Fundamenta Informaticae XII (1989) 221-242
North-Holland

ABSTRACTION AND EMPTY PROCESS IN PROCESS ALGEBRA

J.C.M. BAETEN* and R.J. VAN GLABBEEK**

* *Programming Research Group, University of Amsterdam*
P.O. Box 41882, 1009 DB Amsterdam, The Netherlands

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

Received February 1988 / Accepted September 1988

Abstract: In this paper, we combine the hidden step η of the authors' paper [2] with the empty process ε of VRANCKEN [12] and the authors' [3]. We formulate a system ACPc, which is a conservative extension of the systems ACP_η, $ACP\surd$, but also of ACP_τ. This is a general system, in which most relevant issues can be discussed. Abstraction from internal steps can be achieved in two ways, in two stages: we can abstract to the hidden step η, and then from η to 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

Having been introduced to the Algebra of Communicating Processes of BERGSTRA & KLOP [4], many people ask the question why there is no neutral element for the sequential composition $\cdot$. The neutral element for alternative composition $+$ is the constant δ, that is used to denote deadlock, unsuccessful termination. A constant ε satisfying the laws $\varepsilon \cdot x = x \cdot \varepsilon = x$ must stand for an **empty process**, a process that terminates immediately and successfully. The investigation of what happens when we want to add such a constant to ACP was started by KOYMANS & VRANCKEN [8]. It turned out that the constant ε is very useful, but that the technicalities involved were substantial. For instance, the just quoted paper contained a non-associative merge operator. This problem was remedied in VRANCKEN [12], where the theory ACP was modified and extended to ACP^ε. Recently, the system ACP^ε was reformulated as $ACP\surd$ in BAETEN & VAN GLABBEEK [3], where termination was made explicit. In practice, the constant ε already showed its usefulness in BERGSTRA, KLOP & OLDEROG [6], where ε was needed to define the constant Δ denoting divergence. However, this last paper left an open question: how do we combine the empty process ε with the system ACP_τ of BERGSTRA & KLOP [5]?
Incorporated in ACP_τ is the notion of abstraction, a central issue in concurrency theory. 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 [9] and HOARE [7], 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 abstraction operator that transforms observable actions into internal ones. Now the constant ε stands for a process of no duration, and so cannot be used for an internal action; using it,

we would loose too much information, e.g. information on deadlock behaviour.
The silent step τ of MILNER [9] can be used for an internal step, and the operator τ_I, that renames actions from the set I into τ, can be used for abstraction.
Koymans and Vrancken found that the second law for τ immediately translates to $\tau = \tau + \varepsilon$, so τ contains a summand ε. This raised the question, what the process τ turns into, when we remove the option of immediate termination. The resulting process, called η, together with the abstraction operator η_I, that renames actions from I into η, was extensively investigated in BAETEN & VAN GLABBEEK [2]. There, laws were formulated for the constant η, and it was added to the system ACP (without ε). It was found that η can also be used for an internal step. Also, the hidden step η was compared with the silent step τ. It was found that the η has technical advantages over the τ, but both types of abstraction can co-exist, indeed that one can be applied after the other. What was lacking, is a single system in which both constants appear. While the τ exists in ACP_η in prefix position, it could not be defined as a process.
The solution was already indicated in the last paragraph of [2]: we can define τ, if we have added the empty process ε. Thus, the task is to combine the system ACP$\surd$ of [3] with the system ACP_η of [2]. This is what we do in this paper. The result is a system ACPc (Algebra of Communicating Processes with constants), which has constants η and ε, and in which τ is definable.
We discuss a model for ACPc consisting of finitely branching process graphs modulo an appropriate notion of bisimulation (cf. PARK [11], MILNER [10]). We use this model to establish the consistency of ACPc and the conservativity of ACPc over ACP_η, and most of ACP$\surd$ and ACP_τ.

ACKNOWLEDGEMENT
The original ideas for ε and η, and the central equation $\tau = \eta + \varepsilon$, are due to Karst Koymans and Jos Vrancken.

2. ALGEBRA OF COMMUNICATING PROCESSES WITH EMPTY PROCESS

In this section, we review the theory ACP$\surd$ (Algebra of Communicating Processes with empty process and explicit termination) as defined in BAETEN & VAN GLABBEEK [3]. ACP$\surd$ is a reformulation of the system ACP^ε of VRANCKEN [12].
For a review of related approaches and comparisons with them, we refer to BERGSTRA & KLOP [4].
The axioms of ACP$\surd$ are displayed in table 1, in 2.3 on page 4.

2.1 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 $a,b,c \in A$ (and each side of these equations is defined just when the other side is). 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 can 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$\surd$. Further, ACP$\surd$ has binary operators $+,\cdot,\|,\lfloor\!\lfloor,|$, unary operators ∂_H, ε_K (for $H,K \subseteq A$) and constants δ,ε. $\surd$ is another notation for ∂_A.

2.2 HEURISTICS

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 see this from laws A1-3, since A3 is equivalent to $x+x=x$, by use of A4 and A8). We do not specify whether a choice is made by the process itself, or by the environment. 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 ε denoting the empty process, characterized as the neutral element of sequential composition. See axioms A8,9. In a sum, as in $x + \varepsilon$, it tells us that the process can terminate immediately. We have the operator $\surd$ to indicate whether or not a process can terminate immediately: $\surd(x) = \varepsilon$ if x has the termination option, and $\surd(x) = \delta$ otherwise.

Furthermore, there is a special constant δ denoting deadlock, the acknowledgement of a process that it cannot do anything anymore, the absence of any alternative. See axioms A6-7 (note that axiom A6 is equivalent to $\delta+x=x$, by use of A4, A7 and A8). We can consider δ to stand for unsuccessful termination, and ε for successful termination.

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 $x \| y$, there are 4 possibilities: the process can terminate (only if both x and y have that option), a step from x can be executed, or a step from y, or x and y both synchronously perform an action, which together make up a new action, the communication action. These options are present in axiom EM1. Here, we use the auxiliary operators $\lfloor\!\lfloor$ (left-merge), $|$ (communication merge) and $\surd$ (used to indicate termination). Thus, $x \lfloor\!\lfloor y$ 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. A simple case distinction learns us that the termination summand of $x \| y$ can be represented by $\surd(x) \cdot \surd(y)$. Axioms CF1,2 and EM2-8 give the laws for $\lfloor\!\lfloor$ and $|$. Axioms for $\surd$ are discussed below. It is also possible to use the left-merge $\lfloor\!\lfloor$ to express the termination possibility (as in VRANCKEN [12]), or even the communication merge $|$. Note that it is not a good solution to replace EM2 by the axiom $\varepsilon \lfloor\!\lfloor x = x$ (as in KOYMANS & VRANCKEN [8]), as was shown in [12].

Finally, we have in table 1 the renaming operators ∂_H and ε_K. Here H and K are sets of atoms. ∂_H blocks the actions from H, renames them into δ (axioms D0-4), and ε_K erases actions from K, renames them into ε (axioms E0-4). The operator ∂_H can be used to encapsulate a process, i.e. to block communications with the environment. If we block *all* atomic actions, as in ∂_A, we get an expression for the termination operator $\surd$.

2.3 AXIOMS

The axioms of ACP$\surd$ are presented in table 1, on the following page. There $a,b \in A \cup \{\delta\}$, $H,K \subseteq A$, and x,y,z are arbitrary processes.

Note that axiom CF2 implies that $\delta | a = \delta$ for all $a \in A \cup \{\delta\}$. Since every expression of the form $a | b$ is equal to an element of $A \cup \{\delta\}$, we can assume that axioms EM3,7, D1,2, E1,2 also hold for these expressions instead of a. We call the theory just consisting of the first nine axioms, A1-9, $BPA_{\delta\varepsilon}$ (so

$BPA_{\delta\varepsilon}$ has in the signature only operators $+,\cdot$ and constants $A\cup\{\delta,\varepsilon\}$).

$x + y = y + x$	A1	$\delta + \varepsilon = \varepsilon$		A6
$(x + y) + z = x + (y + z)$	A2	$\delta x = \delta$		A7
$\varepsilon + \varepsilon = \varepsilon$	A3	$\varepsilon x = x$		A8
$(x + y)z = xz + yz$	A4	$x\varepsilon = x$		A9
$(xy)z = x(yz)$	A5			
		$a \mid b = \gamma(a,b)$	if $\gamma(a,b)\downarrow$	CF1
		$a \mid b = \delta$	otherwise	CF2
$x \mathbin{\|} y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x + x \mid y + \surd(x)\surd(y)$	EM1	$x \mid y = y \mid x$		EM5
$\varepsilon \mathbin{\lfloor\!\lfloor} x = \delta$	EM2	$x \mid \varepsilon = \delta$		EM6
$ax \mathbin{\lfloor\!\lfloor} y = a(x \mathbin{\|} y)$	EM3	$x \mid ay = (x \mid a) \mathbin{\lfloor\!\lfloor} y$		EM7
$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$	EM4	$x \mid (y + z) = x \mid y + x \mid z$		EM8
$\partial_H(\varepsilon) = \varepsilon$	D0	$\varepsilon_K(\varepsilon) = \varepsilon$		E0
$\partial_H(a) = a$ if $a \notin H$	D1	$\varepsilon_K(a) = a$ if $a \notin K$		E1
$\partial_H(a) = \delta$ if $a \in H$	D2	$\varepsilon_K(a) = \varepsilon$ if $a \in K$		E2
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D3	$\varepsilon_K(x + y) = \varepsilon_K(x) + \varepsilon_K(y)$		E3
$\partial_H(xy) = \partial_H(x)\cdot\partial_H(y)$	D4	$\varepsilon_K(xy) = \varepsilon_K(x)\cdot \varepsilon_K(y)$		E4

Table 1. ACP$\surd$.

2.4 REMARKS

The system ACP$\surd$ presented above differs in four respects from the system ACP^{ε} in [12]. Firstly, we use an explicit summand in the equation for merge for the termination possibility, as was discussed in 2.1. Next, in [12], γ is a total function from $A\times A$ to $A\cup\{\delta\}$, while in this paper, γ is a partial function from $A\times A$ to A. Also, in the axioms in [12], a varies over A, not over $A\cup\{\delta\}$, which necessitates more axioms. Lastly, we left out the axiom $(x \mid y) \mid z = x \mid (y \mid z)$, as we saw no reason for its inclusion.

