Another look at abstraction in process algebra
(extended abstract)

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 [4, 6]). We introduce a hidden step η, and construct a model for the resulting theory ACP_η. We briefly look at recursive specifications in this theory, and discuss the relations with Milner's silent step τ.
Note: Partial support received from the European Communities under ESPRIT contract no. 432, An Integrated Formal Approach to Industrial Software Development (Meteor).

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 [4, 6]) 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 [11] and HOARE [10], 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 discuss a model for ACP_η consisting of finitely branching process graphs modulo an appropriate notion of bisimulation (see PARK [13], MILNER [12], 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. We touch upon infinite processes, defined by means of recursive specifications. Finally, we discuss the relations of the constant η with Milner's silent step τ, that is also used for abstraction (see MILNER [11], BERGSTRA & KLOP [5]). 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.
All missing proofs, and extra information, can be found in the full paper [3].

2. Algebra of communicating processes.

In this section, we review the theory ACP (Algebra of Communicating Processes) as defined by BERGSTRA & KLOP [4, 6]. 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 parentheses 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 [4]. This will facilitate the 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.

$$\gamma(a,b) = \gamma(b,a) \quad \text{and} \quad \gamma(a,\gamma(b,c)) = \gamma(\gamma(a,b),c)$$

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 $+, \cdot, \|, \mathbin{\|\!\!\lfloor}, |$, unary operators ∂_H (for $H \subseteq A$) and a constant δ.

2.3 AXIOMS: The axioms of ACP are presented in table 1 below. 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:

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

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).

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

Table 1. ACP.

3. Hidden step η.

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.
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:

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

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 $\eta a + a \neq \eta a$. 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 [11]. The considerations made above can be formalised as follows. The theory BPA_η has laws A1-5 and H1,3.

3.1 DEFINITION. We define on BPA_η-terms binary predicates $\to^a$, and unary predicates $\to^a \surd$, for each $a \in A\cup\{\eta\}$.

$x \to^a y$ means that process x can evolve into process y, by starting a;
$x \to^a \surd$ 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. $a \to^a \surd$	5. if $x \to^a \surd$, then $xy \to^a y$
2. if $x \to^a x'$, then $x+y \to^a x'$ and $y+x \to^a x'$	6. $a \to^a \eta$
3. if $x \to^a \surd$, then $x+y \to^a \surd$ and $y+x \to^a \surd$	7. if $x \to^a y$ and $y \to^\eta z$, then $x \to^a z$
4. if $x \to^a x'$, then $xy \to^a x'y$	8. if $x \to^a y$ and $y \to^\eta \surd$, then $x \to^a \surd$.

(Compare these definitions with the ones in VAN GLABBEEK [9].)

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

If there exists a bisimulation between processes p and q, we say p and q **are bisimilar**, and write $p \underline{\leftrightarrow} q$.

3.3 THEOREM. $\underline{\leftrightarrow}$ is a congruence on BPA_η-terms.

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

3.5 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 $a|b = \delta$):

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

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:

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

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.

3.6 AN ALTERNATIVE. An alternative to the solution in 3.5 is, not adopting the law H2, but instead changing the laws CM2 and CM3 of ACP. In accordance with definition 3.1 we would have

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

so that in particular $a \mathbin{\|\!\_} x \to^a \eta\|x$. This leads to the following formulation of laws CM2 and CM3: $a \mathbin{\|\!\_} x = a(\eta x + x)$, and $ax \mathbin{\|\!\_} y = a(\eta(x\|y) + x\|y)$. We do not take this possibility in this paper, because we do not want to change the underlying system ACP.

3.7 **ACP$_\eta$**. Now we collect all axioms discussed so far together in the equational specification ACP$_\eta$. The theory ACP$_\eta$ 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$_\eta$ has, besides the axioms in table 1 (in 2.3) the axioms in table 2 below. We put $C = A \cup \{\delta, \eta\}$, the set of all constants. In tables 1,2 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.

