Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.C.M. Baeten, R.J. van Glabbeek

Another look at abstraction in process algebra

Computer Science/Department of Software Technology Report CS-R8701 January

69F32, 69F43, 69F12, 69D41

Another look at abstraction in process algebra

J.C.M. Baeten,
Dept. of Computer Science, University of Amsterdam,
P.O. Box 41882, 1009 DB Amsterdam, The Netherlands

R.J. van Glabbeek,
Dept. of Software Technology, Centre for Mathematics and Computer Science,
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract: Central to theories of concurrency is the notion of abstraction. Abstraction from internal actions is the most important tool for system verification.
In this paper, we look at abstraction in the framework of the Algebra of Communicating Processes (see Bergstra & Klop [3, 5]). We introduce a hidden step η, and construct a model for the resulting theory ACP_η. We briefly look at recursive specifications, fairness and protocol verification in this theory, and discuss the relations with Milner's silent step τ.

Key words and phrases: concurrency, process algebra, hidden step, hiding, abstraction, silent step, internal action.
1985 Mathematics subject classification: 68Q55, 68Q45, 68Q10, 68N15.
1982 CR categories: F.3.2, F.4.3, F.1.2, D.3.1.

Note: Partial support received from the European Communities under ESPRIT contract no. 432, An Integrated Formal Approach to Industrial Software Development (Meteor). This report will be submitted for publication elsewhere.

1. Introduction.

Central to theories of concurrency is the notion of abstraction. In algebraic concurrency theories such as the Algebra of Communicating Processes (ACP, see BERGSTRA & KLOP [3, 5]) we use operators like alternative, sequential and parallel composition, to build up large systems from smaller processes. Often, such a large system must have a certain prescribed external behaviour, must communicate in a certain way with the environment. To verify that is indeed the case, we need to abstract from all internal behaviour of the system.
Following ideas of MILNER [10] and HOARE [9], abstraction can be modelled by distinguishing two kinds of actions in a process, viz. *external* or *observable* actions, and *internal* or *hidden* actions, and by introducing an explicit hiding operator that transforms observable actions into internal ones. We introduce a constant η for a hidden step, and formulate laws for this constant. Then we discuss the axiom system ACP_η, incorporating the η in the Algebra of Communicating Processes, and consider some properties of this system. We touch upon the usefulness of η in protocol verification, and discuss the issue of fairness. In this context, we formulate a fair

Report CS-R8701
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

abstraction rule HAR.
We also discuss a model for ACP_η consisting of finitely branching process graphs modulo an appropriate notion of bisimulation (see PARK [12], MILNER [11], BAETEN, BERGSTRA & KLOP [2]). We use this model to establish the consistency of ACP_η and the conservativity of ACP_η over $BPA_{\delta\eta}$ and ACP.
Finally, we discuss the relations of the constant η with Milner's silent step τ, that is also used for abstraction (see MILNER [10], BERGSTRA & KLOP [4]). We note that η has nicer technical properties than τ. Then, we consider two ways of combining both constants. First, the constant τ (at least in a system with only prefix multiplication) becomes *definable*, so that τ can be studied in the system ACP_η. Secondly, we can define a homomorphism from ACP_η into ACP_τ, that renames η into τ, and leaves all other constants fixed. This means that we can have a two-tiered abstraction: first we can abstract to η, and then, if further abstraction is desired, we can abstract from η to τ.
The original idea for the η, and some of its laws discussed in this paper, are due to Karst Koymans and Jos Vrancken, to whom the authors express their gratitude.

Table of contents:

2. Algebra of communicating processes.

In this section, we review the theory ACP (Algebra of Communicating Processes) as defined by BERGSTRA & KLOP [3, 5]. In the first paper, also a review of related approaches and comparisons with them can be found.

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

other operators, and + will always bind weaker.
On intuitive grounds $x(y + z)$ and $xy + xz$ present different mechanisms (the moment of choice is different), and therefore, an axiom $x(y + z) = xy + xz$ is not included.
We have a special constant δ denoting deadlock, the acknowledgement of a process that it cannot do anything anymore, the absence of any alternative. Axioms A6-7 give the laws for δ.
Next, we have the parallel composition operator $\|$, called merge. The merge of processes x and y will interleave the actions of x and y, except for the communication actions. In $x\|y$, we can either do a step from x, or a step from y, or x and y both synchronously perform an action, which together make up a new action, the communication action. This trichotomy is expressed in axiom CM1. Here, we use two auxiliary operators $\mathbin{\|\!\lfloor}$ (left-merge) and $|$ (communication merge). Thus, $x \mathbin{\|\!\lfloor} y$ is $x\|y$, but with the restriction that the first step comes from x, and $x\,|\,y$ is $x\|y$ with a communication step as the first step. Axioms CM2-9 and CF1-2 give the laws for $\mathbin{\|\!\lfloor}$ and $|$. The laws CF1-2 differ slightly from laws C1-3 in BERGSTRA & KLOP [3]. This will facilitate the

$x + y = y + x$		A1
$(x + y) + z = x + (y + z)$		A2
$x + x = x$		A3
$(x + y)z = xz + yz$		A4
$(xy)z = x(yz)$		A5
$x + \delta = x$		A6
$\delta x = \delta$		A7
$a\,|\,b = \gamma(a,b)$	if $\gamma(a,b)\downarrow$	CF1
$a\,|\,b = \delta$	otherwise	CF2
$x\|y = x \mathbin{\|\!\lfloor} y + y \mathbin{\|\!\lfloor} x + x\,|\,y$		CM1
$a \mathbin{\|\!\lfloor} x = ax$		CM2
$ax \mathbin{\|\!\lfloor} y = a(x\|y)$		CM3
$(x + y) \mathbin{\|\!\lfloor} z = x \mathbin{\|\!\lfloor} z + y \mathbin{\|\!\lfloor} z$		CM4
$a\,|\,bx = (a\,|\,b)x$		CM5
$ax\,|\,b = (a\,|\,b)x$		CM6
$ax\,|\,by = (a\,|\,b)(x\|y)$		CM7
$(x + y)\,|\,z = x\,|\,z + y\,|\,z$		CM8
$x\,|\,(y + z) = x\,|\,y + x\,|\,z$		CM9
$\partial_H(a) = a$	if $a \notin H$	D1
$\partial_H(a) = \delta$	if $a \in H$	D2
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		D3
$\partial_H(xy) = \partial_H(x)\cdot\partial_H(y)$		D4

Table 1. ACP.

formulation of the system ACP_η later on. Finally, we have in table 1 the encapsulation operator ∂_H. Here H is a set of atoms, and ∂_H blocks those actions, renames them into δ. The operator ∂_H can be used to encapsulate a process, i.e. to block communications with the environment.

2.2 Signature: A is a given (finite) set of atomic actions. On A, we have given a partial binary function γ, which is commutative and associative, i.e.

$$\gamma(a,b) = \gamma(b,a)$$
$$\gamma(a,\gamma(b,c)) = \gamma(\gamma(a,b),c)$$

for all $a,b,c \in A$. γ is the communication function: if $\gamma(a,b)$ is defined (we write $\gamma(a,b)\downarrow$), and $\gamma(a,b) = c$, it means that actions a and b communicate, and their communication is c; if $\gamma(a,b)$ is not defined, we say that a and b do not communicate.
All elements of A are constants of ACP. Further, ACP has binary operators $+,\cdot,\|,\lfloor\!\lfloor,|$, unary operators ∂_H (for $H \subseteq A$) and a constant δ.

2.3 Axioms: The axioms of ACP are presented in table 1 (on the previous page). There $a,b \in A\cup\{\delta\}$, $H \subseteq A$, and x,y,z are arbitrary processes.
Notice that axioms CF1 and CF2 imply that for all $a,b,c \in A\cup\{\delta\}$ we have:

$$a|b = b|a \qquad \text{(C1)}$$
$$a|(b|c) = (a|b)|c \qquad \text{(C2)}$$
$$\delta|a = \delta \qquad \text{(C3).}$$

Since every expression of the form $a|b$ is equal to an element of $A\cup\{\delta\}$, we can assume that axioms CM2,3,5-7 and D1,2 also hold for these expressions. We call the theory just consisting of the first five axioms, A1-5, BPA (so BPA has in the signature only operators $+,\cdot$ and constants A).

3. Hidden step η.

3.1 Let us consider a noisy machine, that is executing a process. When the machine starts the execution of a process, it starts humming. This noise stops upon successful termination. We can observe when the machine starts the execution of an atomic action a. Every atomic action takes some time to be executed, but we do not know how long. Also, this execution time may vary from instance to instance. Deadlock cannot be directly observed: we just see no termination, and no atomic action beginning.
When the machine is executing an internal step η, it is running for some time, but we do not observe any action beginning.
For the moment, we restrict our attention to the theory BPA with extra constant η.

3.2 We can observe no difference between processes $a\eta$ and a: in both cases we see the action a beginning as soon as the machine starts, and then we see the machine stop after a while. Also we can see no difference between $\eta\eta$ and η. This leads us to formulate the following law:

$$x\eta = x \qquad \text{H1.}$$

3.3 We do see a difference between processes ηa and a: in the case of ηa we see a begin some time after the start of the machine; in the case of a, we see a begin immediately. For the same reason, we have $\eta a + a \neq \eta a$.

3.4 The same philosophy leads us to adopt the law

$$a(\eta x + y) = a(\eta x + y) + ax \qquad \text{H3},$$

for, when the process $a(\eta x + y)$ is executed, it might be the case that we observe a begin, as soon as the machine is started, and then after a while, the machine reaches a state where only execution of x is possible. The laws H1 and H3 are (reformulations of) the first and the third τ-law of MILNER [10]. The considerations made above can be formalised as follows. The theory BPA_η has laws A1-5 and H1,3.

3.5 Definition. We define on BPA_η-terms binary predicates $\rightarrow^a$, and unary predicates $\rightarrow^a \surd$, for each $a \in A\cup\{\eta\}$.
$x \rightarrow^a y$ means that process x can evolve into process y, by starting a;
$x \rightarrow^a \surd$ means that process x can terminate (successfully), by starting a.
These predicates are defined by the following rules ($a \in A\cup\{\eta\}$, x,y arbitrary processes):
1. $a \rightarrow^a \surd$
2. if $x \rightarrow^a x'$, then $x+y \rightarrow^a x'$ and $y+x \rightarrow^a x'$
3. if $x \rightarrow^a \surd$, then $x+y \rightarrow^a \surd$ and $y+x \rightarrow^a \surd$
4. if $x \rightarrow^a x'$, then $xy \rightarrow^a x'y$
5. if $x \rightarrow^a \surd$, then $xy \rightarrow^a y$
6. $a \rightarrow^a \eta$
7. if $x \rightarrow^a y$ and $y \rightarrow^\eta z$, then $x \rightarrow^a z$
8. if $x \rightarrow^a y$ and $y \rightarrow^\eta \surd$, then $x \rightarrow^a \surd$.
(Compare these definitions with the ones in VAN GLABBEEK [8].)

3.6 Next, we say that two processes are equal, if they can perform the same actions.
Definition. A **bisimulation** is a binary relation R on process terms, satisfying ($a \in A\cup\{\eta\}$):
1. if $R(p,q)$ and $p \rightarrow^a p'$, then there is a q' such that $q \rightarrow^a q'$ and $R(p',q')$;
2. if $R(p,q)$ and $q \rightarrow^a q'$, then there is a p' such that $p \rightarrow^a p'$ and $R(p',q')$;
3. if $R(p,q)$, then $p \rightarrow^a \surd$ if and only if $q \rightarrow^a \surd$.
If there exists a bisimulation between processes p and q, we say p and q are **bisimilar**, and write $p \leftrightarrow q$.

3.7 Theorem. $\leftrightarrow$ is a congruence on BPA_η-terms.
Proof: Straightforward.

3.8 Definition: A **basic term** is a closed BPA_η-term of the form

$$t = a_0t_0 + \ldots + a_{n-1}t_{n-1} + b_0 + \ldots + b_{m-1}$$

for certain n,m with $n+m>0$, certain $a_i,b_j \in A\cup\{\eta\}$ and basic terms t_i. We usually abbreviate such expressions, in this case to

$$t = \Sigma_{i<n}\, a_i t_i + \Sigma_{j<m}\, b_j.$$

The **depth d(t)** of a basic term t is defined inductively by

$$d(\Sigma_{i<n}\, a_i t_i + \Sigma_{j<m}\, b_j) = 1 + \max(0, d(t_0), \ldots, d(t_{n-1})).$$

By systematically applying the rules of definition 3.5 it turns out that all relations $t \rightarrow^{a} s$ and $t \rightarrow^{a} \surd$ are of the form:

1. $t \rightarrow^{a_i} t_i$ $(i<n)$;
2. $t \rightarrow^{a_i} \eta t_i$ $(i<n)$;
3. $t \rightarrow^{b_j} \eta$ $(j<m)$;
4. $t \rightarrow^{a_i} s$ $(i<n)$ if $t_i \rightarrow^{\eta} s$;
5. $t \rightarrow^{b_j} \surd$ $(j<m)$;
6. $t \rightarrow^{a_i} \surd$ $(i<n)$ if $t_i \rightarrow^{\eta} \surd$.

So if $t \rightarrow^{a} s$, there are four possibilities:

1. t has a summand as, i.e. $t \equiv as + r$ or $t \equiv as$ (in fact, $A1{,}2 \vdash t = as + r$ or $A1{,}2 \vdash t = as$);
2. $s = \eta s'$ and $t \equiv as'$ $(+ r)$;
3. $s = \eta$ and $t \equiv a$ $(+ r)$;
4. $t \equiv at'$ $(+ r)$ and $t' \rightarrow^{\eta} s$.

Now we have the following proposition.

3.9 Proposition. Let t be a basic term and $a \in A \cup \{\eta\}$.

1. if $t \rightarrow^{a} s$, then s is a basic term and $d(s) \leq d(t)$;
2. if $t \rightarrow^{a} s$, then $BPA_\eta \vdash t = as + t$ (we say: as is a **BPA_η-summand** of t);
3. if $t \rightarrow^{a} \surd$, then $BPA_\eta \vdash t = a + t$.

Proof. 1. With induction on $d(t)$: cases 1, 2 and 3 (of 3.8) are trivial; for case 4 we can use the induction hypothesis, since $d(t') < d(t)$.

2. Cases 1, 2 and 3 are again trivial. For case 4 use induction on $d(t)$: assume $BPA_\eta \vdash t' = \eta s + t'$, then $BPA_\eta \vdash t = as + t$ follows by application of H3.

3. As 2. Case 5 is trivial, case 6 follows by induction, using H1 and H3.

3.10 Theorem. For all closed BPA_η-terms t,s we have $t \underline{\leftrightarrow} s \Leftrightarrow BPA_\eta \vdash t{=}s$.

