Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.C.M. Baeten, J.A. Bergstra

Global renaming operators in concrete process algebra

Department of Computer Science Report CS-R8521 September

Global renaming operators in concrete process algebra

J.C.M. Baeten

J.A. Bergstra
Centre for Mathematics and Computer Science, Amsterdam.

Renaming operators are introduced in concrete process algebra (concrete means that abstraction and silent moves are not considered). Examples of renaming operators are given: encapsulation, pre-abstraction and localization. We show that renamings enhance the defining power of concrete process algebra by using the example of a queue. We give a definition of the trace set of a process, see when equality of trace sets implies equality of processes, and use trace sets to define the restriction of a process. Finally, we describe processes with actions that have a side effect on a state space and show how to use this for a translation of computer programs into process algebra.

1980 Mathematics Subject Classification: 68B10, 68C01, 68D25, 68F20.
1982 CR Categories: F.1.1, F.1.2, F.3.2, F.4.3. 69F11, 69F12, 69F32, 69F43
Key Words & Phrases: process algebra, concurrency, renaming operator, trace set.
Note: This report will be submitted for publication elsewhere. Partial support received from the European Communities under ESPRIT project no. 432, An Integrated Formal Approach to Industrial Software Development (Meteor).

INTRODUCTION

In this paper, we describe concrete process algebra. Concrete process algebra is an extension of ACP, the algebra of communicating processes (see BERGSTRA & KLOP [4]). Concrete process algebra does not consider silent moves or abstraction as is done in abstract process algebra (see BERGSTRA & KLOP [5]). The main advantages of not considering abstraction are that it leads to a clearer, and less problematic theory, with an easy to understand axiomatization, that is amenable to a term rewriting analysis and that can be studied using initial algebra semantics. Also, concrete process algebra is the starting point for designing a programming language. It is very useful for specification of processes or protocols, not so much for a verification formalism. Another article about processes without abstraction, and with essentially the same semantics, is DE BAKKER & ZUCKER [3].

In concrete process algebra, we introduce renaming operators. In §2, we define simple renaming operators, called relabelings in CCS (see MILNER [14]), and give three examples of such operators, namely the encapsulation operator (used to shield off a process, to prohibit communications with the environment), the pre-abstraction operator (used to obtain a small degree of abstraction within concrete process algebra), and the localization operator (which allows us to 'view' a process while it is interacting with an environment, or, in other words, allows us to focus on some actions and forget about others).

In §3, we look at the defining power of renamings. We give a specification of a queue in concrete process algebra with renamings, and show that a queue cannot be defined in concrete process algebra without renamings, thus showing that the defining power of concrete process algebra is increased by adding renamings.

In §4, we define a renaming operator with a memory, namely the restriction operator, that restricts a process to a set of possible execution traces. Before we define the restriction operator, we first give a short introduction to the theory of trace sets (for more information, see REM [16]), in which we prove that two processes with identical trace sets, that do not deadlock and are deterministic, must in fact be equal (also see ENGELFRIET [12]). Then we define the restriction operator, and use it in combination with the localization operator to show that in a context (or environment) we can restrict a process to the set of 'localized' traces. In our view, this theorem constitutes an important interface

Report CS-R8521
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

between trace theory and process algebra.

In §5, we introduce the state space of a process, and talk about actions that have a side effect on the state. We implement this with a generalized renaming operator, namely the state operator (for a different approach, see the theory of nonuniform processes in De Bakker & Zucker [3]). We use the state operator to discuss processes having shared variables, and to (mechanically) translate a given computer program into process algebra. We see that this operator can be very useful in the design of a programming language that is based on concrete process algebra. We finish by giving a different specification of the queue.

Table of contents

1. Concrete process algebra

In this section, we describe the axiomatic theory of concrete process algebra. This theory extends the theory ACP (the algebra of communicating processes) as described in Bergstra & Klop [4]. In this paper, we do not consider silent moves (or τ-steps) or abstraction as is done in ACP$_\tau$ (see Bergstra & Klop [5]).

1.1 Atomic actions:. concrete process algebra starts with a set of atomic actions A.
We will assume that A is *finite*, that A contains two special elements δ (for deadlock) and t (for hidden step), and that a *communication function* $\gamma: A \times A \rightarrow A$ is given with the following properties:

1. γ is *commutative*, $\forall a,b \in A\ \gamma(a,b) = \gamma(b,a)$
2. γ is *associative*, $\forall a,b,c \in A\ \gamma(\gamma(a,b),c) = \gamma(a,\gamma(b,c))$.
3. δ is a *neutral element*, $\forall a \in A\ \gamma(a,\delta) = \delta$
4. t does not communicate, $\forall a \in A\ \gamma(a,t) = \delta$.

If a and b are two atomic actions, then $\gamma(a,b)$ is the result of the communication between a and b, the result of executing a and b simultaneously. The *communication merge* $|$ will extend γ to the set of all processes. If $\gamma(a,b) = \delta$, we say a and b *do not communicate.*

Next we define the signature Σ of concrete process algebra. We have three sorts: A, the set of atomic actions was defined in 1.1; P is the set of processes, the subject of investigation, and contains A, and finally $\mathcal{Q}$, the set of subsets of $A-\{\delta\}$, is the set of alphabets.
Functions $+, \cdot, \|, \mathbin{\|\!\_}, |, \partial_H, \pi_n$, and constant δ are discussed in Bergstra & Klop [4] (π_n is called $()_n$ there), and α and t are discussed in Baeten, Bergstra & Klop [1].

1.2 Signature Σ

1. *Sorts*:	A	(see 1.1)
	P	(set of processes; $A \subseteq P$)
	$\mathcal{Q}$	($\mathcal{Q}=Pow(A-\{\delta\})$)
2. *Functions*:	$+: P \times P \rightarrow P$	(alternative composition or sum)
	$\cdot: P \times P \rightarrow P$	(sequential composition or product)
	$\|: P \times P \rightarrow P$	(parallel composition or merge)
	$\mathbin{\|\!\_}: P \times P \rightarrow P$	(left-merge)
	$|: P \times P \rightarrow P$	(communication merge)

	$\partial_H : P \to P$	(encapsulation; $H \subseteq A - \{t\}$)
	$\pi_n : P \to P$	(projection; $n > 0$)
	$\alpha : P \to \mathcal{Q}$	(alphabet function)
3. *Constants*:	$\delta \in A$	(deadlock)
	$t \in A$	(hidden step)

1.3. Equations
Concrete process algebra deals with statements of the form

$$x = y$$

called *equations;* $x,y \in P$.
We use letters $a,b,c,...$ for elements of A, letters $x,y,z,...$ for arbitrary processes, and we use capital letters $X,Y,Z,...$ for variables, ranging over P (often called *formal variables,* since we will use them in specifications to define processes, not, like x,y,z, in quantified statements about processes).

$$e(\vec{X})$$

is an equation with variables among $\vec{X}$, and

$$e(\vec{x})$$

is the same equation with processes $\vec{x}$ substituted for variables $\vec{X}$. Often, we want to focus on *one* of the variables, so writing

$$e(x, -)$$

means that equation e holds for x and a fixed set of other processes.

1.4. Specifications
A *(recursive) specification* E is a set of equations $\{e_j : j \in J\}$ (J an index set), with e_j of the form

$$X_j = t_j(\vec{X}),$$

t_j is a term with variables from $\{X_j : j \in J\}$, and J has a distinguished element j_0.
Process x is a *solution* of E if $E(x, -)$, i.e. substituting x for X_{j_0} (and other processes for the other variables) gives $e_j(x, -)$ for all $j \in J$.
x is *(recursively) definable* if there is a specification E such that $E(y, -) \Leftrightarrow x = y$.
(For these definitions, also see BAETEN, BERGSTRA & KLOP [2]).

1.5. DEFINITION: The set of *finite closed process expressions, FCPE*, is defined inductively:
1. $A \subseteq FCPE$;
2. $x \in FCPE \;\&\; a \in A \;\Rightarrow ax \in FCPE$
3. $x,y \in FCPE \;\Rightarrow x+y \in FCPE$

The set *FCPE* will allow us to use induction in proofs (when combined with the limit rule) and recursion in definitions (on the standard model).

Next we define a notion of guardedness (taken from BAETEN, BERGSTRA & KLOP [2]). Specifications must be guarded in order to prove that they have unique solutions (see 1.9).

1.6. DEFINITION: Let t be an open term, possibly containing variables. An occurrence of a variable X in t is *guarded* if t has a subterm of the form aM, with $a \in A$ and this X occurs in M; otherwise, the occurrence is *unguarded.*
Let $E = \{e_j : j \in J\}$ be a specification.

Define $X_i \xrightarrow{u} X_j \Leftrightarrow X_j$ occurs unguarded in t_j (the right-hand side of e_j), and
E is guarded $\Leftrightarrow \xrightarrow{u}$ is well-founded (i.e. there is no infinite sequence $X_{j_1} \xrightarrow{u} X_{j_2} \xrightarrow{u} \ldots$).

1.7. Axioms

The axioms for concrete process algebra are presented in table 1. We use the following abbreviations: ACP=A1-7+C1-3+CM1-9+D1-4; PR=PR1-4; AB=AB1-6; CPA=ACP+HA+PR +AB+AIP+LR and CPA_γ = CPA + C4, so in table 1 we have CPA_γ + RDP.

$x+y = y+x$	A1	$\pi_n(a)=a$	PR1
$x+(y+z) = (x+y)+z$	A2	$\pi_1(ax)=a$	PR2
$x+x = x$	A3	$\pi_{n+1}(ax)=a\pi_n(x)$	PR3
$(x+y)z = xz+yz$	A4	$\pi_n(x+y)=\pi_n(x)+\pi_n(y)$	PR4
$(xy)z = x(yz)$	A5		
$x+\delta = x$	A6		
$\delta x = \delta$	A7		
$a \mid b = b \mid a$	C1	$\alpha(\delta)=\varnothing$	AB1
$(a \mid b) \mid c = a \mid (b \mid c)$	C2	$\alpha(a)=\{a\}$ if $a \neq \delta$	AB2
$\delta \mid a = \delta$	C3	$\alpha(\delta x)=\varnothing$	AB3
$x \parallel y = x \mathbin{\lfloor\!\!\lfloor} y + y \mathbin{\lfloor\!\!\lfloor} x + x \mid y$	CM1	$\alpha(ax)=\{a\} \cup \alpha(x)$ if $a \neq \delta$	AB4
$a \mathbin{\lfloor\!\!\lfloor} x = ax$	CM2	$\alpha(x+y)=\alpha(x) \cup \alpha(y)$	AB5
$(ax) \mathbin{\lfloor\!\!\lfloor} y = a(x \parallel y)$	CM3	$\alpha(x)=\bigcup_{n=1}^{\infty} \alpha(\pi_n(x))$	AB6
$(x+y) \mathbin{\lfloor\!\!\lfloor} z = x \mathbin{\lfloor\!\!\lfloor} z + y \mathbin{\lfloor\!\!\lfloor} z$	CM4		
$(ax) \mid b = (a \mid b)x$	CM5		
$a \mid (bx) = (a \mid b)x$	CM6		
$(ax) \mid (by) = (a \mid b)(x \parallel y)$	CM7		
$(x+y) \mid z = x \mid z + y \mid z$	CM8	$\dfrac{E \text{ guarded}}{\exists x\ E(x, -)}$	RDP
$x \mid (y+z) = x \mid y + x \mid z$	CM9		
$\partial_H(a) = a$ if $a \notin H$	D1		
$\partial_H(a) = \delta$ if $a \in H$	D2		
$\partial_H(x+y) = \partial_H(x)+\partial_H(y)$	D3	$\dfrac{\forall n\ \pi_n(x)=\pi_n(y)}{x=y}$	AIP
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$	D4		
$x \mid y \mid z=\delta$	HA	$\dfrac{\forall t \in FCPE\ e(t, -)}{e(x, -)}$	limit rule (LR)
$a \mid b=\gamma(a,b)$	C4		

Table 1

1.8. COMMENTS. For a discussion of axioms ACP, see BERGSTRA & KLOP [4]. The *handshaking axiom* HA implies that all (proper) communications are binary. Axiom C4 says that the communication merge $\mid$ extends the communication function γ given in 1.1. Axioms PR1-4 define the projection operator π_n, for $n \in \mathbb{N}, n > 0$. Their intuitive meaning is that π_n 'cuts off' a process at depth n, $\pi_n(x)$ stops after executing n steps. Axioms AB1-6 define the alphabet of a process, and were introduced and discussed in BAETEN, BERGSTRA & KLOP [1].
The *Recursive Definition Principle* (RDP) states that every guarded specification has a solution, and the *Approximation Induction Principle* (AIP) states that two processes are equal if all their projections are equal. For RDP and AIP, see BAETEN, BERGSTRA & KLOP [2]. Finally, the *limit rule* is new here. It states that if an equation holds for all finite processes (more precisely, for all elements of FCPE), then it holds for all processes. It may be, that the limit rule is derivable from the other axioms of concrete process algebra. The limit rule allows us to prove identities by induction, since FCPE is defined recursively.