$x\eta = x$	H1	$\eta_I(a) = a$	if $a \notin I$	HI1
$a(\eta(x+y)+x) = a(x+y)$	H2	$\eta_I(a) = \eta$	if $a \in I$	HI2
$a(\eta x + y) = a(\eta x + y) + ax$	H3	$\eta_I(x+y) = \eta_I(x) + \eta_I(y)$		HI3
		$\eta_I(xy) = \eta_I(x) \cdot \eta_I(y)$		HI4

Table 2. Axioms for ACP$_\eta$.

3.8 NOTE: Axioms CM5 and CM6 are derivable from the other axioms of ACP$_\eta$.

3.9 DEFINITION: A **basic term** is a closed BPA$_{\delta\eta}$-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 $a_i,b_j \in C$ and basic terms t_i. We usually abbreviate such expressions, in this case to $t = \sum_{i<n} a_it_i + \sum_{j<m} b_j$.
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 $\sum_{i<n} a_it_i \in$ BT.
Both these inductive schemes can be used in proofs.

3.10 THEOREM: For every closed ACP$_\eta$-term t there is a basic term s such that ACP$_\eta \vdash t=s$.
This is the so-called **elimination theorem**.

3.11 PROPOSITION: For all closed ACP$_\eta$-terms x,y,z we have the following laws of **standard concurrency**:

$$x|y = y|x \qquad (x \Vert\!\_\, y) \Vert\!\_\, z = x \Vert\!\_\,(y \Vert z)$$
$$x \Vert y = y \Vert x \qquad (x|y) \Vert\!\_\, z = x|(y \Vert\!\_\, z)$$
$$x|(y|z) = (x|y)|z \qquad x \Vert (y \Vert z) = (x \Vert y) \Vert z$$

3.12 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**: $x|y|z = \delta$ (HA).
It says, that all communication is *binary*, i.e. only involves two communication partners.

3.13 PROPOSITION: In ACP$_\eta$ with standard concurrency and handshaking axiom we have the following **expansion theorem** ($n \geq 1$) (where $\Vert_{i\leq n} x_i$ of course means $x_0 \Vert \ldots \Vert x_n$):
$$\Vert_{i\leq n} x_i = \textstyle\sum_{i\leq n} x_i \Vert\!\_\,(\Vert_{k\leq n,\, k\neq i}\, x_k) + \sum_{i<j\leq n} (x_i | x_j) \Vert\!\_\,(\Vert_{k\leq n,\, k\neq i,j}\, x_k).$$

4. The graph model.

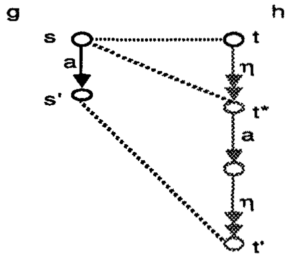
We construct a model for ACP$_\eta$ consisting of equivalence classes of process graphs.

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 $s \to^a s'$; $\twoheadrightarrow^\eta$ is the transitive and reflexive closure of $\to^\eta$, 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**.

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[13], MILNER [12] or BAETEN, BERGSTRA & KLOP [2].

4.2 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∈A (so a≠η, a≠δ), 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.



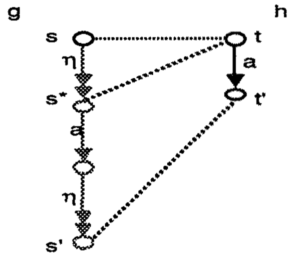


Fig. 1a. Fig. 1b.