Proof: $\Leftarrow$: Straightforward.

$\Rightarrow$: Consider the rewrite system consisting of the following two rules:

$$(x + y)z \rightarrow xz + yz$$
$$(xy)z \rightarrow x(yz)$$

This system is clearly terminating and a normal form of a closed BPA_η-term w.r.t. these rules must be a basic term. Now it is enough to prove the theorem for basic terms t,s.

For, if t',s' are two closed BPA_η-terms with $t' \underline{\leftrightarrow} s'$, and t,s are the corresponding normal forms, then $BPA_\eta \vdash t{=}t'$ and $BPA_\eta \vdash s{=}s'$, so $t \underline{\leftrightarrow} t' \underline{\leftrightarrow} s' \underline{\leftrightarrow} s$ (apply the direction $\Leftarrow$ of the theorem), and $BPA_\eta \vdash t{=}s$ will imply $BPA_\eta \vdash t'{=}s'$.

For basic terms s and t, we use induction on $d(t) + d(s)$.

Thus, suppose s,t are basic terms, $t \underline{\leftrightarrow} s$, and for all basic terms t',s' with $t' \underline{\leftrightarrow} s'$ and $d(s')+d(t')$

$< d(s)+d(t)$ it is already proved that $BPA_\eta \vdash t'=s'$.

Notice that it is enough to prove that any summand as' or a of s is a BPA_η-summand of t, since that implies $BPA_\eta \vdash t = s + t$, and then, for reasons of symmetry, also $BPA_\eta \vdash s = t + s$, which yields $BPA_\eta \vdash t = s$.

Case 1: as' is a summand of s, i.e. $s \equiv as'$ (+ r). Then $s \to^a s'$, so also $t \to^a t'$ for some t' with $s' \leftrightarrow t'$. By 3.9.1, t' is a basic term and $d(t') \le d(t)$. Furthermore $d(s') < d(s)$, so the induction hypothesis can be used and $BPA_\eta \vdash t'=s'$. Hence $BPA_\eta \vdash at' = as'$, and using proposition 3.9.2 this yields $BPA_\eta \vdash t = as' + t$.

Case 2: a is a summand of s, i.e. $s \equiv a$ (+ r). Then $s \to^a \surd$, so also $t \to^a \surd$. Now apply 3.9.3.

3.11 η and merge. The situation becomes more complicated if we consider the interaction of η and merge. Since η is also an action, all axioms that hold for atomic actions must also hold for η. In particular, laws CM2 and CM3 must hold for η instead of a. This leads to the following observation (assume that $a \mid b = \delta$):

$$\eta(ab + ba) = \eta(a\|b) = \eta a \mathbin{|\!\!\lfloor} b = \eta\eta a \mathbin{|\!\!\lfloor} b = \eta(\eta a\|b) = \eta(\eta a \mathbin{|\!\!\lfloor} b + b \mathbin{|\!\!\lfloor} \eta a + \eta a \mid b)$$
$$= \eta(\eta(a\|b) + b\eta a + \delta) = \eta(\eta(ab + ba) + ba).$$

The first term and the last term in this chain of equations cannot be proved equal in the system A1-5, H1,3. It turns out that it is sufficient to add one more law to the theory we have so far:

$$a(\eta(x + y) + x) = a(x + y) \qquad\qquad \text{H2.}$$

The execution of the η in the left-hand side leads from a state to another state that has at least the same possibilities, no options get lost. The philosophy is, that the execution of such an internal step cannot be observed by the environment.

3.12 Alternative. An alternative to the solution in 3.11 is, not adopting the law H2, but instead changing the laws CM2 and CM3 of ACP. In accordance with definition 3.5 we would have

$$\text{if } x \to^a x', \text{ then } x\|y \to^a x'\|y \text{ and } x \mathbin{|\!\!\lfloor} y \to^a x'\|y$$

so that in particular $a \mathbin{|\!\!\lfloor} x \to^a \eta\|x$. This leads to the following formulation of laws CM2 and CM3:

$$a \mathbin{|\!\!\lfloor} x = a(\eta x + x)$$
$$ax \mathbin{|\!\!\lfloor} y = a(\eta(x\|y) + x\|y).$$

We do not take this possibility in this paper, because we do not want to change the underlying system ACP.

3.13 ACP_η. Now we collect all axioms discussed so far together in the equational specification ACP_η. The theory ACP_η has in the signature, besides the elements of the signature from ACP, a constant η ($\eta \notin A$) and unary operators η_I for $I \subseteq A$. η_I is the **hiding operator**, that renames actions from I into η; I is the set of *internal* actions. ACP_η has the axioms in table 2 (on the following page). We put $C = A \cup \{\delta,\eta\}$, the set of all constants. In table 2 we have $a,b \in C$, $H,I \subseteq A$, and x,y,z are arbitrary processes.

The theory $BPA_{\delta\eta}$ consists of laws A1-7 and H1-3.

From now on we write $x=y$ for $ACP_\eta \vdash x=y$; if this holds we say that x is ACP_η-equal to y. If $A1,2 \vdash x=y$ we write $x \equiv y$.

$x + y = y + x$		A1	$x\eta = x$		H1
$(x + y) + z = x + (y + z)$		A2	$a(\eta(x + y) + x) = a(x + y)$		H2
$x + x = x$		A3	$a(\eta x + y) = a(\eta x + y) + ax$		H3
$(x + y)z = xz + yz$		A4			
$(xy)z = x(yz)$		A5			
$x + \delta = x$		A6			
$\delta x = \delta$		A7			
$a \mid b = \gamma(a,b)$	if $\gamma(a,b)\downarrow$	CF1			
$a \mid b = \delta$	otherwise	CF2			
$x \| y = x \mathbin{\lfloor\!\!\lfloor} y + y \mathbin{\lfloor\!\!\lfloor} x + x \mid y$		CM1			
$a \mathbin{\lfloor\!\!\lfloor} x = ax$		CM2			
$ax \mathbin{\lfloor\!\!\lfloor} y = a(x \| y)$		CM3			
$(x + y) \mathbin{\lfloor\!\!\lfloor} z = x \mathbin{\lfloor\!\!\lfloor} z + y \mathbin{\lfloor\!\!\lfloor} z$		CM4			
$a \mid bx = (a \mid b)x$		CM5			
$ax \mid b = (a \mid b)x$		CM6			
$ax \mid by = (a \mid b)(x \| y)$		CM7			
$(x + y) \mid z = x \mid z + y \mid z$		CM8			
$x \mid (y + z) = x \mid y + x \mid z$		CM9			
$\partial_H(a) = a$	if $a \notin H$	D1	$\eta_I(a) = a$	if $a \notin I$	HI1
$\partial_H(a) = \delta$	if $a \in H$	D2	$\eta_I(a) = \eta$	if $a \in I$	HI2
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		D3	$\eta_I(x + y) = \eta_I(x) + \eta_I(y)$		HI3
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$		D4	$\eta_I(xy) = \eta_I(x) \cdot \eta_I(y)$		HI4

Table 2. ACP_η.

3.14 <u>Note:</u> The following identities follow from ACP_η $(a \in C)$:
1. $\eta \mid a = \delta$ 2. $\partial_H(\eta) = \eta$ 3. $\eta_I(\delta) = \delta$.

3.15 <u>Note:</u> Axioms CM5 and CM6 are derivable from the other axioms of ACP_η.
<u>Proof:</u> CM5: $a \mid bx = a\eta \mid bx = (a \mid b)(\eta \| x) = (a \mid b)\eta(\eta \| x) = (a \mid b)(\eta\eta \mathbin{\lfloor\!\!\lfloor} x) = (a \mid b)(\eta \mathbin{\lfloor\!\!\lfloor} x) =$
$= (a \mid b)\eta x = (a \mid b)x$.
For CM6, note that $\eta \mid \eta x = (\eta \mid \eta)x = \delta = (\eta \mid \eta)(x \| \eta) = \eta x \mid \eta\eta = \eta x \mid \eta$. Using this fact, we get $\eta \| \eta x = \eta x \| \eta$ and hence $ax \mid b = a\eta x \mid b\eta = (a \mid b)(\eta x \| \eta) = (a \mid b)(\eta \| \eta x) = a\eta \mid b\eta x = a \mid bx =$
$= (a \mid b)x$.

3.16 As in 3.8, we define a **basic term** as an expression of the form

$$t = \Sigma_{i<n}\, a_i t_i + \Sigma_{j<m}\, b_j$$

with $n+m>0$, $a_i, b_j \in C$, and the t_i are again basic terms. The set of basic terms **BT** can be inductively built up as follows (working modulo law H1):

1. $\eta \in$ BT
2. if $a \in C$ and $x \in$ BT, then $ax \in$ BT
3. if $x,y \in$ BT, then $x+y \in$ BT.

Alternatively, we can build up BT as follows:

1. $\eta \in$ BT
2. if $n>0$, $a_i \in C$ and $t_i \in$ BT (for $i<n$), then $\Sigma_{i<n}\, a_i t_i \in$ BT.

Both these inductive schemes will be used in proofs.

3.17 Theorem: For every closed ACP_η-term t there is a basic term s such that $ACP_\eta \vdash t=s$. This is the so-called **elimination theorem.**

Proof: We will prove that, using the axioms of ACP_η except A1-3, A6,7 and H1-3 as rewrite rules (from left to right), t can be rewritten to a basic term s. Call this rewrite system $RACP_\eta$.

First, we define the **length, width** and **height** of a closed ACP_η-term t inductively as indicated in the following table 3.

t=	l(t)	w(t)	h(t)
$a \in C$	1	1	0
$u+v$	$\max(l(u),l(v))$	$w(u)+w(v)$	0
$u \cdot v$	$l(u)+l(v)$	$w(u)$	0
$u \Vert v$	$l(u)+l(v)$	$w(u)+w(v)+w(u)\cdot w(v)$	0
$u \mathbin{⌊\!⌊} v$	$l(u)+l(v)$	$w(u)$	0
$u \mid v$	$l(u)+l(v)$	$w(u)\cdot w(v)$	0
$\partial_H(u)$	$l(u)$	$w(u)$	$h(u)+1$
$\eta_I(u)$	$l(u)$	$w(u)$	$h(u)+1$

Table 3. Length, width and height of an ACP_η-term.

Roughly, the length of a term indicates the maximal number of steps that can occur when the term is executed, the width gives the number of alternatives at the start of the execution, and the height gives the number of renaming operators ∂_H and η_I around the term. Finally, we define the **size** of t, $s(t)$, to be the triple $<l(t), w(t), h(t)>$. The proof now proceeds via a number of claims.

Claim 1: Let t be a closed ACP_η-term. Then (using the alphabetical ordering on triples):

i. application of a rewrite rule does not increase the size of t;

ii. any proper subterm of t has a smaller size than t.

Proof: Easy.

Claim 2: The rewrite system $RACP_\eta$ is (strongly) terminating for closed ACP_η-terms.

Proof: Suppose it is not terminating. Let t be a closed ACP_η-term of minimal size, such that there is an infinite reduction sequence $t \rightarrow t_1 \rightarrow t_2 \rightarrow$ ..... A reduction on t_i is called **external** (outermost) if it works on the main operator of t_i, and **internal** if it works on a proper subterm of t_i. From claim 1 it follows that it is not possible that from some $i \in \mathbb{N}$ on, the sequence consists of internal reductions only. Therefore, there must be infinitely many external reductions in this sequence. Now note the following facts:

- all external reductions result in a constant $a \in C$, or in a term with + or · as main operator;
- there are no external reductions working on a constant or on a term with + as main operator;
- only the external reductions A4 and A5 work on a term with · as main operator;
- the external reduction A4 results in a term with + as main operator.

Thus, apart from the first one, all external reductions must be A5-reductions. Therefore, in $t \rightarrow t_1 \rightarrow t_2 \rightarrow$ ... we have, from some i on, $t_i = u_i \cdot v_i$, with $l(u_i)$ decreasing with each external reduction. This is impossible, and so claim 2 is proved.

Claim 3: All closed terms, which are normal forms w.r.t. the rewrite system $RACP_\eta$, are basic terms.

Proof: By induction on the structure of closed normal forms t. t must be a constant $a \in C$ or a term $u+v$, $u \cdot v$, $u \| v$, $u \lfloor\!\lfloor v$, $u \mid v$, $\partial_H(u)$ or $\eta_I(u)$. Since also u and v are normal forms, we may assume that they are basic terms. If $t = u \cdot v$ with $u \neq a$ for certain $a \in C$, or if $t = u \| v$, $u \lfloor\!\lfloor v$, $u \mid v$, $\partial_H(u)$ or $\eta_I(u)$, then t cannot be a normal form. In the other cases t is a basic term.

The elimination theorem now follows from claim 2 and claim 3.

3.18 Proposition: For all closed ACP_η-terms x,y,z we have the following laws of **standard concurrency**:

$x \mid y = y \mid x$	SC1
$x \| y = y \| x$	SC2
$x \mid (y \mid z) = (x \mid y) \mid z$	SC3
$(x \lfloor\!\lfloor y) \lfloor\!\lfloor z = x \lfloor\!\lfloor (y \| z)$	SC4
$(x \mid y) \lfloor\!\lfloor z = x \mid (y \lfloor\!\lfloor z)$	SC5
$x \| (y \| z) = (x \| y) \| z$	SC6.

Proof: Because of the elimination theorem we can assume that x,y,z are basic terms. We use the second induction scheme in 3.16. We consider only the case where none of x,y,z is η (the other cases are simpler). Write

$$x = \Sigma_{i \leq n} a_i x_i,\ y = \Sigma_{j \leq m} b_j y_j \text{ and } z = \Sigma_{k \leq p} c_k z_k$$

$(a_i, b_j, c_k \in C)$. By induction hypothesis, we can assume that the proposition holds for all triples (x_i,y,z), (x_i,y_j,z), (x_i,y_j,z_k). Then:

1. $x \mid y = \Sigma_{i,j} (a_i \mid b_j)(x_i \| y_j) = \Sigma_{i,j} (b_j \mid a_i)(y_j \| x_i)$ (C1, induction hypothesis for SC2) $= y \mid x$.
2. $x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x + x \mid y = y \lfloor\!\lfloor x + x \lfloor\!\lfloor y + y \mid x$ (by 1) $= y \| x$.
3. $x \mid (y \mid z) = \Sigma_{i,j,k} (a_i \mid (b_j \mid c_k))(x_i \| (y_j \| z_k)) = \Sigma_{i,j,k} ((a_i \mid b_j) \mid c_k)((x_i \| y_j) \| z_k)$ (C2, induction hypothesis for SC6) $= x \mid (y \mid z)$.