1.9. Definition. The *Recursive Specification Principle* (RSP) is the following rule:

$$\boxed{\dfrac{E(x,-)\quad E(y,-)\quad E \text{ guarded}}{x=y}\ (\text{RSP})}$$

1.10. Lemma: RSP *holds in concrete process algebra.*

Proof. See Baeten, Bergstra & Klop [2].
Since some observations made in this proof will be used more often, we will state these here. Let $E = \{e_j : j \in J\}$ be a guarded recursive specification with solution x (a solution always exists by RDP). Thus, we can substitute x for variable E_{j_0} (j_0 the distinguished element of J) and other processes $\{x_j : j \in J - \{j_0\}\}$ for the other variables. Now, by repeatedly substituting terms t_j for x_j in term t_{j_0}, we obtain a term $t^1_{j_0}$ in which all x_j occur guarded. Then, we see that $\pi_1(t^1_{j_0})$, and therefore $\pi_1(x)$, does not depend on the choice of the x_j ($j \in J$), but is equal to some term in FCPE. In general, for each $n \geqslant 1$, we can obtain an expansion $t^n_{j_0}$ of t_{j_0} in which all x_j are n times guarded, so that $\pi_n(x)$ is equal to some term in FCPE. Thus, if x and y are two solutions of a guarded recursive specification E, we find $\pi_n(x) = \pi_n(y)$ for all $n \geqslant 1$, so $x = y$ by AIP.

1.11. Theorem. *The following identities hold in concrete process algebra:*

Standard Concurrency		*Conditional Axioms*	
$(x \mathbin{\|\!\!\lfloor} y) \mathbin{\|\!\!\lfloor} z = x \mathbin{\|\!\!\lfloor} (y \| z)$	SC1	$\dfrac{\alpha(x) \mid (\alpha(y) \cap H) \subset H}{\partial_H(x \| y) = \partial_H(x \| \partial_H(y))}$	CA1
$(x \mid y) \mathbin{\|\!\!\lfloor} z = x \mid (y \mathbin{\|\!\!\lfloor} z)$	SC2		
$x \mid y = y \mid x$	SC3		
$x \| y = y \| x$	SC4	$\dfrac{\alpha(x) \cap H = \varnothing}{\partial_H(x) = x}$	CA3
$x \mid (y \mid z) = (x \mid y) \mid z$	SC5		
$x \| (y \| z) = (x \| y) \| z$	SC6		
		$\dfrac{H = H_1 \cup H_2}{\partial_H(x) = \partial_{H_1} \circ \partial_{H_2}(x)}$	CA5

Table 2

Expansion Theorem	
$x_1 \| x_2 \| \cdots \| x_n = \sum\limits_{1 \leqslant i \leqslant n} x_i \mathbin{\|\!\!\lfloor} \vec{x}^{j} + \sum\limits_{1 \leqslant i < j \leqslant n} (x_i \mid x_j) \mathbin{\|\!\!\lfloor} \vec{x}^{j,j}$	*ET*

(here $\vec{x}^{j} = \mathop{\|}\limits_{\substack{1 \leqslant k \leqslant n \\ k \neq i}} x_k$ and $\vec{x}^{j,j} = \mathop{\|}\limits_{\substack{1 \leqslant k \leqslant n \\ k \neq i, k \neq j}} x_k$)

Proof. Identities SC1-6 are proved in Bergstra & Klop [6], CA1,3,5 in Baeten, Bergstra & Klop [1], and ET in Bergstra & Tucker [10] for all terms in FCPE. The general identities then follow by applying the limit rule.

1.12. Initial algebra. Suppose we have a guarded finite recursive specification $E = \{e_j : 1 \leqslant j \leqslant n\}$.

By RDP+RSP, E has a unique solution in concrete process algebra. Now if Σ is the signature of concrete process algebra defined in 1.2, let $\Sigma(x_1,...,x_n)$ denote the signature Σ with extra constants $x_1,...,x_n \in P$. Then, the *initial algebra.*

$$\mathbb{A} = T_I(\Sigma(x_1,...,x_n),\ CPA_\gamma + E(x_1,...,x_n))$$

will exist, because $CPA_\gamma + E(x_1,...,x_n)$ is a positive conditional system.
(Of course, we could add constants for more than one specification.)

1.13. EXAMPLES: 1. Suppose $\gamma(a,b) = \delta$ for all $a,b \in A$. Take $a,b \in A$, and consider the initial algebra $\mathbb{A} = T_I(\Sigma(x,y,z,w),\quad CPA_\gamma + \{x = (a+b)x,\quad y = ay,\ z = bz,\ w = y\|z\})$. Then $CPA_\gamma \vdash w = y\|z = \quad y⌊\!⌊z + z⌊\!⌊y + y|z = \quad (ay)⌊\!⌊z + (bz)⌊\!⌊y + \delta = \quad a(y\|z) + b(z\|y) = (a+b)(y\|z) = (a+b)w$ (use induction plus limit rule to show $y|z = \delta$), so $RSP \vdash x = w$.
2. Suppose $a \in A$, and consider the initial algebra $\mathbb{A} = T_I(\Sigma(x,y), CPA_\gamma + \{x = ay, y = ax\})$. Note $RSP \vdash x = y$.

The existence of initial algebras shows that concrete process algebra is consistent. To show however, that there exists a non-trivial model, we need a result from BAETEN, BERGSTRA & KLOP [2]:

1.14. THEOREM. $\mathbb{G} = \mathbb{G}_{\aleph_0}(A - \{\delta\}, \{\delta, \downarrow\})/\leftrightarrow_\delta$, *the set of all finitely branching, rooted directed multigraphs, with edges labeled by elements of* $A - \{\delta\}$, *and endpoints labeled by* δ *or* $\downarrow$, *modulo bisimulation, is a model of concrete process algebra.*
Moreover, each finite element of $\mathbb{G}$ *is the interpretation of an element of FCPE, and each infinite element of* $\mathbb{G}$ *is the unique solution in* $\mathbb{G}$ *of a guarded recursive specification.*

1.15. COROLLARY. If $\mathbb{A}$ is an initial algebra of concrete process algebra as defined in 1.12, then $\mathbb{A}$ is a subalgebra of $\mathbb{G}$.

1.16. COROLLARY. If $x \in \mathbb{G}$, then for each $n \geqslant 1$ there is a term $s \in FCPE$ such that $\mathbb{G} \models \pi_n(x) = s$.

PROOF. From the proof of 1.10.

2. RENAMINGS

In this section, we define global renaming operators in concrete process algebra, and consider three examples of renaming operators, namely the encapsulation operator ∂_H, the pre-abstraction operator t_I and the localization operator ν_B^t. The localization operator will again be used in section 4.

2.1. DEFINITION. If $a \in A$, and $H \subseteq A - \{\delta\}$, then the *renaming operator* a_H will rename all elements of H into a. This renaming operator was introduced in process algebra in BERGSTRA, KLOP & OLDEROG [7]. To be more precise, we extend the signature with operators

$$a_H : P \to P \quad (a \in A,\ H \subseteq A - \{\delta\}),$$

and add axioms

$a_H(b) = b$	if $b \notin H$	RN1
$a_H(b) = a$	if $b \in H$	RN2
$a_H(x+y) = a_H(x) + a_H(y)$		RN3
$a_H(xy) = a_H(x) \cdot a_H(y)$		RN4

Table 3.

We still have the existence of initial algebras (as defined in 1.12) for this extended system.

2.2. EXAMPLE I: ENCAPSULATION The simplest example of a renaming operator is of course the *encapsulation operator* $\partial_H = \delta_H$ (for $H \subseteq A - \{\delta, t\}$). Its usefulness is demonstrated in every paper on the algebra of communicating processes. Usually, we are dealing with the merge of a number of processes, that can communicate, and we shield them off from the outside by encapsulation, i.e. we set the communication 'halves' equal to δ. Thus, if x is the merge of a number of processes, set $H = \{a \in \alpha(x) : \exists b \in \alpha(x)\ \gamma(a,b) \neq \delta\}$, and we consider $\partial_H(x)$.
For example, if $\gamma(a,b) = c \neq \delta$, and $a \neq c, b \neq c$, then $\partial_{\{a,b\}}(a \| b) = c$.

2.3. EXAMPLE II: PRE-ABSTRACTION
We do not consider silent moves or abstraction in concrete process algebra, but using a special constant $t \in A$ we can capture part of abstraction by the renaming operator t_I (for $I \subseteq A - \{\delta\}$), which we call the *pre-abstraction operator.* Note that when we use an encapsulation operator ∂_H, we always require $t \notin H$, so that $\partial_H(t) = t$. In 2.4, we explain some notation for distributed systems, which we use in 2.5-7 to give an example of the use of t_I.

2.4. DISTRIBUTED SYSTEMS. Suppose we have a number of locations, and a number of channels linking them. We assume that at each location a certain process is executed, and that these processes can communicate via the channels, thus obtaining a communication network. These communications will consist of the transferal of a piece of data. So suppose we have a finite set of data D (in practice, $D = \{0,1\}$), and we have communication channels $1,2,...,k$. Then we have the following atomic actions:
$r_i(d)$ = *read d along channel i* ($d \in D, 1 \leqslant i \leqslant k$);
$s_i(d)$ = *send d along channel i* ($d \in D, 1 \leqslant i \leqslant k$);
$c_i(d)$ = *communicate d along channel i* ($d \in D, 1 \leqslant i \leqslant k$),
and on these atomic actions, we define the communication function as follows:
$\gamma(r_i(d), s_i(d)) = \gamma(s_i(d), r_i(d)) = c_i(d)$ ($d \in D, 1 \leqslant i \leqslant k$); $\gamma(a,b) = \delta$ in all other cases.

2.5. DEFINITION. We consider the following communication network:

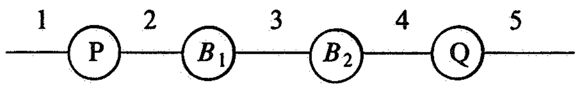

fig. 1

(so we represent locations by circles, with the name of its process inside, and channels by lines).
The processes P, Q, B_1, B_2 are given by the following FCPE terms:

$$P = \sum_{d \in D} r_1(d) \cdot \sum_{e \in D} r_1(e) \cdot s_2(d) \cdot s_2(e)$$

$$Q = \sum_{d \in D} r_4(d) \cdot \sum_{e \in D} r_4(e) \cdot s_5(f(d,e))$$

(here $f: D \times D \rightarrow D$ is some given function)

$$B_i = \sum_{d \in D} r_{i+1}(d) \cdot s_{i+2}(d) \cdot \sum_{e \in D} r_{i+1}(e) \cdot s_{i+2}(e) \quad (i = 1,2).$$

Thus, P is a two-place buffer that works only once, B_1 and B_2 are one-place buffers that work twice, and Q transforms incoming data using f. Put $H = \{r_i(d), s_i(d) : d \in D, i = 2,3,4\}$, so ∂_H will encapsulate all internal communications.

2.6. **Lemma.** $\partial_H(P \| B_1 \| B_2 \| Q) =$
$= \sum_{d \in D} r_1(d) \sum_{e \in D} r_1(e) c_2(d) c_3(d)(c_2(e) c_4(d) + c_4(d) c_2(e)) c_3(e) c_4(e) s_5(f(d,e))$.