The system ACP$\surd$ differs in several aspects from the system ACP of BERGSTRA & KLOP [4]. Most of these differences were a consequence of the addition of the constant ε. Another difference is the inclusion of axiom EM5, the commutativity of the communication merge, which decreased the number of axioms needed.

3. HIDDEN STEP η

3.1 In [2], the constant η is introduced. It stands for a *hidden* or *internal* action: when we want to abstract from the internal actions in systems consisting of several components (essential for verification purposes), we rename these actions into η, the hidden step. η obeys the laws of atomic actions, but has in addition three extra laws, which serve to calculate the η away in certain contexts. We cannot get rid of η altogether (η is not ε), because deadlock behaviour would not be preserved. The three η-laws are presented in table 2 below.

For some intuitive background on these laws, we refer to [2]. Roughly, this intuition amounts to saying that an η-step cannot be directly observed, but will take some time to execute.

$x\eta = x$	H1
$a(\eta(x + y) + x) = a(x + y)$	H2
$a(\eta x + y) = a(\eta x + y) + ax$	H3

Table 2. η-laws.

In a setting with ε, the first law cannot be maintained, for substituting ε for x leads to an unwanted equation. However, by taking $x=\delta$, $y=\varepsilon$ in the second law, we obtain $a\eta = a$, from which the first law can be derived for all basic terms that do not have a subterm with an ε-summand.
Therefore, we will only add the laws H2,3 to ACP$\surd$, and will have a vary over $C = A\cup\{\delta,\eta\}$ (the set of atom-like constants) instead of $A\cup\{\delta\}$. Furthermore, we have to leave out the operator ε_K, which cannot be added (for we would have $\varepsilon = \varepsilon_{\{a\}}(a) = \varepsilon_{\{a\}}(a\eta) = \varepsilon_{\{a\}}(a)\cdot\varepsilon_{\{a\}}(\eta) = \varepsilon\eta = \eta$; it is no solution to have ε_K always rename η into ε as well (the trick we use for τ_I), since then $a(b + c) = \varepsilon_\emptyset(a(\eta b + c) = \varepsilon_\emptyset(a(\eta b + c) + ab) = a(b + c) + ab)$.
Now we will present the system ACPc. We present the system in two parts: the first part, in 3.2, contains the basic system; in 3.3 we discuss in addition the constant τ and the operator τ_η.

3.2 ACPc, THE BASIC SYSTEM

As before, A is a given (finite) set of atomic actions; on A, we have given a partial binary function γ, which is commutative and associative. All elements of A are constants of ACPc. Further, the basic system ACPc has binary operators $+,\cdot,\|,\mathbin{\mathbb{L}},|$, constants δ,ε,η, and unary operators ∂_H, η_I (for $H,I \subseteq A$, but we also allow $H = C = A\cup\{\delta,\eta\}$). $\surd$ now is another notation for ∂_C.

$x + y = y + x$	A1	$\varepsilon x = x$	A8
$(x + y) + z = x + (y + z)$	A2	$x\varepsilon = x$	A9
$\varepsilon + \varepsilon = \varepsilon$	A3		
$(x + y)z = xz + yz$	A4	$a(\eta(x + y) + x) = a(x + y)$	H2
$(xy)z = x(yz)$	A5	$a(\eta x + y) = a(\eta x + y) + ax$	H3
$\delta + \varepsilon = \varepsilon$	A6		
$\delta x = \delta$	A7	$a \mid b = \gamma(a,b)$ if $\gamma(a,b)\downarrow$	CF1
		$a \mid b = \delta$ otherwise	CF2
$x \| y = x \mathbin{\mathbb{L}} y + y \mathbin{\mathbb{L}} x + x \mid y + \surd(x)\surd(y)$	EM1	$x \mid y = y \mid x$	EM5
$\varepsilon \mathbin{\mathbb{L}} x = \delta$	EM2	$x \mid \varepsilon = \delta$	EM6
$ax \mathbin{\mathbb{L}} y = a(x \| y)$	EM3	$x \mid ay = (x \mid a) \mathbin{\mathbb{L}} y$	EM7
$(x + y) \mathbin{\mathbb{L}} z = x \mathbin{\mathbb{L}} z + y \mathbin{\mathbb{L}} z$	EM4	$x \mid (y + z) = x \mid y + x \mid z$	EM8
$\partial_H(\varepsilon) = \varepsilon$	D0	$\eta_I(\varepsilon) = \varepsilon$	HI0
$\partial_H(a) = a$ 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 3. ACPc (basic system).

Note that η can only be renamed into δ, when all atomic actions are renamed into δ at the same time (for otherwise we get an inconsistency from the equation $a=a\eta$). η_I is the **hiding operator**, that renames actions from I into η; I is the set of *internal* actions, that are hidden in order to verify the external behaviour of a system. η_I is also called the **η-abstraction** operator. The axioms of ACPc are presented in table 3 above. There $a,b \in C$, $H,I \subseteq A$ or $H = C$, and x,y,z are arbitrary processes.

3.3 ACPc, THE FULL SYSTEM

The full system ACPc has in the signature, besides the elements mentioned in 3.2, a constant τ, and a unary operator τ_η. τ is the **silent step** of MILNER [9], that, like η, stands for an invisible step; we find however, that it has different properties than η (see [2] and below).

τ_η is a kind of abstraction operator, that renames η into τ. Applying this operator means that we lose some information: we 'forget' that internal actions have to take a positive amount of time (τ has the option of terminating immediately, η does not).

Furthermore, we use the abbreviation τ_I for $\tau_\eta \circ \eta_I$ ($I \subseteq A$). τ_I is the **(τ-)abstraction operator**, that renames atoms from I, and also η, into τ. This operator is also used to abstract from the internal behaviour of a system. Note that we must always rename η into τ, when we use τ_I, for otherwise $\tau = \tau_A(a) = \tau_A(a\eta) = \tau_A(a)\cdot\tau_A(\eta) = \tau\eta = \eta$.

The axioms of ACPc comprise, besides the axioms in table 3, those in table 4.

$\tau = \eta + \varepsilon$	THE
$\tau_\eta(\varepsilon) = \varepsilon$	HT0
$\tau_\eta(\eta) = \tau$	HT1
$\tau_\eta(a) = a \quad \text{if } a \neq \eta$	HT2
$\tau_\eta(x + y) = \tau_\eta(x) + \tau_\eta(y)$	HT3
$\tau_\eta(xy) = \tau_\eta(x)\cdot\tau_\eta(y)$	HT4

Table 4. ACPc (additional axioms).

3.4 LEMMA

The following equations are derivable from the system ACPc ($a,b \in C$, H, $I \subseteq A$):

1. $a\eta = a\tau = a$
2. $\eta\tau = \tau\eta = \eta$
3. $\tau\tau = \tau$
4. $\tau\delta = \eta\delta$
5. $\tau + \varepsilon = \tau + \eta = \tau$
6. $\tau x + x = \tau x$
7. $a(\tau x + y) = a(\tau x + y) + ax$
8. $x \mid \delta = \delta$
9. $\tau x \mathbin{\mathbb{L}} y = \eta(x \| y) + x \mathbin{\mathbb{L}} y$
10. $a(\eta x \| y) = a(\tau x \| y) = a(x \| y)$
11. $ax \| by = by \| ax$
12. $\partial_H(\tau) = \eta_I(\tau) = \tau_I(\tau) = \tau_I(\eta) = \tau$, but $\partial_C(\tau) = \varepsilon$
13. $\tau_I(\varepsilon) = \varepsilon$
14. $\tau_I(a) = a$ if $a \notin I \cup \{\eta\}$, and $\tau_I(a) = \tau$ if $a \in I \cup \{\eta\}$
15. $\tau_I(x + y) = \tau_I(x) + \tau_I(y)$
16. $\tau_I(xy) = \tau_I(x)\cdot\tau_I(y)$

PROOF: Straightforward. We give some hints:

1. $a\eta = a(\eta(\varepsilon + \delta) + \delta) = a(\varepsilon + \delta) = a$ and $a\tau = a(\eta(\delta + \varepsilon) + \varepsilon) = a(\delta + \varepsilon) = a$.
8. $\delta = x \mid \varepsilon = x \mid (\varepsilon + \delta) = x \mid \varepsilon + x \mid \delta = \delta + x \mid \delta = x \mid \delta$.
10. $a(\eta x \| y) = a\eta x \mathbin{\mathbb{L}} y = ax \mathbin{\mathbb{L}} y = a(x \| y) = ax \mathbin{\mathbb{L}} y = a\tau x \mathbin{\mathbb{L}} y = a(\tau x \| y)$.

3.5 NOTE

The second equation of 3.4.1 with 3.4.3, together with 3.4.6 and 3.4.7, form the three τ-laws of MILNER [9]. They form, in the system CCS, and also in ACP_τ (see BERGSTRA & KLOP [5]) the defining equations for the silent step τ. Compared to the defining equations for η, we see that the crucial point is the difference between the law H2 and 3.4.6: H2 is more restrictive, and this difference has many consequences, as was made clear in [2].

3.6 LEMMA

In the system ACPc plus extra axiom $\varepsilon \| x = x$ the following equations are derivable ($a,b \in C$):

1. $x = x \lfloor\!\lfloor \varepsilon + \surd(x)$
2. $x \| \varepsilon = x$
3. $a \lfloor\!\lfloor x = ax$
4. $\tau \lfloor\!\lfloor x = \eta x$
5. $a \mid bx = ax \mid b = (a \mid b)x$
6. $ax \mid by = (a \mid b)(x \| y)$
7. $a(\eta \| x) = a(\tau \| x) = ax$

PROOF: Straightforward.

3.7 NOTE

Equation 3.4.1 states that we can write each process as the sum of its termination option ($\surd(x)$) and the summands that start with an atomic action ($x \lfloor\!\lfloor \varepsilon$). Next, we consider another equation that is of special interest, namely the assertion that $\surd(x)$ must be either ε or δ (here $x = x + \varepsilon$ amounts to saying that x has an ε-summand):

$$\surd(x) = \varepsilon \text{ iff } x = x + \varepsilon, \text{ and } \surd(x) = \delta \text{ otherwise} \qquad (*).$$

3.8 LEMMA

In the system ACPc plus extra axiom (*) the following equations are derivable:

1. $\surd(x) \lfloor\!\lfloor y = \delta$
2. $\surd(x) \mid y = \delta$
3. $\surd(\surd(x)) = \surd(x)$
4. $\surd(x) \cdot \surd(y) = \surd(y) \cdot \surd(x)$
5. $x \| y = y \| x$
6. $\eta x \| \eta y = \eta(x \| y)$

PROOF: Straightforward.

