INFORMATION AND COMPUTATION 78, 205–245 (1988)

Global Renaming Operators in Concrete Process Algebra*

J. C. M. BAETEN

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

AND

J. A. BERGSTRA

Programming Research Group, University of Amsterdam,
P.O. Box 41882, 1009DB Amsterdam, The Netherlands;
Department of Philosophy, State University of Utrecht,
Heidelberglaan 2, 3584 CS Utrecht, The Netherlands.

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.

INTRODUCTION

Concrete process algebra is that part of process algebra that does not involve τ-steps which are the result of abstraction. We introduce concrete process algebras as an extension of ACP, the algebra of communicating processes (see Bergstra and Klop, 1986). Concrete process algebra does not consider silent moves or abstraction as is done in abstract process algebra (see Bergstra and Klop, 1985). The main advantages of not considering abstraction are that it leads to a clearer, and less problematic theory, with an easy to understand axiomatization. Contrary to the case with abstraction, concrete process algebra is amenable to a term rewriting analysis and can be studied using initial algebra semantics. Also, concrete process

* Partial support received from the European Communities under ESPRIT Project 432, An Integrated Formal Approach to Industrial Software Development (Meteor).

0890-5401/88 $3.00

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 and Zucker (1982).

In concrete process algebra, we introduce renaming operators. In Section 2, we define simple renaming operators, called relabelings in CCS (see Milner, 1980), 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 Section 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 Section 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 1983), 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 1985). 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 between trace theory and process algebra.

In Section 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 and Zucker, 1982). 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.

This article is a revision of (Baeten and Bergstra, 1985). Since that report appeared, the operators introduced there have been used in several other papers, notably Vaandrager (1986) and Groenveld (1987), thereby demonstrating their usefulness. We thank the referees for their valuable comments and many suggestions for improvements.

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 and Klop (1986). In this paper, we do not consider silent moves (or τ-steps) or abstraction as is done in ACP$_\tau$ (see Bergstra and Klop, 1985).

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* $\gamma\colon A \times A \to A$ 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 $\mathscr{A}$, the set of subsets of $A - \{\delta\}$, is the set of alphabets. Functions $+$, $\cdot$, $\|$, $\mathbin{\|\!\lfloor}$, $|$, ∂_H, π_n, and constant δ are discussed in Bergstra and Klop (1986) (π_n is called( )$_n$ there), and α and t are discussed in Baeten, Bergstra and Klop (1987a).

1.2. *Signature Σ.* 1. *Sorts.*

A	(see 1.1)
P	(set of processes; $A \subseteq P$)
$\mathscr{A}$	($\mathscr{A} = \mathrm{Pow}(A - \{\delta\})$).

2. *Functions.*

$+ : P \times P \to P$	(alternative composition or sum)
$\cdot : P \times P \to P$	(sequential composition or product)
$\| : P \times P \to P$	(parallel composition or merge)
$\mathbin{\|\!\lfloor} : P \times P \to P$	(left-merge)

$\mid: P \times P \to P$	(communication merge)
$\partial_H: P \to P$	(encapsulation; $H \subseteq A - \{t\}$)
$\pi_n: P \to P$	(projection; $n > 0$)
$\alpha: P \to \mathscr{A}$	(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

$$p = q$$

called *equations*; here p, q are process expressions, possibly containing variables.

We use letters $a, b, c, \ldots$ for elements of A, letters $x, y, z, \ldots$ for arbitrary processes (in some model of the theory), and we use capital letters $X, Y, Z, \ldots$ 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).

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

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

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

is the same equation with processes $\mathbf{x}$ substituted for variables $\mathbf{X}$. 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 is an index set), with e_j of the form

$$X_j = s_j(\mathbf{X}),$$

s_j is a term with variables from $\{X_j : j \in J\}$, and J has a distinguished element j_0.

A set of processes $\mathbf{x} = \{x_j : j \in J\}$ (in a particular model) is a *solution vec-*

tor of E if $E(\mathbf{x})$, i.e., substituting processes $\mathbf{x}$ for the variables of E gives $e_j(\mathbf{x})$ for all $j \in J$. Process x is a *solution* of E, $E(x, -)$, if there is a solution vector of E with x in the j_0-position. x is (*recursively*) *definable* (in a particular model) if there is a specification E such that $E(y, -) \Leftrightarrow x = y$. (For these definitions, also see Baeten, Bergstra and Klop, 1987b.)

