ from $\mathbb{G}^\circ$ to A^∞. Let $g \in \mathbb{G}^\circ$. Then $\varphi(g) = ((g)_1,(g)_2,...)$. It is easy to prove that this is a projective sequence. The hard part is to prove that φ is a surjection. Consider an element $(g_1, g_2, ...) \in A^\infty$. Let g_i be a representing process graph of g_i ($i \geq 1$). We would like to find a graph g such that $(g)_i \leftrightarrows g_i$ for all ≥ 1. (Cf. the construction in Theorem 3.5 of Golson & Rounds [83] by 'blowing up' trees; we wil use another construction.) For the rest of this proof, we will suppose that all process graphs are trees. Let g_i' be $(g_{i+1})_i$. So $g_i \leftrightarrows g_i'$; say R_i is a bisimulation from g_i to g_i'. Let S_i: NODES(g_i') $\rightarrow$ NODES(g_{i+1}) be the obvious embedding function, obtained by the projection mapping. Now if s is a node of depth k in g_k (so s is 'appearing' for the first time in g_k), we define some sequences starting with s, which we will call *fibres*, as follows. Any sequence

$$s = s_k,s_k',s_{k+1},s_{k+1}',s_{k+2},s_{k+2}',\cdots$$

where $s_i \in$ NODES(g_i), $s_i' \in$ NODES(g_i'), $(s_i,s_i') \in R_i$ and $S_i(s_i') = s_{i+1}$ ($i \geq k$) is a fibre. We will say that this fibre starts in g_k. If σ,τ are fibres, starting in g_k and g_{k+1} respectively, we define transitions $\sigma \rightarrow_a \tau$ if there are a-steps between the elements of these sequences:

$$\begin{array}{lllllll} \sigma: & s_k,s_k', & s_{k+1}, & s_{k+1}', & s_{k+2}, & s_{k+2}', & \cdots \\ \downarrow a & & \downarrow a & \downarrow a & \downarrow a & \downarrow a & \\ \tau: & & t_{k+1}, & t_{k+1}', & t_{k+2}, & t_{k+2}', & \cdots \end{array}$$

Now we construct the process graph γ with as nodes the fibres and transitions as just defined. More precisely: the root of γ is the fibre through the roots of $g_1,g_1',g_2,...$, and the other nodes of γ are those fibres reachable from the root of γ via transitions between fibres.

We claim that the projection $(\gamma)_n$ is bisimilar to g_n. A bisimulation ρ_n is given as follows: if s $\in$ NODES(g_n) and $\sigma \in$ NODES($(\gamma)_n$) then $(s,\sigma) \in \rho_n$ iff s is an element of σ. The verification of the claim is easy. An illustration is given in Figure 3.1 where γ is 'reconstructed' from the sequence of process graphs a, $a+a^2$, $a+a^2+a^3$, ... Interestingly, the result is not $\Sigma_{n\geq 1}\, a^n$ but $\Sigma_{n\geq 1}\, a^n + a^\omega$. (See the 'black fibers' in Figure 3.1.)

However, the problem is now to prove that the branching degree of γ is strictly bounded by β. We claim that this is so if $\beta > (\alpha^*)^{\aleph 0}$. Proof of the claim: let us take the g_i ($i \geq 1$) above as small as possible with respect to the cardinalities of their node sets. From the proof of Proposition 3.10(i) it is clear that we can take the g_i such that card(NODES(g_i)) $\leq \alpha_i$ (in fact we can even take card(NODES(g_i)) $\leq \alpha_{i-1}$). Hence we may suppose that the union of the node sets of the g_i, g_i' ($i\geq 1$) is bounded by α^*. Now every fibre (a node of the tree γ) is an ω-sequence of nodes of the g_i, g_i'. Hence there are at most $\kappa = (\alpha^*)^{\aleph 0}$ such fibres; so γ has at most κ nodes, so the branching degree of γ is bounded by κ. □

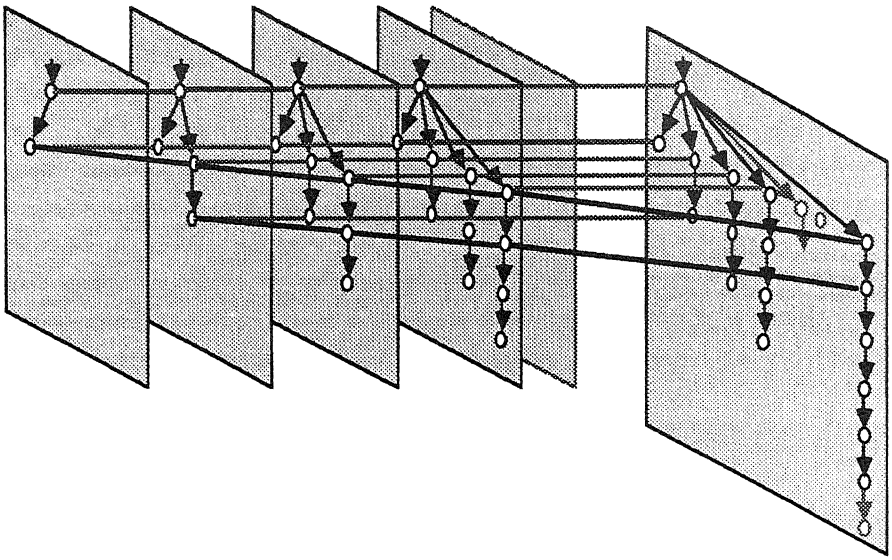


Figure 3.1

3.14. REMARK. (i) In the example above, in Figure 3.1, the process graph γ is closed (see Definition 3.23.1 for the definition of 'closed process graph'). In general, this needs not to be the case: e.g. if in the proof of Proposition 3.13, $g_i = (\Sigma_{n\geq 1} a^n)_i$ for $i \geq 1$ (so g_1 consists of infinitely many a-steps attached at the root) then $\gamma = \Sigma_{n\geq 1} a^n$ and this graph is not closed.