Proof. Since we have binary communication, we can use the expansion theorem proved in §1, so we start with an action from one of the processes, or a communication between two of them. Only the first step we will write out in full.

$$\begin{aligned}
&\partial_H(P \| B_1 \| B_2 \| Q) = \\
&= \partial_H(\sum_{d \in D} r_1(d)((\sum_{e \in D} r_1(e) s_2(d) s_2(e)) \| B_1 \| B_2 \| Q) + \\
&+ \sum_{d \in D} r_2(d)(P \| (s_3(d) \sum_{e \in D} r_2(e) s_3(e)) \| B_2 \| Q) + \\
&+ \sum_{d \in D} r_3(d)(P \| B_1 \| (s_4(d) \sum_{e \in D} r_3(e) s_4(e)) \| Q) + \\
&+ \sum_{d \in D} r_4(d)(P \| B_1 \| B_2 \| (\sum_{e \in D} r_4(e) s_5(f(d,e)))) + \\
&+ \delta) = \\
&= \sum_{d \in D} r_1(d) \cdot \partial_H((\sum_{e \in D} r_1(e) s_2(d) s_2(e)) \| B_1 \| B_2 \| Q) + \delta + \delta + \delta = \\
&= \sum_{d \in D} r_1(d) \cdot \sum_{e \in D} r_1(e) \cdot \partial_H((s_2(d) s_2(e)) \| B_1 \| B_2 \| Q) = \\
&= \sum_{d \in D} r_1(d) \cdot \sum_{e \in D} r_1(e) \cdot c_2(d) \cdot \partial_H(s_2(e) \| (s_3(d) \sum_{f \in D} r_2(f) s_3(f)) \| B_2 \| Q) = \\
&= \sum_{d \in D} r_1(d) \cdot \sum_{e \in D} r_1(e) \cdot c_2(d) \cdot c_3(d) \cdot \partial_H(s_2(e) \| (\sum_{f \in D} r_2(f) s_3(f)) \| (s_4(d) \sum_{g \in D} r_3(g) s_4(g)) \| Q) = \\
&= \sum_{d \in D} r_1(d) \sum_{e \in D} r_1(e) c_2(d) c_3(d)(c_2(e) \cdot \partial_H(s_3(e) \| (s_4(d) \sum_{g \in D} r_3(g) s_4(g)) \| Q) + \\
&+ c_4(d) \cdot \partial_H(s_2(e) \| (\sum_{f \in D} r_2(f) s_3(f)) \| (\sum_{g \in D} r_3(g) s_4(g)) \| (\sum_{h \in D} r_4(h) s_5(f(d,h))))) = \\
&= \sum_{d \in D} r_1(d) \sum_{e \in D} r_1(e) c_2(d) c_3(d)(c_2(e) c_4(d) \cdot \partial_H(s_3(e) \| (\sum_{g \in D} r_3(g) s_4(g)) \| (\sum_{h \in D} r_4(h) s_5(f(d,h)))) + \\
&+ c_4(d) c_2(e) \cdot \partial_H(s_3(e) \| (\sum_{g \in D} r_3(g) s_4(g)) \| (\sum_{h \in D} r_4(h) s_5(f(d,h))))) = \\
&= \sum_{d \in D} r_1(d) \sum_{e \in D} r_1(e) c_2(d) c_3(d)(c_2(e) c_4(d) + c_4(d) c_2(e)) \cdot \\
&\cdot c_3(e) \cdot \partial_H(s_4(e) \| (\sum_{h \in D} r_4(h) s_5(f(d,h)))) = \\
&= \sum_{d \in D} r_1(d) \sum_{e \in D} r_1(e) c_2(d) c_3(d)(c_2(e) c_4(d) + c_4(d) c_2(e)) \cdot c_3(e) c_4(e) s_5(f(d,e)).
\end{aligned}$$

2.7. We can simplify the expression derived in 2.6 considerably, if we use pre-abstraction. Put $I = \{c_i(d) : d \in D,\ i = 2,3,4\}$, the set of all internal communications, then

$$\begin{aligned}
&t_I \circ \partial_H(P \| B_1 \| B_2 \| Q) = \\
&= \sum_{d \in D} r_1(d) \sum_{e \in D} r_1(e) \cdot t \cdot t(t \cdot t + t \cdot t) \cdot t \cdot t \cdot s_5(f(d,e)) = \\
&= \sum_{d \in D} r_1(d) \sum_{e \in D} r_1(e) \cdot t^6 \cdot s_5(f(d,e)).
\end{aligned}$$

Thus, we only see the input and output actions, and we no longer see the alternatives in formula 2.6.

2.8. EXAMPLE III: LOCALIZATION
Let $B \subseteq A - \{\delta\}$ be such that

1. for all $b_1, b_2 \in B$ and all $a_1, a_2 \in A$, if $\gamma(b_1,a_1) = \gamma(b_2,a_2) \neq \delta$, then $b_1 = b_2$ (we express this by saying that communication is one-to-one on B); and
2. for all $b \in B$ and all $a \in A$ $\gamma(b,a) \notin B$ (if we put, for $B_1, B_2 \subseteq A$,

$$B_1 | B_2 = \{\gamma(b_1,b_2) : b_1 \in B_1, b_2 \in B_2\} - \{\delta\},$$

then this requirement is equivalent to:

$$B | A \cap B = \varnothing.$$

If B satisfies these two requirements, then γ has an 'inverse' on $B|A$, i.e. if $c \in B|A$, then there is exactly one $b \in B$ so that an $a \in A$ exists with $\gamma(b,a) = c$. In this case, we put $b = \gamma^{-1}(c)$.
Now we can define the localization operator.

2.9. DEFINITION. Let $B \subseteq A - \{\delta, t\}$ satisfy 2.8.1 and 2.8.2, and suppose $B|A = \{c_1, ..., c_n\}$. We define the *localization operator* ν_B^t by:

$$\nu_B^t = \gamma^{-1}(c_1)_{\{c_1\}} \circ \cdots \circ \gamma^{-1}(c_n)_{\{c_n\}} \circ t_{A-(A|B \cup B \cup \{\delta\})}$$

It is easy to see that ν_B^t has the following properties:

1.	$\nu_B^t(\gamma(b,a)) = b$	if $b \in B, a \in A, \gamma(b,a) \neq \delta$
2.	$\nu_B^t(b) = b$	if $b \in B$
3.	$\nu_B^t(c) = t$	if $c \in A - (A|B \cup B \cup \{\delta\})$
4.	$\nu_B^t(\delta) = \delta$	
5.	$\nu_B^t(x+y) = \nu_B^t(x) + \nu_B^t(y)$	
6.	$\nu_B^t(xy) = \nu_B^t(x) \cdot \nu_B^t(y)$.	

Intuitively, we think of ν_B^t as the operator that 'localizes' a process to actions from B, so that, in a context, typically a merge of communicating processes, we can focus on some actions and (pre-) abstract from others.
We give an example of the use of localization in 2-10,11 and will discuss this example again in §4.

2.10. DEFINITION. We consider the following communication network

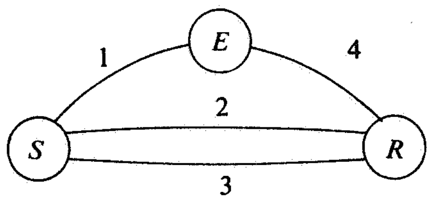

fig.2

Think of S as a *sender,* R as a *receiver* and E as an *environment.*
We define S and R as the unique solutions in concrete process algebra of the following two guarded recursive specifications:

$$\boxed{\begin{array}{l} S = \sum\limits_{d\in D} r_1(d)s_2(d)r_3(ack)s_1(ack)S \\ R = \sum\limits_{d\in D} r_2(d)s_4(d)s_3(ack)R \end{array}}$$

Here we have a special element *ack* denoting acknowledgement (it is easiest to take $ack \notin D$), so S sends a $d \in D$ to R, receives an acknowledgement, and then sends the next d; R receives the data and sends back an acknowledgement.
S and R also communicate with the environment E; so E can send data along 1, receive an acknowledgement along 1 or receive data along 4. Thus we can put

$$\boxed{E = (\sum\limits_{d\in D} s_1(d) + \sum\limits_{d\in D} r_4(d) + r_1(ack))E}$$

2.11. But, we have the feeling that E cannot do any of these things at any time: first we must have a $s_1(d)$, then a $r_4(d)$, and then a $r_1(ack)$, before a next $s_1(d')$ can follow.
We can express this by using the localization operator.
We put $H = \{s_i(d), r_i(d) : d \in D, i \in \{1,2,3,4\}\}$, and look at

$$\boxed{\nu^t_{\alpha(E)} \circ \partial_H(E \| S \| R)}$$

so in the process $\partial_H(E\|S\|R)$ we focus on actions from E, we localize to E. It is easily seen that $\alpha(E) = \{s_1(d), r_4(d) : d \in D\} \cup \{r_1(ack)\}$, and that $\alpha(E)$ satisfies 2.8.1 and 2.8.2. We see

$$\begin{aligned}
&\nu^t_{\alpha(E)} \circ \partial_H(E\|S\|R) = \\
&= \nu^t_{\alpha(E)}(\sum_{d\in D} c_1(d)\cdot\partial_H(E\|(s_2(d)r_3(ack)s_1(ack)S)\|R) = \\
&= \sum_{d\in D} s_1(d)\cdot\nu^t_{\alpha(E)}(c_2(d)\cdot\partial_H(E\|(r_3(ack)s_1(ack)S\|(s_4(d)s_3(ack)R)) = \\
&= \sum_{d\in D} s_1(d)t\cdot\nu^t_{\alpha(E)}(c_4(d)\cdot\partial_H(E\|(r_3(ack)s_1(ack)S)\|(s_3(ack)R)) = \\
&= \sum_{d\in D} s_1(d)\cdot t\cdot r_4(d)\cdot\nu^t_{\alpha(E)}(c_3(ack)\cdot\partial_H(E\|(s_1(ack)S)\|R) = \\
&= \sum_{d\in D} s_1(d)\cdot t\cdot r_4(d)\cdot t\cdot\nu^t_{\alpha(E)}(c_1(ack)\cdot\partial_H(E\|S\|R)) = \\
&= \sum_{d\in D} s_1(d)\cdot t\cdot r_4(d)\cdot t\cdot r_1(ack)\cdot\nu^t_{\alpha(E)}\circ\partial_H(E\|S\|R),
\end{aligned}$$

so that the actions from E indeed occur in the right order.

3. Defining power of renamings

3.1. Suppose we want to give a recursive specification of a queue Q with input channel 1 and output channel 2 as in fig.3.

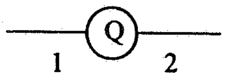

fig. 3

An *infinite* guarded specification can be given by the following equations:

$$Q = Q_\epsilon = \sum_{d\in D} r_1(d)\cdot Q_d$$

$$Q_{\sigma * d} = s_2(d)\cdot Q_\sigma + \sum_{e \in D} r_1(e)\cdot Q_{e * \sigma * d}$$

(for any word $\sigma \in D^*$ and any $d \in D$)

(see BERGSTRA & TIURYN [9]).

Now we look for a *finite* recursive specification of Q, in the context of handshaking communication. Then we may assume that $r_1(d)$ and $s_2(d)$ are not the result of communications, for we need the following interactions with the environment:

$$\gamma(r_i(d),s_i(d)) = c_i(d) \ \ (i = 1,2;\ d \in D).$$

That is why we want to specify Q under the condition

$$\alpha(Q) \cap \mathrm{ran}\,(\gamma) = \varnothing \text{ (since } \alpha(Q) = \{r_1(d),s_2(d) : d \in D\}).$$

3.2. Next, we will prove two theorems:

1. we cannot define Q by a finite guarded recursive specification in concrete process algebra without renaming under the condition $\alpha(Q) \cap \mathrm{ran}\,(\gamma) = \varnothing$.
2 we can define Q by a finite guarded recursive specification in concrete process algebra with renaming operators (as defined in §2) under the condition $\alpha(Q) \cap \mathrm{ran}(\gamma) = \varnothing$.

We will prove theorem 1 in 3.10. First we need a number of intermediate results.

We define the following axiom systems:
PA = A1-5 + M1-4 + PR1-4
(here M 1 is axiom $x \| y = x \mathbin{\|\!\_} y + y \mathbin{\|\!\_} x$ and M2-4 = CM2-4) and
PA_δ = PA + A6,7.

3.3. LEMMA. *Q is not definable by a finite guarded recursive specification in PA.*

PROOF. see BERGSTRA & TIURYN [9].

3.4. DEFINITION. Let $x \in P$. We say x *does not deadlock* (or $\neg DL(x)$) if for each $n \geqslant 1$, there is a term $s \in FCPE$ such that $\pi_n(x) = s$ and δ does not occur in s.

3.5. LEMMA. *Suppose process x is definable by a finite guarded specification in* PA_δ *and x does not deadlock. Then x is definable by a finite guarded specification in* PA.

PROOF. Let $E = \{e_j : 1 \leqslant j \leqslant n\}$ be a guarded recursive specification in PA_δ defining x (so $x = t_1(x,x_2,...,x_n)$ for some $x_2,...,x_n$). We define a theory PA^*, intermediate between PA and PA_δ, as follows:
the set of PA *-terms is defined inductively:

1. any $a \in A - \{\delta\}$ and any variable X is a PA *-term;
2. if s,t are PA *-terms, then so are $s+t, s\cdot t, s\|t, s \mathbin{\|\!\_} t$ and $s\cdot\delta$.