3.9 DEFINITION

A **basic term** is a closed term of the form

$$t = a_0 t_0 + \ldots + a_{n-1} t_{n-1} + b_0 + \ldots + b_{m-1} \ (+ \varepsilon)$$

for certain $n,m \in \mathbb{N}$, certain $a_i, b_j \in C$, basic terms t_i and the summand ε may or may not occur. If the summand ε does not occur, we must have $n+m > 0$.

We usually abbreviate such expressions, in this case to $t = \sum_{i<n} a_i t_i + \sum_{j<m} b_j \ (+ \varepsilon)$. Note that we can always write $t = \sum_{i<n} a_i t_i + \sum_{j<m} b_j + \surd(t)$, for it is easy to see that $\surd(t) = \varepsilon$ iff t has a summand ε, and $\surd(t) = \delta$ otherwise.

The set of basic terms **BT** can be inductively built up as follows (working modulo laws A1-3 and A9):

1. $\varepsilon \in BT$
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, if $\sum_{i<0} x_i$ denotes δ, we can build up BT as follows:

– If $n \in \mathbb{N}$, $a_i \in C$ and $t_i \in BT$ (for $i<n$), then $\sum_{i<n} a_i t_i \ (+ \varepsilon) \in BT$.

Both these inductive schemes can be used in proofs.

3.10 THEOREM

For every closed ACPc-term t there is a basic term s such that ACPc $\vdash t=s$. This is the so-called **elimination theorem.**

PROOF: The proof is very similar to the case of ACP$\surd$, treated in [3]. We mention the differences briefly: η_I should be treated like ∂_H and τ_η like ε_K. Take $Te(\tau) = 1$, $I(\tau) = 1$ and $w(\tau) = 2$. In claim 3, τ is not a normal form of RACPc.

3.11 LEMMA

The following statements hold for all closed ACPc-terms:

1. $\varepsilon \| x = x$, and hence the equations of lemma 3.6;
2. $\surd(x) = \varepsilon$ iff $x + \varepsilon = x$ and $\surd(x) = \delta$ otherwise, and hence the equations of lemma 3.8;
3. $\surd(x \between y) = \surd(x \mid y) = \delta$
4. $\eta x \mid y = \eta \mid x = \tau \mid x = \delta$
5. $\tau x \mid y = x \mid y$
6. $x \between \tau y = x \between y$
7. $\tau x \| y = \tau(x \| y)$
8. $x \| (\eta y + z) = x \| (\eta y + z) + x \between y$

PROOF: By theorem 3.10, it is enough to prove these equations for basic terms. This proof can be done by structural induction, following one of the schemes in 3.9.

Write $x = \sum_{i<n} a_i x_i \ (+ \varepsilon)$ and $y = \sum_{j<m} b_j y_j \ (+ \varepsilon)$ $(m,n \in \mathbb{N}, a_i, b_j \in C)$.

Now 1 - 5 follow immediately, and 6 follows from 3.8.5 and 3.4.10 (note that only in the proof of statement 1, we actually need the induction hypothesis).

The proof of 7: $\tau x \| y = \tau x \between y + y \between \tau x + \tau x \mid y + \surd(\tau x) \cdot \surd(y) =$

$= \eta x \between y + x \between y + y \between x + x \mid y + \surd(x) \cdot \surd(y) = \eta(x \| y) + x \| y = \tau(x \| y)$.

In the proof of 8, we use the following notation: $p \leq q$ iff $p + q = q$.

It follows from A1, A2 and A3 that $\leq$ is antisymmetrical, transitive and reflexive, respectively, and hence a partial ordering. Now H3 can be reformulated as a conditional equation:

$$\eta p \leq q \Rightarrow ap \leq aq \qquad \text{(H3)}.$$

Now we can prove 8: $\eta(x_i \| y) = \eta(y \| x_i) = \eta y \between x_i \leq (\eta y + z) \between x_i \leq x_i \| (\eta y + z)$, and thus

$x \between y = \sum_i a_i x_i \between y = \sum_i a_i (x_i \| y) \leq \sum_i a_i (x_i \| (\eta y + z)) = x \between (\eta y + z) \leq x \| (\eta y + z)$.

3.12 PROPOSITION

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

$\varepsilon \| x = x$

$\surd(x) = \varepsilon$ iff $x + \varepsilon = x$, and $\surd(x) = \delta$ otherwise

$\surd(x \| y) = \surd(x) \cdot \surd(y)$

$x \mid (y \mid z) = (x \mid y) \mid z$

$(x \between y) \between z = x \between (y \| z)$

$(x \mid y) \between z = x \mid (y \between z)$

$x \| (y \| z) = (x \| y) \| z$.

PROOF: As in [3].

3.13 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**:

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

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

3.14 PROPOSITION

In ACPc with standard concurrency and handshaking axiom we have the following **expansion theorem** ($n \geq 1$):

$$\|_{i\leq n} x_i = \sum_{i\leq n} x_i \mathbin{\mathbb{L}} (\|_{k\leq n, k\neq i} x_k) + \sum_{i<j\leq n} (x_i \mid x_j) \mathbin{\mathbb{L}} (\|_{k\leq n, k\neq i,j} x_k) + \prod_{i\leq n} \surd(x_i)$$

(Where $\|_{i\leq n} x_i$ means $x_0 \| \ldots \| x_n$, and $\Pi_{i\leq n} x_i$ means $x_0 \cdot \ldots \cdot x_n$.)

PROOF: As in [3].

3.15 PROPOSITION

The operator τ_η is a homomorphism on closed terms, w.r.t. the operators $+, \cdot, \|, \mathbb{L}, \partial_H$, for $H \subseteq A$.

PROOF: The proof is only non-trivial for the case of $\|, \mathbb{L}$. We prove the case of $\|$.

Thus, let x,y be closed ACPc-terms. We have to prove that ACPc $\vdash \tau_\eta(x\|y) = \tau_\eta(x) \| \tau_\eta(y)$.

Because of the elimination theorem, we can assume that x,y are basic terms. We use the second induction scheme of 3.9.

Write $x = \sum_{i<n} a_i x_i + \sum_{j<m} \eta x'_j + \surd(x)$ and $y = \sum_{k<p} b_k y_k + \sum_{l<q} \eta y'_l + \surd(y)$ $\quad (a_i, b_k \in A \cup \{\delta\})$.

Then $\tau_\eta(x) = \sum_i a_i \cdot \tau_\eta(x_i) + \sum_j \eta \cdot \tau_\eta(x'_j) + \sum_j \tau_\eta(x'_j) + \surd(x)$. Also, note that $\surd(\tau_\eta(x)) =$

$= \sum_j \surd(\tau_\eta(x'_j)) + \surd(x)$, and $\surd(\tau_\eta(y)) = \sum_l \surd(\tau_\eta(y'_l)) + \surd(y)$, so $\surd(\tau_\eta(x)) \cdot \surd(\tau_\eta(y)) =$

$= \sum_{j,l} \surd(\tau_\eta(x'_j)) \cdot \surd(\tau_\eta(y'_l)) + \sum_l \surd(x) \cdot \surd(\tau_\eta(y'_l)) + \sum_j \surd(\tau_\eta(x'_j)) \cdot \surd(y) + \surd(x) \cdot \surd(y)$

(use A4, in combination with 3.8.4) =

$= \sum_l \surd(\tau_\eta(x)) \cdot \surd(\tau_\eta(y'_l)) + \sum_j \surd(\tau_\eta(x'_j)) \cdot \surd(\tau_\eta(y)) + \surd(x) \cdot \surd(y)$.

Now, using 3.8.1-2, we get $\tau_\eta(x) \| \tau_\eta(y) =$

$= \sum_i a_i(\tau_\eta(x_i) \| \tau_\eta(y)) + \sum_j \eta(\tau_\eta(x'_j) \| \tau_\eta(y)) + \sum_j \tau_\eta(x'_j) \mathbb{L} \tau_\eta(y) +$

$+ \sum_k b_k(\tau_\eta(x) \| \tau_\eta(y_k)) + \sum_l \eta(\tau_\eta(x) \| \tau_\eta(y'_l)) + \sum_l \tau_\eta(y'_l) \mathbb{L} \tau_\eta(x) +$

$\sum_{i,k} (a_i \mid b_k)(\tau_\eta(x_i) \| \tau_\eta(y_k)) + \sum_j \tau_\eta(x'_j) \mid \tau_\eta(y) + \sum_l \tau_\eta(x) \mid \tau_\eta(y'_l) +$

$+ \sum_l \surd(\tau_\eta(x)) \cdot \surd(\tau_\eta(y'_l)) + \sum_j \surd(\tau_\eta(x'_j)) \cdot \surd(\tau_\eta(y)) + \surd(x) \cdot \surd(y)$.

Using 3.11.8, which involves an application of H3, we may add to this expression the term

$\sum_l \tau_\eta(x) \mathbb{L} \tau_\eta(y'_l) + \sum_j \tau_\eta(y) \mathbb{L} \tau_\eta(x'_j)$, which yields

$\tau_\eta(x) \| \tau_\eta(y) = \sum_i a_i(\tau_\eta(x_i) \| \tau_\eta(y)) + \sum_j \tau(\tau_\eta(x'_j) \| \tau_\eta(y)) +$

$+ \sum_k b_k(\tau_\eta(x) \| \tau_\eta(y_k)) + \sum_l \tau(\tau_\eta(x) \| \tau_\eta(y'_l)) +$

$+ \sum_{i,k} (a_i \mid b_k)(\tau_\eta(x_i) \| \tau_\eta(y_k)) + \surd(x) \cdot \surd(y) =$

$= \sum_i a_i(\tau_\eta(x_i \| y)) + \sum_j \tau(\tau_\eta(x'_j \| y)) + \sum_k b_k(\tau_\eta(x \| y_k)) + \sum_l \tau(\tau_\eta(x \| y'_l)) +$

$+ \sum_{i,k} (a_i \mid b_k)(\tau_\eta(x_i \| y_k)) + \surd(x) \cdot \surd(y)$ (by induction hypothesis) $= \tau_\eta(x \| y)$.

3.16 NOTE

The mapping τ_η is *not* a homomorphism w.r.t. $\mid$ or $\surd$. E.g., if $\gamma(a,b)$ is defined, then $\tau a \mid b = \gamma(a,b)$, while $\eta a \mid b = \delta$. Also the termination behaviour is different, viz. $\tau_\eta(\surd(\eta)) = \delta$ but $\surd(\tau_\eta(\eta)) = \varepsilon$. This is not so bad, however, as operators $\mid, \surd$ are only auxiliary operators, needed to define the merge operator, and τ_η *is* a homomorphism for merge.