4. $(x\mathbin{\lfloor\!\lfloor}y)\mathbin{\lfloor\!\lfloor}z = (\Sigma_i a_ix_i\mathbin{\lfloor\!\lfloor}y)\mathbin{\lfloor\!\lfloor}z = \Sigma_i (a_i(x_i\|y))\mathbin{\lfloor\!\lfloor}z = \Sigma_i a_i((x_i\|y)\|z) = \Sigma_i a_i(x_i\|(y\|z))$ (induction hypothesis for SC6) $= \Sigma_i a_ix_i\mathbin{\lfloor\!\lfloor}(y\|z) = x\mathbin{\lfloor\!\lfloor}(y\|z)$.
5. $(x\,|\,y)\mathbin{\lfloor\!\lfloor}z = (\Sigma_{i,j}\,(a_i\,|\,b_j)(x_i\|y_j))\mathbin{\lfloor\!\lfloor}z = \Sigma_{i,j}\,(a_i\,|\,b_j)((x_i\|y_j)\|z) =$
$= \Sigma_{i,j}\,(a_i\,|\,b_j)(x_i\|(y_j\|z))$ (induction hypothesis for SC6) $= \Sigma_{i,j}\,a_ix_i\,|\,b_j(y_j\|z) = x\,|\,(y\mathbin{\lfloor\!\lfloor}z)$.
6. $x\|(y\|z) = x\mathbin{\lfloor\!\lfloor}(y\|z) + (y\|z)\mathbin{\lfloor\!\lfloor}x + x\,|\,(y\|z) =$
$= x\mathbin{\lfloor\!\lfloor}(y\|z) + (y\mathbin{\lfloor\!\lfloor}z)\mathbin{\lfloor\!\lfloor}x + (z\mathbin{\lfloor\!\lfloor}y)\mathbin{\lfloor\!\lfloor}x + (y\,|\,z)\mathbin{\lfloor\!\lfloor}x + x\,|\,(y\mathbin{\lfloor\!\lfloor}z) + x\,|\,(z\mathbin{\lfloor\!\lfloor}y) + x\,|\,(y\,|\,z) =$
$= (x\mathbin{\lfloor\!\lfloor}y)\mathbin{\lfloor\!\lfloor}z + y\mathbin{\lfloor\!\lfloor}(z\|x) + z\mathbin{\lfloor\!\lfloor}(y\|x) + (z\,|\,y)\mathbin{\lfloor\!\lfloor}x + (x\,|\,y)\mathbin{\lfloor\!\lfloor}z + (x\,|\,z)\mathbin{\lfloor\!\lfloor}y + (x\,|\,y)\,|\,z$ (by 1,3,4,5) $=$
$= (x\mathbin{\lfloor\!\lfloor}y)\mathbin{\lfloor\!\lfloor}z + y\mathbin{\lfloor\!\lfloor}(x\|z) + z\mathbin{\lfloor\!\lfloor}(y\|x) + z\,|\,(y\mathbin{\lfloor\!\lfloor}x) + (x\,|\,y)\mathbin{\lfloor\!\lfloor}z + (z\,|\,x)\mathbin{\lfloor\!\lfloor}y + z\,|\,(x\,|\,y)$ (by 1,2,5) $=$
$= (x\mathbin{\lfloor\!\lfloor}y)\mathbin{\lfloor\!\lfloor}z + (y\mathbin{\lfloor\!\lfloor}x)\mathbin{\lfloor\!\lfloor}z + z\mathbin{\lfloor\!\lfloor}(x\|y) + z\,|\,(y\mathbin{\lfloor\!\lfloor}x) + (x\,|\,y)\mathbin{\lfloor\!\lfloor}z + z\,|\,(x\mathbin{\lfloor\!\lfloor}y) + z\,|\,(x\,|\,y)$ (by 2,4,5) $=$
$= (x\|y)\mathbin{\lfloor\!\lfloor}z + z\mathbin{\lfloor\!\lfloor}(x\|y) + z\,|\,(x\|y) = (x\|y)\|z$.

3.19 Note: We usually assume that the laws of Standard Concurrency hold for all processes. Therefore, they are often called the *axioms* of Standard Concurrency.
Often, we also assume the following **Handshaking Axiom**:

$$x\,|\,y\,|\,z = \delta \qquad \text{(HA).}$$

It says, that all communication is *binary*, i.e. only involves two communication partners.

3.20 Proposition: In ACP_η with standard concurrency and handshaking axiom we have the following **expansion theorem** ($n\geq1$):

$$\|_{i\leq n}\, x_i = \Sigma_{i\leq n}\, x_i\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i}\, x_k) + \Sigma_{i<j\leq n}\,(x_i\,|\,x_j)\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i,j}\, x_k)$$

(Where $\|_{i\leq n}\, x_i$ of course means $x_0\|\ldots\|x_n$.)
Proof: We use induction on n. The case $n=1$ is exactly the axiom CM1. The induction step is as follows: $\|_{i\leq n+1}\, x_i = (\ \|_{i\leq n}\, x_i)\|x_{n+1} =$
$= (\ \|_{i\leq n}\, x_i)\mathbin{\lfloor\!\lfloor}x_{n+1} + x_{n+1}\mathbin{\lfloor\!\lfloor}(\ \|_{i\leq n}\, x_i) + (\ \|_{i\leq n}\, x_i)\,|\,x_{n+1}$.
We consider these three terms in turn. The first:
$(\ \|_{i\leq n}\, x_i)\mathbin{\lfloor\!\lfloor}x_{n+1} =$
$= \{\ \Sigma_{i\leq n}\, x_i\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i}\, x_k) + \Sigma_{i<j\leq n}\,(x_i\,|\,x_j)\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i,j}\, x_k)\ \}\mathbin{\lfloor\!\lfloor}x_{n+1} =$
$= \Sigma_{i\leq n}\,(x_i\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i}\, x_k))\mathbin{\lfloor\!\lfloor}x_{n+1} + \Sigma_{i<j\leq n}\,((x_i\,|\,x_j)\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i,j}\, x_k))\mathbin{\lfloor\!\lfloor}x_{n+1} =$
$= \Sigma_{i\leq n}\, x_i\mathbin{\lfloor\!\lfloor}((\ \|_{k\leq n,\ k\neq i}\, x_k)\|x_{n+1}) + \Sigma_{i<j\leq n}\,(x_i\,|\,x_j)\mathbin{\lfloor\!\lfloor}((\ \|_{k\leq n,\ k\neq i,j}\, x_k)\|x_{n+1})$ (use SC4,5) $=$
$= \Sigma_{i\leq n}\, x_i\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n+1,\ k\neq i}\, x_k) + \Sigma_{i<j\leq n}\,(x_i\,|\,x_j)\mathbin{\lfloor\!\lfloor}((\ \|_{k\leq n+1,\ k\neq i,j}\, x_k)$.
The second term is equal to $x_{n+1}\mathbin{\lfloor\!\lfloor}(\ \|_{i\leq n+1,\ k\neq n+1}\, x_i)$, and the third:
$(\ \|_{i\leq n}\, x_i)\,|\,x_{n+1} =$
$= \{\ \Sigma_{i\leq n}\, x_i\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i}\, x_k) + \Sigma_{i<j\leq n}\,(x_i\,|\,x_j)\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i,j}\, x_k)\ \}\,|\,x_{n+1} =$
$= \Sigma_{i\leq n}\,(x_i\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i}\, x_k))\,|\,x_{n+1} + \Sigma_{i<j\leq n}\,((x_i\,|\,x_j)\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i,j}\, x_k))\,|\,x_{n+1} =$
$= \Sigma_{i\leq n}\,(x_i\,|\,x_{n+1})\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i}\, x_k) + \Sigma_{i<j\leq n}\,(x_i\,|\,x_j\,|\,x_{n+1})\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n,\ k\neq i,j}\, x_k)$ (use SC1,3,5) $=$
$= \Sigma_{i<n+1}\,(x_i\,|\,x_{n+1})\mathbin{\lfloor\!\lfloor}(\ \|_{k\leq n+1,\ k\neq i,n+1}\, x_k)$ (by handshaking axiom).
Adding the three obtained expressions gives the desired result.

3.21 Note: In the next section we will prove that ACP_η is a **conservative extension** of $BPA_{\delta\eta}$ and of ACP, i.e. for all closed $BPA_{\delta\eta}$-terms t,s we have

$$ACP_\eta \vdash t=s \quad \text{iff} \quad BPA_{\delta\eta} \vdash t=s,$$

and for all closed ACP-terms t,s we have

$$ACP_\eta \vdash t{=}s \text{ iff } ACP \vdash t{=}s.$$

4. The graph model.

We construct a model for ACP_η consisting of equivalence classes of process graphs.

4.1 Definition: A process graph is a *labeled, rooted, finitely branching, directed multigraph.* An edge goes from a node to another (or the same) node, and is labeled with an element of C, the set of constants. We consider only finitely branching graphs, so each node has only finitely many outgoing edges. Graphs need not be finite (have finitely many nodes and edges), but we must be able to reach every node from the root in finitely many steps, so our graphs never have height more than ω. Finite graphs are also called **regular** graphs. $\mathbb{G}$ is the set of all process graphs, except the trivial graph $\mathbb{0}$, just consisting of one node. For more information about process graphs, see e.g. BAETEN, BERGSTRA & KLOP [2].

An **a-step** in a graph from s to s' is an edge going from s to s' with label $a \in C$, notation $s \rightarrow^a s'$; $\twoheadrightarrow^\eta$ is the transitive and reflexive closure of $\rightarrow^\eta$, so $s \twoheadrightarrow^\eta s'$ if there is a number of η-labeled edges (≥ 0), starting in s, and ending in s'. $\twoheadrightarrow^\eta$ is called a **generalized** η**-step**.

4.2 In order to define when two graphs denote the same process, we have the notion of bisimulating process graphs. For more information about bisimulations, see PARK [12], MILNER [11] or BAETEN, BERGSTRA & KLOP [2].

4.3 Definition. Let g,h be process graphs, and let R be a relation between nodes of g and nodes of h. R is a **rooted** η**-bisimulation** between g and h, notation R: $g \underline{\leftrightarrow}_{r\eta}$ h, iff

1. The roots of g and h are related.

2. If R(s,t) and $s \rightarrow^a s'$ is an edge in g with label $a \in A$ (so $a \neq \eta$, $a \neq \delta$), then, in h, we can do a generalized η-step $t \twoheadrightarrow^\eta t^*$ to a node t^* with $R(s,t^*)$, and from t^*, there is an a-step, followed by a generalized η-step to a node t' with R(s',t'). See fig. 1a.

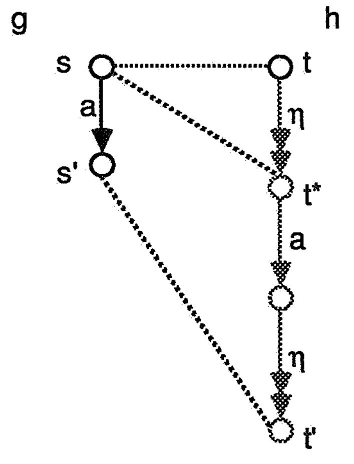

Fig. 1a.

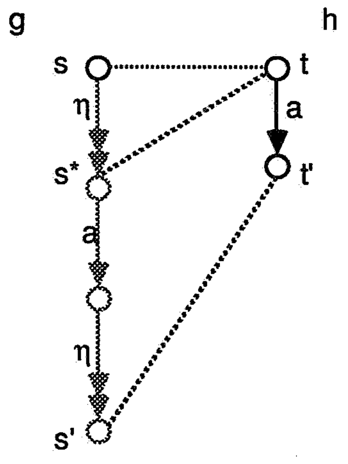

Fig. 1b.

In case (s,t) is the pair of roots, we must have $t \equiv t^*$ (this is part of the so-called **root condition**).
3. Vice versa: if R(s,t) and $t \rightarrow^{a} t'$ is an edge in h with label $a \notin A$, then, in g, we can do a generalized η-step $s \twoheadrightarrow^{\eta} s^*$ to a node s* with R(s*,t), and from s*, there is an a-step, followed by a generalized η-step to a node s' with R(s',t'). See fig. 1b. In case (s,t) is the pair of roots, we must have $s \equiv s^*$ (another part of the root condition).
4. If R(s,t) and $s \rightarrow^{\eta} s'$ is an edge in g, then, in h, we can do a generalized η-step $t \twoheadrightarrow^{\eta} t'$ to a node t' with R(s',t'). In case (s,t) is the pair of roots, the step $t \twoheadrightarrow^{\eta} t'$ must contain at least one edge (the third part of the root condition).
5. Vice versa: if R(s,t) and $t \rightarrow^{\eta} t'$ is an edge in h, then, in g, we can do a generalized η-step $s \twoheadrightarrow^{\eta} s'$ to a node s' with R(s',t'). In case (s,t) is the pair of roots, the step $s \twoheadrightarrow^{\eta} s'$ must contain at least one edge (the last part of the root condition).
6. If R(s,t) and s is an endpoint in g (i.e. s has no outgoing edges), then, in h, we can do a generalized η-step to an endnode of h.
7. Vice versa: if R(s,t) and t is an endpoint in h, then, in g, we can do a generalized η-step to an endnode of g.

A relation R between nodes of g and nodes of h is an **η-bisimulation** between g and h, $g \leftrightarrows_{\eta} h$, if we do not require the root condition in points 2-5.
Graphs g and h are **rη-bisimilar**, $g \leftrightarrows_{r\eta} h$, if there is a rooted η-bisimulation between g and h; g and h are **η-bisimilar**, $g \leftrightarrows_{\eta} h$, if there is a η-bisimulation between g and h.