(ii) Another way of constructing a process graph g with projections $(g)_n$ bisimilar to g_n as in the proof above, is by taking g as the *canonical process graph* of the projective sequence $(g_1, g_2, ...) \in \mathbb{A}^\infty$. See Definition 3.23.2. One can prove that this graph is closed indeed, for $\beta > (\alpha^*)^{\aleph 0}$.

3.15. DEFINITION. Let $X,X' \subseteq \mathbb{G}_{\alpha,\beta}$. (i) Then $(X)_n = \{(g)_n \mid g \in X\}$.

(ii) $X \equiv_n X'$ if $\forall g \in X\ \exists g' \in X'\ g \equiv_n g'$ and $\forall g' \in X'\ \exists g \in X\ g \equiv_n g'$.

(iii) $X \equiv X'$ if $X \equiv_n X'$ for all n.

3.16. DEFINITION. Let $g \in \mathbb{G}_{\alpha,\beta}$. The a-*derivation* of g is the set of all subgraphs of g reachable by an a-step from the root. Notation: g/a.

3.17. PROPOSITION. *Let* $g,h \in \mathbb{G}_{\alpha,\beta}$. *Then* g,h *determine the same element in* $\mathbb{G}^\circ_{\alpha,\beta}$ *iff for all* a, $g/a \equiv h/a$.

PROOF. Routine. □

3.18. PROPOSITION. *Let* $X \subseteq \mathcal{G}_{\alpha,\beta}$. *Then there is an* $X' \subseteq \mathcal{G}_{\alpha,\beta}$ *such that* $X \equiv X'$ *and* $\mathrm{card}(X') \leq \alpha^*$.

PROOF. Consider the collection $\bigcup_{n\geq 1}(X)_n$ of finitely deep process graphs. We will construct a graph (not a process graph) with node set $\bigcup_{n\geq 1}(X)_n$, and arrows $g \to h$ for $g \in (X)_n$, $h \in (X)_{n+1}$ whenever $g = (h)_n$. See Figure 3.2.

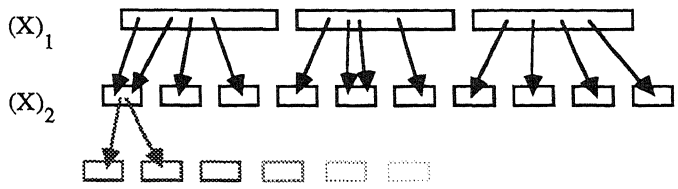


Figure 3.2

The boxes in Figure 3.2 are the $\triangleq$-equivalence classes. We note (Proposition 3.10(i)) that there are at most α_n boxes at level n, hence at most α^* boxes in total. Now every $g \in X$ corresponds with a path in this huge graph (not necessarily vice versa). We now construct X' as follows. If $g \in X$ is finitely deep (i.e. determines a terminating path in the graph of Figure 3.2), then $g \in X'$. Furthermore, in each box we select one node (i.e. a process graph $g \in (X)_n$ for some n) and choose an *arbitrary* path through this node. This path (which in fact is a projective sequence of process *graphs*) determines a process graph, call it $g^{\sim}$. Now we put $g^{\sim} \in X'$. Obviously, $\mathrm{card}(X') \leq \alpha^*$ and it is not hard to prove that $X \equiv X'$. □

3.19. PROPOSITION. *For all* α,κ: $\mathbb{G}^{\circ}_{\alpha,(\alpha^*)^+} \cong \mathbb{G}^{\circ}_{\alpha,(\alpha^*)^+ +\kappa}$

PROOF. Consider a process graph $g \in \mathcal{G}_{\alpha,(\alpha^*)^+ +\kappa}$. We must show that g can be pruned to a $g' \in \mathcal{G}_{\alpha,(\alpha^*)^+}$ such that g and g' determine the same element after dividing out $\triangleq$ and AIP (or dividing out $\equiv$ at once). This follows directly from the preceding two propositions. □

3.20. COROLLARY. *For all* α,κ,λ: $\mathbb{A}^{\infty}_{\alpha,\alpha^* + \kappa} \cong \mathbb{G}^{\circ}_{\alpha,(\alpha^*)^+ + \lambda}$

PROOF. This follows from Theorem 3.11 and Propositions 3.13 and 3.19. □

The cardinality of the models constructed above is for infinite alphabets quite large (this was already noticed in Golson & Rounds [83] for the process model of De Bakker & Zucker [82a,b];

see our remarks below). In fact:

3.21. PROPOSITION. (i) *For finite* α: $\mathrm{card}(A^{\infty}{}_{\alpha,\aleph 0}) = 2^{\aleph 0}$

(ii) *For countably infinite alphabet*: $\mathrm{card}(A^{\infty}{}_{\aleph 0,\beth_\omega}) = \beth_{\omega+1}$

(iii) *For general* α: $\mathrm{card}(A^{\infty}{}_{\alpha,\alpha^*}) = 2^{(\alpha^*)} = (\alpha^*)^{\aleph 0}$.

PROOF. (We will assume the Axiom of Choice in our calculations with cardinals.) Statements (i) and (ii) follow from (iii). Proof of (iii): Let λ be $\mathrm{card}(A^{\infty}{}_{\alpha,\alpha^*})$. Using Proposition 3.10 and noting that every element of $A^{\infty}{}_{\alpha,\alpha^*}$ is a map from ω into the union of the $\mathbb{G}^{n}{}_{\alpha,\alpha^*}$, we have $\lambda \leq (\alpha^*)^{\aleph 0}$. In view of the isomorphism with the graph models (Corollary 3.20), we find $\lambda \geq 2^{(\alpha^*)}$. The argument is as follows: there are α^* finitely deep process graphs which are mutually not bisimilar. (This is in fact Proposition 3.10(ii).) Let $\mathcal{F}$ be the set of these process graphs. For every subset $\mathcal{X}$ of $\mathcal{F}$ we define a process graph $g_{\mathcal{X}}$ as $\Sigma_{g\in\mathcal{X}}\, a.g$ for a fixed atom a. Now $g_{\mathcal{X}} \leftrightarrows g_{\mathcal{Y}}$ iff $\mathcal{X} = \mathcal{Y}$. Moreover, for different $\mathcal{X},\mathcal{Y}$ the corresponding graphs are not identified after dividing out AIP. So we now have:

$$2^{(\alpha^*)} \leq \lambda \leq (\alpha^*)^{\aleph 0}.$$

We also have: $2^{(\alpha^*)} = (\alpha^*)^{\alpha^*} \geq (\alpha^*)^{\aleph 0}$ (here AC is used, in the equality step). Hence the result follows. □