3.17 NOTE

In the next section we will prove that ACPc is a **conservative extension** of ACP_η, and most of ACP$\surd$ and of ACP_τ, i.e. for all closed ACP_η-terms t,s we have

$$ACPc \vdash t=s \text{ iff } ACP_\eta \vdash t=s,$$

for all closed ACP$\surd$-terms t,s in which the operator ε_K does not appear, we have

$$ACPc \vdash t=s \text{ iff } ACP\surd \vdash t=s,$$

and for all closed ACP_τ-terms t,s in which the operator $\between$ does not appear, we have

$$ACPc \vdash t=s \text{ iff } ACP_\tau \vdash t=s.$$

4. THE GRAPH MODEL

We construct a model for ACPc consisting of equivalence classes of process graphs. This model will also contain *infinite* processes, processes that cannot be represented by a closed term. The way to talk about such processes algebraically, is by means of recursive equations, or, more generally, recursive specifications. We will not discuss recursive specifications in this paper, but refer the reader e.g. to [2, 3].

4.1 DEFINITIONS

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 $A\cup\{\delta,\eta,\varepsilon\}$, the set of constants (but there is no label τ). 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. $\mathbb{G}$ is the set of all process graphs. $\mathbb{0}$ is the trivial graph, just consisting of a single node and no edges. A **tree** is a graph in which the root has no incoming edge, and all other nodes have exactly one incoming edge. Note that a tree has no **cycles**, i.e. there is no series of edges that leads from a node back to the same node. $\mathbb{T}$ is the set of all nontrivial process trees.

A **path** π in a process graph g is a finite alternating sequence of connected nodes and edges of g. The **length** of π is the number of non-ε-edges in π. A node s of g is **reached** by the path π if π ends in s. If s is a node of graph g, then $(g)_s$ is the subgraph of g that consists of all nodes and edges that can be reached from s, with root s. Note that in a process *tree* every node is reached by exactly one path from the root. The **depth** $d(g)$ of a finite tree g is the length of its longest path. A node in a graph is an **endnode** if it has no outgoing edges. An edge is **intermediate** if it ends in a non-endnode.

A **u-step** in a graph from s to s' is an edge going from s to s' with label $u \in A\cup\{\delta,\eta,\varepsilon\}$, notation $s \rightarrow^u s'$; $\twoheadrightarrow^\varepsilon$ is the transitive and reflexive closure of $\rightarrow^\varepsilon$, so $s \twoheadrightarrow^\varepsilon s'$ if there is path of (≥ 0) ε-labeled edges, starting in s, and ending in s'. $\twoheadrightarrow^\varepsilon$ is called a **generalized ε-step**. Further, we will also need a **generalized ε/η-step** $\twoheadrightarrow^{\varepsilon/\eta}$: this is a path with edges labeled ε or η, of which the last one (if there is a last one) has label η. For more information about process graphs, see e.g. BAETEN, BERGSTRA & KLOP [1].

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 [11], MILNER [10] or BAETEN, BERGSTRA & KLOP [1]. The present definition is obtained by 'putting together' the ε-bisimulation of VRANCKEN [12] (also used in [3]) with the $r\eta$-bisimulation of [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 $\eta\varepsilon$-bisimulation** between g and h, notation $R: g \leftrightarrows_{r\eta\varepsilon} h$, iff

1. The roots of g and h are related.
2. If $R(s,t)$ and from s, we can do a generalized ε-step followed by an a-step to a node s' ($s \twoheadrightarrow^\varepsilon \rightarrow^a s'$) with $a\in A$ (so $a\neq\eta$, $a\neq\varepsilon$, $a\neq\delta$), then, in h, we can do a generalized ε/η-step $t \twoheadrightarrow^{\varepsilon/\eta} t^*$ to a node t^* with

R(s,t*), and from t*, we can do a generalized ε-step, followed by an a-step, followed by a generalized ε/η-step, to a node t' with R(s',t'), i.e. $t^* \twoheadrightarrow^{\varepsilon} \rightarrow^{a} \twoheadrightarrow^{\varepsilon/\eta} t'$. See fig. 1a.

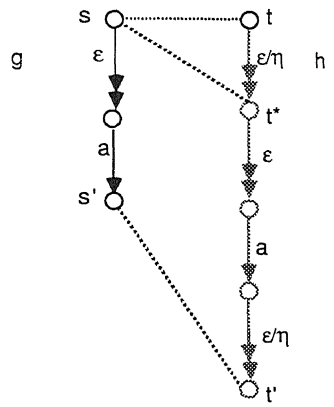

Fig. 1a.

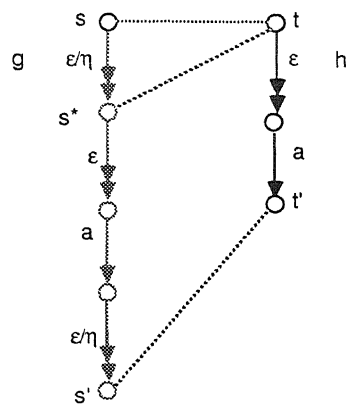

Fig. 1b.

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 $t \twoheadrightarrow^{\varepsilon} \rightarrow^{a} t'$ is a path in h with a∈A, then, in g, there are nodes s*,s' such that $s \twoheadrightarrow^{\varepsilon/\eta} s^* \twoheadrightarrow^{\varepsilon} \rightarrow^{a} \twoheadrightarrow^{\varepsilon/\eta} s'$ and R(s*,t), R(s',t'). See fig. 1b. In case (s,t) is the pair of roots, we must have s≡s* (the second part of the root condition).

4. If R(s,t) and $s \twoheadrightarrow^{\varepsilon} \rightarrow^{\eta} s'$ is a path in g, then, in h, there is a node t' such that $t \twoheadrightarrow^{\varepsilon/\eta} t'$ and R(s',t'). In case (s,t) is the pair of roots, the step $t \twoheadrightarrow^{\varepsilon/\eta} t'$ must contain at least one edge (the third part of the root condition).

5. Vice versa: if R(s,t) and $t \twoheadrightarrow^{\varepsilon} \rightarrow^{\eta} t'$ is a path in h, then, in g, there is a node s' such that $s \twoheadrightarrow^{\varepsilon/\eta} s'$ and R(s',t'). In case (s,t) is the pair of roots, the step $s \twoheadrightarrow^{\varepsilon/\eta} s'$ must contain at least one edge (the fourth part of the root condition).

6. If R(s,t), s' is an endpoint in g, and $s \twoheadrightarrow^{\varepsilon} s'$, then, in h, there are nodes t*,t' such that t' is an endpoint, $t \twoheadrightarrow^{\varepsilon/\eta} t^* \twoheadrightarrow^{\varepsilon} t'$ and R(s,t*). In case (s,t) is the pair of roots, we must have t≡t* (the fifth part of the root condition).

7. Vice versa: if R(s,t), t' is an endpoint in h, and $t \twoheadrightarrow^{\varepsilon} t'$, then, in g, there are nodes s*,s' such that s' is an endpoint, $s \twoheadrightarrow^{\varepsilon/\eta} s^* \twoheadrightarrow^{\varepsilon} s'$ and R(s*,t). In case (s,t) is the pair of roots, we must have s≡s* (the last part of the root condition).

A relation R between nodes of g and nodes of h is an **ηε-bisimulation** between g and h, $g \leftrightarrows_{\eta\varepsilon} h$, if we do not require the root condition in points 2-7.

Graphs g and h are **rηε-bisimilar**, $g \leftrightarrows_{r\eta\varepsilon} h$, if there is a rooted ηε-bisimulation between g and h; g and h are **ηε-bisimilar**, $g \leftrightarrows_{\eta\varepsilon} h$, if there is a ηε-bisimulation between g and h.

4.3 EXAMPLES

See fig. 2. We have a,b,c ∈ A, so ≠δ,ε,η.

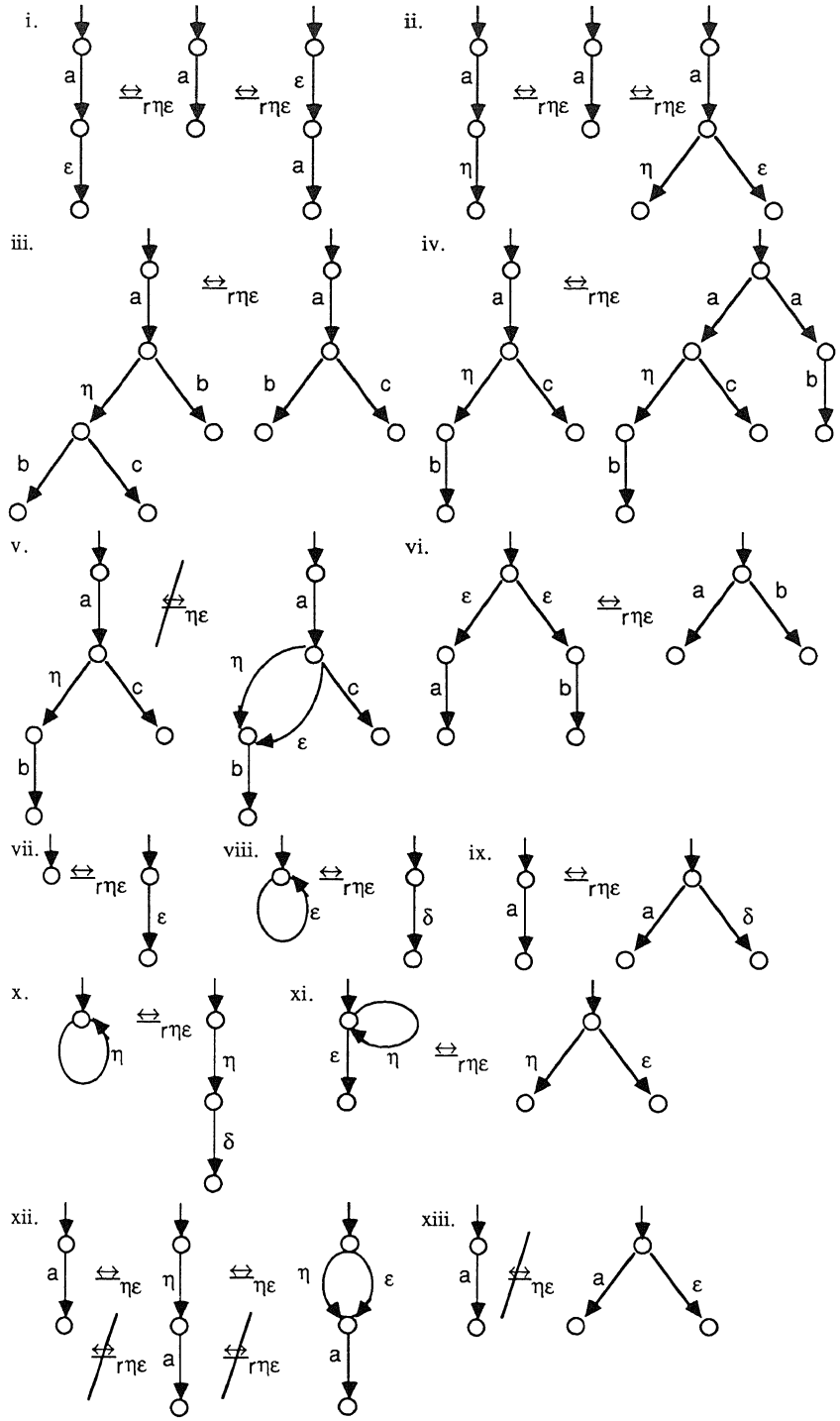

Fig. 2.