The *axioms of PA^** are:
$\mathrm{PA}^* = \mathrm{PA} + \{a\delta \mathbin{\|\!\_} x = ax\delta\}$.
Note that applying a PA*-axiom to a PA*-term will yield a PA*-term.
Note also that CPA $\vdash a\delta \mathbin{\|\!\_} x = ax\delta$ (induction on $FCPE$ plus limit rule). Now we can assume that all right hand sides of the equations in E are PA*-terms (if some are not, apply axioms A6 and A7, and if an equation $X_j = \delta$ appears, substitute δ for occurrences of X_j in right hand sides, and leave out equation $X_j = \delta$). Let E' be the specification obtained from E by leaving out all occurring δ.
Then E' is a finite guarded specification in PA, and we claim that E' also defines x.
To see that x is a solution of E', let $n \geqslant 1$ be given. By 1.10, there is an expansion t_1^n of t_1 in which all x_j are n times guarded, so that we can reduce, in PA_δ, $\pi_n(t_1^n)$ to a term in $FCPE$. Likewise, for E' there is an expansion t'^n_1 of t'_1, so that $\pi_n(t'^n_1)$ reduces in PA to a s'_n in $FCPE$. Now the reduction

from $\pi_n(t'^n_1)$ to s'_n in PA can be exactly transcribed to a reduction from $\pi_n(t^n_1)$ to a $s_n \in FCPE$ in PA* (sometimes using the PA*-axiom instead of CM2).
Since x is a solution of E we have $\pi_n(x) = s_n$. But s_n is a PA*-term, and so A6 and A7 cannot be applied to s_n, so since x does not deadlock, δ cannot occur in s_n. But that means that $s_n = s'_n$, so $\pi_n(t^n_1) = s_n = s'_n = \pi_n(t'^n_1)$, and x is a solution of E'.

3.6. Corollary. *Q is not definable by a finite guarded recursive specification in PA_δ.*

Proof. That Q does not deadlock can be seen using the infinitary specification given in 3.1. Now use 3.3 and 3.5.

3.7. Lemma. *Let x be a solution of the guarded recursive specification E. Then $\partial_H(x)$ is a solution of $\partial_H(E)$, where $\partial_H(E)$ is obtained from E by replacing each right hand side t_j by $\partial_H(t_j)$ (where $H \subseteq A - \{\delta\}$).*

Proof. Use 1.10 and the observation

$$\text{CPA} \vdash \pi_n \circ \partial_H(x) = \partial_H \circ \pi_n(x).$$

3.8. Lemma. *Suppose process x is definable by a finite guarded specification in* ACP$_\gamma$+PR *and $\alpha(x) \cap \mathrm{ran}(\gamma) = \varnothing$. Then x is definable by a finite guarded specification in* PA$_\delta$. (ACP$_\gamma$ = ACP+C4)

Proof. Let E be a finite guarded recursive specification in ACP$_\gamma$+PR defining x. Put $H = A\,|\,A = \mathrm{ran}\,\gamma - \{\delta\}$, then by 3.7 $\partial_H(E)$ defines $\partial_H(x)$. But $\partial_H(x) = x$, by applying rule CA3 (see 1.11). But it is not hard to see that applying ∂_H to an open (ACP$_\gamma$+PR)-term amounts to leaving out (i.e. set equal to δ) all communication (sub)terms of the form $x\,|\,y$, and has the same effect as having a trivial communication function γ with $\gamma(a,b) = \delta$ for all $a,b \in A$. The theory ACP$_\gamma$+PR with trivial γ is the same as theory PA$_\delta$.

3.9. Corollary. *Q is not definable by a finite guarded recursive specification in* ACP$_\gamma$+PR, *if we require $\alpha(Q) \cap \mathrm{ran}(\gamma) = \varnothing$.*

3.10. Theorem. *Q is not definable by a finite guarded recursive specification in concrete process algebra, if we require $\alpha(Q) \cap \mathrm{ran}(\gamma) = \varnothing$.*

Proof. This immediately follows from 3.9, since any specification in concrete process algebra uses the signature of ACP$_\gamma$+PR, so is a (ACP$_\gamma$+PR)-specification.

3.11. Theorem. *Q is definable by a finite guarded recursive specification in concrete process algebra with renaming operators, such that $\alpha(Q) \cap \mathrm{ran}(\gamma) = \varnothing$.*

Proof. Let A contain atoms $r_1(d), s_2(d), l(d)$ and $u(d)$, for each $d \in D$, and suppose

$$\gamma(l(d), l(d)) = u(d) \quad (d \in D)$$

are the only non-trivial communications.
Suppose $D = \{d_1, \ldots, d_n\}$, and define $s_{2\{u\}} = s_2(d_1)_{\{u(d_1)\}} \circ s_2(d_2)_{\{u(d_2)\}} \circ \cdots \circ s_2(d_n)_{\{u(d_n)\}}$, and $l_{\{s_2\}} = l(d_1)_{\{s_2(d_1)\}} \circ \cdots \circ l(d_n)_{\{s_2(d_n)\}}$, two renaming operators, and $H = \{l(d) : d \in D\}$. We claim we can define Q by:

$$\boxed{\begin{array}{l} Q = \sum_{d\in D} r_1(d)\cdot s_{2_{\{u\}}} \circ \partial_H(l_{\{s_2\}}(Q) \| s_2(d)\cdot Z) \\ Z = \sum_{d\in D} l(d)\cdot Z \end{array}}$$

To show this specification is correct, define R_σ, for $\sigma \in D^*$, inductively:
$R_\epsilon = Q$ (as given above)
$R_{\sigma * d} = s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \| s_2(d)Z)$.
First we need the following observation:
$R_\sigma = s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \| Z)$.
We prove this by RSP, showing that both sides satisfy the same equations: we have, on the one hand

$$R_\epsilon = \sum_{d\in D} r_1(d)\cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\epsilon) \| s_2(d)Z) =$$

$$= \sum_{d\in D} r_1(d)\cdot R_d, \text{ and}$$

$$R_{\sigma * d} = s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \| s_2(d)Z) =$$

$$= s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \lfloor\!\lfloor s_2(d)Z) + s_{2\{u\}} \partial_H(s_2(d)Z \lfloor\!\lfloor l_{\{s_2\}}(R_\sigma))$$

(since $s_2(d)$ does not communicate) =

$$= s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(\sum_{e\in D} r_1(e))[l_{\{s_2\}}(R_{e*\sigma}) \| s_2(d)Z] +$$

(by induction)

$$+ \, l_{\{s_2\}}(s_2(f))[l_{\{s_2\}}(R_{\sigma' * f}) \| s_2(d)Z])$$

(if $\sigma = \sigma' * f$; if $\sigma = \epsilon$, this term doesn't appear)+

$$+ \, s_2(d)\cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \| Z) =$$

$$= \sum_{e\in D} r_1(e)\cdot s_{2\{u\}}(\partial_H(l_{\{s_2\}}(R_{e*\sigma}) \| s_2(d)Z) + \delta) +$$

$$+ \, s_2(d)\cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \| Z) =$$

$$= \sum_{e\in D} r_1(e)\cdot R_{e*\sigma*d} + s_2(d)\cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \| Z).$$

On the other hand, we have

$$s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\epsilon) \| Z) =$$

$$= s_{2\{u\}} \circ \partial_H(\sum_{d\in D} r_1(d)\cdot [l_{\{s_2\}}(s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(Q) \| s_2(d)Z)) \| Z] +$$

$$+ \, l(d)\cdot [l_{\{s_2\}}(R_\epsilon) \| Z]) =$$

$$= \sum_{d\in D} r_1(d)\cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_d) \| Z), \text{ and}$$

$$s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_{\sigma * d}) \| Z) =$$

$$= s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma)) \| s_2(d)Z) \| Z) =$$

$$= s_{2\{u\}} \circ \partial_H([l_{\{s_2\}}(\sum_{e\in D} r_1(e)[s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_{e*\sigma})) \| s_2(d)Z)]) +$$

$$+ \, l_{\{s_2\}}(s_2(d)[s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \| Z)])] \| Z) =$$

$$= s_{2\{u\}} \circ \partial_H(\sum_{e\in D} r_1(e)(l_{\{s_2\}}(R_{e*\sigma*d}) \| Z) +$$

$$+\ s_{2\{u\}}\circ\partial_H(u(d)(s_{2\{u\}}\circ\partial_H(l_{\{s_2\}}(R_\sigma)\|Z)\|Z)) =$$
$$= \sum_{e\in D} r_1(e)\cdot s_{2\{u\}}\circ\partial_H(l_{\{s_2\}}(R_{e*\sigma*d})\|Z)+$$
$$+\ s_2(d)\cdot s_{2\{u\}}\circ\partial_H(s_{2\{u\}}\circ\partial_H(l_{\{s_2\}}(R_\sigma)\|Z)\|Z).$$

Therefore, we have shown $R_\sigma = s_{2\{u\}}\circ\partial_H(l_{\{s_2\}}(R_\sigma)\|Z)$, and so the equations for R_σ simplify to:

$$R_\epsilon = \sum_{d\in D} r_1(d)\cdot R_d$$
$$R_{\sigma*d} = \sum_{e\in D} r_1(e)\cdot R_{e*\sigma*d} + s_2(d)\cdot R_\sigma.$$

But that means that the R_σ satisfy the equations for Q_σ in 3.1, so by RSP $R_\sigma = Q_\sigma$, in particular $R_\epsilon = Q_\epsilon$, so the equations above indeed define the queue Q. Finally, $\alpha(Q)\cap \text{ran}(\gamma) = \varnothing$ is obvious.

4. Traces and restriction

In this section, we define the trace set (set of execution paths) of a process. It is well-known that in concrete process algebra, two processes with identical trace sets need not be equal (consider e.g. processes $a(b+c)$ and $ab+ac$). We will define for a process, what it means that it does not deadlock, and when it is deterministic, and then we show that if two processes have the same trace set, do not deadlock and are deterministic, then they must be equal. Next, we define a restriction operator, that can limit a process to a set of possible traces, and we give an example of the use of this operator by again considering example 2.10,11.

4.1. Note. From now on, our discussion will take place in the standard model $\mathbb{G}$ (see 1.14), but everything will work equally well in an initial algebra $\mathbb{A}$ as defined in 1.12. The crucial property we need is 1.16: each projection must be a finite term.

4.2. Deadlock behaviour. We define a predicate DL on $\mathbb{G}$ as follows:
on FCPE, we define DL inductively by
1. $DL(\delta)$
2. $DL(x) \Rightarrow DL(ax+y) \quad (a\in A-\{\delta\},\ x,y\in FCPE)$;
and on $\mathbb{G}$, we define
3. $DL(x) \Leftrightarrow \exists n \quad DL(\pi_n(x))$.

4.3. Lemma. *The following hold on* $\mathbb{G}$:
1. $\neg DL(a)$ if $a \in A-\{\delta\}$,
2. $\neg DL(x) \Rightarrow \neg DL(ax) \quad (a\in A-\{\delta\},\ x\in\mathbb{G})$,
3. $\neg DL(x)\&\neg DL(y) \Rightarrow \neg DL(x+y) \quad (x,y\in\mathbb{G})$.

Proof. Easy.

4.4. Theorem. *Let* $a\in A-\{\delta\}$, $H\subseteq A-\{\delta\}$ *and* $x\in\mathbb{G}$, *then* $DL(a_H(x)) \Leftrightarrow DL(x)$.

Proof. $\Rightarrow$: suppose $\neg DL(x)$. Take $n\geqslant 1$. Then $\neg DL(\pi_n(x))$. Let $s\in FCPE$ such that $\pi_n(x) = s$. Then $\neg DL(s)$, and by applying 4.3, we find $\neg DL(a_H(s))$. Then $\neg DL(a_H(\pi_n(x)))$, and since by limit rule we can easily show $a_H\circ\pi_n = \pi_n\circ a_H$, we have $\neg DL(\pi_n(a_H(x)))$. Since n was chosen arbitrarily, we have $\neg DL(a_H(x))$.
$\Leftarrow$: just as simple.

4.5. Notes.

1. In 4.4, we can take $a = t$, so pre-abstraction is safe with respect to deadlocks.
2. In 4.4, we cannot take $a = \delta$, as the following counterexamples show.

2.1. if $a,b \in A - \{\delta\}$ with $a \neq b$, then
$DL(a\delta + b)$ but $\neg DL(\partial_{\{a\}}(a\delta + b))$;
2.2. if $a \in A - \{\delta\}$, then $\neg DL(a)$ but $DL(\partial_{\{a\}}(a))$.
Thus, *neither* of the implications

$$DL(x) \Rightarrow DL(\partial_H(x))$$

$$DL(\partial_H(x)) \Rightarrow DL(x)$$

holds in general.
3. By 4.2.3, the predicate DL is semi-recursive on $\mathbb{G}$. In general, DL will however not be decidable.

4.6. NOTE. The following statements are easily proved:
1. $DL(x \| y) \Rightarrow DL(x) \lor DL(y)$
2. $DL(x.y) \Rightarrow DL(x) \lor DL(y)$

(the converse only holds for 1, if both x and y are finite, and the converse holds for 2, if x is finite).

We define determinism in 4.8. First we need another definition, which appeared earlier in e.g. BERGSTRA & KLOP [4].

4.7. DEFINITION. The set of *subprocesses* of x is the set of all processes obtained by executing a number of steps from x. We have the following inductive definition:
1. $x \in Sub(x)$
2. $ay + z \in Sub(x) \Rightarrow y \in Sub(x)$.