3.22. QUESTIONS. At present we do not know the answers to the following questions. For what α,β is $\mathbb{G}^{\circ}{}_{\alpha,\beta}$ a complete metric space? What is the cardinality of $\mathbb{G}^{\circ}{}_{\alpha,\beta}$ and $A^{\infty}{}_{\alpha,\beta}$? If $\mathbb{G}^{\circ}{}_{\alpha,\beta}$ is a complete metric space, is $\mathbb{G}^{\circ}{}_{\alpha,\beta'}$ for $\beta' > \beta$ also complete?

It is interesting to compare the projective model $A^{\infty}{}_{\alpha,\alpha^*}$ with the process model $\mathbb{P}_{\alpha}$ as constructed by De Bakker & Zucker [82a,b] as a solution of the domain equation

$$P \cong \{p_0\} \cup \wp_c(A \times P).$$

In $\mathbb{P}_{\alpha}$, processes can terminate with p_0 or with $\varnothing$ ('successfully' or 'unsuccesfully'). Leaving this double termination possibility aside (one can extend PA to PA_{δ} and have the same double termination possibility) or using a variant of the domain equation:

$$P \cong \wp_c(A \cup (A \times P)),$$

we can state that our projective model $\mathbb{A}^{\infty}_{\alpha,\alpha^*}$ is *isometric* to the process domain $\mathbb{P}_\alpha$. For finite α, this follows from the proof in Golson & Rounds [83] that $\mathbb{P}_\alpha$ is isometric to the graph domain $\mathbb{G}^{\circ}_{\alpha,\aleph_1}$; hence it is also isometric to $\mathbb{A}^{\infty}_{\alpha,\aleph_0}$, by Corollary 3.20. For infinite α the proof is similar. (The proof proceeds by noting that our spaces of finitely deep processes $\mathbb{G}^{n}_{\alpha,\alpha^*}$ are isometric to the P_n in De Bakker & Zucker [82a,b] or Golson & Rounds [83]; hence the completions of $\bigcup_{n\geq 1} \mathbb{G}^{n}_{\alpha,\alpha^*}$ and $\bigcup_{n\geq 1} P_n$, respectively, must also be isometric.) So the cardinality statements in Proposition 3.21 apply also to the models in De Bakker & Zucker [82a,b].

For a systematic (category-theoretic) treatment of De Bakker-Zucker domain equations like the two above, showing that they have unique solutions modulo isometry, we refer to America & Rutten [88].

3.23. Closed process graphs

We conclude with some remarks about a trade-off between closure properties of processes and the Approximation Induction Principle used in the construction of $\mathbb{G}^{\circ}_{\alpha,\alpha^*}$. These remarks are suggested by the fact that the model of De Bakker and Zucker is a solution of their domain equation; loosely speaking this means that the elements of that model can be perceived as 'hereditarily closed sets'. (Note, however, that these 'sets' are not well-founded. For a treatment of non-well-founded sets, including the connection with bisimulations, see Aczel [87].) One may ask whether the closure property can replace, when constructing a model from process graphs such as $\mathbb{G}^{\circ}_{\alpha,\alpha^*}$, taking the quotient with respect to AIP. We will make this question more precise using the definition of 'closed process tree' which was suggested to us by R. van Glabbeek (personal communication).

3.23.1. DEFINITION. (i) For process trees $g,h \in \mathcal{G}_{\alpha,\beta}$ we define the distance $\delta(g,h)$ as follows:

$$\delta(g,h) = \begin{cases} 2^{-m} & \text{if } \exists n\ g \not\equiv_n h;\ m = \min\{n \mid g \not\equiv_n h\} \\ 0 & \text{otherwise, i.e. } g \equiv h. \end{cases}$$

(ii) Let $\mathcal{H} \subseteq \mathcal{G}_{\alpha,\beta}$ be a set of process trees. Then $\mathcal{H}$ is *closed* if every Cauchy sequence $(g_i)_{i\geq 1}$ with respect to δ in $\mathcal{H}$ converges to a limit g in $\mathcal{H}$ (i.e. $\forall k\, \exists N\, \forall n>N\ g \equiv_k g_n$).

(iii) Let $g \in \mathcal{G}_{\alpha,\beta}$ be a process tree. Then g is *closed* if all its nodes s are closed; and a node s in g is closed when $(g)_s/a$ is a closed set of trees for every $a \in A$. Here $(g)_s$ is the subtree of g at s. Futhermore, a process graph is closed if its tree unwinding is closed. The set of all closed process graphs is $\mathcal{G}^{c}_{\alpha,\beta}$.

3.23.1.1. REMARK. Note that the closure property of process graphs is invariant under

bisimulation equivalence: if $g \leftrightarrows h$ and g is closed, then h is closed.

3.23.2. DEFINITION. Let $\mathbb{M}$ be a a process algebra for PA.

(i) From the elements x,y,z,... of $\mathbb{M}$ we construct a transition diagram (i.e. a 'process graph' without root and not necessarily connected) as follows. Whenever $x = ay + z$ there is a transition $x \rightarrow_a y$. In the case that $x = ay$ we have the same transition. If $x = a$, then there is a transition $x \rightarrow_a \mathbf{0}$ where $\mathbf{0}$ is the *termination node*. More concisely, we have $x \rightarrow_a y$ iff $x = ay + x$ and $x \rightarrow_a \mathbf{0}$ iff $x = a + x$. (To see this, use the axiom $x + x = x$.)