4.4 LEMMA
$\leftrightarrows_{\tau\eta\varepsilon}$ and $\leftrightarrows_{\eta\varepsilon}$ are equivalence relations on $\mathbb{G}$.
PROOF: Straightforward.

4.5 LEMMA
Each $\leftrightarrows_{\tau\eta\varepsilon}$-equivalence class contains a nontrivial process tree.
PROOF: As in [3].

4.6 $\mathbb{G}/\leftrightarrows_{\tau\eta\varepsilon}$ will be the domain of the graph model for ACPc. The interpretation of a constant $u \in A\cup\{\delta,\eta,\varepsilon\}$ is the equivalence class of the graph with two nodes and a single edge between them labeled u. The interpretation of the constant τ is the equivalence class of the graph with two nodes and two edges between them (with the same direction), one labeled η, the other labeled ε.
What remains is the definition of the operators of ACPc on $\mathbb{G}/\leftrightarrows_{\tau\eta\varepsilon}$. We will define these operators on $\mathbb{G}$ (the parallel operators only on $\mathbb{T}$) and will then show that $\leftrightarrows_{\tau\eta\varepsilon}$ is a congruence relation w.r.t. them. These definitions are also given in [3]. We repeat them here, but for comments and examples we refer the reader to [3].

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, that will be the root of $g+h$. Then, we add two edges labeled ε: from r to the root of g, and from r to the root of h.
2. $\cdot$. 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.
3. $\|$. The definition of the merge on $\mathbb{G}$ is rather complicated. Therefore, we will only define the merge on nontrivial process *trees*. Using lemma 4.5, this definition can be extended to $\mathbb{G}$.
If $g,h \in \mathbb{T}$, graph $g\|h$ is the cartesian product graph of graphs g and h, with 'diagonal' edges added for communication steps, and with non-ε-edges 'orthogonal' to an incoming ε-step turned into δ-steps. By this, we mean the following: if (s,t) is a node in $g\|h$, then it has the following outgoing edges ($u,v \in A\cup\{\delta,\eta,\varepsilon\}$, $a,b \in A$):
i. an edge $(s,t) \rightarrow^u (s',t)$ if $s \rightarrow^u s'$ is an edge in g, and $u = \varepsilon$ or h has no edge $t'' \rightarrow^\varepsilon t$;
ii. an edge $(s,t) \rightarrow^\delta (s',t)$ if $s \rightarrow^u s'$ is an edge in g, $u \neq \varepsilon$ and h has an edge $t'' \rightarrow^\varepsilon t$;
iii. an edge $(s,t) \rightarrow^v (s,t')$ if $t \rightarrow^v t'$ is an edge in h, and $u = \varepsilon$ or g has no edge $s'' \rightarrow^\varepsilon s$;
iv. an edge $(s,t) \rightarrow^\delta (s,t')$ if $t \rightarrow^v t'$ is an edge in h, $u \neq \varepsilon$ and g has an edge $s'' \rightarrow^\varepsilon s$;
v. an edge $(s,t) \rightarrow^{\gamma(a,b)} (s',t')$ if $s \rightarrow^a s'$ is an edge in g, $t \rightarrow^b t'$ is an edge in h and $\gamma(a,b)$ is defined (these are the *diagonal* edges).
The root of $g\|h$ is the pair of roots of g and h.
Edges $(s,t) \rightarrow^u (s',t)$ are called *vertical* edges, and edges $(s,t) \rightarrow^u (s,t')$ are *horizontal* edges.
4. $\lfloor\!\lfloor$. If $g,h \in \mathbb{T}$, graph $g\lfloor\!\lfloor h$ is obtained from graph $g\|h$ by turning all horizontal and diagonal edges, that are reachable from the root by a generalized ε-step, into δ-edges.
5. $\lfloor$. Similar to 4: If $g,h \in \mathbb{T}$, graph $g\lfloor h$ is obtained from graph $g\|h$ by turning all horizontal and vertical edges, that are reachable from the root by a generalized ε-step, and do not have label ε, or do have label ε but lead to an endpoint, into δ-edges.
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.
7. τ_η. If $g \in \mathbb{G}$, obtain $\tau_\eta(g)$ by adding an edge $s \rightarrow^\varepsilon t$ for each edge $s \rightarrow^\eta t$ in g.

This finishes the definition of the operators of ACPc on $\mathbb{G}$. Then we also have the operators on $\mathbb{G}/\leftrightarrows_{r\eta\varepsilon}$, if we use the following theorem 4.9.

4.8 NOTE

If g,h are process trees, $g \| h$ does not have to be a tree (but is equivalent to one). In VRANCKEN [12], the parallel operators $\|, \lfloor, |$ are defined on a wider class of graphs, a class which is closed under these operators. This makes proofs of statements about them much easier.

4.9 THEOREM

$\leftrightarrows_{r\eta\varepsilon}$ is a congruence relation on $\mathbb{G}$.

PROOF: Tedious, but straightforward. As an example, consider the case of $\|$.

So suppose $g,g',h,h' \in \mathbb{G}$ and $g \leftrightarrows_{r\eta\varepsilon} g'$, $h \leftrightarrows_{r\eta\varepsilon} h'$. We have to prove that $g \| h \leftrightarrows_{r\eta\varepsilon} g' \| h'$. Using lemma 4.5, we can suppose that g,g',h,h' are nontrivial process trees.

Take an $r\eta\varepsilon$-bisimulation R between g and g', such that R does not relate endpoints of intermediate ε-edges, and an $r\eta\varepsilon$-bisimulation S between h and h' with the same restriction (such bisimulations always exist, since we are dealing with trees). Let $R \times S$ be the cartesian product of R and S, i.e. $R \times S((s,t),(s',t'))$ iff $R(s,s')$ and $S(t,t')$ (s a node in g, s' in g', t in h, t' in h').

CLAIM: $R \times S$ is an $r\eta\varepsilon$-bisimulation between $g \| h$ and $g' \| h'$.

PROOF OF THE CLAIM: Tedious, but straightforward. As an example, consider the verification of condition 4.2.2 (without the root condition).

(1) Let $(s_1,t_1) \rightarrow^a (s_1,t_2)$ be a horizontal step in $g \| h$, with $a \in A$. Let $(s_0,t_0) \twoheadrightarrow^{\varepsilon} (s_1,t_1) \rightarrow^a (s_1,t_2)$, and $R \times S((s_0,t_0),(s'_0,t'_0))$. Then we must have that $s_0 \equiv s_1$, that the vertical component of $(s_0,t_0) \twoheadrightarrow^{\varepsilon} (s_1,t_1)$ is empty, for otherwise, the step $(s_1,t_1) \rightarrow^a (s_1,t_2)$ would be orthogonal to an incoming ε-edge, and would be turned into a δ-edge. Now $t_0 \twoheadrightarrow^{\varepsilon} t_1 \rightarrow^a t_2$ in h and $S(t_0,t'_0)$, hence we can find nodes t^*_0 and t'_2 such that $t'_0 \twoheadrightarrow^{\varepsilon/\eta} t^*_0 \twoheadrightarrow^{\varepsilon} \rightarrow^a \twoheadrightarrow^{\varepsilon/\eta} t'_2$ and $S(t_0,t^*_0)$ and $S(t_2,t'_2)$. This path can be 'lifted' to $g' \| h'$, since s'_0 cannot have an incoming ε-edge by the restriction on R, and we obtain $(s'_0,t'_0) \twoheadrightarrow^{\varepsilon/\eta} (s'_0,t^*_0) \twoheadrightarrow^{\varepsilon} \rightarrow^a \twoheadrightarrow^{\varepsilon/\eta} (s'_0,t'_2)$ and $R \times S((s_0,t_0),(s'_0,t^*_0))$ and $R \times S((s_0,t_2),(s'_0,t'_2))$.

(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 look at a path $(s_0,t_0) \twoheadrightarrow^{\varepsilon} (s_1,t_1) \rightarrow^c (s_2,t_2)$ such that $R \times S((s_0,t_0),(s'_0,t'_0))$. We consider the 'projections' of this path, i.e. there are paths $s_0 \twoheadrightarrow^{\varepsilon} s_1 \rightarrow^a s_2$ in g and $t_0 \twoheadrightarrow^{\varepsilon} t_1 \rightarrow^b t_2$ in h with $\gamma(a,b) = c$. Since $R(s_0,s'_0)$, there are nodes s^*_0 and s'_2 in g' such that $s'_0 \twoheadrightarrow^{\varepsilon/\eta} s^*_0 \twoheadrightarrow^{\varepsilon} \rightarrow^a \twoheadrightarrow^{\varepsilon/\eta} s'_2$ and $R(s_0,s^*_0)$ and $R(s_2,s'_2)$. Likewise, $t'_0 \twoheadrightarrow^{\varepsilon/\eta} t^*_0 \twoheadrightarrow^{\varepsilon} \rightarrow^b \twoheadrightarrow^{\varepsilon/\eta} t'_2$, $S(t_0,t^*_0)$ and $S(t_2,t'_2)$ for certain t^*_0, t'_2 in h'. We compose these paths in $g' \| h'$:

$$(s'_0,t'_0) \twoheadrightarrow^{\varepsilon/\eta} (s'_0,t^*_0) \twoheadrightarrow^{\varepsilon/\eta} (s^*_0,t^*_0) \twoheadrightarrow^{\varepsilon} \twoheadrightarrow^{\varepsilon} \rightarrow^c \twoheadrightarrow^{\varepsilon/\eta} \twoheadrightarrow^{\varepsilon/\eta} (s'_2,t'_2)$$

and $R \times S((s_0,t_0),(s^*_0,t^*_0))$ and $R \times S((s_2,t_2),(s'_2,t'_2))$. Since R does not relate endpoints of intermediate ε-edges, s'_0 does not have an incoming ε-edge, so the η-edges in $(s'_0,t'_0) \twoheadrightarrow^{\varepsilon/\eta} (s'_0,t^*_0)$ do not turn into δ-edges; since the last edge in this sequence is not an ε-edge, the η-edges in $(s'_0,t^*_0) \twoheadrightarrow^{\varepsilon/\eta} (s^*_0,t^*_0)$ do not turn into δ-edges (use the restriction on S in case the sequence is empty); next, the first sequence of η-edges in the sequence following $\rightarrow^c$ are orthogonal to a component of c, and the second sequence is either orthogonal to the last η-edge in the first sequence, or to the other component of c.