1.5. DEFINITION. The set of *finite closed process expressions*, FCPE, is defined inductively:

1. $A \subseteq \text{FCPE}$;
2. $x \in \text{FCPE}$ and $a \in A \Rightarrow ax \in \text{FCPE}$
3. $x, y \in \text{FCPE} \Rightarrow x + y \in \text{FCPE}$.

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

Next we define a notion of guardedness (taken from Baeten, Bergstra and Klop 1987b). Specifications must be guarded in order to prove that they have unique solutions (see 1.11).

1.6. DEFINITION. Let s be an open term, possibly containing variables. An occurrence of a variable X in s is *guarded* if s has a subterm of the form aM, with $a \in A$ and this occurrence of X is in M; otherwise, the occurrence is *unguarded*.

Let $E = \{e_j : j \in J\}$ be a specification. Define $X_i \rightarrow^u X_j \Leftrightarrow X_j$ occurs unguarded in s_i (the right-hand side of e_i), and E is guarded $\Leftrightarrow$ $\rightarrow^u$ is well founded (i.e., there is no infinite sequence $X_{j_1} \rightarrow^u X_{j_2} \rightarrow^u \ldots$).

1.7. AXIOMS. The axioms for concrete process algebra are presented in Table I. We use the following abbreviations: ACP = A1–7 + C1–3 + CM1–9 + D1–4; P = PR1–4; AB = AB1–6; CPA = ACP + PR + AB + AIP, and $\text{CPA}_\gamma = \text{CPA} + \text{C4}$, so in Table I we have $\text{CPA}_\gamma + \text{RDP} + \text{HNF}$.

1.8. *Comments.* For a discussion of axioms ACP, see Bergstra and Klop (1986 or 1984). Axiom C4 says that the communication merge $|$ extends the communication function γ given in 1.1. We often have an extra restriction on the communication function, namely the *handshaking axiom* (HA), that says that all communications are binary:

$$x | y | z = \delta.$$

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 and Klop (1987a). Note

TABLE I

Concrete Process Algebra

$x+y=y+x$	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
		$\alpha(ax)=\{a\}\cup\alpha(x)$ if $a\neq\delta$	AB4
$x \parallel y=x \mathbin{\|\!\!\lfloor} y+y \mathbin{\|\!\!\lfloor} x+x \mid y$	CM1	$\alpha(x+y)=\alpha(x)\cup\alpha(y)$	AB5
$a \mathbin{\|\!\!\lfloor} x=ax$	CM2	$\alpha(x)=\bigcup_{n=1}^{\infty}\alpha(\pi_n(x))$	AB6
$(ax) \mathbin{\|\!\!\lfloor} y=a(x \parallel y)$	CM3		
$(x+y) \mathbin{\|\!\!\lfloor} z=x \mathbin{\|\!\!\lfloor} z+y \mathbin{\|\!\!\lfloor} z$	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	$\dfrac{\forall n\ \pi_n(x)=\pi_n(y)}{x=y}$	AIP
$\partial_H(a)=\delta$ if $a\in H$	D2		
$\partial_H(x+y)=\partial_H(x)+\partial_H(y)$	D3		
$\partial_H(xy)=\partial_H(x)\cdot\partial_H(y)$	D4		
		$\forall x\ \exists n, m\ \exists a_i, b_j \in A, x_i \in P(i<n, j<m)$	
$a \mid b=\gamma(a,b)$	C4	$x=\sum_{i<n} a_i\cdot x_i+\sum_{j<m} b_j$	HNF

that it is not necessary to use the extra sort $\mathscr{A}$, for instead of saying that the alphabet is a *set* of atomic actions, it works equally well to say that the alphabet is a *sum* of atomic actions. If we take the latter option, we preserve the algebraic framework more strictly.

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, and Klop (1987b). Finally, the principle of *head normal forms* (HNF) is formulated in Baeten and van

Glabbeek (1987). It says that every process has a *head normal form*, i.e., can be written as a sum of finitely many alternatives, each of which starts with an atomic action. The second sum is needed, in case some of these alternatives consist of just an atomic action.

1.9. THEOREM. *For every closed term t there is a term $s \in$ FCPE such that* $\mathrm{CPA}_\gamma \vdash t = s$. *(This is the so-called elimination theorem.)*

Proof. See Bergstra and Klop (1984).