(ii) The *canonical process graph of* x *in* $\mathbb{M}$ is the process graph with root x, and as nodes all the elements of $\mathbb{M}$ reachable from x in zero or more transition steps as just defined, including possibly the termination node. Notation: $can_{\mathbb{M}}(x)$ or just can(x) when it is clear what $\mathbb{M}$ is meant. (See Figure 3.3 for the canonical process graph of $(\Sigma_{n\geq 1} a^n)/\equiv$ in $\mathbb{G}^{\circ}_{\alpha,\beta}$.)

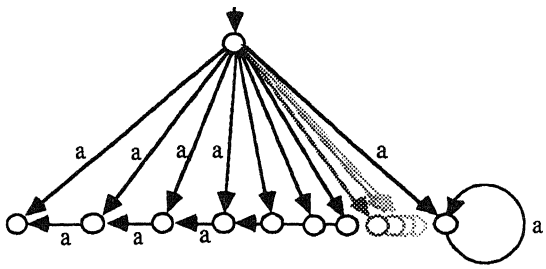


Figure 3.3

3.23.3. PROPOSITION. *Let* $g/\equiv$ *be an element of* $\mathbb{G}^{\circ}_{\alpha,\beta}$. *Then:*

(i) $can(g/\equiv) \equiv g$.

(ii) $can(g/\equiv) \equiv_n can(h/\equiv) \Leftrightarrow g \equiv_n h$.

(iii) $can(g/\equiv)$ *is a closed proces graph.*

PROOF. (i) With induction on n we prove that $g \equiv_n can(g/\equiv)$ for $n \geq 0$ (see Definition 3.3). The basis of the induction, $n = 0$, is trivial. Suppose (induction hypothesis) that we have proved $\forall g \; g \equiv_n can(g/\equiv)$. In order to prove $g \equiv_{n+1} can(g/\equiv)$, we have to show (1) and (2):

(1) *for every transition* $g \rightarrow_a g'$ *there is an initial step in* $can(g/\equiv)$: $g/\equiv \rightarrow_a h/\equiv$ *such that*

$$g' \equiv_n (can(g/\equiv))_{(h/\equiv)} = can(h/\equiv).$$

(Remember that $g/\equiv$, $h/\equiv$ are nodes in $can(g/\equiv)$.) Now $g/\equiv \rightarrow_a h/\equiv$ is (by definition of canonical

process graph) the same as: $g/\equiv \;=\; a(h/\equiv) + r/\equiv$ for some graph r. Or, equivalently: $g \equiv ah + r$. So, given the transition $g \twoheadrightarrow_a g'$ we have to find h,r with $g \equiv ah + r$ and $g' \equiv_n \text{can}(h/\equiv)$. This is simple: take $h = g'$ and r as given by $g \twoheadrightarrow_a g'$ (i.e. $g = a\cdot g' + r$ for some r). Now apply the induction hypothesis.

(2) *For every initial step in* $\text{can}(g/\equiv)$: $g/\equiv \;\to_a h/\equiv$ *there is a transition* $g \twoheadrightarrow_a g'$ *such that* $g' \equiv_n \text{can}(h/\equiv)$.

So, let $g/\equiv \;\to_a h/\equiv$ be given. This means $g \equiv ah + r$ for some r. In particular, $g \equiv_{n+1} ah + r$, i.e.

$$(g)_{n+1} \triangleq (ah + r)_{n+1} = a(h)_n + (r)_{n+1}. \qquad (*)$$

From the induction hypothesis we know that $h \equiv_n \text{can}(h/\equiv)$, i.e.

$$(h)_n \triangleq (\text{can}(h/\equiv))_n. \qquad (**)$$

Combining (*),(**) we have

$$(g)_{n+1} \triangleq a(\text{can}(h/\equiv))_n + (r)_n. \qquad (***)$$

Now we have to find a step $g \to_a g'$ such that $g' \equiv_n \text{can}(h/\equiv)$, i.e. $(g')_n \triangleq (\text{can}(h/\equiv))_n$. This is easily obtained from (***): consider the a-occurrence displayed in the right-hand side of (***). By definition of $\triangleq$, this a-step is matched in $(g)_{n+1}$ by an a-step $(g)_{n+1} \to_a (g')_n$ with $(g')_n \triangleq (\text{can}(h/\equiv))_n$.

(ii) Write $g^* = \text{can}(g/\equiv)$. To prove ($\Leftarrow$), suppose $g \equiv_n h$. Then $g^* \equiv g \equiv_n h \equiv h^*$, using (i). So $g^* \equiv_n h^*$. The proof of ($\Rightarrow$) is similar.

(iii) Consider $\text{can}(g/\equiv)$. (See Figure 3.4.) Let s be a node of this graph (so $s \in \mathbb{G}^\circ_{\alpha,\beta}$). Consider the a-derivation of s, i.e. the set of subgraphs of $\text{can}(g/\equiv)$ determined by the a-successors of s. Clearly, this a-derivation is the set of canonical graphs of some elements t_i ($i \in I$) of $\mathbb{G}^\circ_{\alpha,\beta}$. Suppose this set $\{\text{can}(t_i) \mid i \in I\}$ contains a Cauchy sequence (with respect to δ as in Definition 3.23.1):

$$\text{can}(t_{i0}),\ \text{can}(t_{i1}),\ \ldots,\ \text{can}(t_{in}),\ \ldots .$$

We claim that the elements $t_{i0}, t_{i1}, \ldots, t_{in}, \ldots$ form a Cauchy sequence in $\mathbb{G}^\circ_{\alpha,\beta}$. This follows at once

from (ii) of this proposition. So there is a limit $t \in \mathbb{G}^{\circ}_{\alpha,\beta}$ of the last Cauchy sequence. Now can(t) is easily seen (using again (ii)) to be a limit (in the δ-sense) for the Cauchy sequence $can(t_{i0})$, $can(t_{i1})$, ...