The remainder of the verification is not too hard.

4.10 THEOREM
$\mathbb{G}/\leftrightarrows_{\eta\varepsilon}$ is a model of ACPc.
PROOF: For every axiom of ACPc, it has to be checked that if we substitute process graphs for the variables, and use the definitions 4.6 and 4.7, then there exists a $\eta\varepsilon$-bisimulation between the two graphs resulting from both sides of the equality sign. The construction of these $\eta\varepsilon$-bisimulations is routine, tedious and omitted (cf. BERGSTRA & KLOP [5], 2.5). The only interesting cases concern the η and ε laws, of which instances are presented in examples 4.3.

4.11 REMARK
We also obtain models of ACPc, 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 $\mathbb{G}_\kappa/\leftrightarrows_{\eta\varepsilon}$. $\mathbb{G}/\leftrightarrows_{\eta\varepsilon}$ is the model $\mathbb{G}_{\aleph_0}/\leftrightarrows_{\eta\varepsilon}$. Also, the set $\mathbb{R}$ of all finite (or *regular*) process graphs modulo $\leftrightarrows_{\eta\varepsilon}$ and the set $\mathbb{F}$ of all finite and acyclic process graphs modulo $\leftrightarrows_{\eta\varepsilon}$ form models of ACPc.

4.12 In the sequel, we will show that these models are *complete* for closed ACPc-terms, i.e. if t,s are closed ACPc-terms, and graph(t) denotes the graph corresponding to the closed term t (following definitions 4.6 and 4.7), then graph(t) $\leftrightarrows_{\eta\varepsilon}$ graph(s) implies ACPc $\vdash$ t=s.
For the completeness proof, it is simplest if we write the terms in the form of basic terms according to the first inductive scheme of 3.9: this means that every subterm terminates with an ε-step, and that a subterm may have more than one ε-summand. If s is such a basic term, this means that graph(s) is a nontrivial finite process tree, with no intermediate ε-edges, and with all edges leading to an endnode having label ε. We call such process trees **basic trees**.
Some notation: from now on we write $x = y$ for ACPc $\vdash x = y$; if this holds we say that x is **ACPc-equal** to y. If A1,2 $\vdash x = y$ we write $x \equiv y$. We say x is an **ACPc-summand** of y if ACPc $\vdash y = x + y$.

4.13 PROPOSITION
i. If t is a basic term then graph(t) is a basic tree.
ii. If graph(t) $\equiv$ graph(s) for $s,t \in BT$, then $t \equiv s$.
iii. For any basic tree g, there is a basic term t with graph(t) $\equiv$ g.
PROOF: Easy.

4.14 DEFINITION
Let term be a function that maps a basic tree g onto a basic term t with graph(t) $\equiv$ g.
By proposition 4.13 we have term(graph(t)) $\equiv$ t for $t \in BT$.

4.15 PROPOSITION
If R is an $\eta\varepsilon$-bisimulation between process graphs g,h (not necessarily rooted), and R(s,t), then $(g)_s \leftrightarrows_{\eta\varepsilon} (h)_t$.
PROOF: R, restricted to the nodes of $(g)_s$ and $(h)_t$, will be an $\eta\varepsilon$-bisimulation.

4.16 PROPOSITION
Let g be a basic tree, and $a \in C$.
i. term(g) has a summand $s \equiv as'$ iff there is an edge root(g) $\rightarrow^a$ p in g with $a \cdot term((g)_p) \equiv s$.
ii. term(g) has a summand ε iff there is an edge root(g) $\rightarrow^\varepsilon$ p to an endnode p.

PROOF: Easy.

4.17 PROPOSITION
Let q be a node of a basic tree h, such that root(h) $\rightarrow^a \twoheadrightarrow^\eta$ q. Then term(h) = a·term($(h)_q$) + term(h) (i.e. a·term($(h)_q$) is an ACPc-summand of term(h)).
PROOF: By induction on the length of the path from root(h) to q. Let this length be n.
The induction base n=1 follows from proposition 4.16.
Now suppose root(h) $\rightarrow^a$ p $\twoheadrightarrow^\eta$ q such that the path from p to q has length $n \geq 1$, and the proposition is already proved for n. Then we can write p $\rightarrow^\eta \twoheadrightarrow^\eta$ q.
Thus term(h) = a·term($(h)_p$) + term(h) and term($(h)_p$) = η·term($(h)_q$) + term($(h)_p$).
Hence term(h) = a(η·term($(h)_q$) + term($(h)_p$)) + term(h) = (using H3)
= a·term($(h)_q$) + a(η·term($(h)_q$) + term($(h)_p$)) + term(h) = a·term($(h)_q$) + term(h).

4.18 PROPOSITION
For basic trees g,h we have:
i. If g $\leftrightarrows_{\eta\varepsilon}$ h then ACPc $\vdash$ a·term(g) = a·term(h) for each $a \in C$.
ii. If g $\leftrightarrows_{\eta\varepsilon}$ h then ACPc $\vdash$ term(g) = term(h).
PROOF: (i) will be proved with induction on d(g) + d(h). So suppose g $\leftrightarrows_{\eta\varepsilon}$ h, say R: g $\leftrightarrows_{\eta\varepsilon}$ h, and for any basic trees g',h' with d(g') + d(h') < d(g) + d(h) (i) is already proved.
CLAIM: One of the following statements holds:
I: term(h) = η·term(g) + term(h)
II: term(h) = term(g) + term(h)
III: a·term(h) = a·term(g) for $a \in C$.
PROOF OF THE CLAIM:
I: Suppose there is a node q in h with root(h) $\rightarrow^\eta \twoheadrightarrow^\eta$ q and R(root(g),q).
Then, by proposition 4.15, g $\leftrightarrows_{\eta\varepsilon}$ $(g)_{root(g)}$ $\leftrightarrows_{\eta\varepsilon}$ $(h)_q$ and d($(h)_q$) < d(h), so by induction η·term(g) = η·term($(h)_q$). Furthermore, using proposition 4.17, term(h) = η·term($(h)_q$) + term(h), and hence term(h) = η·term(g) + term(h).
II: Suppose there is no such node. We will prove that any summand s = ε or at of term(g) either is an ACPc-summand of term(h), or is ACPc-equal to η·term(h). If all summands of term(g) are ACPc-summands of term(h) we get II. The case that there are summands of term(g) ACPc-equal to η·term(h) will be considered in part III of this proof.
So let s = ε or s = as' be a summand of term(g).
CASE 1: s = ε. It follows from proposition 4.16 that there is an edge root(g) $\rightarrow^\varepsilon$ p in g to an endnode p. Since R: g $\leftrightarrows_{\eta\varepsilon}$ h, and h is a basic tree, there must be nodes q,q* in h, with q an endnode, such that root(h) $\twoheadrightarrow^\eta$ q* $\rightarrow^\varepsilon$ q and R(root(g),q*). By the assumption above (in the first sentence of II), root(h) = q*, so we are done using proposition 4.16.
CASE 2: s = as'. By proposition 4.16 there is an edge root(g) $\rightarrow^a$ p in g with a·term($(g)_p$) = s.
CASE 2.1: $a \in A$. Since R: g $\leftrightarrows_{\eta\varepsilon}$ h, and h is a basic tree, there must be nodes q* and q in h with root(h) $\twoheadrightarrow^\eta$ q* $\rightarrow^a \twoheadrightarrow^\eta$ q, R(root(g),q*) and R(p,q).
Hence, by proposition 4.15, $(g)_p$ $\leftrightarrows_{\eta\varepsilon}$ $(h)_q$, and since d($(g)_p$) < d(g) (and also d($(h)_q$) < d(h)), the induction hypothesis yields a·term($(g)_p$) = a·term($(h)_q$). By the assumption above root(h) = q* and proposition 4.17 gives term(h) = a·term($(h)_q$) + term(h). Thus term(h) = s + term(h).
CASE 2.2: a = η. Since R: g $\leftrightarrows_{\eta\varepsilon}$ h, and h is a basic tree, there must be a node q in h with root(h) $\twoheadrightarrow^\eta$ q and R(p,q). Hence $(g)_p$ $\leftrightarrows_\eta$ $(h)_q$ and since d($(g)_p$) < d(g), the induction hypothesis yields