4.4 Examples: See fig. 2. We have $a,b,c \in A \cup \{\eta\}$, so $\neq \delta$.

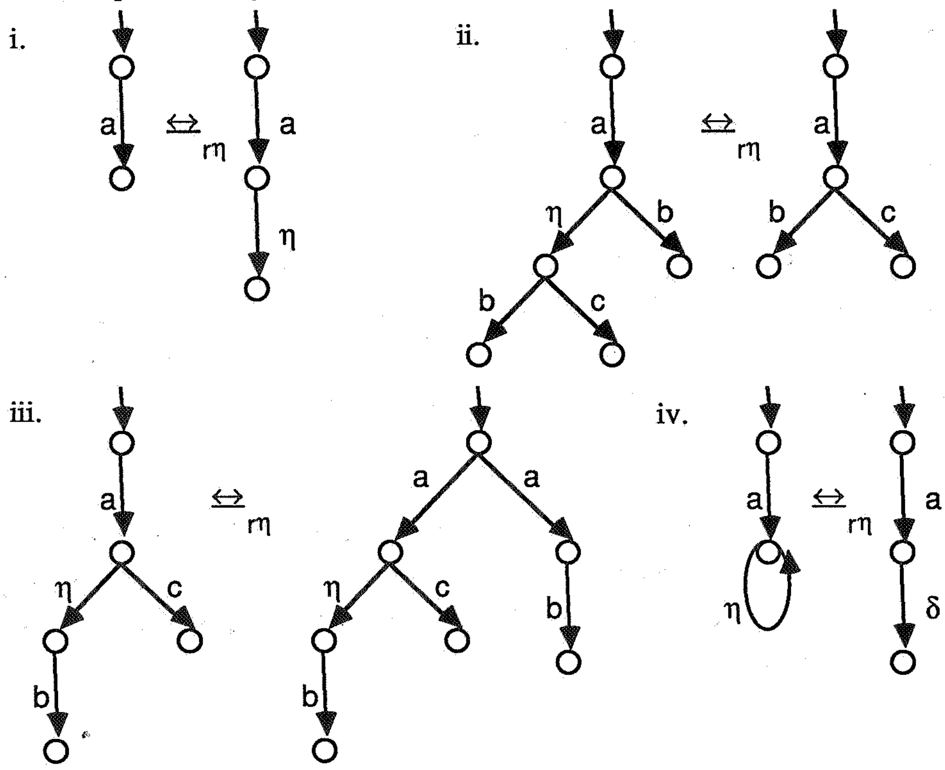

Fig. 2, i-iv.

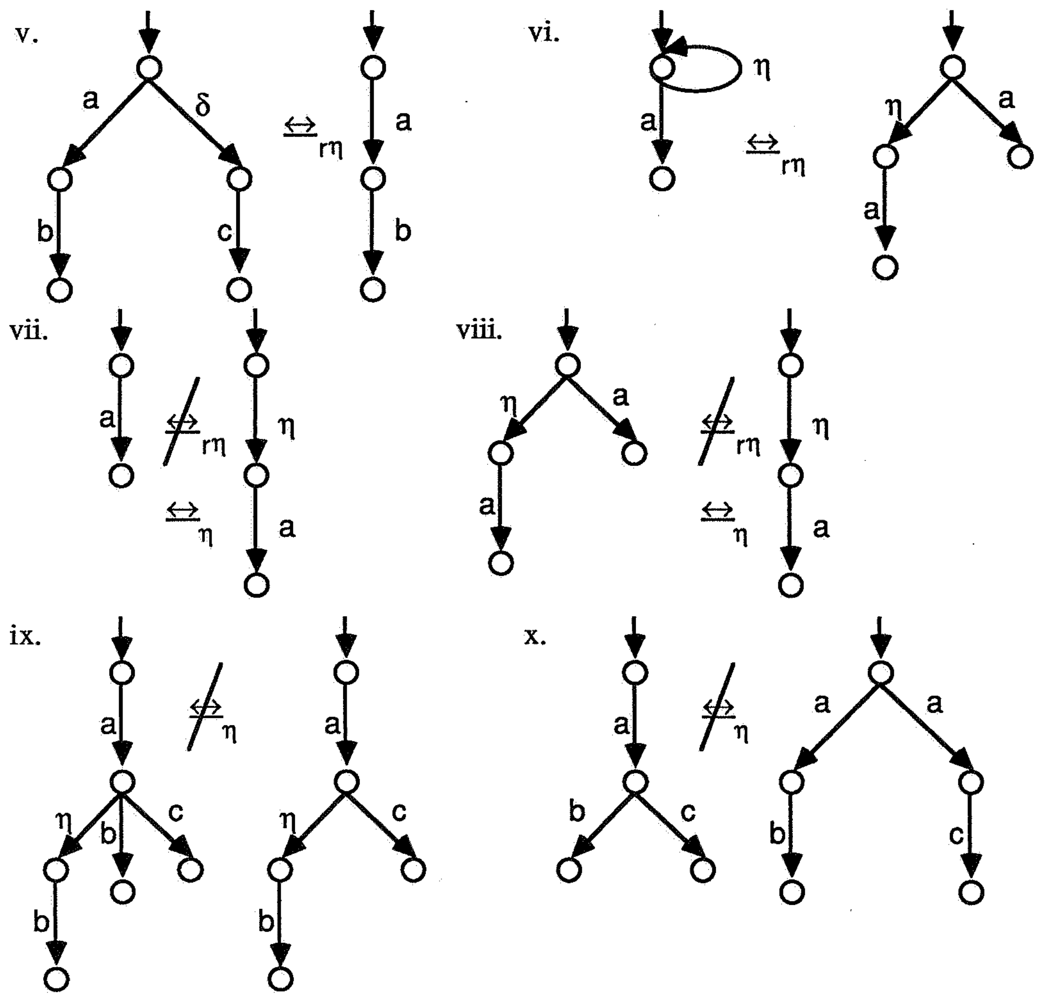
v.
vi.
vii.
viii.
ix.
x.
$\underline{\leftrightarrow}_{r\eta}$
$\not\underline{\leftrightarrow}_{r\eta}$
$\underline{\leftrightarrow}_{\eta}$
$\not\underline{\leftrightarrow}_{\eta}$
a
b
c
δ
η

Fig. 2, v-x.

4.5 Lemma: $\underline{\leftrightarrow}_{r\eta}$ and $\underline{\leftrightarrow}_{\eta}$ are equivalence relations on $\mathbb{G}$.
Proof: Straightforward.

4.6 $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ will be the domain of the graph model for ACP_η. The interpretation of a constant $a \in C$ is the equivalence class of the graph with two nodes and a single edge between them labeled a. What remains is the definition of the operators of ACP_η on $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$. We will define these operators on $\mathbb{G}$, and will then show that $\underline{\leftrightarrow}_{r\eta}$ is a congruence relation w.r.t. them.

4.7 Definitions.
1. +. If $g,h \in \mathbb{G}$, graph $g+h$ is obtained by taking the graphs of g and h and adding one new node r. For each edge $s \rightarrow^a s'$ in g from the root of g, we add an edge $r \rightarrow^a s'$; similarly, for each edge $t \rightarrow^a t'$ in h from the root of h, we add an edge $r \rightarrow^a t'$. Then, we discard nodes and edges that cannot be reached from the new root r.

Example:

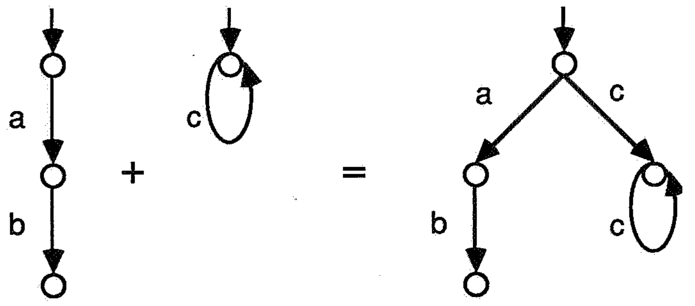

Fig. 3.

2. ·. If g,h ∈ $\mathbb{G}$, graph g·h is obtained by identifying all endpoints of g with the root node of h. If g has no endpoints, the result is just g. The root of g·h is the root of g.

Example:

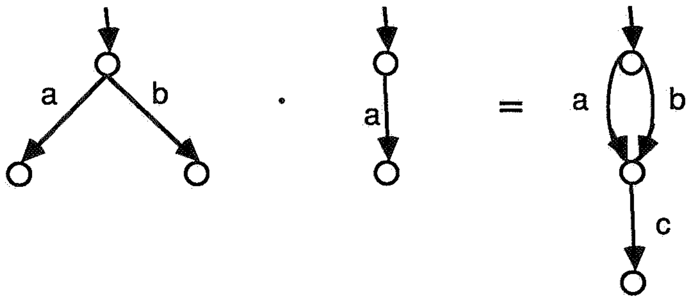

Fig. 4.

3. ||. If g,h ∈ $\mathbb{G}$, graph g||h is the cartesian product graph of graphs g and h, with 'diagonal' edges added for communication steps, i.e. if (s,t) is a node in g||h, then it has an outgoing edge (s,t) $\rightarrow^{a}$ (s',t) for each edge s $\rightarrow^{a}$ s' in g, an outgoing edge (s,t) $\rightarrow^{b}$ (s,t') for each edge t $\rightarrow^{b}$ t' in h, and moreover, whenever γ(a,b) is defined, outgoing edges (s,t) $\rightarrow^{\gamma(a,b)}$ (s',t'), so-called *diagonal* edges. The root of g||h is the pair of roots of g and h.

Example: Suppose γ(a,a) = d, and γ(a,b) and γ(a,c) are not defined. Then:

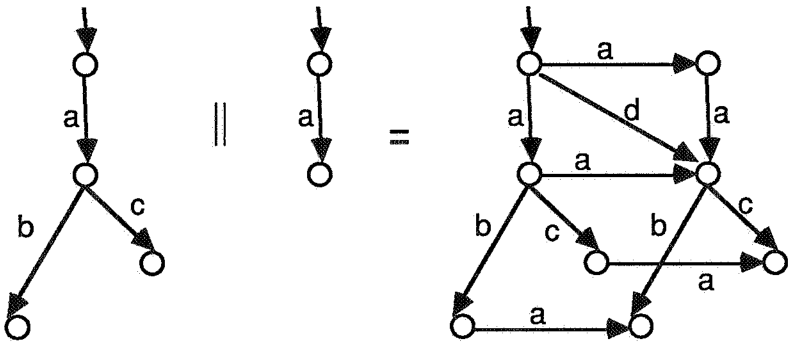

Fig. 5.

4. $\lfloor\lfloor$. If g,h ∈ $\mathbb{G}$, graph g$\lfloor\lfloor$h is obtained from graph g||h by adding a new node r, and, if s is the root of g and t the root of h, then we add, for each each edge s $\rightarrow^{a}$ s' in g, an edge r $\rightarrow^{a}$ (s',t). Then, we discard nodes and edges that cannot be reached from the new root r.

Example: (communications as in the previous example)

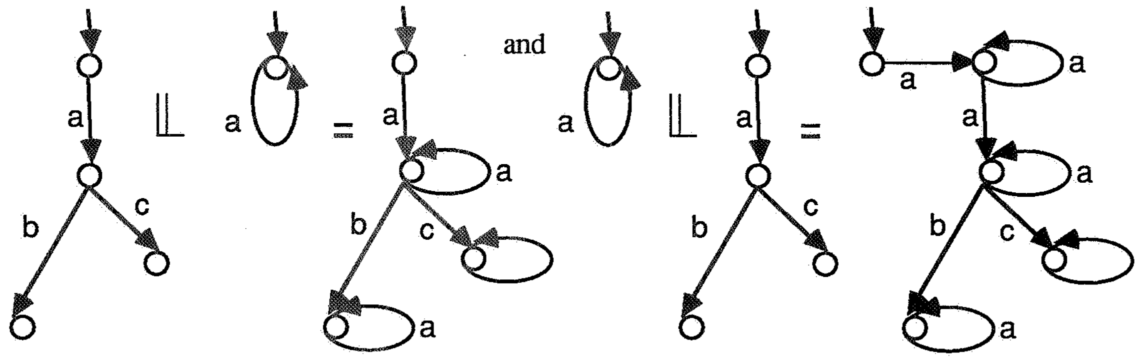

Fig. 6.

5. |. Similar to 4: if g,h ∈ 𝔾, graph g|h is obtained from graph g||h by adding a new node r, and adding, if s is the root of g and t the root of h, an edge $r \rightarrow^a (s',t')$ for each diagonal edge $(s,t) \rightarrow^a (s',t')$ in g||h. Then, we discard nodes and edges that cannot be reached from the new root r.

Example: (communications as before)

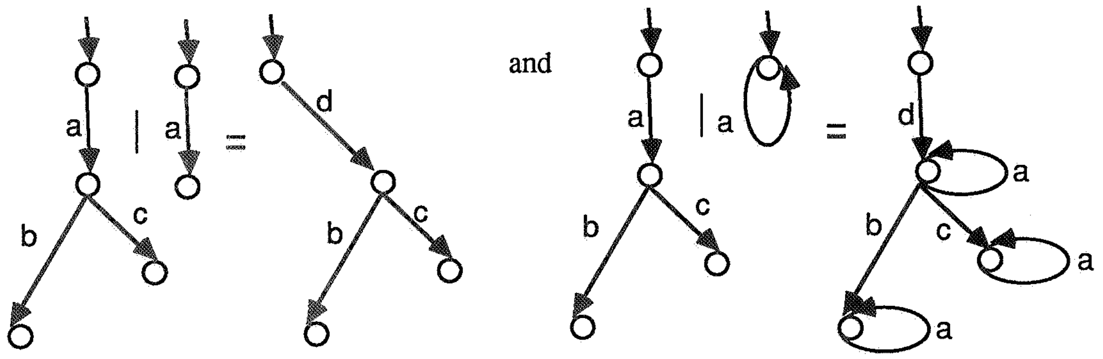

Fig. 7.

6. ∂_H, η_I. If g ∈ 𝔾, obtain $\partial_H(g)$ by replacing all labels in g from H by δ, and obtain $\eta_I(g)$ by replacing all labels from I by η.

This finishes the definition of the operators of ACP_η on 𝔾. Then we also have the operators on $𝔾/\underline{\leftrightarrow}_{r\eta}$, if we use the following proposition.

4.8 Proposition. $\underline{\leftrightarrow}_{r\eta}$ is a congruence relation on 𝔾.

Proof: Straightforward. As an example, consider the case of ||. So suppose g,g',h,h' ∈ 𝔾 and g $\underline{\leftrightarrow}_{r\eta}$ g', h $\underline{\leftrightarrow}_{r\eta}$ h'. We have to prove that g||h $\underline{\leftrightarrow}_{r\eta}$ g'||h'.

Take an rooted η-bisimulation R between g and g', and an rooted η-bisimulation S between h and h'. Let R×S be the cartesian product of R and S, i.e. R×S((s,t),(s',t')) iff R(s,s') and S(t,t') (s a node in g, s' in g', t in h, t' in h').

We claim that R×S is a rooted η-bisimulation between g||h and g'||h'.

Proof of the claim: (1) Let $(s_1,t_1) \rightarrow^a (s_1,t_2)$ be a 'horizontal' step in g||h, with a∈ A. Let $R\times S((s_1,t_1),(s'_1,t'_1))$. Then $t_1 \rightarrow^a t_2$ in h and $S(t_1,t'_1)$, hence we can find nodes t^*_1 and t'_2 such that $t'_1 \twoheadrightarrow^\eta t^*_1 \rightarrow^a \twoheadrightarrow^\eta t'_2$ and $S(t_1,t^*_1)$ and $S(t_2,t'_2)$. This path can be 'lifted' to g'||h', i.e. we get $(s'_1,t'_1) \twoheadrightarrow^\eta (s'_1,t^*_1) \rightarrow^a \twoheadrightarrow^\eta (s'_1,t'_2)$ and $R\times S((s_1,t_1),(s'_1,t^*_1))$ and $R\times S((s_1,t_2),(s'_1,t'_2))$.