We still have to prove that $s \rightarrow_a t$, or equivalently (see Definition 3.23.2(i)) $s = at + s$ in $\mathbb{G}^{\circ}_{\alpha,\beta}$. Let $\underline{s}$ denote a representing process graph from the $\equiv$-equivalence class s, and likewise for t etc. Then we must prove that $\underline{s} \equiv a\underline{t} + \underline{s}$. To this end, take $\underline{t}_{ik}$ such that $\underline{t}_{ik} \equiv_n \underline{t}$. Since $\underline{s} \equiv a\underline{t}_{ik} + \underline{s}$ we have $\underline{s} \equiv_n a\underline{t} + \underline{s}$. Hence $\underline{s} \equiv a\underline{t} + \underline{s}$. □

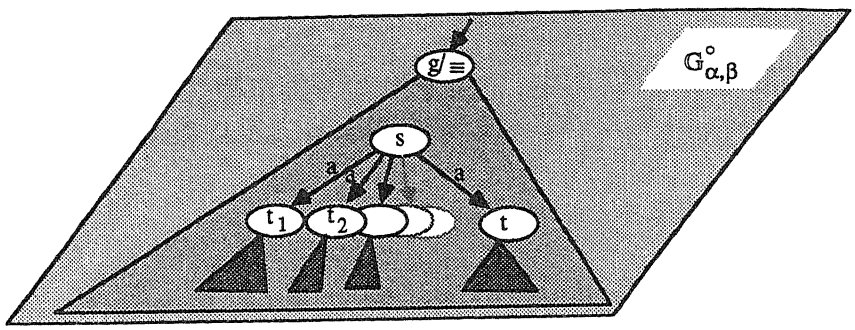


Figure 3.4

The preceding proposition enables us to define the *closure* of a process graph $g \in \mathbb{G}_{\alpha,\beta}$, notation g^c, as can(g/≡) w.r.t. $\mathbb{G}^{\circ}_{\alpha,\beta}$, such that $g \equiv g^c$. Next, we define operations $+^c$, $\cdot^c$, $\|^c$, $\between^c$ on $\mathbb{G}^c_{\alpha,\beta}$ as follows: $g \|^c h = (g \| h)^c$ and likewise for the other operators. Here $\|$ is the merge operation on $\mathbb{G}_{\alpha,\beta}$.

5.4. REMARK. If $\mathbb{G}^c_{\alpha,\beta}$ would have been closed under the operations $+$, $\cdot$, $\|$, $\between$ the preceding closure operation in $(g \| h)^c$ (etc.) would not have been necessary. However, for an *infinite* alphabet $\mathbb{G}^c_{\alpha,\beta}$ is not necessarily closed under $\|$, as the following example shows. (We conjecture that for finite alphabets $\mathbb{G}^c_{\alpha,\beta}$ is closed under the operations $\|$ etc.)

Let the alphabet be $\{a_i \mid i \geq 1\} \cup \{b,c\}$. We define process graphs H, G, g_n ($n \geq 1$):

$$H = \|_{i \geq 1} a_i^{\omega}$$

$$g_n = a_n \| b^n$$

$$G = \Sigma_{n \geq 1} c \cdot g_n.$$

Now H is a closed process graph. This can be easily seen, noting that H is a *deterministic* process graph, i.e. a graph where two different edges leaving the same node must have different label, and noting that deterministic graphs are always closed. Also G is closed: the c-derivation G/c, consisting of the graphs g_n, does not contain a Cauchy sequence since the graphs g_n are already different in their first level, due to the 'spoiling effect' of the a_n in g_n. Now G || H is, we claim, not closed. For, consider the c-derivation

$$(G \parallel H)/c = \{H \parallel g_n \mid n \geq 1\}.$$

Since $H \parallel a_n \leftrightarrows H$, we have

$$(G \parallel H)/c = \{H \parallel b^n \mid n \geq 1\},$$

modulo $\leftrightarrows$ which does not affect the closure properties (as remarked in 5.1.1). The last set is a Cauchy sequence: in general, if $\{q_i \mid i \geq 1\}$ is a Cauchy sequence of process graphs, then $\{p \parallel q_i \mid i \geq 1\}$ is again a Cauchy sequence for arbitrary p. However, there is no limit for this sequence in the set (G || H)/c, and hence it is not closed. So G || H is not closed.

This counterexample may seem somewhat surprising in view of a related result in De Bakker, Bergstra, Klop & Meyer [84], where it is stated (Theorem 2.9) that the collection of closed *trace languages* (containing possibly infinite traces) is closed under the merge operation, for arbitrary alphabet. Here a trace language is obtained as the set of all maximal traces of a process (or process graph). Note however that closure of processes does not very well correspond to closure of the corresponding trace sets; cf. also Example 4.4 in De Bakker, Bergstra, Klop & Meyer [84] of a closed process graph with a trace set which is not closed.

Next, we define the quotient structure

$$\mathbb{G}^c_{\alpha,\beta} = \mathfrak{G}^c_{\alpha,\beta} / \leftrightarrows.$$

Here $\mathfrak{G}^c_{\alpha,\beta}$ is supposed to be equipped with the operations as just defined. It is left to the reader to show that $\leftrightarrows$ is indeed a congruence with respect to these operations. Now there is the following fact, showing that indeed taking the quotient with respect to the congruence induced by AIP can be exchanged for the restriction to closed process graphs:

3.23.5. THEOREM. $\mathbb{G}^c_{\alpha,\beta} \cong \mathbb{G}^\circ_{\alpha,\beta}$.

PROOF. Remember that $\mathbb{G}^c_{\alpha,\beta} = \mathfrak{G}^c_{\alpha,\beta} / \triangleq$ and $\mathbb{G}^\circ_{\alpha,\beta} = \mathfrak{G}_{\alpha,\beta} / \equiv$. Define the map

$$\varphi: \mathfrak{G}^c_{\alpha,\beta} / \triangleq \ \rightarrow \ \mathfrak{G}_{\alpha,\beta} / \equiv$$

by $\varphi(g/\triangleq) = (g/\equiv)$. Here $g \in \mathfrak{G}^c_{\alpha,\beta}$ and $g/\triangleq$ is the equivalence class modulo $\triangleq$; likewise $g/\equiv$ is the equivalence class of g modulo $\equiv$.