$\eta\cdot term((g)_p) = \eta\cdot term((h)_q)$.
Now there are two possibilities: if $root(h) \neq q$, then $root(h) \rightarrow^\eta \twoheadrightarrow^\eta q$ and proposition 4.17 gives $term(h) = \eta\cdot term((h)_q) + term(h)$. Thus $term(h) = s + term(h)$.
On the other hand, if $root(h) = q$, then $s = \eta\cdot term((g)_p) = \eta\cdot term((h)_q) = \eta\cdot term(h)$.
CASE 2.3: $a = \delta$. In this case $term(h) = s + term(h)$ follows from laws A6 and A7.
III: Finally suppose that some summands of term(g) are ACPc-equal to $\eta\cdot term(h)$, while the others are ACPc-summands of term(h). Then, there is a term t such that $term(g) = \eta\cdot term(h) + t$ and $term(h) = t + term(h)$. Hence, using H2, $a\cdot term(g) = a(\eta(t + term(h)) + t) = a(t + term(h)) = a\cdot term(h)$, 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: $term(g) = \eta\cdot term(h) + term(g)$
B: $term(g) = term(h) + term(g)$
C: $a\cdot term(g) = a\cdot term(h)$, 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 $a \in C$
$a\cdot term(h) = a(\eta\cdot term(g) + term(h)) =$ (using H3)
$= a(\eta\cdot term(g) + term(h)) + a\cdot term(g) = a\cdot term(h) + a\cdot term(g)$.
Likewise, A implies $a\cdot term(g) = a\cdot term(g) + a\cdot term(h)$. Putting these two statements together yields $a\cdot term(g) = a\cdot term(h)$.
• Suppose that II and B hold. Then $term(g) = term(h) + term(g) = term(h)$, so $a\cdot term(g) = a\cdot term(h)$ for $a \in C$.
• Suppose that II and A hold. Then, for $a \in C$, $a\cdot term(g) = a(\eta(term(g) + term(h)) + term(g)) = a\cdot term(h)$, 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 $g \leftrightarrows_{\tau\eta\varepsilon} h$, say $R: g \leftrightarrows_{\tau\eta\varepsilon} h$. We will prove that any summand ε or as' of term(g) is an ACPc-summand of term(h) (and vice versa), which yields the desired result.
So let s be a summand of term(g).
CASE 1: $s = \varepsilon$. It follows from proposition 4.16 that there is an edge $root(g) \rightarrow^\varepsilon p$ in g to an endnode p. Since $R: g \leftrightarrows_{\tau\eta\varepsilon} h$, and h is a basic tree, there must be nodes q, q^* in h, with q an endnode, such that $root(h) \twoheadrightarrow^\eta q^* \rightarrow^\varepsilon q$ and $R(root(g), q^*)$. By the root condition, $root(h) = q^*$, so we are done using proposition 4.16.
CASE 2: $s = as'$. By proposition 4.16 there is an edge $root(g) \rightarrow^a p$ in g with $a\cdot term((g)_p) = s$.
CASE 2.1: $a \in A$. Since $R: g \leftrightarrows_{\tau\eta\varepsilon} h$, and h is a basic tree, there must be nodes q^* and q in h with $root(h) \twoheadrightarrow^\eta q^* \rightarrow^a \twoheadrightarrow^\eta q$, $R(root(g), q^*)$ and $R(p,q)$. Moreover, the rootedness condition gives $root(h) = q^*$. By proposition 4.15, $(g)_p \leftrightarrows_{\tau\eta\varepsilon} (h)_q$, and (i) yields $a\cdot term((g)_p) = a\cdot term((h)_q)$. Furthermore, proposition 4.17 gives $term(h) = a\cdot term((h)_q) + term(h)$, so $term(h) = s + term(h)$.
CASE 2.2: $a = \eta$. Since $R: g \leftrightarrows_{\tau\eta\varepsilon} h$, and h is a basic tree, there must be a node q in h with $root(h) \twoheadrightarrow^\eta q$ and $R(p,q)$. Moreover, the rootedness condition gives $root(h) \rightarrow^\eta \twoheadrightarrow^\eta q$. By proposition 4.15, $(g)_p \leftrightarrows_{\tau\eta\varepsilon} (h)_q$, and (i) yields $\eta\cdot term((g)_p) = \eta\cdot term((h)_q)$. Furthermore, proposition 4.17 gives $term(h) = \eta\cdot term((h)_q) + term(h)$, so $term(h) = s + term(h)$.

CASE 2.3, $a=\delta$, follows from A6 and A7.
This finishes the proof of part (ii).

4.19 THEOREM
ACPc is sound and complete for closed terms, with respect to $\mathbb{G}/\leftrightarrows_{\tau\eta\varepsilon}$, i.e.
for all closed ACPc-terms t,s we have: $graph(t) \leftrightarrows_{\tau\eta\varepsilon} graph(s) \Leftrightarrow ACPc \vdash t = s$.
PROOF: Direction $\Leftarrow$, the soundness, follows from theorem 4.10.
For direction $\Rightarrow$, the completeness, note that the elimination theorem 3.10 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 4.14: if $t,s \in BT$ and $graph(t) \leftrightarrows_{\tau\eta\varepsilon} graph(s)$, then
$t \equiv term(graph(t)) = term(graph(s)) \equiv s$.

4.20 THEOREM
ACP_η is sound and complete for closed terms, with respect to $\mathbb{G}/\leftrightarrows_{\tau\eta\varepsilon}$, i.e.
for all closed ACP_η-terms t,s we have: $graph(t) \leftrightarrows_{\tau\eta\varepsilon} graph(s) \Leftrightarrow ACP_\eta \vdash t=s$.
PROOF: If we only consider those process graphs in $\mathbb{G}/\leftrightarrows_{\tau\eta\varepsilon}$ that are nontrivial and have no ε-edges, then we have the model $\mathbb{G}/\leftrightarrows_{\tau\eta}$ of BAETEN & VAN GLABBEEK [2] (we do have to change the definition of + to an equivalent one that does not use ε-edges). In [2], soundness and completeness of closed ACP_η-terms w.r.t. this model was proven. This result transfers completely to the present case.
Note that ACP_η is *not* sound for all open terms, since the axiom H1 of ACP_η, $x\eta = x$, does not hold for $x \equiv \varepsilon$. However, if we replace H1 by $a\eta = a$, we do get soundness for open terms, and $\mathbb{G}/\leftrightarrows_{\tau\eta\varepsilon}$ becomes a model for ACP_η.

4.21 THEOREM
$ACP\surd$ is sound and complete for closed terms, that do not contain the ε_K-operator, with respect to $\mathbb{G}/\leftrightarrows_{\tau\eta\varepsilon}$, i.e. for all closed $ACP\surd$-terms t,s without ε_K we have:

$$graph(t) \leftrightarrows_{\tau\eta\varepsilon} graph(s) \Leftrightarrow ACP\surd \vdash t=s.$$

PROOF: Exactly as the proof of theorem 4.18 above, but much simpler, by omitting all η's from this chapter. Essentially, this proof is given in [3].
Note that $ACP\surd$ is sound for all open terms, if we omit the operator ε_K and its axioms. Thus, $\mathbb{G}/\leftrightarrows_{\tau\eta\varepsilon}$ becomes a model for $ACP\surd$.

4.22 THEOREM
ACP_τ is sound and complete for closed terms, that do not contain the $\between$-operator, with respect to $\mathbb{G}/\leftrightarrows_{\tau\eta\varepsilon}$, i.e. for all closed ACP_τ-terms t,s without $\between$ we have:

$$graph(t) \leftrightarrows_{\tau\eta\varepsilon} graph(s) \Leftrightarrow ACP_\tau \vdash t=s.$$

PROOF: We consider the graph model $\mathbb{F}_\tau/\leftrightarrows_{r\tau}$ for ACP_τ, as presented in BERGSTRA & KLOP [5]. $\mathbb{F}_\tau$ consists of all finite acyclic nontrivial process graphs with labels from $A\cup\{\delta,\tau\}$. The definition of rooted τ-bisimulation $\leftrightarrows_{r\tau}$ is like definition 4.2, with the following changes:
1. Omit all reference to ε-steps, so $s \twoheadrightarrow^{\varepsilon} t$ just amounts to $s \equiv t$, and $s \twoheadrightarrow^{\varepsilon/\eta} t$ becomes $s \twoheadrightarrow^{\eta} t$;
2. Change all η's into τ's;
3. Drop all requirements $R(s^*,t)$ and $R(s,t^*)$;
4. Drop the root conditions in 2,3,6,7, keep only the root conditions in 4,5.
Concerning the definition of the operators, we use the definition for + alluded to in 4.19, that does not involve ε-edges, but is equivalent to the present one, and we can use the same definitions for the other

operators, except for the communication merge $|$ (keeping in mind the changes 1 - 4 above). If g and h are two finite nontrivial process trees, then $g|h$ can be obtained as follows. First, obtain graph $g|^*h$ from graph $g\|h$ by turning all horizontal and vertical edges, that are reachable from the root by a generalized τ-step, and do not have label τ, or do have label τ but lead to an endpoint, into δ-edges. Then, $g|h$ is the sum of all subgraphs of $g|^*h$, that are reachable from the root by a generalized τ-step.

Let φ be the mapping from $\mathbb{F}_\tau$ into $\mathbb{G}$, that replaces each edge $s \to^\tau t$ by two edges $s \to^\varepsilon t$ and $s \to^\eta t$. In BERGSTRA & KLOP [5] it is shown that $\mathbb{F}_\tau/\leftrightarrows_{r\tau}$ is a sound and complete model for ACP_τ, w.r.t. closed terms, i.e. $graph_\tau(t) \leftrightarrows_{r\tau} graph_\tau(s) \Leftrightarrow ACP_\tau \vdash t = s$ for closed t,s, where $graph_\tau(t)$ is defined as graph(t), but w.r.t. the ACP_τ-operators.

Thus, we are done if we prove the following two claims:

CLAIM 1: For any $g,h \in \mathbb{F}_\tau$: $g \leftrightarrows_{r\tau} h \Leftrightarrow \varphi(g) \leftrightarrows_{r\eta\varepsilon} \varphi(h)$.

CLAIM 2: φ is a homomorphism w.r.t. the constants a,δ,τ and the operators $+,\cdot,\|,|,\partial_H$ and τ_I ($H,I \subseteq A$), so for $\|$-free closed terms t we have $\varphi(graph_\tau(t)) \leftrightarrows_{r\eta\varepsilon} graph(t)$.

PROOF OF CLAIM 1: $\Rightarrow$: Suppose $g \leftrightarrows_{r\tau} h$, take a relation R: $g \leftrightarrows_{r\tau} h$. We claim that R also is an $r\eta\varepsilon$-bisimulation between $\varphi(g)$ and $\varphi(h)$. For, suppose $s \twoheadrightarrow^\varepsilon s'' \to^a s'$ is a path in $\varphi(g)$. Since every ε-edge has a 'neighbouring' η-edge, $s \twoheadrightarrow^\tau s'' \to^a s'$ is a path in g. Since R is an $r\tau$-bisimulation, there are nodes t'',t' in h such that $t \twoheadrightarrow^\tau t'' \twoheadrightarrow^\tau \to^a \twoheadrightarrow^\tau t'$ is a path in h, and R(s'',t''), R(s',t'). By definition of φ, $t \twoheadrightarrow^\varepsilon t'' \twoheadrightarrow^\varepsilon \to^a \twoheadrightarrow^\eta t'$ is a path in $\varphi(h)$, and we can take $t^* \equiv t$. It is not hard to finish the verification.