(2) Likewise for a 'vertical' step in $g||h$.

(3) Suppose $(s_1,t_1) \rightarrow^c (s_2,t_2)$ is a diagonal step in $g||h$, and $R{\times}S((s_1,t_1),(s'_1,t'_1))$. Then there are steps $s_1 \rightarrow^a s_2$ in g and $t_1 \rightarrow^b t_2$ in h with $\gamma(a,b) = c$. Since $R(s_1,s'_1)$, there are nodes s^*_1 and s'_2 in g' such that $s'_1 \twoheadrightarrow^\eta s^*_1 \rightarrow^a \twoheadrightarrow^\eta s'_2$ and $R(s_1,s^*_1)$ and $R(s_2,s'_2)$. Likewise, $t'_1 \twoheadrightarrow^\eta t^*_1 \rightarrow^b \twoheadrightarrow^\eta t'_2$, $S(t_1,t^*_1)$ and $S(t_2,t'_2)$ for certain t^*_1, t'_2 in h'.
We compose these paths in
$g'||h'$: $(s'_1,t'_1) \twoheadrightarrow^\eta (s'_1,t^*_1) \twoheadrightarrow^\eta (s^*_1,t^*_1) \rightarrow^c \twoheadrightarrow^\eta \twoheadrightarrow^\eta (s'_1,t'_2)$ and $R{\times}S((s_1,t_1),(s^*_1,t^*_1))$ and $R{\times}S((s_1,t_2),(s'_2,t'_2))$.
The remainder of the verification is straightforward.

4.9 Theorem: $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ is a model of ACP_η.
Proof: It has to be checked that for any closed instance of an axiom of ACP_η, $t=s$, the interpretation of both sides of the equality sign in $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ yields the same equivalence class of process graphs. Thus, if $graph(t)$ denotes the graph corresponding to the closed term t, following definitions 4.6 and 4.7, then it has to be checked that $graph(t) \underline{\leftrightarrow}_{r\eta} graph(s)$. The construction of these rooted η-bisimulations is routine, tedious and omitted (cf. BERGSTRA & KLOP [4], 2.5). The only interesting cases concern the η-laws, of which instances are presented in examples 4.4.

4.10 Remark: We also obtain models of ACP_η, if instead of limiting ourselves to finitely branching graphs, we allow all graphs of branching degree less than some infinite cardinal number. Thus we get models $\mathbb{G}_\kappa/\underline{\leftrightarrow}_{r\eta}$. $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ is the model $\mathbb{G}_{\aleph_0}/\underline{\leftrightarrow}_{r\eta}$. Also, the set $\mathbb{R}$ of all finite process graphs modulo $\underline{\leftrightarrow}_{r\eta}$ and the set $\mathbb{F}$ of all finite and acyclic process graphs modulo $\underline{\leftrightarrow}_{r\eta}$ form models of ACP_η.
In the sequel, we will show that these models are also *complete* for closed ACP_η-terms, i.e. if t,s are closed ACP_η-terms, then $graph(t) \underline{\leftrightarrow}_{r\eta} graph(s)$ implies $ACP_\eta \vdash t=s$.

4.11 Definitions: A **rooted path** π in a process graph g is a finite alternating sequence of connected nodes and edges of g, starting in the root, and ending in the so-called **endnode** of π. The **length** of π is the number of edges in π. A node s of g is **reached** by the rooted path π if s is the endnode of π. A process graph g is a **tree** if any node of g is reached by exactly one rooted path. A graph is **finite** if it has finitely many nodes and edges. Note that a tree g is finite iff there are finitely many rooted paths in g. The **depth** $d(g)$ of a finite tree g is the length of its longest path. Note that the set of graphs $\mathbb{F}$, introduced in 4.10, is the set of graphs with only finitely many rooted paths.

4.12 Proposition: i. If t is a basic term then $graph(t)$ is a finite nontrivial process tree.
ii. If $graph(t) \equiv graph(s)$ for $s,t \in BT$, then $t \equiv s$ (i.e. A1,2 $\vdash t=s$).
iii. For any finite nontrivial process tree g, there is a basic term t with $graph(t) \equiv g$.
Proof: Easy.

4.13 Definition. Let $term$ be a function that maps a finite nontrivial process tree g onto a basic term t with $\mathrm{graph(t)}\equiv\mathrm{g}$. By proposition 4.12 we have $\mathrm{term(graph(t))}\equiv\mathrm{t}$ for $\mathrm{t}\in BT$. Although $\mathrm{term}(\mathbb{0})$ is undefined we write $\mathrm{a}\cdot\mathrm{term}(\mathbb{0})$ for a.

4.14 Definition. If s is a node of a process graph g then $(\mathrm{g})_\mathrm{s}$ denotes the subgraph of g, obtained from g by leaving out all nodes and edges which are not reachable from s. s will be the root of $(\mathrm{g})_\mathrm{s}$. Of course $(\mathrm{g})_{\mathrm{root(g)}} = \mathrm{g}$.

4.15 Proposition: If R is an η-bisimulation between process graphs g,h (not necessarily rooted), and R(s,t), then $(\mathrm{g})_\mathrm{s} \underline{\leftrightarrow}_\eta (\mathrm{h})_\mathrm{t}$.
Proof: R, restricted to the nodes of $(\mathrm{g})_\mathrm{s}$ and $(\mathrm{h})_\mathrm{t}$, will be an η-bisimulation.

4.16 Proposition: Let g be a finite nontrivial process tree. Then term(g) has a summand s = as' or s = a iff there is an edge $\mathrm{root(g)} \rightarrow^{a} \mathrm{p}$ in g with $\mathrm{a}\cdot\mathrm{term}((\mathrm{g})_\mathrm{p}) = \mathrm{s}$.
Proof: Easy.

4.17 Proposition: Let p and q be nodes of a finite process tree g, such that $\mathrm{p} \rightarrow^{a} \twoheadrightarrow^{\eta} \mathrm{q}$ (i.e. from p, we can do an a-step, followed by a generalized η-step, ending in q). Then $\mathrm{term}((\mathrm{g})_\mathrm{p}) = \mathrm{a}\cdot\mathrm{term}((\mathrm{g})_\mathrm{q}) + \mathrm{term}((\mathrm{g})_\mathrm{p})$ (i.e. $\mathrm{a}\cdot\mathrm{term}((\mathrm{g})_\mathrm{q})$ is an ACP_η-summand of $\mathrm{term}((\mathrm{g})_\mathrm{p})$).
Proof: By induction on the number of η-steps in $\twoheadrightarrow^{\eta}$. Let this number be n.
The induction base n=0 follows from proposition 4.15.
Now suppose $\mathrm{p} \rightarrow^{a} \mathrm{p}' \rightarrow^{\eta} \twoheadrightarrow^{\eta} \mathrm{q}$ and for $\mathrm{n}=\mathrm{length}(\twoheadrightarrow^{\eta})$ the proposition is already proved.
Then $\mathrm{term}((\mathrm{g})_\mathrm{p}) = \mathrm{a}\cdot\mathrm{term}((\mathrm{g})_{\mathrm{p}'}) + \mathrm{term}((\mathrm{g})_\mathrm{p})$ and $\mathrm{term}((\mathrm{g})_{\mathrm{p}'}) = \eta\cdot\mathrm{term}((\mathrm{g})_\mathrm{q}) + \mathrm{term}((\mathrm{g})_{\mathrm{p}'})$.
Hence $\mathrm{term}((\mathrm{g})_\mathrm{p}) = \mathrm{a}(\eta\cdot\mathrm{term}((\mathrm{g})_\mathrm{q}) + \mathrm{term}((\mathrm{g})_{\mathrm{p}'})) + \mathrm{term}((\mathrm{g})_\mathrm{p}) =$ (using H3)
$= \mathrm{a}\cdot\mathrm{term}((\mathrm{g})_\mathrm{q}) + \mathrm{a}(\eta\cdot\mathrm{term}((\mathrm{g})_\mathrm{q}) + \mathrm{term}((\mathrm{g})_{\mathrm{p}'})) + \mathrm{term}((\mathrm{g})_\mathrm{p}) = \mathrm{a}\cdot\mathrm{term}((\mathrm{g})_\mathrm{q}) + \mathrm{term}((\mathrm{g})_\mathrm{p})$.
In case q is an endnode, we have $\mathrm{term}((\mathrm{g})_{\mathrm{p}'}) = \eta + \mathrm{term}((\mathrm{g})_{\mathrm{p}'})$, and hence
$\mathrm{term}((\mathrm{g})_\mathrm{p}) = \mathrm{a}(\eta + \mathrm{term}((\mathrm{g})_{\mathrm{p}'})) + \mathrm{term}((\mathrm{g})_\mathrm{p}) =$ (using H1 and H3)
$= \mathrm{a}\eta + \mathrm{a}(\eta\eta + \mathrm{term}((\mathrm{g})_{\mathrm{p}'})) + \mathrm{term}((\mathrm{g})_\mathrm{p}) = \mathrm{a} + \mathrm{term}((\mathrm{g})_\mathrm{p})$.

4.18 Proposition: For finite process trees g,h we have:
i. If $\mathrm{g} \underline{\leftrightarrow}_\eta \mathrm{h}$ then $\mathrm{ACP}_\eta \vdash \mathrm{a}\cdot\mathrm{term(g)} = \mathrm{a}\cdot\mathrm{term(h)}$ for each $\mathrm{a}\in C$.
ii. If $\mathrm{g,h} \neq \mathbb{0}$ and $\mathrm{g} \underline{\leftrightarrow}_{\mathrm{r}\eta} \mathrm{h}$ then $\mathrm{ACP}_\eta \vdash \mathrm{term(g)} = \mathrm{term(h)}$.
Proof: (i) will be proved with induction on d(g) + d(h). So suppose $\mathrm{g} \underline{\leftrightarrow}_\eta \mathrm{h}$, say $\mathrm{R}: \mathrm{g} \underline{\leftrightarrow}_\eta \mathrm{h}$, and for any finite process trees g',h' with $\mathrm{d(g')} + \mathrm{d(h')} < \mathrm{d(g)} + \mathrm{d(h)}$ (i) is already proved.

Claim: One of the following statements holds:
I: $\mathrm{h} \neq \mathbb{0}$ and $\mathrm{term(h)} = \eta\cdot\mathrm{term(g)} + \mathrm{term(h)}$
II: $\mathrm{g,h} \neq \mathbb{0}$ and $\mathrm{term(h)} = \mathrm{term(g)} + \mathrm{term(h)}$
III: $\mathrm{a}\cdot\mathrm{term(h)} = \mathrm{a}\cdot\mathrm{term(g)}$ for $\mathrm{a}\in C$.
Proof of the claim: I: Suppose there is a node q in h with $\mathrm{root(h)} \rightarrow^{\eta} \twoheadrightarrow^{\eta} \mathrm{q}$ and R(root(g),q).
Then, by proposition 4.15, $\mathrm{g} \underline{\leftrightarrow}_\eta (\mathrm{h})_\mathrm{q}$ and $\mathrm{d}((\mathrm{h})_\mathrm{q}) < \mathrm{d(h)}$, so by induction $\eta\cdot\mathrm{term(g)} = \eta\cdot\mathrm{term}((\mathrm{h})_\mathrm{q})$. Furthermore, using proposition 4.17, $\mathrm{term(h)} = \eta\cdot\mathrm{term}((\mathrm{h})_\mathrm{q}) + \mathrm{term(h)}$, and

hence $\mathsf{term}(h) = \eta\cdot\mathsf{term}(g) + \mathsf{term}(h)$. (Of course $h \neq \mathbb{0}$.)
II: Suppose there is no such node. If $g = \mathbb{0}$, then there must be an endnode q in h with $\mathsf{root}(h) \twoheadrightarrow^{\eta} q$. Of course, $R(\mathsf{root}(g),q)$. Hence, by the assumption just made, $\mathsf{root}(h) = q$ so $h = \mathbb{0}$. Thus III holds. Therefore, we can suppose $g \neq \mathbb{0}$. In that case $\mathsf{term}(g)$ is well-defined. We call a summand δ or δt of $\mathsf{term}(g)$ **redundant** if $\mathsf{term}(g)$ also has a summand a or at with $a{\neq}\delta$. We will prove that any non-redundant summand at or a of $\mathsf{term}(g)$ either is an ACP_η-summand of $\mathsf{term}(h)$ or is ACP_η-equal to $\eta\cdot\mathsf{term}(h)$. If all non-redundant summands of $\mathsf{term}(g)$ are ACP_η-summands of $\mathsf{term}(h)$ we get II. The case that there are summands of $\mathsf{term}(g)$ ACP_η-equal to $\eta\cdot\mathsf{term}(h)$ will be considered in part III of this proof.
So let $s = as'$ or $s = a$ be a summand of $\mathsf{term}(g)$. By proposition 4.16 there is an edge $\mathsf{root}(g) \xrightarrow{a} p$ in g with $a\cdot\mathsf{term}((g)_p) = s$.
<u>Case 1:</u> $a \in A$. Since $R\colon g \underline{\leftrightarrow}_\eta h$ there must be nodes q^* and q in h with $\mathsf{root}(h) \twoheadrightarrow^{\eta} q^* \xrightarrow{a} \twoheadrightarrow^{\eta} q$ $R(\mathsf{root}(g),q^*)$ and $R(p,q)$. Hence, by proposition 4.15, $(g)_p \underline{\leftrightarrow}_\eta (h)_q$, and since $d((g)_p) < d(g)$ (and $d((h)_q) < d(h)$), the induction hypothesis yields $a\cdot\mathsf{term}((g)_p) = a\cdot\mathsf{term}((h)_q)$. By the assumption above $\mathsf{root}(h) = q^*$ and proposition 4.17 gives $\mathsf{term}(h) = a\cdot\mathsf{term}((h)_q) + \mathsf{term}(h)$. Thus $\mathsf{term}(h) = s + \mathsf{term}(h)$.
<u>Case 2:</u> $a = \eta$. Since $R\colon g \underline{\leftrightarrow}_\eta h$, there must be a node q in h with $\mathsf{root}(h) \twoheadrightarrow^{\eta} q$ and $R(p,q)$. Hence $(g)_p \underline{\leftrightarrow}_\eta (h)_q$ and since $d((g)_p) < d(g)$, the induction hypothesis yields $\eta\cdot\mathsf{term}((g)_p) = \eta\cdot\mathsf{term}((h)_q)$. Now there are two possibilities:
<u>Case 2.1:</u> $\mathsf{root}(h) \neq q$. Then $\mathsf{root}(h) \xrightarrow{\eta} \twoheadrightarrow^{\eta} q$ and proposition 4.17 gives $\mathsf{term}(h) = \eta\cdot\mathsf{term}((h)_q) + \mathsf{term}(h)$. Thus $\mathsf{term}(h) = s + \mathsf{term}(h)$.
<u>Case 2.2:</u> $\mathsf{root}(h) = q$. Then $s = \eta\cdot\mathsf{term}((g)_p) = \eta\cdot\mathsf{term}((h)_q) = \eta\cdot\mathsf{term}(h)$.
<u>Case 3:</u> $a = \delta$. If $h \neq \mathbb{0}$ then $\mathsf{term}(h)$ is defined and $\mathsf{term}(h) = s + \mathsf{term}(h)$ follows from laws A6 and A7. If $h = \mathbb{0}$ then clause 7 of definition 4.3 implies that there is an endnode q in g with $\mathsf{root}(g) \twoheadrightarrow^{\eta} q$. Since we also have $\mathsf{root}(g) \xrightarrow{\delta} p$, $\mathsf{root}(g) \neq q$. Hence $\mathsf{term}(g)$ has a summand η or ηt, so s is redundant.
III: Finally suppose that $g \neq \mathbb{0}$, and some summands of $\mathsf{term}(g)$ are ACP_η-equal to $\eta\cdot\mathsf{term}(h)$, while the others are redundant or ACP_η-summands of $\mathsf{term}(h)$. Then, if $h \neq \mathbb{0}$,
$\mathsf{term}(g) = \eta\cdot\mathsf{term}(h) + t$ and $\mathsf{term}(h) = t + \mathsf{term}(h)$. Hence, using H2, $a\cdot\mathsf{term}(g) = a(\eta(t + \mathsf{term}(h)) + t) = a(t + \mathsf{term}(h)) = a\cdot\mathsf{term}(h)$, for $a{\in}C$. If $h = \mathbb{0}$, then $\mathsf{term}(g) = \eta\cdot\mathsf{term}(h)$ and H1 yields $a\cdot\mathsf{term}(g) = a\cdot\mathsf{term}(h)$ for $a{\in}C$.