(1) To prove that φ is injective, let $g,h \in \mathfrak{G}^c_{\alpha,\beta}$ and suppose $g \equiv h$. We must prove $g \triangleq h$. Define $R \subseteq \text{NODES}(g) \times \text{NODES}(h)$ as follows: $(s,t) \in R$ iff $(g)_s \equiv (g)_t$. We claim that R is a bisimulation from g to h. Proof of the claim: The roots are related, by the assumption $g \equiv h$. Further, suppose $(s,t) \in R$ and suppose there is a step $s \rightarrow_a s'$ in g. (See Figure 3.5.)

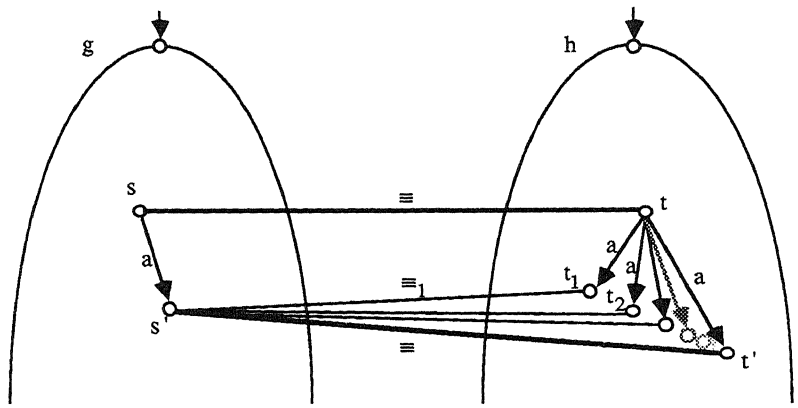


Figure 3.5

Since $(g)_s \equiv (h)_t$ we have for all $n \geq 1$: $(g)_s \equiv_n (h)_t$. This means that there are t_n such that $(g)_{s'} \equiv_n (h)_{t_n}$ for all $n \geq 1$. The t_n (or rather the $(h)_{t_n}$) form a Cauchy sequence with respect to δ, hence there is, since h is closed, a node t' such that $t \rightarrow_a t'$ and $(h)_{t'}$ is a limit for the Cauchy sequence t_n, $n \geq 1$. So $(h)_{t'} \equiv_n (h)_{t_m}$ for some $m \geq n$. Therefore $(h)_{t'} \equiv_n (h)_{t_m} \equiv_m (g)_{s'}$, and since $m \geq n$, $(h)_{t'} \equiv_n (g)_{s'}$. This holds for all $n \geq 1$, so $(h)_{t'} \equiv (g)_{s'}$, i.e. $(s',t') \in R$.

The same argument shows that if $(s,t) \in R$ and there is a step $t \rightarrow_a t'$ in h, then there is a step $s \rightarrow_a s'$ with $(s',t') \in R$.

This shows that R is a bisimulation from g to h, and ends the proof of (1).

(2) To prove that φ is surjective, we have to show that

$$\forall g \in \mathfrak{G}^c_{\alpha,\beta} \ \exists g' \in \mathfrak{G}^c_{\alpha,\beta} \ \ g \equiv g'.$$

This follows by taking $g' = \text{can}(g/\equiv)$ and applying Proposition 3.23.3(iii). □

In the case that β is large enough, so that $\mathbb{G}^c_{\alpha,\beta}$ is isometric to the process model $\mathbb{P}_\alpha$ of De Bakker and Zucker, this isometry leads to an 'explicit representation' of $\mathbb{P}_\alpha$, as follows. First a definition:

3.23.6. DEFINITION. (i) A process graph g is *minimal* if

$$\forall s,t \in \mathrm{NODES}(g)\ \ (g)_s \leftrightarrows (g)_t \Rightarrow s = t.$$

(ii) A process graph is *normal* if

$$\forall s,t,t' \in \mathrm{NODES}(g)\ \forall a \in A\ \ s \rightarrow_a t \ \&\ s \rightarrow_a t' \ \&\ (g)_s \leftrightarrows (g)_t \Rightarrow s = t.$$

Clearly, normality is implied by minimality. Also note that a process tree can never be minimal, unless it is linear (has only one branch); this is the reason for introducing the concept 'normal'.

It is not hard to prove that if g,h are minimal process graphs and $g \leftrightarrows h$, then g,h are in fact identical. Moreover, the canonical process graphs (of elements of $\mathbb{G}^\circ_{\alpha,\beta}$) are precisely the closed and minimal process graphs in $\mathcal{G}_{\alpha,\beta}$. Thus every element in $\mathbb{P}_\alpha$ can be represented by a *closed, minimal process graph with branching degree at most* α^*, and the operations in $\mathbb{P}_\alpha$ can be represented by the corresponding operations in $\mathcal{G}^c_{\alpha,\beta}$ followed by minimalisation (collapsing all bisimilar subgraphs). Another explicit representation can be given, using trees instead of graphs and observing that normal, bisimilar process trees are identical. Then the elements of $\mathbb{P}_\alpha$ correspond to *closed, normal process trees with branching degree at most* α^*. This representation is closer to the idea of elements of $\mathbb{P}_\alpha$ as 'hereditarily closed and possibly not well-founded sets'.