4.8. DEFINITIONS. Let $x \in \mathbb{G}$.
1. x is *root nondeterministic* if there is an $a \in A - \{\delta\}$ and $x_1, x_2, y \in \mathbb{G}$ such that (in $\mathbb{G}$) $x_1 \neq x_2$ and

$$x = ax_1 + ax_2 + y.$$

2. x is *nondeterministic* if there is a $y \in Sub(x)$ which is root nondeterministic.
3. $DET(x) \Leftrightarrow x$ is not nondeterministic.

Thus, DET is also a predicate on $\mathbb{G}$.

4.9. NOTES. 1. *Neither* of the implications

$$DET(x) \Rightarrow DET(a_H(x))$$

$$DET(a_H(x)) \Rightarrow DET(x)$$

holds in general (for $a \in A - \{\delta\}$, $H \subseteq A - \{\delta\}$), as the following counterexamples show.
If $a,b \in A - \{\delta\}$ with $a \neq b$, then
1.1. $DET(aa + b\delta)$ but $\neg DET(a_{\{b\}}(aa + b\delta))$;
1.2. $\neg DET(aa + ab)$ but $DET(a_{\{b\}}(aa + ab))$.
2. In case $a = \delta$, the implication

$$DET(x) \Rightarrow DET(\partial_H(x))$$

does hold, as is easily shown by induction, but the implication

$$DET(\partial_H(x)) \Rightarrow DET(x)$$

does *not*, as the following counterexample shows.
If $a \in A - \{\delta\}$, then

$DET(\partial_{\{a\}}(aa+a\delta))$ but $\neg DET(aa+a\delta)$.

4.10. It is easily seen that $DET(x)$ & $DET(y)$ is not a sufficient condition to conclude to $DET(x \| y)$ (take $x = aa, y = ab$), so we need some extra condition(s). Without proof, we mention the following

PROPOSITION.

$$\frac{\begin{array}{l} DET(x) \quad DET(y) \\ \alpha(x) \cap \alpha(y) \subseteq H \\ (\alpha(x) | \alpha(y)) \cap (\alpha(x) \cup \alpha(y)) \subseteq H \end{array}}{DET(\partial_H(x \| y))}.$$

4.11. THEOREM. *Let* $x \in \mathbb{G}$. *Then*

$$DET(x) \Leftrightarrow \text{for all } n \geqslant 1 \; DET(\pi_n(x)).$$

PROOF. $\Rightarrow$ Suppose $n \geqslant 1$ is such that $\neg DET(\pi_n(x))$, so there is a $y \in Sub(\pi_n(x))$ and $a \in A-\{\delta\}$, $x_1,x_2,x_3 \in \mathbb{G}$ with $x_1 \neq x_2$ and $y = ax_1 + ax_2 + x_3$. Using induction, it is not hard to show that for each $y \in Sub(\pi_n(x))$, there is a $y' \in Sub(x)$ and an $m \leqslant n$ such that $\pi_m(y') = y$. It follows that $y' = ax'_1 + ax'_2 + x'_3$, with $\pi_{m-1}(x'_1) = x_1$, $\pi_{m-1}(x'_2) = x_2$ ($x'_1 = x_1$ and $x'_2 = x_2$ if $m=1$) and $\pi_m(x'_3) = x_3$. Since $x_1 \neq x_2$, a fortiori $x'_1 \neq x'_2$, whence $\neg DET(x)$.
$\Leftarrow$ Suppose $\neg DET(x)$, so there is a $y \in Sub(x)$ and $a \in A-\{\delta\}$, $x_1,x_2,x_3 \in \mathbb{G}$ with $y = ax_1+ax_2+x_3$ and $x_1 \neq x_2$. Since $x_1 \neq x_2$, there must be an $n \geqslant 1$ with $\pi_n(x_1) \neq \pi_n(x_2)$ (by *AIP*). Therefore we have that $\pi_{n+1}(y)$ is root nondeterministic. Now if the top of y is 'at depth m' in x (y is reached after executing m steps from x), then $\neg DET(\pi_{n+m+1}(x))$.

4.12. NOTE. By 4.11, the predicate *DET* is co-semi-recursive on $\mathbb{G}$ In general, *DET* will however not be decidable.

4.13. DEFINITION. Now, we will define the *trace set* of a process in $\mathbb{G}$. A trace set is a set of words from $A-\{\delta\}$, so we will have a function

$$tr: \mathbb{G} \to Pow((A-\{\delta\})^*).$$

On *FCPE*, we define *tr* inductively:

1. $tr(a) = \{\epsilon, a\}$	if $a \in A-\{\delta\}$
2. $tr(\delta) = \{\epsilon\}$	
3. $tr(ax) = \{\epsilon\} \cup \{a*\sigma : \sigma \in tr(x)\}$	if $a \in A-\{\delta\}$
4. $tr(x+y) = tr(x) \cup tr(y)$	

and we extend this definition to $\mathbb{G}$ by:

$$\boxed{5. \quad tr(x) = \bigcup_{n=1}^{\infty} tr(\pi_n(x))}$$

4.14. Definition 4.13.5 is correct, because trace sets are *prefix closed*, i.e. if $\sigma*\rho$ is in some trace set ($\sigma,\rho \in (A-\{\delta\})^*$, $\sigma*\rho$ is word σ followed by word ρ), then σ is too.
We define the *set of trace sets* $\mathfrak{T}$ as the set of all prefix closed subsets of $(A-\{\delta\})^*$.
Note that $\mathfrak{T} = \{\emptyset\} \cup ran(tr)$.

4.15. DEFINITIONS. On $\mathfrak{T}$, we define three operations:

1. if $Z \in \mathfrak{T}$, $\frac{\partial}{\partial a}(Z) = \{\sigma : a^*\sigma \in Z\}$, so $\frac{\partial}{\partial a} : \mathfrak{T} \rightarrow \mathfrak{T}$ for $a \in A - \{\delta\}$.
2. If $Z \in \mathfrak{T}$, $first(Z) = \{a : \exists \sigma \in (A-\{\delta\})^* \; a^*\sigma \in Z\}$, so $first : \mathfrak{T} \rightarrow \mathcal{Q}(=Pow(A-\{\delta\}))$.
3. If $Z \in \mathfrak{T}$ and $a \in A - \{\delta\}$, $a^*Z = \{\epsilon\} \cup \{a^*\sigma : \sigma \in Z\}$, so $^* : (A-\{\delta\}) \times \mathfrak{T} \rightarrow \mathfrak{T}$.

Note that $a^* \varnothing = \{\epsilon\}$.

4.16. LEMMA. *For all* $Z \in \mathfrak{T} - \{\varnothing\}$ $Z = \bigcup_{a \in A - \{\delta\}} a^* \frac{\partial}{\partial a}(Z)$.

PROOF. This follows easily from 4.15.

Next we see when equality of trace sets implies equality of processes. A theorem similar to this one was proved by ENGELFRIET [12], in the setting of CCS (MILNER [14]) or CSP (BROOKES, HOARE & ROSCOE [11]). Here, in the setting of concrete process algebra, we need an extra condition, because we have both successful and unsuccessful termination (the process a vs. the process $a\delta$).

4.17. THEOREM. *Let* $x,y \in G$. *If* $DET(x) \& DET(y) \& \neg DL(x) \& \neg DL(y) \& tr(x) = tr(y)$, *then* $x = y$.

PROOF. By the limit rule, it suffices to prove this for $x,y \in FCPE$. We use induction, but in a little different form as in the definition of *FCPE*.
We will prove the following statement:
suppose $DET(x) \& DET(y) \& \neg DL(x) \& \neg DL(y) \& tr(x) = tr(y)$,

$$x = \sum_{i=1}^{n} a_i x_i + \sum_{j=1}^{m} b_j \quad (a_i, b_j \in A, \; n+m>0)$$

$$y = \sum_{k=1}^{r} c_k y_k + \sum_{l=1}^{s} d_l \quad (c_k, d_l \in A, \; r+s>0)$$

and the theorem holds for all x_i, y_k. Then $x = y$. So suppose x and y are as specified.
By applying A6 and A7, we can assume $a_i, b_j, c_k, d_l \in A - \{\delta\}$. By applying A3, we can assume all the b_j are distinct, and all the d_l are distinct.
Since $DET(x)$ and $DET(y)$, we can assume all the a_i are distinct, and all the c_k are distinct.
Since $\neg DL(x)$ and $\neg DL(y)$, we can assume that $x_i \neq \delta$ (if $1 \leq i \leq n$) and $y_k \neq \delta$ (if $1 \leq k \leq r$).
Using definition 4.13, we see that this implies that there is a $\sigma \in tr(x_i)$ and a $\sigma \in tr(y_k)$ with $\sigma \neq \epsilon$. Now $\{a_i : 1 \leq i \leq n\} \cup \{b_j : 1 \leq j \leq m\} = first\,(tr(x)) = first\,(tr(y)) = \{c_k : 1 \leq k \leq r\} \cup \{d_l : 1 \leq l \leq s\}$. If $1 \leq i \leq n$, then there is a $\sigma \in tr(x_i)$ with $\sigma \neq \epsilon$. Then $a_i^*\sigma \in tr(x)$, so $a_i^*\sigma \in tr(y)$. It follows that there must be a k $(1 \leq k \leq r)$ with $a_i = c_k$ and $\sigma \in tr(y_k)$. Thus $\{a_i : 1 \leq i \leq n\} = \{c_k : 1 \leq k \leq r\}$ and also $\{b_j : 1 \leq j \leq m\} = \{d_l : 1 \leq l \leq s\}$, whence $\sum_{j=1}^{m} b_j = \sum_{l=1}^{s} d_l$.
Therefore, we can write $y = \sum_{i=1}^{n} a_i y_i + \sum_{j=1}^{m} b_j$ (maybe after a renumbering of the y_i).
Let $1 \leq i \leq n$, then $tr(x_i) = \frac{\partial}{\partial a_i}(tr(x)) = \frac{\partial}{\partial a_i}(tr(y)) = tr(y_i)$ (since all the a_i are distinct). Since $x_i \in Sub(x)$, $y_i \in Sub(y)$, we have $DET(x_i) \& DET(y_i)$ immediately from definition 4.8; we have $\neg DL(x_i) \& \neg DL(y_i)$ from 4.2.2. Thus, applying the induction hypothesis, we have $x_i = y_i$, and therefore $x = y$.

Now we define the *restriction of a process to a trace set.* If x is a process, and Z a trace set, then $\nabla_Z(x)$ is the result of disallowing every step in x that will result in a trace outside Z. ∇_Z is not strictly a renaming operator (axiom RN4 will not be satisfied), so if we formally want to define the restriction operator in concrete process algebra, we need to extend our signature and set of axiomas. We formulate this in 4.18.

4.18. DEFINITION. Extend signature Σ with operators

$$\nabla_Z : P \to P \text{ for each } Z \in \mathfrak{T}$$

($\mathfrak{T}$, defined in 4.14, is an algebra with functions *first*, $\frac{\partial}{\partial a}$, $*$, defined in 4.15), and extend CPA_γ + RDP with axioms

$$\nabla_Z(a) = \partial_{A-first(Z)}(a)$$
$$\nabla_Z(ax) = \partial_{A-first(Z)}(a)\cdot\nabla_{\frac{\partial}{\partial a}(Z)}(x)$$
$$\nabla_Z(x+y) = \nabla_Z(x) + \nabla_Z(y)$$

4.19. LEMMA *Let $x \in \mathbb{G}$ and $Z \in \mathfrak{T}$. Then:*
1. $tr(\nabla_Z(x)) \subseteq Z$
2. $tr(x) \subseteq Z \Rightarrow \nabla_Z(x) = x.$

PROOF. Use induction on x.

4.20. DEFINITION. Let $I \subseteq A-\{\delta\}$. For a word σ in $(A-\{\delta\})^*$, let $\epsilon_I(\sigma)$ be the word obtained from σ by leaving out all elements from I. Then, if $x \in \mathbb{G}$, we define $tr_I(x)$, the *set of I-abstracted traces* of x, by:

$$tr_I(x) = \{\epsilon_I(\sigma) : \sigma \in tr(x)\}.$$

The following theorem constitutes an important interface between trace theory and process algebra.

4.21. THEOREM. *Let $p,q \in \mathbb{G}$. If $Z \supseteq tr_{\{t\}}\circ v^t_{\alpha(p)}\circ\partial_H(p\|q)$, then $\partial_H(p\|q) = \partial_H(\nabla_Z(p)\|q)$.*

PROOF. Let $p,q \in \mathbb{G}$, and suppose

$$Z \supseteq tr_{\{t\}}\circ v^t_{\alpha(p)}\circ\partial_H(p\|q).$$

Note that by 2.8, communication is one-to-one on $\alpha(p)$ and $\alpha(p)|A \cap \alpha(p) = \varnothing$. It suffices to prove the theorem for $p,q \in FCPE$ (by limit rule). We will use simultaneous induction on p and q to prove the following five statements:
1. $\partial_H(p\|q) = \partial_H(\nabla_Z(p)\|q)$
2. $\partial_H(p \mathbin{|\!\lfloor} q) = \partial_H(\nabla_Z(p) \mathbin{|\!\lfloor} q)$
3. $\partial_H(q \mathbin{|\!\lfloor} p) = \partial_H(q \mathbin{|\!\lfloor} \nabla_Z(p))$
4. $\partial_H(p|q) = \partial_H(\nabla_Z(p)|q)$.
5. $\partial_H(p) = \partial_H(\nabla_Z(p))$.