Thus we have proved the claim. Now we return to the proof of the proposition, part (i).
For reasons of symmetry also one of the following statements must hold:
A: $g \neq \mathbb{0}$ and $\mathsf{term}(g) = \eta\cdot\mathsf{term}(h) + \mathsf{term}(g)$
B: $g,h \neq \mathbb{0}$ and $\mathsf{term}(g) = \mathsf{term}(h) + \mathsf{term}(g)$
C: $a\cdot\mathsf{term}(g) = a\cdot\mathsf{term}(h)$, for $a{\in}C$.
Now the remainder of the proof consists of a simple case distinction.
• Suppose that I and A hold. From I it follows that for $a{\in}C$
$a\cdot\mathsf{term}(h) = a(\eta\cdot\mathsf{term}(g) + \mathsf{term}(h))$ = (using H3, and H1 in case $g = \mathbb{0}$)
$= a(\eta\cdot\mathsf{term}(g) + \mathsf{term}(h)) + a\cdot\mathsf{term}(g) = a\cdot\mathsf{term}(h) + a\cdot\mathsf{term}(g)$.

Likewise, A implies $a\cdot\text{term}(g) = a\cdot\text{term}(g) + a\cdot\text{term}(h)$. Putting these two statements together yields $a\cdot\text{term}(g) = a\cdot\text{term}(h)$.

- Suppose that II and B hold. Then $\text{term}(g) = \text{term}(h) + \text{term}(g) = \text{term}(h)$, so $a\cdot\text{term}(g) = a\cdot\text{term}(h)$ for $a\in C$.
- Suppose that II and A hold. Then, for $a\in C$, $a\cdot\text{term}(g) = a(\eta(\text{term}(g) + \text{term}(h)) + \text{term}(g)) = a\cdot\text{term}(h)$, using H2.
- Likewise the case that I and B hold.
- If III or C hold there is nothing left to show.

This finishes the proof of part (i).

For part (ii), suppose $g,h \neq 0$ and $g \underline{\leftrightarrow}_{r\eta}\ h$, say $R: g \underline{\leftrightarrow}_{r\eta}\ h$. We will prove that any summand as' or a of $\text{term}(g)$ is an ACP_η-summand of $\text{term}(h)$ (and vice versa), which yields the desired result. So let $s = as'$ or $s = a$ be a summand of $\text{term}(g)$. By proposition 4.16 there is an edge $\text{root}(g) \xrightarrow{a} p$ in g with $a\cdot\text{term}((g)_p) = s$.
<u>Case 1:</u> $a\in A$. Since $R: g \underline{\leftrightarrow}_{r\eta}\ h$, there must be nodes q^* and q in h with $\text{root}(h) \twoheadrightarrow^{\eta} q^* \xrightarrow{a} \twoheadrightarrow^{\eta} q$ $R(\text{root}(g),q^*)$ and $R(p,q)$. Moreover, the rootedness condition gives $\text{root}(h) = q^*$. By proposition 4.15, $(g)_p \underline{\leftrightarrow}_{\eta}\ (h)_q$, and (i) yields $a\cdot\text{term}((g)_p) = a\cdot\text{term}((h)_q)$. Furthermore, proposition 4.17 gives $\text{term}(h) = a\cdot\text{term}((h)_q) + \text{term}(h)$, so $\text{term}(h) = s + \text{term}(h)$.
<u>Case 2:</u> $a=\eta$. Since $R: g \underline{\leftrightarrow}_{r\eta}\ h$, there must be a node q in h with $\text{root}(h) \twoheadrightarrow^{\eta} q$ and $R(p,q)$. Moreover, the rootedness condition gives $\text{root}(h) \xrightarrow{\eta} \twoheadrightarrow^{\eta} q$. By proposition 4.15, $(g)_p \underline{\leftrightarrow}_{\eta}\ (h)_q$, and (i) yields $\eta\cdot\text{term}((g)_p) = \eta\cdot\text{term}((h)_q)$. Furthermore, proposition 4.17 gives $\text{term}(h) = \eta\cdot\text{term}((h)_q) + \text{term}(h)$, so $\text{term}(h) = s + \text{term}(h)$.
<u>Case 3</u>, $a=\delta$, follows from A6 and A7.
This finishes the proof of part (ii).

4.19 <u>Theorem:</u> ACP_η is sound and complete for closed terms, with respect to $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$, i.e. for all closed ACP_η-terms t,s we have: $\text{graph}(t) \underline{\leftrightarrow}_{r\eta}\ \text{graph}(s) \Leftrightarrow \text{ACP}_\eta \vdash t=s$.
<u>Proof:</u> Direction $\Leftarrow$, the soundness, follows from theorem 4.9.
For direction $\Rightarrow$, the completeness, note that the elimination theorem 3.17.1 and direction $\Leftarrow$ imply that it is enough to prove $\Rightarrow$ for basic terms t,s. This amounts to an application of 4.18.ii, using definition 4.13: if $t,s \in \text{BT}$ and $\text{graph}(t) \underline{\leftrightarrow}_{r\eta}\ \text{graph}(s)$, then
$t \equiv \text{term}(\text{graph}(t)) = \text{term}(\text{graph}(s)) \equiv s$.

4.20 <u>Theorem:</u> $\text{BPA}_{\eta\delta}$ is sound and complete for closed terms, with respect to $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$, i.e. for all closed $\text{BPA}_{\eta\delta}$-terms t,s we have: $\text{graph}(t) \underline{\leftrightarrow}_{r\eta}\ \text{graph}(s) \Leftrightarrow \text{BPA}_{\eta\delta} \vdash t=s$.
<u>Proof:</u> Subsections 4.16 through 4.19 remain valid if every occurrence of ACP_η is replaced by $\text{BPA}_{\eta\delta}$ and $x=y$ stands for $\text{BPA}_{\eta\delta} \vdash x=y$ instead of $\text{ACP}_\eta \vdash x=y$.

4.21 <u>Theorem:</u> ACP is sound and complete for closed terms, with respect to $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$, i.e. for all closed ACP-terms t,s we have: $\text{graph}(t) \underline{\leftrightarrow}_{r\eta}\ \text{graph}(s) \Leftrightarrow \text{ACP} \vdash t=s$.
<u>Proof:</u> As above for ACP_η, but much simpler, by removing all η's from this chapter.

4.22 Theorem: ACP_η is a **conservative extension** of $BPA_{\delta\eta}$ and of ACP, i.e. for all closed $BPA_{\delta\eta}$-terms t,s we have

$$ACP_\eta \vdash t{=}s \quad \text{iff} \quad BPA_{\delta\eta} \vdash t{=}s,$$

and for all closed ACP-terms t,s we have

$$ACP_\eta \vdash t{=}s \quad \text{iff} \quad ACP \vdash t{=}s.$$

Proof: Combine 4.19 with 4.20 and with 4.21.

5. Recursive specifications and fairness.

Most processes encountered in practice cannot be represented by a closed term, by an element of the initial algebra of ACP_η, but will be specified recursively. Therefore, the model presented in the previous section also contains infinite processes, processes that can perform infinitely many actions consecutively. The algebraic way to represent such processes is by means of recursive specifications. First, we develop some terminology.

5.1 Definition. A **recursive specification** over ACP_η is a set of equations $\{x = t_x : x \in X\}$, with X a set of variables, and t_x a term over ACP_η and variables X. No other variables may occur in t_x. There is exactly one equation $x{=}t_x$ for each variable x. X contains one designated variable, called the **root variable.**
A process p (in a certain model of ACP_η) is a **solution** of the recursive specification E if substituting p for the root variable of E, and substituting other elements of the model for the other variables of E, yields a set of statements that hold in the model.

5.2 Definition. i. Let t be a term over ACP_η, and x a variable in t. We call the occurrence of x in t **guarded** if x is *preceded* by an atomic action, i.e. t has a subterm of the form $a \cdot s$, with $a \in A$, and this x occurs in s. Otherwise, we call x **unguarded.**
ii. A recursive specification $\{x{=}t_x : x \in X\}$ is **guarded** if no t_x contains an abstraction operator η_I, and each occurrence of a variable in each t_x is guarded.

5.3 Remark: We will establish in the sequel that each guarded recursive specification has a unique solution in the graph model of section 4. We see that the constant η cannot be guard, since the equation $x = \eta x$ has infinitely many solutions (for each process p, ηp is a solution).
A definition of guardedness involving abstraction operators η_I is very complicated. Therefore, we limit ourselves to the case where no abstraction operators appear in a recursive specification. Of course, we can apply abstraction to a process that has been defined by means of a guarded recursive specification.

5.4 Definition. We formulate two principles that will be shown to hold in the model $\mathbb{G}/{\underline{\leftrightarrow}}_{r\eta}$ of section 4. First, the **Recursive Definition Principle (RDP)** is the assumption that every guarded recursive specification has at least one solution. Second, the **Recursive Specification**

Principle (RSP) is the assumption that every guarded recursive specification has at most one solution.

5.5 In order to show that RDP and RSP hold in $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ we need some auxiliary notions, that may also be interesting in their own right. First, we define the **projections** of a process. To that end, we enlarge the signature of ACP$_\eta$ with unary operators π_n, for $n \in \mathbb{N}$. Then we add the following axioms PR (using the inductive structure of basic terms from 3.16; $a \in A \cup \{\delta\}$).

$$\begin{aligned}
&\pi_n(\eta) = \eta \\
&\pi_n(\eta x) = \eta \cdot \pi_n(x) \\
&\pi_0(ax) = \delta \\
&\pi_{n+1}(ax) = a \cdot \pi_n(x) \\
&\pi_n(x + y) = \pi_n(x) + \pi_n(y)
\end{aligned}$$

Table 4. Projection.

We see that the operator π_n cuts off the process after it has executed n visible steps; the remaining visible steps are replaced by δ.

5.6 Lemma: Suppose process p is a solution of the guarded recursive specification E. Then for each n, $\pi_n(p)$ is equal to a closed ACP$_\eta$-term (independent of p).
Proof: This follows easily from the definition of guardedness. For details, see BAETEN, BERGSTRA & KLOP [1].

5.7 Definition. Let p be a process (in a certain model of ACP$_\eta$). We say p has **bounded nondeterminism** if for each sequence σ of atomic actions, there are only finitely many different processes to which p can evolve by performing σ. For more details concerning this notion, and an axiomatisation of it, see VAN GLABBEEK [8].

5.8 Theorem: Let E be a guarded recursive specification. Then, in $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$, E has a solution that has bounded nondeterminism. Thus, $\mathbb{G}/\underline{\leftrightarrow}_{r\eta} \models$ RDP.
Proof: We build up such a solution in stages, using the finite projections that we can calculate from E (by the proof of lemma 5.6). It is easy to see that the graph we obtain is finitely branching, and no infinite sequence of η-steps can occur. This is enough to conclude that this graph has bounded nondeterminism. For more details, see BAETEN, BERGSTRA & KLOP [2].

5.9 Finally, we need the **Approximation Induction Principle (AIP$^-$)** which says that a process that has bounded nondeterminism, is completely determined by its finite projections, i.e. if p has bounded nondeterminism, and q is such that for all n $\pi_n(p) = \pi_n(q)$, then $p = q$.
(The "-" refers to a version of AIP without the restriction to bounded processes.)

5.10 Theorem: AIP^- holds in $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$.
Proof: See VAN GLABBEEK [8] or BAETEN, BERGSTRA & KLOP [2].

5.11 Theorem: RSP holds in $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$.
Proof: 5.8 plus 5.10.

5.12 Thus, we have established that every guarded recursive specification has a unique solution in the graph model. On the other hand, it is not hard to show that each element of the graph model can be given by a guarded recursive specification (maybe after applying abstraction first). (For the proof, associate a variable to each node of the tree, and then the equation for node x enumerates the outgoing edges of x; in case an infinite sequence of η-steps occurs, first rename them into a fresh atom h, and then apply $\eta_{\{h\}}$ to the solution of the specification.)
In BAETEN, BERGSTRA & KLOP [2], it is even shown that each computable element of $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ can be obtained from a *finite* guarded recursive specification.

5.13 Definitions. i. A process p is **definable** if it can be obtained from the constants C by means of guarded recursion and the operators of ACP_η.
ii. A process p can be written in **head normal form** if there is an $n>0$, constants $a_1,...,a_n \in C$ and processes $p_1,...,p_n$ such that $p = \Sigma_{i\leq n}\, a_i p_i$.