Summarizing our comparisons with $\mathbb{P}_\alpha$ we have established isometries (for all κ):

$$\mathbb{P}_\alpha \cong \mathbb{A}^\infty_{\alpha,\alpha^*+\kappa} \cong \mathbb{G}^\circ_{\alpha,(\alpha^*)^+ +\kappa}.$$

Furthermore, writing $\mathcal{G}^{cm}_{\alpha,\beta}$ for the set of closed minimal graphs in $\mathcal{G}_{\alpha,\beta}$ and $\mathcal{T}^{cn}_{\alpha,\beta}$ for the set of closed normal trees in $\mathcal{G}_{\alpha,\beta}$, there are the isometries

$$\mathbb{P}_\alpha \cong \mathbb{G}^c_{\alpha,(\alpha^*)^+ +\kappa} \cong \mathcal{G}^{cm}_{\alpha,(\alpha^*)^+ +\kappa} \cong \mathcal{T}^{cn}_{\alpha,(\alpha^*)^+ +\kappa},$$

where the last two complete metric spaces can be seen as 'explicit representations' of $\mathbb{P}_\alpha$.

References

ACZEL, P. (87), *Lecture Notes on Non-Well-Founded sets*, CSLI, Lecture Notes Nr.9, 1987

AMERICA, P. & RUTTEN, J.J.M.M. (88), *Solving reflexive domain equations in a category of complete metric spaces*, in: Proc. of the Third Workshop on Mathematical Foundations of Programming Language Semantics (M. Main, A. Melton, M. Mislove, D. Schmidt, eds.), Springer LNCS 298, 1988, p.2540288. Also to appear in the Journal of Computer and System Sciences.

BAETEN, J.C.M. & BERGSTRA, J.A. (88), *Global renaming operators in concrete process algebra*, Information and Computation, Vol.78, Nr.3 (1988), 205-245.

BAETEN, J.C.M., BERGSTRA, J.A. & KLOP, J.W. (86), *Syntax and defining equations for an interrupt mechanism in process algebra*, Fund. Inf. IX (2), p.127-168, 1986.

BAETEN, J.C.M., BERGSTRA, J.A. & KLOP, J.W. (87) *On the consistency of Koomen's Fair Abstraction Rule*, TCS 51 (1987), 129-176.

BAETEN, J.C.M., BERGSTRA, J.A. & KLOP, J.W. (87a), *Decidability of bisimulation equivalence for processes generating context-free languages*, in: Proc. PARLE, Vol.II (Parallel Languages), (eds. J.W. de Bakker, A.J. Nijman, P.C. Treleaven), Eindhoven 1987, Springer LNCS 259, p.94-113, 1987.

BAETEN, J.C.M., BERGSTRA, J.A. & KLOP, J.W. (87b), *Conditional axioms and α/β calculus in process algebra*, in: Proc. IFIP Conf. on Formal Description of Programming Concepts—III, Ebberup 1986, (M. Wirsing, ed.) North-Holland, Amsterdam 1987, p.53-75.

BAETEN, J.C.M & VAN GLABBEEK, R.J. (87), *Another look at abstraction in process algebra*, in: Proc. 14th ICALP 87, Karlsruhe (Th. Ottman, ed.), Springer LNCS 267, p.84-94, 1987.

DE BAKKER, J.W., BERGSTRA, J.A., KLOP, J.W. & MEYER, J.-J.CH. (84), *Linear time and branching time semantics for recursion with merge*. Theoretical Computer Science 34 (1984), p.135-156.

DE BAKKER, J.W. & ZUCKER, J.I. (82a), *Denotational semantics of concurrency*, Proc. 14th ACM Symp. Theory of Comp., p. 153 - 158, 1982.

DE BAKKER, J.W. & ZUCKER, J.I. (82b), *Processes and the denotational semantics of concurrency*, Information and Control 54 (1/2), p. 70 - 120, 1982.

BERGSTRA, J.A. & KLOP, J.W. (84a), *Process algebra for synchronous communication*, Information & Control 60 (1/3), p. 109 - 137, 1984.

BERGSTRA, J.A. & KLOP, J.W. (84b), *The algebra of recurively defined processes and the algebra of regular processes*, in: Proc. 11th ICALP (ed. J. Paredaens), Antwerpen 1984, Springer LNCS 172, p.82-95, 1984.

BERGSTRA, J.A. & KLOP, J.W. (85), *Algebra of communicating processes with abstraction*, TCS 37 (1), p. 77 - 121, 1985.

BERGSTRA, J.A. & KLOP, J.W. (86a), *Verification of an alternating bit protocol by means of process algebra*, in: Math. Methods of Spec. and Synthesis of Software Systems '85 (eds. W. Bibel and K.P. Jantke), Math. Research 31, Akademie-Verlag Berlin, p.9-23. 1986.

BERGSTRA, J.A. & KLOP, J.W. (86b), *Algebra of communicating processes*, in: CWI Monographs I, Proceedings of the CWI Symposium Mathematics and Computer Science (eds. J.W. de Bakker, M. Hazewinkel & J.K. Lenstra) North-Holland, Amsterdam, 1986, p.89-138.

BERGSTRA, J.A. & KLOP, J.W. (86c), *Process algebra: specification and verification in bisimulation semantics*, in: CWI Monograph 4, Proceedings of the CWI Symposium Mathematics and Computer Science II (eds. M. Hazewinkel, J.K. Lenstra & L.G.L.T. Meertens), North-Holland, Amsterdam 1986, p.61-94.

BERGSTRA, J.A. & KLOP, J.W. (87), *A convergence theorem in process algebra*, CWI Report CS-R8733, Centre for Mathematics and Computer Science, Amsterdam, 1987.

BERGSTRA, J.A. & KLOP, J.W. (88), *A complete inference system for regular processes with silent moves*, in: Proc. of Logic Colloquium, Hull '86, (eds. F.R. Drake and J.K. Truss), North-Holland 1988.

BERGSTRA, J.A., KLOP, J.W. & OLDEROG, E.-R. (86), *Failure semantics with fair abstraction*, CWI Report CS-R8609, Amsterdam 1986.