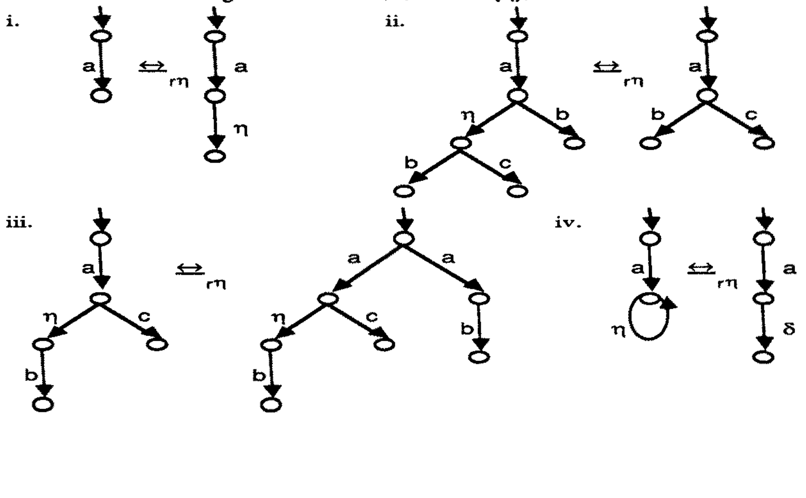
In case (s,t) is the pair of roots, we must have t≡t* (this is part of the so-called **root condition**).

3. Vice versa: if R(s,t) and t $\rightarrow^a$ t' is an edge in h with label a∉ 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≡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 $\underline{\leftrightarrow}_\eta$ h, if we do not require the root condition in points 2-5.

Graphs g and h are **rη-bisimilar**, g $\underline{\leftrightarrow}_{r\eta}$ h, if there is a rooted η-bisimulation between g and h; g and h are **η-bisimilar**, g $\underline{\leftrightarrow}_\eta$ h, if there is a η-bisimulation between g and h.

4.3 EXAMPLES: See fig. 2. We have a,b,c ∈ A∪{η}, so ≠δ.


i.
a
$\underline{\leftrightarrow}_{r\eta}$
a
η
ii.
a
η
b
b
c
$\underline{\leftrightarrow}_{r\eta}$
a
b
c
iii.
a
η
c
b
$\underline{\leftrightarrow}_{r\eta}$
a
a
η
c
b
b
iv.
a
η
$\underline{\leftrightarrow}_{r\eta}$
a
δ

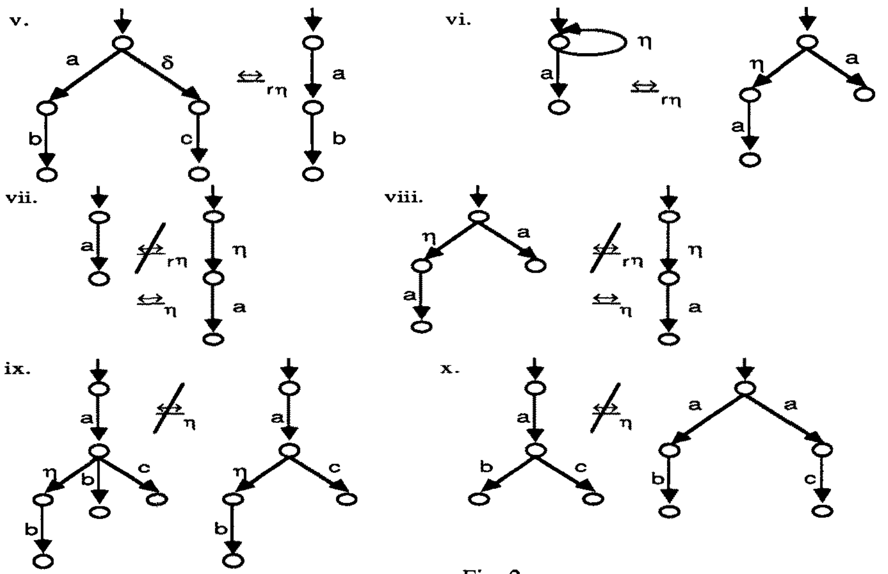


Fig. 2.

4.4 We see that $\underline{\leftrightarrow}_{r\eta}$ and $\underline{\leftrightarrow}_{\eta}$ are equivalence relations on $\mathbb{G}$. $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ 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.