$\Leftarrow$: Suppose $\varphi(g) \leftrightarrows_{r\eta\varepsilon} \varphi(h)$, take a relation R: $\varphi(g) \leftrightarrows_{r\eta\varepsilon} \varphi(h)$. Now we claim that R is an $r\tau$-bisimulation between g and h. For, suppose $s \to^a s'$ is an edge in g. Then $s \to^a s'$ also is an edge in $\varphi(g)$. Since R is an $r\eta\varepsilon$-bisimulation, there are nodes t^*,t' in $\varphi(h)$ such that $t \twoheadrightarrow^{\varepsilon/\eta} t^* \twoheadrightarrow^\varepsilon \to^a \twoheadrightarrow^{\varepsilon/\eta} t'$ is a path in $\varphi(h)$, and R(s,t*), R(s',t'). Hence, by definition of φ, $t \twoheadrightarrow^\tau \to^a \twoheadrightarrow^\tau t'$ is a path in h. Again, it is not hard to finish the verification.

PROOF OF CLAIM 2: The proof is easy for the constants a,δ,τ and the operators $+,\cdot,\partial_H$ and τ_I. The operators $\|$ and $|$ are defined on trees, so if $g \in \mathbb{F}_\tau$, we have to 'unshare' $\varphi(g)$, in the sense that whenever an edge $s \to^\tau t$ appears in g, we make two copies of $(\varphi(g))_t$, with root nodes t_1,t_2, and have edges $s \to^\eta t_1$ and $s \to^\varepsilon t_2$ in the tree constructed from $\varphi(g)$.

Now let $g,h \in \mathbb{F}_\tau$ be two process trees, then for the operator $\|$ we have to prove that

$$\varphi(g\|h) \leftrightarrows_{r\eta\varepsilon} tree(\varphi(g))\|tree(\varphi(h)),$$

where the operation tree is as just explained. Let R be the relation that relates each node (s,t) in $\varphi(g\|h)$ (s a node of g, t a node of h) to the corresponding node in $tree(\varphi(g))\|tree(\varphi(h))$, or, if there is more than one copy, to all copies which have no incoming ε-step. We claim that R is a $r\eta\varepsilon$-bisimulation. The crux of the verification is the following: if $(s,t) \to^\varepsilon (s',t) \to^a (s',t')$ is a path in $\varphi(g\|h)$ ($a \in A$), then the corresponding a-edge in $tree(\varphi(g))\|tree(\varphi(h))$ is turned into a δ-edge by definition 4.7.3. However, we can take the path $(s,t) \to^a (s,t') \to^\eta (s',t')$ in $tree(\varphi(g))\|tree(\varphi(h))$ for the bisimulation. The rest of the verification is left to the reader.

For the operator $|$, we have to prove that

$$\varphi(g|h) \leftrightarrows_{r\eta\varepsilon} tree(\varphi(g))|tree(\varphi(h)).$$

First remark that $\varphi(g|h) \leftrightarrows_{r\eta\varepsilon} \psi(\varphi(g|^*h))$, where $\psi(g)$ is obtained from g by changing all η-edges, that are reachable from the root by a generalized ε-step, into δ-edges. Now there is a clear correspondence between the nodes of $\psi(\varphi(g|^*h))$ and $tree(\varphi(g))|tree(\varphi(h))$ so that R: $\psi(\varphi(g|^*h)) \leftrightarrows_{r\eta\varepsilon} tree(\varphi(g))|tree(\varphi(h))$ can be defined as above. Also the verification that R is a bisimulation follows the case of $\|$.

We remark that ACP_τ is sound for all open terms, apart from the law T1 ($x\tau = x$) that does not hold for

ε (and could be replaced by $a\tau = a$ and $\tau\tau = \tau$), and the laws for the operator $\between$.

4.23 EXAMPLE
The mapping φ in the proof of 4.22 is *not* a homomorphism w.r.t. the operator $\between$. For, let $g = \tau$, and $h = a$ ($a \in A$). Then the graph $\varphi(g \between h)$ in fig. 3a does not $r\eta\varepsilon$-bisimilate with the graph $tree(\varphi(g)) \between tree(\varphi(h))$ in fig. 3b.

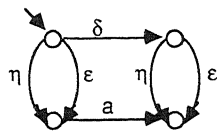

Fig. 3a.

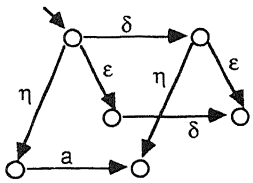

Fig. 3b.

We see that when we compare the process τ in ACP_τ with the process $\eta + \varepsilon$ in ACPc, they interact in the same way with the operators except for the auxiliary operator $\between$. On the other hand, we found in 3.15 that when we compare τ with η in ACPc by means of the homomorphism τ_η, we also get the same interaction with the operators, except for the auxiliary operators $|$ and $\surd$. Both comparisons however work for the merge operator, which is defined by means of these auxiliary operators. This is the same duality that we found in [2].

4.24 THEOREM
ACPc is a **conservative extension** of ACP_η, and most of $ACP\surd$ and of ACP_τ, i.e. for all closed ACP_η-terms t,s we have

$$ACPc \vdash t=s \text{ iff } ACP_\eta \vdash t=s,$$

for all closed $ACP\surd$-terms t,s in which the operator ε_K does not appear, we have

$$ACPc \vdash t=s \text{ iff } ACP\surd \vdash t=s,$$

and for all closed ACP_τ-terms t,s in which the operator $\between$ does not appear, we have

$$ACPc \vdash t=s \text{ iff } ACP_\tau \vdash t=s.$$

PROOF: Combine 4.19 with 4.20, 4.21 and 4.22.

4.25 THEOREM
Let g be a finitely branching process graph, and let h be a finitely branching process graph with no η-label, such that $\tau_\eta(g) \leftrightarrows_{r\eta\varepsilon} h$. Then $g \leftrightarrows_{r\eta\varepsilon} h$.
PROOF: Let g,h be as stated. $\tau_\eta(g)$ is obtained from g by adding an ε-step for each η-step, but has the same nodes. Let R be a $r\eta\varepsilon$-bisimulation between $\tau_\eta(g)$ and h. We claim that R is also a $r\eta\varepsilon$-bisimulation between g and h. We only give the most difficult part of the verification.
Let $R(s,t)$ and let $t \twoheadrightarrow^{\varepsilon} \rightarrow^{a} t'$ be a path in h with $a \in A$. Since R is a $r\eta\varepsilon$-bisimulation, there are nodes s^*,s' in g such that $s \twoheadrightarrow^{\varepsilon/\eta} s^* \twoheadrightarrow^{\varepsilon} \rightarrow^{a} \twoheadrightarrow^{\varepsilon/\eta} s'$ is a path in $\tau_\eta(g)$ and $R(s^*,t)$, $R(s',t')$. Some of the ε-steps in this path might not appear in g. All of such steps however, have a 'neighbouring' η-step. We now take the path in g with, where necessary, ε-steps replaced by the neighbouring η-steps. Thus $s \twoheadrightarrow^{\varepsilon/\eta} s'' \twoheadrightarrow^{\varepsilon} \rightarrow^{a} \twoheadrightarrow^{\varepsilon/\eta} s'$ is a path in g for some node s" (that may be further along the path than node s^*). Since $s \twoheadrightarrow^{\varepsilon/\eta} s''$, there is a node t" in h such that $t \twoheadrightarrow^{\varepsilon/\eta} t''$ and $R(s'',t'')$. But h contains no η-labels, so the number of η-steps in $\twoheadrightarrow^{\varepsilon/\eta}$ must be 0, whence $t'' = t$. This means $R(s'',t)$ holds.

4.26 Theorem 4.25 allows us to formulate the following proof principle:

if x is an ACPc-process and y is an ACP$\surd$-process,

and $\tau_\eta(x) = y$, then $x = y$.

We call this the **Two-tiered Abstraction Principle (TAP)**. It follows from the completeness of the graph model that TAP is derivable for closed terms.

4.27 REMARK

The principles RDP (the Recursive Definition Principle), RSP (the Recursive Specification Principle), AIP$^-$ (the Approximation Induction Principle) and HAR (the η Abstraction Rule) from BAETEN & VAN GLABBEEK [2] also hold in the graph model $\mathbb{G}/\leftrightarrows_{\eta\varepsilon}$, and the same proofs and definitions can be used. Note that the fairness principle HAR has a nice formulation in this setting (equivalent to the one in [2]) (cf. 4.3.xi):

if $x = ix + \varepsilon$, and $i \in I$, then $\eta_I(x) = \tau$.

We conjecture that the principles FAP (the Fresh Atom Principle) and LR$^-$ (the Limit Rule) from BAETEN & VAN GLABBEEK [3] hold in the graph model $\mathbb{G}/\leftrightarrows_{\eta\varepsilon}$. In the setting of ACPc, the Limit Rule says that any equation without abstraction operators (η_I, τ_η, τ_I), that holds for all basic terms, holds for all processes.

REFERENCES

[1] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP, *On the consistency of Koomen's fair abstraction rule,* Theor. Comp. Sci. 51 (1/2), pp. 129 - 176, 1987.

[2] J.C.M. BAETEN & R.J. VAN GLABBEEK, *Another look at abstraction in process algebra,* in: Proc. 14th ICALP, Karlsruhe (Th. Ottman, ed), Springer LNCS 267, pp. 84 - 94, 1987. Full version: report CS-R8701, Centre for Math. & Comp. Sci., Amsterdam 1987.

[3] J.C.M. BAETEN & R.J. VAN GLABBEEK, *Merge and termination in process algebra,* in: Proc. 7th FST&TCS, Pune, India (K.V. Nori, ed.), Springer LNCS 287, pp. 153 - 172, 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 & E.-R. OLDEROG, *Failures without chaos: a new process semantics for fair abstraction,* in: Proc. IFIP Conf. on Formal Description of Progr. Concepts - III, Ebberup 1986 (M. Wirsing, ed.), North-Holland, Amsterdam, pp. 77 - 103, 1987.

[7] C.A.R. HOARE, *Communicating sequential processes,* Prentice-Hall International 1985.

[8] C.P.J. KOYMANS & J.L.M. VRANCKEN, *Extending process algebra with the empty process ε,* report LGPS 1, Dept. of Philosophy, State University of Utrecht, The Netherlands 1985.

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

[10] 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.

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

[12] J.L.M. VRANCKEN, *The algebra of communicating processes with empty process,* report FVI 86-01, Dept. of Comp. Sci., Univ. of Amsterdam 1986.