BERGSTRA, J.A., KLOP, J.W. & OLDEROG, E.-R. (87), *Failures without chaos: a new process semantics for fair abstraction*, in: Proceedings IFIP Conference on Formal Description of Programming Concepts—III, Gl. Avernaes (Ebberup) 1986 (ed. M. Wirsing), North-Holland, Amsterdam, p.77-103, 1987.

BERGSTRA, J.A., KLOP, J.W. & OLDEROG, E.-R. (88), *Readies and failures in the algebra of communicating processes*, CWI Report CS-R8523, Amsterdam 1985. To appear in SIAM J. of Computing, 1988.

BERGSTRA, J.A. & TIURYN, J. (87), *Process algebra semantics for queues*, Fund. Inf. X, p.213-224, 1987.

BERGSTRA, J.A. & TUCKER, J.V. (84), *Top down design and the algebra of communicating processes*, Sci. of Comp. Progr. 5 (2), p. 171 - 199, 1984.

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

BROOKES, S.D., HOARE, C.A.R. & ROSCOE, A.W. (84), A theory of Communicating Sequential Processes, JACM Vol.31, No.3 (1984) 560-599.

DE NICOLA, R. & HENNESSY, M. (83), *Testing equivalences for processes*, TCS 34, p.83-133.

VAN GLABBEEK, R.J. (87), *Bounded nondeterminism and the approximation principle in process algebra*. In: Proc. of the 4th Annual Symposium on Theoretical Aspects of Computer Science (eds. F.J. Brandenburg, G. Vidal-Naquet and M. Wirsing), Passau (W. Germany) 1987, Springer LNCS 247, 336-347.

VAN GLABBEEK, R.J. & VAANDRAGER, F.W. (88), *Modular specifications in process algebra—with curious queues*, Centre for Mathematics and Computer Science, Report CS-R8821, Amsterdam 1988; extended abstract to appear in: Proc. of the METEOR Workshop on Algebraic Methods: Theory, Tools and Applications, Springer LNCS.

GOLSON, W.G. & ROUNDS, W.C. (83), *Connections between two theories of concurrency: metric spaces and synchronization trees*. Information and Control 57 (1983), 102-124.

HENNESSY, M. (88), *Algebraic theory of processes*, The MIT Press, 1988.

HENNESSY, M. & MILNER, R. (85), *Algebraic laws for nondeterminism and concurrency*, JACM 32, 137-161.

HESSELINK, W. (88), *Deadlock and fairness in morphisms of transition systems*, Theor. Comp. Sci. 59 (1988) 235-257.

HOARE, C.A.R. (78), *Communicating sequential processes*, Comm. ACM 21, p. 666 - 677, 1978.

HOARE, C.A.R. (84), *Notes on communicating sequential processes*, International Summer School in Marktoberdorf: Control Flow and Data Flow, Munich 1984.

HOARE, C.A.R. (85), *Communicating sequential processes*, Prentice Hall 1985.

KOYMANS, C.P.J. & MULDER, J.C. (86), *A modular approach to protocol verification using process algebra*, Logic Group Preprint Series Nr.6, Dept. of Philosophy, State University of Utrecht, 1986; to appear in: Applications of Process Algebra, (J.C.M. Baeten, ed.), CWI Monograph, North-Holland, 1988.

KOYMANS, C.P.J. & VRANCKEN, J.L.M. (85), *Extending process algebra with the empty process* ε, Logic Group Preprint Series Nr.1, Dept. of Philosophy, State University of Utrecht, 1985.

KOSSEN, L. & WEIJLAND, W.P. (87), *Correctness proofs for systolic algorithms: palindromes and sorting*, Report FVI 87-04, Computer Science Department, University of Amsterdam, 1987.

KRANAKIS, E. (86), *Approximating the projective model*, in: Proc. Conf. on Math. Logic & its Applications, Druzhba (Bulgaria), 1986 (Pergamon Press).

KRANAKIS, E. (87), *Fixed point equations with parameters in the projective model*, Information and Computation, Vol.75, No.3, 1987.

MAUW, S. (87), *A constructive version of the Approximation Induction Principle*, Report FVI 87-09, Computer Science Department, University of Amsterdam, 1987.

MILNER, R. (80), *A calculus of communicating systems*, Springer LNCS 92, 1980.

MILNER, R. (84a), *Lectures on a Calculus for Communicating Systems*, Working Material for the Summer School Control Flow and Data Flow, Munich, July 1984.

MILNER, R. (84b), *A complete inference system for a class of regular behaviours*, Journal of Computer and System Sciences, Vol.28, Nr.3, 439-466, 1984.

MILNER, R. (85), *Lectures on a calculus for communicating systems*, in: Seminar on Concurrency, Springer LNCS 197 (1985), 197-220.

MILNER, R. (88), *A complete axiomatisation for observational congruence of finite-state behaviours*, Preprint, Univ. of Edinburgh 1985; to appear in Information and Computation 1988.

MOLLER, F. (88), *Non-finite axiomatisability in Process Algebras*, preprint, Univ. of Edinburgh, 1988

MULDER, J.C. (88), *On the Amoeba protocol*, CWI Report CS-R8827, Centre for Mathematics and Computer Science, Amsterdam 1988.

PARK, D.M.R. (81), *Concurrency and automata on infinite sequences*. Proc. 5th GI Conference, Springer LNCS 104, 1981.

PHILLIPS, I.C.C. (87), *Refusal testing*, TCS 50 (2), 1987.

VAANDRAGER, F.W. (86), *Verification of two communication protocols by means of process algebra*, CWI Report CS-R8608, Centre for Mathematics and Computer Science, Amsterdam 1986.

VRANCKEN, J.L.M. (86), *The Algebra of Communicating Processes with empty process*, Report FVI 86-01, Computer Science Department, University of Amsterdam, 1986.

WEIJLAND, W.P. (87), *A systolic algorithm for matrix-vector multiplication*, Report FVI 87-08, Computer Science Department, University of Amsterdam, 1987; also in: Proc. SION Conf. CSN 87, p.143-160, CWI, Amsterdam 1987.