If $\nabla_Z(p) = p$, there is nothing to prove, so we can assume $\nabla_Z(p) \neq p$.

Case 1: $p \equiv a \in A$. Since $\nabla_Z(p) \neq p$, we must have $a \neq \delta$, and $\nabla_Z(a) = \delta$, so $a \notin first(Z)$. If we would have $a \notin H$, then $a \in first(tr_{\{t\}} \circ v^t_{\{a\}}\circ\partial_H(a)) \subseteq first(tr_{\{t\}} \circ v^t_{\{a\}} \circ\partial_H(a \mathbin{|\!\lfloor} q)) \subseteq first(tr_{\{t\}}\circ v^t_{\{a\}}\circ\partial_H(a\|q)) \subseteq first(Z)$, which is a contradiction. Therefore $a \in H$. Now we use induction on q.

Case 1.1: $q \equiv b \in A$. Since $v^t_{\{a\}}\circ\partial_H(a|b) = a$, if $a|b \neq \delta$ and $a|b \notin H$, and that will give a contradiction as above, we must have $\partial_H(a|b) = \delta$. But then $\partial_H(a\|b) = \partial_H(\delta\|b)$ and 2-5 follow.

Case 1.2: $q \equiv bx$, $b \in A$. Again $\partial_H(a|b) = \delta$, so $\partial_H(a\|bx) = \partial_H(a)\partial_H(bx)+\partial_H(b)\partial_H(a\|x) + \partial_H(a|b)\partial_H(x) = \delta\partial_H(bx) + \partial_H(b)\partial_H(\delta\|x)+\delta\partial_H(x)$ (use induction for the middle term) $= \partial_H(\delta\|bx)$.

Statements 2-5 are easier.

Case 1.3: $q \equiv x+y$.
$\partial_H(a \| (x+y)) = \partial_H(a)\partial_H(x+y) + \partial_H(x \lfloor\!\lfloor a) + \partial_H(y \lfloor\!\lfloor a) +$
$\partial_H(a \mid x) + \partial_H(a \mid y) = \delta\partial_H(x+y) + \partial_H(x \lfloor\!\lfloor \delta) + \partial_H(y \lfloor\!\lfloor \delta) +$
$+ \partial_H(\delta \mid x) + \partial_H(\delta \mid y)$ (induction on terms 2-5) $= \partial_H(\delta \| (x+y))$, and again, 2-5 are easier.
This finishes case 1.

Case 2: $p \equiv ax$, $a \in A$. If $a \notin \mathit{first}(Z)$, we conclude as in case 1 that $\partial_H(p \| q) = \partial_H(\delta \| q) = \partial_H(\nabla_Z(p) \| q)$. Therefore, we can assume $a \in \mathit{first}(Z)$, so $a \notin H$.

Claim: $\frac{\partial}{\partial a}(Z) \supseteq tr_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \| q)$.

Proof. Suppose $\sigma \in tr_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \| q)$.
Then

$$\begin{aligned}
a^*\sigma &\in a^* tr_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \| q) = \\
&= tr_{\{t\}}(a \cdot v^t_{\alpha(x)} \circ \partial_H(x \| q)) = \\
&= tr_{\{t\}} \circ v^t_{\alpha(x) \cup \{a\}}(a \cdot \partial_H(x \| q)) = \\
&= tr_{\{t\}} \circ v^t_{\alpha(p)} \circ \partial_H(ax \lfloor\!\lfloor q) \subseteq \\
&\subseteq tr_{\{t\}} \circ v^t_{\alpha(p)} \circ \partial_H(p \| q) \subseteq Z,
\end{aligned}$$

so by definition $\sigma \in \frac{\partial}{\partial a}(Z)$.
Now we use induction on q.

Case 2.1. $q \equiv b \in A$. Then

$$\begin{aligned}
\partial_H(ax \| b) &= \partial_H(a) \cdot \partial_H(x \| b) + \partial_H(b)\partial_H(a)\partial_H(x) + \\
&\quad + \partial_H(a \mid b)\partial_H(x) = \\
&= \partial_H(a) \cdot \partial_H(\nabla_{\frac{\partial}{\partial a}Z}(x) \| b) + \partial_H(b)\partial_H(a\nabla_{\frac{\partial}{\partial a}Z}(x)) + \\
&\quad + \partial_H(a \mid b)\partial_H(\nabla_{\frac{\partial}{\partial a}Z}(x)) = \\
&= \partial_H((a\nabla_{\frac{\partial}{\partial a}Z}(x)) \| b) = \partial_H(\nabla_Z(ax) \| b),
\end{aligned}$$

and 2-5 are easier.

Case 2.2. $q \equiv by$, $b \in A$. Then

$$\begin{aligned}
\partial_H(ax \| by) &= \partial_H(a)\partial_H(x \| by) + \partial_H(b)\partial_H(ax \| y) + \partial_H(a \mid b)\partial_H(x \| y) = \\
&= \partial_H(a)\partial_H(\nabla_{\frac{\partial}{\partial a}Z}(x) \| by) + \partial_H(b)\partial_H(\nabla_Z(ax) \| y) + \\
&\quad + \partial_H(a \mid b)\partial_H(\nabla_{\frac{\partial}{\partial a}Z}(x) \| y) = \partial_H(\nabla_Z(ax) \| by),
\end{aligned}$$

and 2-5 are easier.

Case 2.3. $q \equiv y+z$. Then

$$\partial_H(ax \| (y+z)) = \partial_H(a)\partial_H(x \| (y+z)) + \partial_H(y \lfloor\!\lfloor ax) + \partial_H(z \lfloor\!\lfloor ax) +$$

$$+ \partial_H(ax\,|\,y)+\partial_H(ax\,|\,z) =$$
$$= \partial_H(a)\partial_H(\nabla_{\frac{\partial}{\partial a}Z}(x)\|(y+z))+\partial_H(y\lfloor\!\lfloor \nabla_Z(ax)) \quad +$$
$$+ \partial_H(z\lfloor\!\lfloor \nabla_Z(ax))+\partial_H(\nabla_Z(ax)\,|\,y)+\partial_H(\nabla_Z(ax)\,|\,z) =$$
$$= \partial_H(\nabla_Z(ax)\|(y+z)),$$

and 2-5 are easier.
This finishes case 2.

Case 3. $p \equiv x+y$.

Claim. $Z \supseteq tr_{\{t\}}\circ v^t_{\alpha(x)}\circ\partial_H(x\|q) \cup tr_{\{t\}}\circ v^t_{\alpha(y)}\circ\partial_H(y\|q)$.

PROOF. Suppose $\sigma \in tr_{\{t\}}\circ v^t_{\alpha(x)}\circ\partial_H(x\|q)$. We can suppose $\sigma \neq \epsilon$.
Then

$$\sigma \in tr_{\{t\}}\circ v^t_{\alpha(x)}\circ\partial_H(x\lfloor\!\lfloor q) \cup tr_{\{t\}}\circ v^t_{\alpha(x)}\circ\partial_H(q\lfloor\!\lfloor x) \cup$$
$$tr_{\{t\}}\circ v^t_{\alpha(x)}\circ\partial_H(x\,|\,q).$$

If

$$\sigma \in tr_{\{t\}}\circ v^t_{\alpha(x)}\circ\partial_H(x\lfloor\!\lfloor q),$$

then

$$\sigma \in tr_{\{t\}}\circ v^t_{\alpha(x)}\circ\partial_H((x+y)\lfloor\!\lfloor q)\subseteq tr_{\{t\}}\circ v^t_{\alpha(x)}\circ\partial_H(p\|q) \subseteq$$
$$tr_{\{t\}}\circ v^t_{\alpha(p)}\circ\partial_H(p\|q) \subseteq Z.$$

If $\sigma\in tr_{\{t\}}\circ v^t_{\alpha(x)}\circ\partial_H(q\lfloor\!\lfloor x)$, take a trace

$$\tau\in tr\circ v^t_{\alpha(x)}\circ\partial_H(q\lfloor\!\lfloor x)$$

with

$$\epsilon_{\{t\}}(\tau) = \sigma.$$

If $\tau \equiv t{*}\tau'$, for some τ', this t-step came from a b-step in q, i.e. $q = bz+w$ with $\tau'\in tr\circ v^t_{\alpha(x)}\circ\partial_H(x\|z)$.
In the other case, if $\tau\equiv a{*}\tau'$, for some τ' and $a \in \alpha(x)$, we have $q = az+w$ with $\tau'\in tr\circ v^t_{\alpha(x)}\circ\partial_H(x\|z)$.
In either case, we can conclude by using an induction argument that $\tau'\in tr\circ v^t_{\alpha(x)}\circ\partial_H((x+y)\|z)$, whence, in the first case

$$\tau = t{*}\tau'\in t{*}tr\circ v^t_{\alpha(x)}\circ\partial_H((x+y)\|z)$$
$$= tr(t\cdot v^t_{\alpha(x)}\circ\partial_H((x+y)\|z)$$
$$= tr\circ v^t_{\alpha(x)}(b\cdot\partial_H((x+y)\|z)$$
$$= tr\circ v^t_{\alpha(x)}\circ\partial_H(bz\lfloor\!\lfloor(x+y))$$
$$\subseteq tr\circ v^t_{\alpha(x)}\circ\partial_H(q\lfloor\!\lfloor p)$$
$$\subseteq tr\circ v^t_{\alpha(p)}\circ\partial_H(p\|q),$$

so $\sigma = \epsilon_{\{t\}}(\tau)\in tr_{\{t\}}\circ v^t_{\alpha(p)}\circ\partial_H(p\|q)\subseteq Z$.
The other case is similar.
Lastly, if $\sigma\in tr_{\{t\}}\circ v^t_{\alpha(x)}\circ\partial_H(x\,|\,q)$, then

$$\sigma \in tr_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H((x+y)\mid q) \subseteq tr_{\{t\}} \circ v^t_{\alpha(p)} \circ \partial_H(p \| q) \subseteq Z.$$

Thus $tr_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \| q) \subseteq Z$. Similarly $tr_{\{t\}} \circ v^t_{\alpha(y)} \circ \partial_H(y \| q) \subseteq Z$, and the claim is proved.

Then we prove case 3 by induction on q.

Case 3.1. $q \equiv a \in A$. Then

$$\begin{aligned}\partial_H((x+y)\|a) &= \partial_H(x \mathbin{||\!\_} a)+\partial_H(y \mathbin{||\!\_} a)+\partial_H(a)(\partial_H(x)+\partial_H(y))+\\ &+ \partial_H(x\mid a)+\partial_H(y\mid a) =\\ &= \partial_H(\nabla_Z(x) \mathbin{||\!\_} a)+\partial_H(\nabla_Z(y) \mathbin{||\!\_} a)+\partial_H(a)(\partial_H(\nabla_Z(x))+\\ &+ \partial_H(\nabla_Z(y)))+\partial_H(\nabla_Z(x)\mid a)+\partial_H(\nabla_Z(y)\mid a) =\\ &= \partial_H((\nabla_Z(x)+\nabla_Z(y))\|a) = \partial_H(\nabla_Z(x+y)\|a),\end{aligned}$$

and 2-5 are easier.

Case 3.2. $q \equiv az, a \in A$. Then

$$\begin{aligned}\partial_H((x+y)\|az) &= \partial_H(x \mathbin{||\!\_} az)+\partial_H(y \mathbin{||\!\_} az)+\partial_H(a)\cdot\partial_H((x+y)\|z)+\\ &+ \partial_H(x\mid az)+\partial_H(y\mid az) =\\ &= \partial_H(\nabla_Z(x) \mathbin{||\!\_} az)+\partial_H(\nabla_Z(y) \mathbin{||\!\_} az)+\partial_H(a)\partial_H(\nabla_Z(x+y)\|z)+\\ &+ \partial_H(\nabla_Z(x)\mid az)+\partial_H(\nabla_Z(y)\mid az) = \partial_H(\nabla_Z(x+y)\|az),\end{aligned}$$

and 2-5 are easier.