4.5 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:

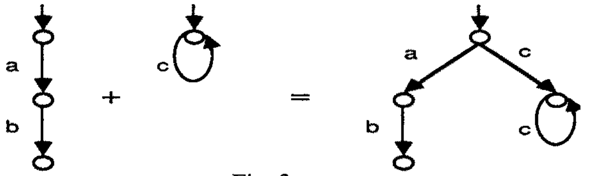


Fig. 3.

2. ·. If $g,h \in \mathbb{G}$, graph $g \cdot h$ 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 $g \cdot h$ is the root of g. EXAMPLE:

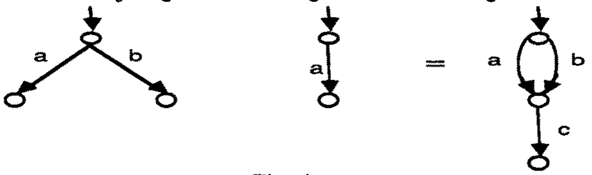


Fig. 4.

3. ||. If $g,h \in \mathbb{G}$, 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 $\gamma(a,b)$ 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 $\gamma(a,a) = d$, and $\gamma(a,b)$ and $\gamma(a,c)$ are not defined. Then:

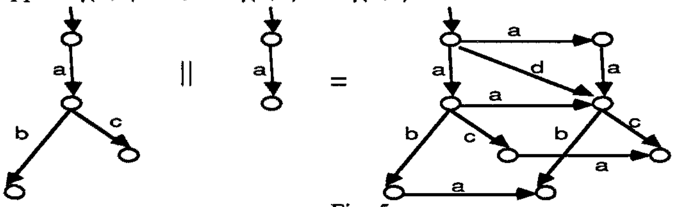


Fig. 5.

4. $⫫$. If $g,h \in \mathbb{G}$, graph $g⫫h$ 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 $s \to^a s'$ in g, an edge $r \to^a (s',t)$. Then, we discard nodes and edges that cannot be reached from the new root r.

EXAMPLE: (communications as in the previous example)

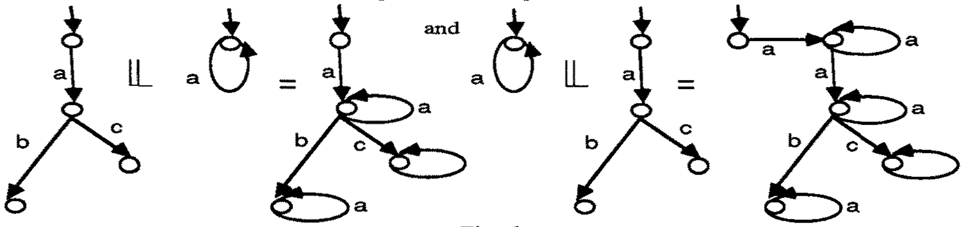


Fig. 6.

5. $|$. Similar to 4: if $g,h \in \mathbb{G}$, 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 $r \to^a (s',t')$ for each diagonal edge $(s,t) \to^a (s',t')$ in $g||h$. Then, we discard nodes and edges that cannot be reached from the new root r.

EXAMPLE: (communications as before)

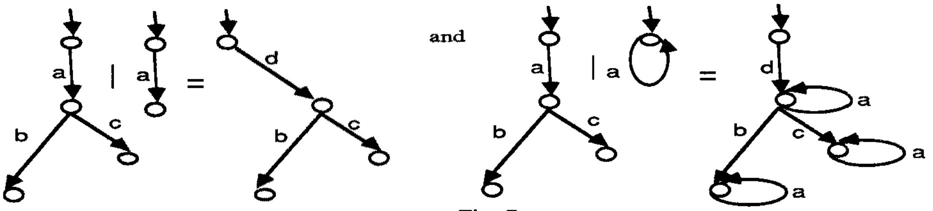


Fig. 7.