5.14 Proposition: Each definable process can be written in head normal form.
Proof: Straightforward.

Note that all elements of $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ are definable. As an application of proposition 5.14, we prove the following proposition.

5.15 Proposition: Let p,q be definable processes. Then $p\|q = q\|p$.
Proof: We saw in the proof of 3.15 that $\eta x\|\eta y = \eta y\|\eta x$ for all processes x,y.
It follows that $ax\,|\,by = a\eta x\,|\,b\eta y = (a\,|\,b)(\eta x\|\eta y) = (b\,|\,a)(\eta y\|\eta x) = by\,|\,ax$ for all x,y and all $a,b \in C$. Since $|$ distributes over $+$, we get that $|$ is commutative for all head normal forms. Thus $p\,|\,q = q\,|\,p$, and it is easy to deduce $p\|q = q\|p$.

5.16 Next, we briefly discuss the issue of *fairness*. As a motivation, we first consider an example (from VAANDRAGER [13]).
Example: A statistician performs a simple experiment: he tosses a coin until tail comes up; then he goes to report success. The behaviour of the statistician is given by the recursive equation

$$S = \mathit{head}\cdot S + \mathit{tail}\cdot \mathit{success}.$$

For an observer that cannot see the coin, the actions *head*, *tail* are hidden, so he observes the process $\eta_I(S)$, with $I = \{\mathit{head}, \mathit{tail}\}$. If we assume that the coin is fair, S will perform a *tail* action sooner or later, which yields the identity

$$\eta_I(S) = \eta\cdot \mathit{success}.$$

What is needed is an algebraic framework in which one can prove this equation.
Looking in the graph model, we see indeed that there is a rooted η-bisimulation between the processes in fig. 8.

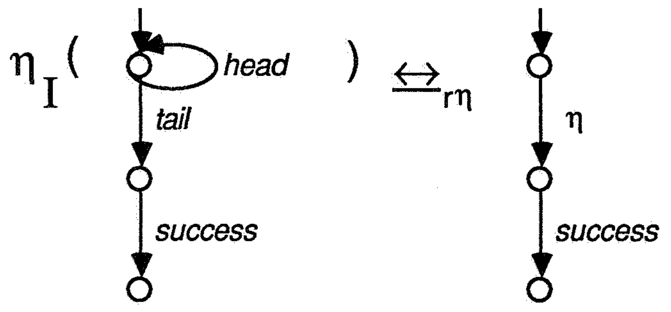

Fig. 8.

The algebraical rule that expresses that a process will not perform an infinite sequence of internal steps, but will perform an external step eventually (if possible) is the η **Abstraction Rule (HAR):**

$$\frac{}{\begin{array}{l}\text{if } x = ix + y, \text{ and } i \in I,\\ \text{then } \eta_I(x) = \eta\cdot\eta_I(y) + \eta_I(y)\end{array}} \quad \text{HAR}$$

The process x will not remain in the i-loop, but will either exit immediately (start executing $\eta_I(y)$) or exit after some internal action (execute $\eta\cdot\eta_I(y)$).
Applied to the example above, we obtain

$$\eta_I(S) = \eta\cdot\eta_I(\mathit{tail}\cdot\mathit{success}) + \eta_I(\mathit{tail}\cdot\mathit{success}) =$$
$$= \eta\cdot\eta\cdot\mathit{success} + \eta\cdot\mathit{success} = \eta\cdot\mathit{success}.$$

A particular consequence of HAR should be mentioned: if we define the process x by the recursive equation $x = ix$, then $\eta_{\{i\}}(x)$ is the process that only performs an infinite sequence of internal steps, a situation that is often called *livelock*. Since $x = ix = ix + \delta$, an application of HAR yields

$$\eta_{\{i\}}(x) = \eta\cdot\eta_{\{i\}}(\delta) + \eta_{\{i\}}(\delta) = \eta\delta + \delta = \eta\delta.$$

This equation we can call *livelock = deadlock.*
HAR can be generalized to the case where we have a loop of internal steps of length $n>1$; this gives us the following rules HAR_n. Note that HAR_1 is just HAR.

$$\frac{\forall k\in\mathbb{Z}_n \quad x_k = i_k x_{k+1} + y_k, \quad i_k \in I}{\eta_I(x_0) = \eta_I(y_0) + \eta\cdot \sum_{k\in\mathbb{Z}_n} \eta_I(y_k).} \quad \text{HAR}_n$$

We conjecture that, with the use of renaming operators, the rules HAR_n (and even more complicated generalizations, dealing with *clusters* of internal steps) can be derived from HAR. For a similar sitation with the rules KFAR and KFAR_n, see VAANDRAGER [13].
We claim that the rule HAR (and its generalization) is well-suited to deal with fairness considerations in the verification of communication protocols, as in VAANDRAGER [13] and other papers.

6. Relations with τ.

6.1 This paper is called *another* look at abstraction in process algebra, because a different abstraction mechanism has already been in use in process algebra for some time, starting with BERGSTRA & KLOP [4]. This abstraction mechanism is based on Milner's silent step τ.
The three laws of the constant τ are from MILNER [10], and are presented in table 5 below.

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

Table 5. τ-laws.

The crucial difference with the η-laws presented in section 3 is the second law: the second τ-law implies the second η-law, since $\tau(x + y) = \tau(x + y) + (x + y) = \tau(x + y) + x + y + x = \tau(x + y) + x$, but not the other way around, as example 4.4.ix illustrates.
We can motivate the second τ-law by reconsidering the action relations of 3.5. If we change their meaning to:
$x \rightarrow^a y$ means that process x can evolve into process y, during a period in which a starts;
$x \rightarrow^a \surd$ means that process x can terminate (successfully), after performing an a-step,
then we have for τ the same definition as the one in 3.5 for the η, with two extra clauses:
9. if $x \rightarrow^\tau y$ and $y \rightarrow^a z$, then $x \rightarrow^a z$
10. if $x \rightarrow^\tau y$ and $y \rightarrow^a \surd$, then $x \rightarrow^a \surd$
(see VAN GLABBEEK [8]). It can be argued that the presence of these clauses makes the τ less operational than the η. It turns out that the τ-laws give a complete axiomatisation for these modified action relations, i.e. for all closed BPA_τ-terms t,s we have $t \leftrightarrow s \Leftrightarrow BPA_\tau \vdash t=s$. In this philosophy, τ and η denote the same process, but in ACP_η, more subtle differences between processes are observable.
A different motivation for τ is along the lines of 3.1: when the machine is executing an internal step τ, it is running for a period of time, which can also have no duration; therefore, we can consider η to stand for 1 or more machine-steps, and τ to stand for 0 or more machine-steps (clauses 9 and 10 above can also be used with this motivation, if we keep the original meaning of predicates $\rightarrow^a$). This philosophy does not mesh nicely with the abstraction operator τ_I: it now abstracts an atomic action of some duration to a process which might have no duration.

The difference between τ and η has far-reaching consequences, of which we will mention a few in the sequel. The first of these differences is that not all laws of ACP, that hold for all atomic actions a, also hold for τ. For if $\gamma(a,b)$ is defined, we obtain

$$\tau a \mid b = (\tau a + a) \mid b = \tau a \mid b + a \mid b = \tau a \mid b + \gamma(a,b),$$

so $\tau a \mid b$ contains a summand $\gamma(a,b)$, contrary to the situation with η. Thus, in order to axiomatise the theory ACP_τ, the relation of τ and merge had to be explicitly defined, which necessitated careful deliberations. The result was the axiom system ACP_τ, presented in table 6. There $a,b \in A \cup \{\delta\}$, $H,I \subseteq A$ and x,y,z are arbitrary processes.

$x + y = y + x$		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			
$a \mid b = \gamma(a,b)$	if $\gamma(a,b)\downarrow$	CF1			
$a \mid b = \delta$	otherwise	CF2			
$x \| y = x ⌊⌊ y + y ⌊⌊ x + x \mid y$		CM1			
$a ⌊⌊ x = ax$		CM2	$\tau ⌊⌊ x = \tau x$		TM1
$ax ⌊⌊ y = a(x \| y)$		CM3	$\tau x ⌊⌊ y = \tau(x \| y)$		TM2
$(x + y) ⌊⌊ z = x ⌊⌊ z + y ⌊⌊ z$		CM4	$\tau \mid x = \delta$		TC1
$a \mid bx = (a \mid b)x$		CM5	$x \mid \tau = \delta$		TC2
$ax \mid b = (a \mid b)x$		CM6	$\tau x \mid y = x \mid y$		TC3
$ax \mid by = (a \mid b)(x \| y)$		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 6. ACP_τ.

6.2 The theory ACP_η, as developed in this paper, has nicer technical properties than the theory ACP_τ. For instance, the proofs of theorem 3.17 (Elimination Theorem) and propositions 3.18 (laws of Standard Concurrency) and 3.20 (Expansion Theorem) become more cumbersome (see BERGSTRA & KLOP [4]). Also, not every definable process can be written in head normal form, and the set of all finitely branching process graphs cannot be made into a model for ACP_τ, because it is not closed under the communication merge (see BAETEN, BERGSTRA & KLOP [2]).
However, the authors do not favor one theory over the other, and feel they are not in competition. Depending on the particular application, one theory may be more suited than the other, and the theories can even be applied one after the other (the τ represents a *further* abstraction than the η; in the sequel we will define a mapping $\tau_{\{\eta\}}$ that abstracts more, by renaming η into τ), or even

simultaneously (in the sequel we *define* τ in ACP$_\eta$, at least in prefix position).

6.3 Definition. We will define a mapping $\tau_{\{\eta\}}$ from ACP$_\eta$-processes to ACP$_\tau$-processes on the graph model $\mathbb{G}$. First some notes on the graph model of ACP$_\tau$.
The graph model of countably branching process graphs modulo rooted τ-bisimulation for ACP$_\tau$ is defined in BAETEN, BERGSTRA & KLOP [2]. We denote this model by $\mathbb{G}_{\aleph_1}/\underline{\leftrightarrow}_{r\tau}$. The definition of rooted τ-bisimulation is very similar to the notion of rooted η-bisimulation; the differences are (using the notation of 4.3) that in 2 we do not require R(s,t*), in 3 we do not require R(s*,t), and we drop the root condition in point 2 and 3 (but not in 4 and 5). For the proof that ACP$_\tau$ is sound and complete for closed terms w.r.t. $\mathbb{G}_{\aleph_1}/\underline{\leftrightarrow}_{r\tau}$, we also refer to [2]. The definition of the operators on $\mathbb{G}_{\aleph_1}/\underline{\leftrightarrow}_{r\tau}$ is the same as on $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$, except for the definition of the communication merge (the set of *finitely* branching process graphs cannot be made into a model for ACP$_\tau$, because it is not closed under this communication merge).
Then, the mapping $\tau_{\{\eta\}}$ simply changes all η-labels of a graph to τ-labels. Since this is a renaming operator, it is on closed terms completely axiomatised by the equations in table 7 below.
There $a \in A \cup \{\delta\}$.

$$\tau_{\{\eta\}}(a) = a$$
$$\tau_{\{\eta\}}(\eta) = \tau$$
$$\tau_{\{\eta\}}(x + y) = \tau_{\{\eta\}}(x) + \tau_{\{\eta\}}(y)$$
$$\tau_{\{\eta\}}(xy) = \tau_{\{\eta\}}(x) \cdot \tau_{\{\eta\}}(y)$$

Table 7. η-to-τ-abstraction (HT)

It is immediate that the following lemma holds for all process graphs not containing a label η or τ. Using the axioms HT above, we give the algebraic proof for closed terms.

6.4 Lemma: Let x be a closed ACP-term and let $I \subseteq A$. Then $\vdash \tau_{\{\eta\}}(\eta_I(x)) = \tau_I(x)$.
Proof: Write x as a basic term. Then we can use induction on the structure of x.
Case 1: x is an atomic action or δ, so $x \in A \cup \{\delta\}$. If $x \in I$, $\tau_{\{\eta\}}(\eta_I(x)) = \tau_{\{\eta\}}(\eta) = \tau = \tau_I(x)$; if $x \notin I$, $\tau_{\{\eta\}}(\eta_I(x)) = \tau_{\{\eta\}}(x) = x = \tau_I(x)$.
Case 2: Otherwise, x is of the form $y \cdot z$ or $y+z$, and the lemma holds for y and z. Write * for $\cdot$ or +. Then $\tau_{\{\eta\}}(\eta_I(x)) = \tau_{\{\eta\}}(\eta_I(y) * \eta_I(z)) = \tau_{\{\eta\}}(\eta_I(y)) * \tau_{\{\eta\}}(\eta_I(z)) = \tau_I(y) * \tau_I(z) = \tau_I(x)$.

6.5 Proposition: The mapping $\tau_{\{\eta\}}$ is a homomorphism on closed terms, w.r.t. the operators $+, \cdot, \|, \lfloor\!\lfloor, \partial_H$.
Proof: The proof is only non-trivial for the case of $\|$. Thus, let x,y be closed ACP$_\eta$-terms. We have to prove that ACP$_\tau$ $\vdash \tau_{\{\eta\}}(x\|y) = \tau_{\{\eta\}}(x)\|\tau_{\{\eta\}}(y)$. We use induction on basic terms (see 3.16). We consider four cases:
Case 1: $x=\eta$ and $y=\eta$. Then $\tau_{\{\eta\}}(x\|y) = \tau_{\{\eta\}}(\eta) = \tau = \tau\|\tau = \tau_{\{\eta\}}(x)\|\tau_{\{\eta\}}(y)$.
Case 2: $x = \Sigma_{i<n}\, a_i x_i + \Sigma_{j<m}\, \eta x'_j$ and $y = \eta$ $(n+m>0,\ a_i \in A \cup \{\delta\})$. Then $\tau_{\{\eta\}}(x\|y) =$

$= \Sigma_i\, a_i\cdot\tau_{\{\eta\}}(x_i \| y) + \Sigma_j\, \tau\cdot\tau_{\{\eta\}}(x'_j \| y) + \tau\cdot\tau_{\{\eta\}}(x) + \delta = \Sigma_i\, a_i(\tau_{\{\eta\}}(x_i) \| \tau_{\{\eta\}}(y)) +$
$+ \Sigma_j\, \tau(\tau_{\{\eta\}}(x'_j) \| \tau_{\{\eta\}}(y)) + \tau \mathbin{|\!\lfloor} \tau_{\{\eta\}}(x) + \tau \mid \tau_{\{\eta\}}(x)$ (by induction hypothesis) $= \tau_{\{\eta\}}(x) \| \tau_{\{\eta\}}(y)$.

Case 3: $x = \eta$ and $y = \Sigma_{k<p}\, b_k y_k + \Sigma_{l<q}\, \eta y'_l$ ($p+q>0$, $b_k \in A\cup\{\delta\}$). Like case 2.