Case 3.3. $q \equiv z+w$. Then

$$\begin{aligned}&\partial_H((x+y)\|(z+w)) = \partial_H(x \mathbin{||\!\_} (z+w))+\partial_H(y \mathbin{||\!\_} (z+w))+\\ &\partial_H(z \mathbin{||\!\_} (x+y))+\partial_H(w \mathbin{||\!\_} (x+y))+\partial_H(x\mid(z+w))+\\ &\partial_H(y\mid(z+w)) = \partial_H(\nabla_Z(x+y)\|(z+w)),\end{aligned}$$

and 2-5 are easier.

This finishes the proof of the theorem.

4.22. **Example.** We will illustrate the use of theorem 4.21 by again considering example 2.10. Define $F = \sum_{d\in D} s_1(d)r_4(d)r_1(ack)F$ (meaning that F is the unique solution of this guarded recursive specification), then 2.11 shows that in the context

$$\partial_H(\ldots\|S\|R),$$

F should do the same as E. We use 4.21 to prove this. Define the trace set Z inductively by:

1. for all $d \in D$

$$\epsilon, s_1(d), s_1(d)r_4(d), s_1(d)r_4(d)r_1(ack) \in Z;$$

2. if $\sigma \in Z$, then $s_1(d)r_4(d)r_1(ack)*\sigma \in Z$ (for all $d \in D$).

Claim 1. $Z = tr_{\{t\}} \circ v^t_{\alpha(E)} \circ \partial_H(E\|S\|R)$.

Proof. Using 2.11, we get

$$tr_{\{t\}} \circ \nu^t_{\alpha(E)} \circ \partial_H(E \| S \| R) =$$
$$tr_{\{t\}}(\sum_{d \in D} s_1(d)\ t\ r_4(d)\ t\ r_1(ack) \cdot \nu^t_{\alpha(E)} \circ \partial_H(E \| S \| R)) =$$
$$= \bigcup_{d \in D} (s_1(d) * tr_{\{t\}}(t\ r_4(d)\ t\ r_1(ack) \cdot \nu^t_{\alpha(E)} \circ \partial_H(E \| S \| R)) =$$
$$= \bigcup_{d \in D} (s_1(d) * (r_4(d) * (r_1(ack) * tr_{\{t\}} \circ \nu^t_{\alpha(E)} \circ \partial_H(E \| S \| R)))).$$

It is not hard to finish the proof.

Thus $\partial_H(E \| S \| R) = \partial_H(\nabla_Z(E) \| S \| R)$, and so we are done if we prove $\nabla_Z(E) = F$.

Claim 2. $\nabla_Z(E) = F$.

PROOF. Since $Z = \bigcup_{d \in D} (s_1(d) * (r_4(d) * (r_1(ack) * Z)))$, we have, for $d \in D$,

$$\frac{\partial}{\partial s_1(d)} Z = r_4(d) * (r_1(ack) * Z), \frac{\partial}{\partial r_4(d)} (\frac{\partial}{\partial s_1(d)} Z) = r_1(ack) * Z$$

and

$$\frac{\partial}{\partial r_1(ack)} (\frac{\partial}{\partial r_4(d)} (\frac{\partial}{\partial s_1(d)} Z)) = Z.$$

Then

$$\nabla_Z(E) = \nabla_Z((\sum_{d \in D} s_1(d) + \sum_{d \in D} r_4(d) + r_1(ack))E) =$$
$$= \sum_{d \in D} \nabla_Z(s_1(d)E) + \sum_{d \in D} \nabla_Z(r_4(d)E) + \nabla_Z(r_1(ack)E) =$$
$$= \sum_{d \in D} \partial_{A - \{s_1(d) : d \in D\}}(s_1(d)) \nabla_{\frac{\partial}{\partial s_1(d)} Z}(E) + \delta + \delta =$$
$$= \sum_{d \in D} s_1(d) \nabla_{r_4(d) * (r_1(ack) * Z)}(E) =$$
$$= \sum_{d \in D} s_1(d) \partial_{A - \{r_4(d)\}}(r_4(d)) \nabla_{r_1(ack) * Z}(E) + \delta =$$
$$= \sum_{d \in D} s_1(d) r_4(d) \partial_{A - \{r_1(ack)\}}(r_1(ack)) \nabla_Z(E) + \delta =$$
$$= \sum_{d \in D} s_1(d) r_4(d) r_1(ack) \nabla_Z(E).$$

Thus $\nabla_Z(E)$ satisfies the defining specification of F, so by RSP $\nabla_Z(E) = F$.

5. PROCESSES WITH SIDE EFFECT ON A STATE SPACE

In this section, we introduce processes that can be in different states. In fact, we introduce an operator λ_s (the state operator) so that $\lambda_s(x)$ is process x in state s. We give some examples, and show that we can use the state operator to translate computer programs (in some high level language) into the language of concrete process algebra.

5.1. STATE OPERATOR. We want to define the state operator λ_s, for $s \in S$, the *state space*. The principal idea is that executing a step of a process will result in a certain *effect* on the state, so our main equation will look like the following:

$$\lambda_s(ax) = a'\lambda_{s'}(x),$$

and here a' is the action resulting from execution of a in state s, and s' is the state resulting from execution of a in state s.
In fact, when we talk about a state, what we have in mind is the state of a certain *object.* Therefore, we will have a set of *names* M, and we will also index the state operator with a name $m \in M$, so λ_s^m symbolizes that the object named by m is in state s.
The action and effect functions will also depend on m. Now we are ready to give the formal definition. The basic idea for this definition came from BERGSTRA, KLOP & TUCKER [8].

5.2. DEFINITION. Let M and S be two given finite sets, so that sets A,M,S are pairwise disjoint. Suppose two functions *act*, *eff* are given:
act: $A \times M \times S \rightarrow A$,
eff: $A \times M \times S \rightarrow S$. We will write $a(m,s)$ for *act*(a,m,s) and $s(m,a)$ for *eff*(a,m,s).
We require the following:

$$\delta(m,s) = \delta \qquad s(m,\delta) = s \qquad (\text{for } m \in M,\ s \in S).$$

Now we extend the signature of concrete process algebra with operators

$$\lambda_s^m: P \rightarrow P \quad (\text{for } m \in M,\ s \in S),$$

and extend the set of axioms by:

$\lambda_s^m(a) = a(m,s)$	11
$\lambda_s^m(ax) = a(m,s)\lambda_{s(m,a)}^m(x)$	12
$\lambda_s^m(x+y) = \lambda_s^m(x) + \lambda_s^m(y)$	13

5.3. NOTE: The state operator is a generalization of the renaming operator defined in 2.1. For, if $b \in A$ and $H \subseteq A - \{\delta\}$ are given, define $M = \{m\}$ and $S = \{s\}$, and

$$a(m,s) = \begin{cases} b & \text{if } a \in H \\ a & \text{if } a \notin H, \end{cases}$$

then $\lambda_s^m = b_H$ follows.

5.4. DEFINITION. We define the *alphabet* of an object $m \in M$, $\alpha(m)$, as the set of all actions and states that can be changed, so
$\alpha(m) = \{a \in A$: there is $s \in S$ with $a(m,s) \neq a\} \cup \{s \in S$: there is $a \in A$ with $s(m,a) \neq s\}$.

5.5. THEOREM. *If there is no communication, and*
$\alpha(x) \cap \alpha(m_2) = \alpha(y) \cap \alpha(m_1) = \varnothing$, *then*
$\lambda_{s_1}^{m_1} \circ \lambda_{s_2}^{m_2}(x \| y) = \lambda_{s_1}^{m_1}(x) \| \lambda_{s_2}^{m_2}(y)$.

PROOF. The phrase 'there is no communication' can be specified by the statement

$$\alpha(x) | \alpha(y) = \varnothing \ \& \ \{a(m_1,s_1) : a \in \alpha(x)\} | \{a(m_2,s_2) : a \in \alpha(y)\} = \varnothing.$$

The proof consists of a simultaneous induction on x and y to prove a number of statements as in 4.21, and is straightforward, which is why it is omitted here.

5.6. REMARK. In 5.5, we have the case where we can separate the 'variables' m_1,m_2. If either $\alpha(x) \cap \alpha(m_2) \neq \varnothing$ or $\alpha(y) \cap \alpha(m_1) \neq \varnothing$, we have so-called *shared variables,* a situation that is also described in *GPL* (in the absence of communication), see OWICKI & GRIES [15].

5.7. EXAMPLE I. Suppose we have a serial switch as depicted in fig. 4.

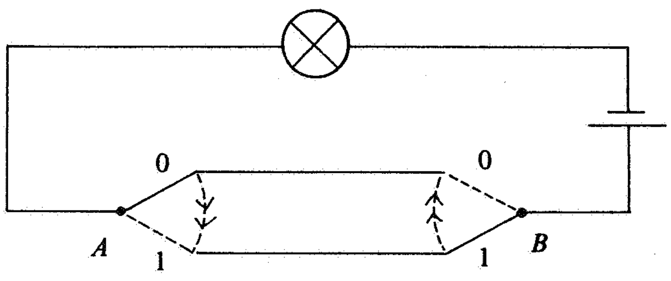

fig. 4

(For the formulation of this example, we are grateful to Jos Vrancken.)
The switches A and B are given by equations

$$A = aA$$

$$B = bB$$

(action a is the action of flipping switch A, and action b is the action of flipping switch B).
Define $M = \{m\}$, $S = \{0,1\}\times\{0,1\}$ (state $<i,j>$ is the state when switch A is in position i and switch B in position j, with $i,j \in \{0,1\}$). Now we define functions *act* and *eff*:

$$a(m,<i,j>) = \begin{cases} on(a) & \text{if } i \neq j \\ off(a) & \text{if } i = j \end{cases}$$

(on(a) means that the lamp is turned on by doing a, off(a) it is turned off by a),

$$b(m,<i,j>) = \begin{cases} on(b) & \text{if } i \neq j \\ off(b) & \text{if } i = j \end{cases}$$

$$<i,j>(m,a) = <1-i,j>$$

$$<i,j>(m,b) = <i,1-j>,$$

and functions *act* and *eff* are trivial otherwise. We assume there is no communication, so $\gamma(a,b) = \delta$. Now suppose we start in state $<0,1>$ (so the lamp is off), then we have process $P = \lambda^m_{<0,1>}(A \| B)$.

CLAIM: $P = (on(a)+on(b))(off(a)+off(b))P$.

PROOF. We use RSP to show

$$\lambda^m_{<0,1>}(A \| B) = \lambda^m_{<1,0>}(A \| B)$$

and

$$\lambda^m_{<0,0>}(A \| B) = \lambda^m_{<1,1>}(A \| B).$$

Then

$$P = \lambda^m_{<0,1>}(A \| B) =$$

$$= \lambda^m_{<0,1>}((a+b)(A \| B)) =$$
$$= on(a).\lambda^m_{<1,1>}(A \| B) + on(b).\lambda^m_{<0,0>}(A \| B) =$$
$$= (on(a)+on(b))\lambda^m_{<0,0>}(A \| B) =$$
$$= (on(a)+on(b))(off(a)\lambda^m_{<1,0>}(A \| B) +$$
$$+ off(b)\lambda^m_{<0,1>}(A \| B)) =$$
$$= (on(a)+on(b))(off(a)+off(b))\lambda^m_{<0,1>}(A \| B) =$$
$$= (on(a)+on(b))(off(a)+off(b))P.$$

5.8. EXAMPLE II. Random walk.

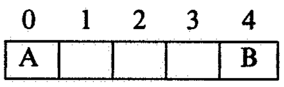

fig. 5.

Suppose we have given squares as in fig.5, and processes A and B each occupying one square. Then both start a random walk, so

$$A = (l_A + r_A)A + h_A$$
$$B = (l_B + r_B)B + h_B$$

(possible actions are *left, right* and *halt*). We implement this using the following state operator: Take $M = \{m\}$ (we will omit this m in the sequel) and $S = \{0,1,2,3,4\} \times \{0,1,2,3,4\}$. Then:

$$l_A(<i,j>) = \begin{cases} \delta & \text{if } i=0 \text{ or } j=i-1 \\ l_A & \text{otherwise} \end{cases}$$
$$r_A(<i,j>) = \begin{cases} \delta & \text{if } i=4 \text{ or } j=i+1 \\ r_A & \text{otherwise} \end{cases}$$
$$h_A(<i,j>) = h_A(i) \quad (A \text{ halts at } i),$$
$$<i,j>(l_A) = \begin{cases} <i,j> & \text{if } i=0 \text{ or } j=i-1 \\ <i-1,j> & \text{otherwise} \end{cases}$$
$$<i,j>(r_A) = \begin{cases} <i,j> & \text{if } i=4 \text{ or } j=i-1 \\ <i+1,j> & \text{otherwise} \end{cases}$$
$$<i,j>(h_A) = <i,j>,$$

and we have similar definitions for B:

$$l_B(<i,j>) = \delta \text{ if } j=0 \textit{ or } i=j-1,\ l_B \text{ otherwise}$$
$$r_B(<i,j>) = \delta \text{ if } j=4 \text{ or } i=j+1,\ r_B \text{ otherwise}$$
$$h_B(<i,j>) = h_B(j)$$
$$<i,j>(l_B) = <i,j> \text{ if } j=0 \text{ or } i=j-1,\ <i,j-1> \text{ otherwise}$$
$$<i,j>(r_B) = <i,j> \text{ if } j=4 \text{ or } i=j+1,\ <i,j+1> \text{ otherwise}$$
$$<i,j>(h_B) = <i,j>.$$

Then the situation pictured in fig.5 is described by:

$\lambda_{<0,4>}(A \| B)$.