1.10. DEFINITION. The *recursive specification principle* (RSP) is

$$\frac{\begin{array}{c} E(x,-)\ E(y,-) \\ E \text{ guarded} \end{array}}{x = y}\ (\mathrm{RSP}).$$

Note that the solution vector in $E(x,-)$ may be specified completely different than the solution vector in $E(y,-)$ (see 1.18).

1.11. LEMMA. HNF *implies that for every x and n, $\pi_n(x)$ is equal to a term in* FCPE.

Proof. By induction on n. Write $x = \sum_{i<n} a_i \cdot x_i + \sum_{j<m} b_j$, for certain n, m, a_i, x_i, b_j. For $n = 1$, we see immediately that $\pi_1(x) = \sum_{i<n} a_i + \sum_{j<m} b_j$. For $n > 1$, we write $\pi_n(x) = \sum_{i<n} a_i \cdot \pi_{n-1}(x_i) + \sum_{j<m} b_j$, and use the induction hypothesis for the x_i.

1.12. LEMMA. RSP *holds in concrete process algebra.*

Proof. See Baeten, Bergstra and Klop (1987a).

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. Now the condition of guardedness ensures that we can write each equation in the form $X_j = \sum_{i<n} a_i \cdot s_i + \sum_{j<m} b_j$, for certain expressions s_i. Thus, the head normal form of each variable in a guarded recursive specification is completely determined by the specification and does not depend on the chosen solution (vector). Using 1.11, we see that we can calculate the finite projections of each variable as a term in FCPE. Now, 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.

1.13. DEFINITION. The *limit rule* (LR) is

$$\frac{\forall t \in \mathrm{FCPE}\ e(t,-)}{e(x,-)}\ (\mathrm{LR}).$$

In words, if an equation holds for all finite processes (more precisely, for all elements of FCPE), then it holds for all processes. The limit rule allows us to prove identities by induction, since FCPE is defined recursively.

1.14. LEMMA. LR *holds in concrete process algebra.*

Proof. Suppose that $s(x) = t(x)$ is an equation in which a variable x occurs, and we know that it holds, when we substitute a term in FCPE for x. We have to prove that $s(x) = t(x)$ holds. By AIP, it is enough to prove $\pi_n(s(x)) = \pi_n(t(x))$, for each $n \geqslant 1$. Fix $n \geqslant 1$. For terms in FCPE, we can prove by structural induction that $\pi_n(x \,\Box\, y) = \pi_n(\pi_n(x) \,\Box\, \pi_n(y))$ for $\Box = +, \cdot, \|, \mathbin{\|\!\!\!\lfloor}, |$ and $\pi_n(\partial_H(x)) = \pi_n(\partial_H(\pi_n(x))$. For general processes, we can prove these equations by use of HNF. It follows that in the equation $\pi_n(s(x)) = \pi_n(t(x))$, we can replace each occurrence of x by $\pi_n(x)$, by 1.11 equal to a term in FCPE. By assumption, the equation holds for this term. It is easy to finish the proof.

1.15. EXAMPLES. 1. In the previous theorem, the use of the principle of head normal forms is essential. For, if we consider the initial algebra of the theory with atomic actions $\{a, b\}$, but restrict the signature to $\{a\}$ (so that the element b cannot be described by a closed term), then AIP does hold, but HNF and LR do not. HNF does not hold, since the element b does not have a head normal form, and LR does not hold, since the equation $\pi_1(x) + a = a$ holds for all closed terms (since they must consist of a-steps or be equal to δ) but not for the element b.

2. Even when we replace the principle HNF by the weaker assumption, that every finite projection of every process is equal to a term in FCPE (see 1.11), we cannot derive LR. For, consider the initial algebra of the theory with infinitely many atomic actions $\{a_i: i \in \mathbb{N}\}$, also containing one constant c that does not correspond to an element in the signature. Then, we define $\pi_n(c) = a_n$, and further define projection in accordance with the axioms. Then, the model does satisfy AIP (as it contains no infinite elements) and the assumption, but not HNF (as c does not have a head normal form) or LR (the equation $\pi_1(\pi_2(x)) = \pi_1(x)$ holds for all closed terms, as projection is defined normally on the a_i, but not for c). To see that the model satisfies AIP, use the fact that each element has an alphabet, containing only finitely many of the a_i. Thus, the occurrence of a c in a term can be detected in the sequence of its projections, as infinitely many a_i appear in that position.