Case 4: $x = \Sigma_{i<n}\, a_i x_i + \Sigma_{j<m}\, \eta x'_j$ and $y = \Sigma_{k<p}\, b_k y_k + \Sigma_{l<q}\, \eta y'_l$ ($m+n>0$, $p+q>0$, $a_i, b_k \in A\cup\{\delta\}$).

Note that as a consequence of the second τ-law we have the following equation for all processes t,s: $\quad \tau(t\|s) = \tau(t\|s) + t\,|\,s$.

We get $\tau_{\{\eta\}}(x\|y) =$

$= \Sigma_i\, a_i\cdot\tau_{\{\eta\}}(x_i\|y) + \Sigma_j\, \tau\cdot\tau_{\{\eta\}}(x'_j\|y) + \Sigma_k\, b_k\cdot\tau_{\{\eta\}}(x\|y_k) + \Sigma_l\, \tau\cdot\tau_{\{\eta\}}(x\|y'_l) +$
$+ \Sigma_{i,k}\, (a_i\,|\,b_k)\cdot\tau_{\{\eta\}}(x_i\|y_k) =$
$= \Sigma_i\, a_i(\tau_{\{\eta\}}(x_i)\|\tau_{\{\eta\}}(y)) + \Sigma_j\, \tau(\tau_{\{\eta\}}(x'_j)\|\tau_{\{\eta\}}(y)) + \Sigma_k\, b_k(\tau_{\{\eta\}}(x)\|\tau_{\{\eta\}}(y_k)) +$
$+ \Sigma_l\, \tau(\tau_{\{\eta\}}(x)\|\tau_{\{\eta\}}(y'_l)) + \Sigma_{i,k}\, (a_i\,|\,b_k)(\tau_{\{\eta\}}(x_i)\|\tau_{\{\eta\}}(y_k))$

(by induction hypothesis). We can add to this last expression the following terms, as noted above: $\Sigma_j\, \tau_{\{\eta\}}(x'_j)\,|\,\tau_{\{\eta\}}(y) + \Sigma_l\, \tau_{\{\eta\}}(x)\,|\,\tau_{\{\eta\}}(y'_l)$. Straightforward calculations show that the result is equal to $\tau_{\{\eta\}}(x)\|\tau_{\{\eta\}}(y)$. This finishes the proof.

6.6 Note: The mapping $\tau_{\{\eta\}}$ is *not* a homomorphism w.r.t. $|$. If e.g. $\gamma(a,b)$ is defined, then $\tau a\,|\,b = \gamma(a,b)$, while $\eta a\,|\,b = \delta$.

It is not hard to see that proposition 6.5 also holds in the graph model, since the operators $+,\cdot,\|,\mathbin{|\!\lfloor},\partial_H$ have the same definition in both cases. We will usually assume that propositions 6.4 and 6.5 hold for all processes (similar to the situation with the equations of Standard Concurrency, see note 3.19).

6.7 Theorem. Let g be a finitely branching process graph with labels from $A\cup\{\delta,\eta\}$, and let h be a finitely branching process graph with labels from $A\cup\{\delta\}$, such that $\tau_{\{\eta\}}(g) \leftrightarrow_{r\tau} h$.
Then $g \leftrightarrow_{r\eta} h$.

Proof: Let g,h be as stated. $\tau_{\{\eta\}}(g)$ is obtained from g by changing all η-labels to τ-labels, but has the same nodes. Let R be a rooted τ-bisimulation between $\tau_{\{\eta\}}(g)$ and h. We claim that R is also a rooted η-bisimulation between g and h. We see that it is enough to check the extra conditions of the rooted η-bisimulation.

1. Let $R(s,t)$ and let $s \rightarrow^a s'$ be an edge in g with $a\in A$. Since R is a rooted τ-bisimulation, there are nodes t^*,t' in h such that $t \twoheadrightarrow^\tau t^* \rightarrow^a \twoheadrightarrow^\tau t'$ and $R(s',t')$. But h contains no τ-labels, so the number of τ-steps in $\twoheadrightarrow^\tau$ must be 0, and $t^* = t$. This means $R(s,t^*)$ holds.

The root condition is also taken care of, since $t^*=t$.

2. Vice versa: let $R(s,t)$ and let $t \rightarrow^a t'$ be an edge in h with $a\in A$. Since R is a rooted τ-bisimulation, there are nodes s^*,s' in g such that $s \twoheadrightarrow^\tau s^* \rightarrow^a \twoheadrightarrow^\tau s'$ and $R(s',t')$. If the number of τ-steps in $s \twoheadrightarrow^\tau s^*$ is 0, we are done. Otherwise, consider the first τ-step $s \rightarrow^\tau s''$. Since R is a rooted τ-bisimulation, there is a node t'' in h with $t \twoheadrightarrow^\tau t''$ and $R(s'',t'')$. But h contains no τ-labels, so $t = t''$ and $R(s'',t)$. By repeating this procedure, we obtain $R(s^*,t)$.

If (s,t) is the pair of roots, we must have that the number of τ-steps in $s \twoheadrightarrow^\tau s^*$ is 0. Otherwise, $s \rightarrow^\tau s''$ for some node s'' in g, and since R is a rooted τ-bisimulation, there is a node t'' in h with t

$\twoheadrightarrow^{\tau}$ t" and R(s",t"). But by the root condition of the rooted τ-bisimulation, t" can be chosen such that the number of τ-steps in t $\twoheadrightarrow^{\tau}$ t" is at least 1. This is a contradiction, since h contains no τ-labels.

6.8 Theorem 6.7 allows us to formulate the following proof principle:

if x is an ACP_{η}-process and y is an ACP-process,
and $\tau_{\{\eta\}}(x) = y$,
then x = y.

We will call this the **Two-tiered Abstraction Principle (TAP).** It follows from the completeness of the graph model that TAP is derivable for closed terms. In the following example we apply this proof principle.

6.9 Example: A *bag* (a channel that does not preserve the order of its contents) with input port i and output port j is given by the following guarded recursive equation:

$$B^{ij} = \Sigma_{d \in D}\, ri(d) \cdot (B^{ij} \| sj(d)).$$

Here D is a finite set of data, ri(d) is the atomic action *receive* datum d at port i, and sj(d) is the atomic action *send* datum d at port j. For more information, see BAETEN, BERGSTRA & KLOP [1]. Now we connect two bags in series. If we abstract from the communications between the bags, the resulting process should again be a bag. We express this as follows.
In fig. 9, we have bags B^{12} and B^{23}. Communications between them are given by defining

$$\gamma(r2(d),s2(d)) = \gamma(s2(d),r2(d)) = c2(d)$$

(*communicate* d at 2). γ is undefined in all other cases. When we merge these bags, we have to encapsulate unsuccessful communications, actions from H = {r2(d), s2(d) : d∈ D}, and abstract from internal steps, actions from I = {c2(d) : d∈ D}. With these definitions, we have the following theorem.

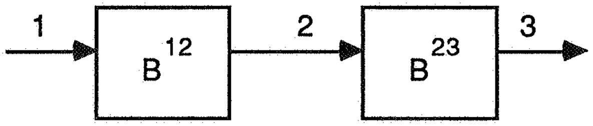

Fig. 9.

6.10 Theorem: $\eta_I \circ \partial_H(B^{12} \| B^{23}) = B^{13}$.
Proof: In BAETEN, BERGSTRA & KLOP [1] it is proved that

$$\tau_I \circ \partial_H(B^{12} \| B^{23}) = B^{13}.$$

By 6.4, we have $\tau_I \circ \partial_H(B^{12} \| B^{23}) = \tau_{\{\eta\}} \circ \eta_I \circ \partial_H(B^{12} \| B^{23})$. An application of the Two-tiered Abstraction Principle finishes the proof.

6.11 Thus, we can see that we can consider ACP_{τ} to be a homomorphic image of ACP_{η} (viewing | as a hidden operator). (Note that this mapping cannot be an isomorphism, as example 4.4.ix shows.)
When verifying statements about processes, we often use abstraction from internal behaviour. Now

it is possible to use the operator η_I, and the theory ACP_η, to implement this abstraction. Then later, if we want to abstract further, if we want to identify more processes, we can apply the operator $\tau_{\{\eta\}}$ and use the theory ACP_τ to see if the obtained expression can be simplified further. Of course, having done that, it is still possible to add more identifications, for instance the laws of failure semantics (see BROOKES, HOARE & ROSCOE [7] or BERGSTRA, KLOP & OLDEROG [6]).
In the sequel we will consider a different way to use the constants τ and η together, by defining (part of) τ in ACP_η. Where the approach with the operator $\tau_{\{\eta\}}$ is closest to the first motivation given in 6.1, the following approach is more consistent with the second motivation in 6.1.

6.12 Definition: Enrich the signature of ACP_η with a unary operator τ and add the following law:

$$\tau(x) = \eta x + x \qquad \text{TH.}$$

We call the resulting theory $ACP_{\eta\tau}$.

6.13 Note: An immediate consequence of law A4 is that $\tau(x)\cdot y = \tau(x\cdot y)$. This allows to write $\tau\cdot x$ instead of $\tau(x)$.

6.14 Lemma: The following equations hold in the theory $ACP_{\eta\tau}$ ($a \in C$):
1. $a\tau x = ax$ 2. $\tau\tau x = \tau x$
3. $\tau\eta = \eta$ 4. $\tau\delta = \eta\delta$
5. $\tau x + x = \tau x$
6. $a(\tau x + y) = a(\tau x + y) + ax$
7. $\tau x \mathbin{|\!\lfloor} y = \eta(x \| y) + x \mathbin{|\!\lfloor} y$
Proof: Easy.

6.15 Lemma: The following equations hold for all closed $ACP_{\eta\tau}$-terms x,y:
1. $x\tau y = xy$
2. $\tau x \mid y = x \mid y = x \mid \tau y$
3. $x \mathbin{|\!\lfloor} \tau y = x \mathbin{|\!\lfloor} \eta y = x \mathbin{|\!\lfloor} y$
4. $\tau x \| y = \tau(x \| y)$.
Proof: First note that each closed $ACP_{\eta\tau}$-term is again equal to a basic term (the operator τ can easily be eliminated). Therefore, we only have to prove the lemma for basic terms x,y. This proof proceeds by an induction on the structure of x and y, and is easy to complete.
Remark: Statements 2-4 are even provable for all definable processes.

6.16 We have seen that we can introduce τ in the theory ACP_η as a unary operator, so that we can simulate prefix multiplication by τ. In prefix position, this τ obeys all laws of the τ in ACP_τ (at least for closed terms), *except* for the law for left-merge. Strangely enough, obtaining τ from ACP_η by means of the mapping $\tau_{\{\eta\}}$ respects all operators except the auxiliary operator $|$, and obtaining τ in ACP_η as a unary operator respects all operators except the auxiliary operator $\mathbin{|\!\lfloor}$. Thus, viewed with the greater discriminating power of the η, we see that the laws used in ACP_τ to define the relation of τ and merge, generate friction amongst eachother, so they cannot be united with the laws for η.

In both cases though, we do get the same laws for τ and merge.

6.17 Remark: It still can be viewed as a drawback that we were not able to define a *process* τ in the theory ACP_η. We can do this, however, with the use of another constant, namely the empty process ε discussed in VRANCKEN [14]. The constant ε has the characteristic laws

$$\varepsilon x = x\varepsilon = x.$$

We see that ε is the process that terminates immediately (stands for zero machine-steps), and we can formulate the following definition:

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

Thus, τ becomes definable in a theory ACP_η with ε, and in this theory, we can also define a mapping $\tau_{\{\eta\}}$, renaming η into τ, so that $\tau_I = \tau_{\{\eta\}} \circ \eta_I$.

References

[1] J.C.M.BAETEN, J.A.BERGSTRA & J.W.KLOP, *Conditional axioms and α/β-calculus in process algebra,* report CS-R8502, Centre for Math. & Comp. Sci., Amsterdam 1985, to appear in: Proc. IFIP Conf. on Formal Description of Progr. Concepts (M.Wirsing, ed.), Gl. Avernæs 1986, North-Holland.

[2] J.C.M.BAETEN, J.A.BERGSTRA & J.W.KLOP, *On the consistency of Koomen's fair abstraction rule,* report CS-R8511, Centre for Math. & Comp. Sci., Amsterdam 1985, to appear in Theor. Comp. Sci.

[3] J.A.BERGSTRA & J.W.KLOP, *Process algebra for synchronous communication,* Inf. & Control 60 (1/3), pp. 109 - 137, 1984.

[4] J.A.BERGSTRA & J.W.KLOP, *Algebra of communicating processes with abstraction,* Theor. Comp. Sci. 37 (1), pp. 77 - 121, 1985.

[5] J.A.BERGSTRA & J.W.KLOP, *Algebra of communicating processes,* Proc. CWI Symp. Math. & Comp. Sci. (J.W.de Bakker, M.Hazewinkel & J.K.Lenstra, eds.), pp. 89 - 138, North-Holland, 1986.

[6] J.A.BERGSTRA, J.W.KLOP & E.-R. OLDEROG, *Failures without chaos: a new process semantics for fair abstraction,* report CS-R8625, Centre for Math. & Comp. Sci., Amsterdam 1986, to appear in: Proc. IFIP Conf. on Formal Description of Progr. Concepts (M.Wirsing, ed.), Gl. Avernæs 1986, North-Holland.

[7] S.D.BROOKES, C.A.R.HOARE & A.W.ROSCOE, *A theory of communicating sequential processes,* JACM 31 (3), pp. 560 - 599, 1984.

[8] R.J.VAN GLABBEEK, *Bounded nondeterminism and the approximation induction principle in process algebra,* report CS-R8634, Centre for Math. & Comp. Sci., Amsterdam 1986.

[9] C.A.R.HOARE, *Communicating sequential processes,* Prentice Hall 1985.

[10] R.MILNER, *A calculus of communicating systems,* Springer LNCS 92, 1980.

[11] R.MILNER, *Lectures on a calculus of communicating systems,* Seminar on concurrency (S.D.Brookes, A.W.Roscoe & G.Winskel, eds.), pp. 197 - 220, Springer LNCS 197, 1985.

[12] D.M.R.PARK, *Concurrency and automata on infinite sequences,* Proc. 5th GI Conf., Springer LNCS 104, 1981.
[13] F.W.VAANDRAGER, *Verification of two communication protocols by means of process algebra,* report CS-R8608, Centre for Math. & Comp. Sci., Amsterdam 1986.
[14] J.L.M.VRANCKEN, *The algebra of communicating processes with empty process,* report FVI 86-01, Dept. of Comp. Sci., Univ. of Amsterdam 1986.