Using *abstract* process algebra, we can establish the following claim:

CLAIM: Process $\lambda_{<0,4>}(A \| B)$ terminates, and will do so in a state $<i,j>$ with $i<j$. To be more precise, if we define $I = \{l_A, r_A, l_B, r_B\}$, then

$$\tau_I \circ \lambda_{<0,4>}(A \| B) = \tau\left(\sum_{i=0}^{3} h_A(i)\cdot \sum_{j=i+1}^{4} h_B(j) + \sum_{j=1}^{4} h_B(j)\cdot \sum_{i=0}^{j-1} h_A(i)\right).$$

The main tool used in proving this is Koomen's Fair Abstraction Rule (KFAR, see BAETEN, BERGSTRA & KLOP [2]). We will not give the proof here, since it is outside the scope of this paper.

5.9. Without proof, we mention two more propositions:

1. If $H \cap \alpha(m) = \varnothing$ and if $a \notin H$ implies $a(m,s) \notin H$ (for all $s \in S$), then $\lambda_s^m \circ \partial_H(x) = \partial_H \circ \lambda_s^m(x)$.
2. If $\alpha(x) \cap \alpha(m_1) \cap \alpha(m_2) = \varnothing$ and if $a \in \alpha(x) \cap \alpha(m_1)$ implies $a(m_1,s) \notin \alpha(m_2)$ (for all $s \in S$) and $a \in \alpha(x) \cap \alpha(m_2)$ implies $a(m_2,s) \notin \alpha(m_1)$ (for all $s \in S$), then $\lambda_{s_1}^{m_1} \circ \lambda_{s_2}^{m_2}(x) = \lambda_{s_2}^{m_2} \circ \lambda_{s_1}^{m_1}(x)$.

5.10. DEFINITION. In order to be able to give the following examples, we need to extend the notion of a state operator a little. What we need is that executing a step in a process can result in several possible actions, i.e. $a(m,s) \subseteq A$, not $a(m,s) \in A$. Thus $act: A \times M \times S \to Pow(A)$. The state following will depend on the alternative chosen, so $eff: A \times A \times M \times S \to S$, where $s(m,a,b)$ will matter only when $b \in a(m,s)$. We still have $\delta(m,s) = \{\delta\}$ and $s(m,\delta,\delta) = s$, and then we can define the *extended state operator* Λ_s^m by the following equations:

$\Lambda_s^m(a) = \sum_{b \in a(m,s)} b$	L1
$\Lambda_s^m(ax) = \sum_{b \in a(m,s)} b \cdot \Lambda_{s(m,a,b)}^m(x)$	L2
$\Lambda_s^m(x+y) = \Lambda_s^m(x) + \Lambda_s^m(y)$	L3

Note that if each $a(m,s)$ is a singleton, we get back the (simple) state operator defined in 5.2.

5.11. EXAMPLE III: we will describe a small part of a CSP language (see HOARE [13]). We have finite sets C (channels), X (variables) and D (data). We have atomic actions $c?x$ (for $c \in C$, $x \in X$; *receive*) and $c!d$ (*send* d). The only non-trivial communications are $c?d | c!d = c\sharp d$ (d is *communicated along channel* c).
We implement this as follows:
take $M = X$, $S = D$, then we define *act* and *eff* so that

$$\Lambda_d^x(c!x \cdot Z) = c!d \cdot \Lambda_d^x(Z) \quad \text{(for any process } Z, \text{ and } x \in X,\ d \in D,\ c \in C)$$

$$\Lambda_d^x(c?x \cdot Z) = \sum_{e \in D} c?e \cdot \Lambda_e^x(Z) \quad \text{(any } Z,\ x \in X,\ d \in D,\ c \in C).$$

(thus, in environment Λ_d^x, *variable* x has *value* d).

5.12. EXAMPLE IV: We can mechanically translate any computer program into process algebra. We illustrate this by means of the following example, a simple program to double a given number. Suppose we work on data structure $\mathbb{Z}_n = \{0,1,...,n-1\}$ with functions s (successor modulo n) and p (predecessor modulo n), and constant 0. We have the following program P:

```
read($x$)
$y := 0$
while $x \neq 0$ do $y := ss(y)$; $x := p(x)$
write($y$).
```

We translate this into process algebra as follows: all simple statements will become atomic actions, and program constructs become process algebra constructs, for instance a *while-* loop will become a recursive specification. Thus:

$$P = \text{read}(x)\cdot(y:=0)\cdot Z\cdot\text{write}(y),$$

$$Z = (x\neq 0)\cdot(y:=ss(y))\cdot(x:=p(x))\cdot Z + (x=0).$$

Now we describe the state operator:

$$M = \{x,y\},\ S = \mathbb{Z}_n,$$

1. $\text{read}(x)(x,d) = \{r(e) : e\in\mathbb{Z}_n\}\ \ (d\in\mathbb{Z}_n)$
   $d(x, \text{read}(x), r(e)) = e\ \ (d,e\in\mathbb{Z}_n)$
2. $d(y, (y:=0), (y:=0)) = 0\ \ (d\in\mathbb{Z}_{n)}$
3. $\text{write}(y)(y,d) = \{w(d)\}\ \ (d\in\mathbb{Z}_n)$
4. $(x\neq 0)(x, 0) = \{\delta\}$
5. $d(y, (y:=ss(y)), (y:=ss(y))) = s(s(d))\ \ (d\in\mathbb{Z}_n)$
6. $d(x, (x:=p(x)), (x:=p(x))) = p(d)\ \ (d\in\mathbb{Z}_n)$
7. $(x=0)(x,d) = \{\delta\}\ \ (d\in\mathbb{Z}_n-\{0\})$.
8. The functions *act* and *eff* are trivial in all other cases.

In order to see what happens, we will use pre-abstraction as defined in 2.3.
Take $I = \{y:=0, x\neq 0, y:=ss(y), x:=p(x), x=0\}$.

CLAIM. $t_I\circ\Lambda_0^x\circ\Lambda_0^y(P) = \sum_{e\in\mathbb{Z}_n} r(e)t^{2+3e}w(2e(\text{mod}\,n))$

(Note: if the program contains statements of the form $x:=y$, we have to use an operator $\Lambda_{<d,e>}^{<x,y>}$ instead of $\Lambda_d^x\circ\Lambda_e^y$.) We will sketch the proof of the claim by taking $n=2$, so $\mathbb{Z}_n = \{0,1\}$.
Then

$$t_I\circ\Lambda_0^x\circ\Lambda_0^y(P) =$$
$$= t_I\circ\Lambda_0^x(\text{read(x)}\cdot\Lambda_0^y((y:=0)Z\text{write}(y)) =$$
$$= t_I(r(0)\cdot\Lambda_0^x((y:=0)\cdot\Lambda_0^y(Z\text{write}(y)))+$$
$$+ r(1)\cdot\Lambda_1^x((y:=0)\cdot\Lambda_0^y(Z\text{write}(y)))) =$$
$$= r(0)t\cdot t_I(\delta+(x=0)\cdot\Lambda_0^x\circ\Lambda_0^y(\text{write}(y)))+$$
$$+ r(1)t\cdot t_I((x\neq 0)\cdot\Lambda_1^x\circ\Lambda_0^y((y: = ss(y)(x: = p(x))\cdot Z\cdot\text{write}(y))) =$$
$$= r(0)\cdot t\cdot t\cdot w(0)\ +$$
$$+ r(1)\cdot t\cdot t\cdot t_I((y:=ss(y))(x:=p(x))\Lambda_0^x\circ\Lambda_0^y(Z\cdot\text{write}(y))) =$$
$$= r(0)ttw(0)\ +$$
$$+ r(1)ttttt_I(\delta+(x=0)\Lambda_0^x\circ\Lambda_0^y(\text{write}(y))) =$$
$$= r(0)ttw(0)\ +\ r(1)tttttw(0).$$

5.13. EXAMPLE V: Consider again the queue defined in 3.1. Looked at in a certain way, all it does is actions *read* and *write,* so we might want to say

$$\text{queue} = read^{\omega}\|write^{\omega}$$

where for an atom a, the process a^{ω} is defined by the recursive specification $X = aX$. We can realise this view in the following way:: take $M = \{<1,2>\}$ (we have input channel 1 and output channel 2) and $S = D^*$ (if we want the state space to be finite, we need to limit the capacity of the queue).
Now we define *act* and *eff:*

1. $read(<1,2>,\sigma) = \{r_1(d) : d \in D\}$
   $\sigma(<1,2>, read, r_1(d)) = \sigma * d.$
2. $write(<1,2>,\epsilon) = \{\delta\}$
   $write(<1,2>,\sigma * d) = \{s_2(d)\} \quad (d \in D)$
   $\epsilon(<1,2>, write, \delta) = \epsilon$
   $\sigma * d(<1,2>, write, s_2(d)) = \sigma. \quad (d \in D)$

CLAIM: $Q = \Lambda_\epsilon^{<1,2>}(read^\omega \| write^\omega)$.

PROOF. We define, for $\sigma \in D^*$

$$R_\sigma = \Lambda_\sigma^{<1,2>}(read^\omega \| write^\omega).$$

Then:

1. $R_\epsilon = \Lambda_\epsilon^{<1,2>}(read(read^\omega \| write^\omega) + write(read^\omega \| write^\omega)) =$
   $= \sum_{d \in D} r_1(d) \cdot \Lambda_d^{<1,2>}(read^\omega \| write^\omega) + \delta = \sum_{d \in D} r_1(d) R_d.$
2. $R_{\sigma * d} = \Lambda_{\sigma * d}^{<1,2>}(read(read^\omega \| write^\omega) + write(read^\omega \| write^\omega)) =$
   $= \sum_{e \in D} r_1(e) \cdot \Lambda_{\sigma * d * e}^{<1,2>}(read^\omega \| write^\omega) + s_2(d) \Lambda_\sigma^{<1,2>}(read^\omega \| write^\omega) =$
   $= \sum_{e \in D} r_1(e) R_{\sigma * d * e} + s_2(d) R_\sigma.$

Therefore, the R_σ satisfy the equations for the Q_σ in 3.1, so by RSP $R_\sigma = Q_\sigma$.

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 Mathematics and Computer Science, Amsterdam, 1985.

[2] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP, *On the consistency of Koomen's Fair Abstraction Rule*, report CS-R8511, Centre for Mathematics and Computer Science, Amsterdam, 1985.

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

[4] J.A. BERGSTRA & J.W. KLOP, *Algebra of communicating processes*, in: Proceedings of the CWI symposium Mathematics and Computer Science, eds. J.W. de Bakker, M. Hazewinkel & J.K. Lenstra, Amsterdam, 1985.

[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, *Process algebra for synchronous communication*, Information and Control 60 (1/3), pp.109-137, 1984.

[7] J.A. BERGSTRA, J.W. KLOP & E.-R. OLDEROG, *Readies and failures in the algebra of communicating processes*, report CS-R85.., Centre for Mathematics and Computer Science, Amsterdam, 1985.

[8] J.A. BERGSTRA, J.W. KLOP & J.V. TUCKER, *Process algebra with asynchronous communication mechanisms*, report CS-R8410, Centre for Mathematics and Computer Science, Amsterdam, 1984, to appear in Proc. of the CMU workshop on concurrency.

[9] J.A. BERGSTRA & J. TIURYN, *Process algebra semantics for queues*, report IW 241/83, Mathematical Centre, Amsterdam 1983.

[10] J.A. BERGSTRA & J.V. TUCKER, *Top-down design and the algebra of communicating processes*, Sci. of Comp. Prog. 5 (2), pp. 171-199, 1985.

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

[12] J. ENGELFRIET, *Determinacy $\rightarrow$ (observation equivalence = trace equivalence)*, memo ♯ inf-84-10, Twente University of Technology, Enschede 1984.

[13] C.A.R. HOARE, *Communicating sequential processes*, Comm. Assoc. Comp. Mach. 21, pp.666-677, 1978.

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

[15] S. OWICKI & D. GRIES, *An axiomatic proof technique for parallel programs,* Acta Inf. 6, pp.319-340, 1976.

[16] M. REM, *Partially ordered computations, with applications to VLSI design,* in: Proc. 4th Advanced Course on Found. of Comp. Sci. part 2, eds. J.W. de Bakker & J. van Leeuwen, MC Tract 159, Mathematical centre, pp. 1-44, Amsterdam 1983.