6. ∂_H, η_I. If $g \in \mathbb{G}$, 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 $\mathbb{G}$. Then we also have the operators on $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$, if we use the following proposition.

4.6 PROPOSITION. $\underline{\leftrightarrow}_{r\eta}$ is a congruence relation on $\mathbb{G}$.

4.7 THEOREM: $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ is a model of ACP_η.

4.8 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.

4.9 THEOREM: 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: $graph(t) \underline{\leftrightarrow}_{r\eta}\ graph(s) \Leftrightarrow ACP_\eta \vdash t=s$.

4.10 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 $ACP_\eta \vdash t=s$ iff $BPA_{\delta\eta} \vdash t=s$,
and for all closed ACP-terms t,s we have $ACP_\eta \vdash t=s$ iff $ACP \vdash t=s$.

5. Infinite processes.

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. We state some principles for infinite processes, that hold in the graph model.

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 see 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 THEOREM. The graph model $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ satisfies the **Recursive Definition Principle (RDP)**: Every guarded recursive specification has at least one solution.

5.5 THEOREM. The graph model $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ satisfies the **Recursive Specification Principle (RSP)**: Every guarded recursive specification has at most one solution.

5.6 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 $p = \sum_{i \leq n} a_i p_i$.

5.7 PROPOSITION: Each definable process can be written in head normal form.
Note that all elements of $\mathbb{G}/\underline{\leftrightarrow}_{r\eta}$ are definable. As an application of proposition 5.7, we have the following proposition.

5.8 PROPOSITION: Let p,q be definable processes. Then $\vdash p||q = q||p$.

5.9 FAIRNESS. To conclude this section, we want to mention the fairness principle HAR. This is the η **Abstraction Rule**, and is comparable to Koomen's Fair Abstraction Rule KFAR, see BAETEN, BERGSTRA & KLOP [2]. HAR states that a process will not perform an infinite sequence of internal steps, but will perform an external step eventually (if possible):

$$\text{if } x = ix + y, \text{ and } i \in I, \text{ then } \eta_I(x) = \eta \cdot \eta_I(y) + \eta_I(y).$$

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*. Taking $y = \delta$, an application of HAR yields $\eta_{\{i\}}(x) = \eta\delta$, an equation that we can call *livelock = deadlock*.

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 [5]. This abstraction mechanism is based on Milner's silent step τ.
The three laws of the constant τ are from MILNER [11], and are as follows:

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

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 4.3.ix illustrates. We can motivate the second τ-law by reconsidering the action relations of 3.1. 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 3.1 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 [9]). 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 $t \underline{\leftrightarrow} s \Leftrightarrow BPA_\tau \vdash t{=}s$. In this philosophy, τ and η denote the same process, but in ACP_η, more subtle differences between processes are observable. The η of ACP_η is between the η of the action relations 3.1 and τ.
A different motivation for τ is along the lines of the beginning of section 3: 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, which consists of the axioms in table 1 (in 2.3), the three τ-laws above, and the axioms in table 3 below. In all these axioms, we have $a,b \in A\cup\{\delta\}$, $H,I \subseteq A$ and x,y,z are arbitrary processes.

$\tau \mathbin{\lfloor\!\lfloor} x = \tau x$	TM1	$\partial_H(\tau) = \tau$	DT
$\tau x \mathbin{\lfloor\!\lfloor} y = \tau(x \| y)$	TM2	$\tau_I(\tau) = \tau$	TI1
$\tau \mid x = \delta$	TC1	$\tau_I(a) = a \quad$ if $a \notin I$	TI2
$x \mid \tau = \delta$	TC2	$\tau_I(a) = \tau \quad$ if $a \in I$	TI3
$\tau x \mid y = x \mid y$	TC3	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	TI4
$x \mid \tau y = x \mid y$	TC4	$\tau_I(xy) = \tau_I(x) \cdot \tau_I(y)$	TI5

Table 3. Axioms of 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 3.10 (Elimination Theorem) and propositions 3.11 (laws of Standard Concurrency) and 3.13 (Expansion Theorem) become more cumbersome (see BERGSTRA & KLOP [5]). 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_η, at least in prefix position).

6.3 DEFINITION. We will define a mapping $\tau_{\{\eta\}}$ from ACP_η-processes to ACP_τ-processes on the graph model $\mathbb{G}$. First some notes on the graph model of ACP_τ.
The graph model of countably branching process graphs modulo rooted τ-bisimulation for ACP_τ 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 4.2) 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_τ 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_τ, 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. Thus, for a process graph g that does not have a η or τ label, we have $\tau_{\{\eta\}}(\eta_I(g)) = \tau_I(g)$.

6.4 PROPOSITION: The mapping $\tau_{\{\eta\}}$ is a homomorphism w.r.t. the operators $+,\cdot,\|,⫽,\partial_H$.

6.5 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 $\eta a\,|\,b = \delta$.

6.6 THEOREM. Let g be a finitely branching graph with labels from $A\cup\{\delta,\eta\}$, and let h be a finitely branching graph with labels from $A\cup\{\delta\}$, such that $\tau_{\{\eta\}}(g)\ \underline{\leftrightarrow}_{r\tau}\ h$. Then $g\ \underline{\leftrightarrow}_{r\eta}\ h$.

6.7 Theorem 6.6 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.

6.8 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:

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

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. 8, 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 $H = \{r2(d), s2(d) : d\in D\}$, and abstract from internal steps, actions from $I = \{c2(d) : d\in D\}$. With these definitions, we have the following theorem.

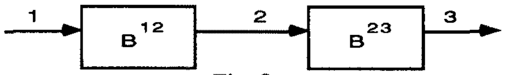


Fig. 8.

6.9 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 the remark at the end of 6.3, 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.

6.10 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 by example 4.3.ix.)
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 [8] or BERGSTRA, KLOP & OLDEROG [7]).
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.

6.11 DEFINITION: Enrich the signature of ACP_η with a unary operator τ and add the following law:

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

We call the resulting theory $ACP_{\eta\tau}$. Note that, as an immediate consequence of law A4, we have that $\tau(x)\cdot y = \tau(x\cdot y)$. This allows to write $\tau\cdot x$ instead of $\tau(x)$.

6.12 LEMMA: The following equations are provable 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. $\tau x \mathbin{\lfloor\!\lfloor} y = \eta(x \| y) + x \mathbin{\lfloor\!\lfloor} y$

6.13 LEMMA: The following equations are provable for all closed $ACP_{\eta\tau}$-terms x,y:

1. $x\tau y = xy$ 3. $x \mathbin{\lfloor\!\lfloor} \tau y = x \mathbin{\lfloor\!\lfloor} \eta y = x \mathbin{\lfloor\!\lfloor} y$
2. $\tau x | y = x | y = x | \tau y$ 4. $\tau x \| y = \tau(x \| y)$.

6.14 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 $\lfloor\!\lfloor$. 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.

6.15 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,* to appear in Theor. Comp. Sci. 51 (1/2).

[3] J.C.M.BAETEN & R.J.VAN GLABBEEK, *Another look at abstraction in process algebra,* report CS-R8701, Centre for Math. & Comp. Sci., Amsterdam 1987.

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

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

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

[7] 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.

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

[9] R.J.VAN GLABBEEK, *Bounded nondeterminism and the approximation induction principle in process algebra,* Proc. STACS 87 (F.Brandenburg, G.Vidal-Naquet & M.Wirsing, eds.), Springer LNCS 247, pp. 336 - 347, 1987.

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

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

[12] 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.

[13] D.M.R.PARK, *Concurrency and automata on infinite sequences,* Proc. 5th GI Conf.(P. Deussen, ed.), Springer LNCS 104, pp. 167 - 183, 1981.

[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.