3. We leave it as an open problem, to construct models of concrete process algebra, that satisfy LR, but not AIP or HNF.

1.16. Theorem. *The identities in Table* II *are provable from the axioms of concrete process algebra. Also the expansion theorem*:

$$x_1 \| x_2 \| \cdots \| x_n = \sum_{1 \leqslant i \leqslant n} x_i \mathbin{\lfloor\!\lfloor} \mathbf{x}^j + \sum_{1 \leqslant i < j \leqslant n} (x_i | x_j) \mathbin{\lfloor\!\lfloor} \mathbf{x}^{i,j} \quad \text{ET}$$

(*here* $\mathbf{x}^i = \|_{1 \leqslant k \leqslant n, k \neq i} x_k$ and $\mathbf{x}^{i,j} = \|_{1 \leqslant k \leqslant n, k \neq i, k \neq j} x_k$).

Proof. Identities SC1–6 are proved in Bergstra and Klop (1984), CA 1, 3, 5 in Baeten, Bergstra and Klop (1987a), and ET in Bergstra and Tucker (1985), for all terms in FCPE. The general identities then follow by applying the limit rule.

1.17. *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*

$$\mathbf{A} = T_I\left(\Sigma\,(x_1, ..., x_n), \mathrm{CPA}_\gamma + E(x_1, ..., x_n)\right)$$

will exist, because $\mathrm{CPA}_\gamma + E(x_1, ..., x_n)$ is a positive conditional system. In $\mathbf{A}$, the principles AIP, LR, HNF (by 1.12) and RSP hold, but RDP does not hold (not *all* guarded recursive specifications have solutions). (Of course, we could add constants for more than one specification.)

1.18. Examples. 1. Suppose $\gamma(a, b) = \delta$ for all $a, b \in A$. Take $a, b \in A$, and consider the initial algebra $\mathbf{A} = T_I(\Sigma(x, y, z, w), \mathrm{CPA}_\gamma + \{x = (a + b)x, y = ay, z = bz, w = y \| z\})$. In $\mathbf{A}$ we have $w = y \| z = y \mathbin{\lfloor\!\lfloor} z + z \mathbin{\lfloor\!\lfloor} y + y | z =$

TABLE II

Standard concurrency		Conditional axioms	
$(x \mathbin{\lfloor\!\lfloor} y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} (y \| z)$	SC1	$\dfrac{\alpha(x) \| (\alpha(y) \cap H) \subset H}{\partial_H(x \| y) = \partial_H(x \| \partial_H(y))}$	CA1
$(x \| y) \mathbin{\lfloor\!\lfloor} z = x \| (y \mathbin{\lfloor\!\lfloor} z)$	SC2		
$x \| y = y \| x$	SC3		
$x \| y = y \| x$	SC4	$\dfrac{\alpha(x) \cap H = \varnothing}{\partial_H(x) = x}$	CA3
$x \| (y \| z) = (x \| y) \| z$	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

$(ay) \mathbin{\|\!\_} z + (bz) \mathbin{\|\!\_} y + ay \mid bz = a(y \| z) + b(z \| y) + (a \mid b)(y \| z) = (a+b)(y \| z) + \delta = (a+b)w$, so with RSP $x = w$. (Note that $z \| y = y \| z$ since $y \mid z = \delta$.)

2. Suppose $a \in A$, and consider the initial algebra $\mathbf{A} = T_I(\Sigma(x, y), \mathrm{CPA}_\gamma + \{x = ay, y = ax\})$. Using RSP we obtain $x = y$.

The existence of inital algebras shows that concrete process algebra is consistent. To show however, that there exists a model also satisfying RDP, we need a result from Bergstra and Klop (1986).

1.19. Theorem. $\mathbb{G}$, *the set of all finitely branching process graphs modulo bisimulation, is a model of concrete process algebra,* HNF *and* RDP.

1.20. Theorem. $\mathbb{G}$ *is the* final *model for concrete process algebra plus* HNF, *in the sense that any other model that has the same equalities for closed terms, must be a submodel of* $\mathbb{G}$.

Proof. If $\mathbb{B}$ is any other model for concrete process algebra plus HNF, and x is an element of $\mathbb{B}$, construct a process graph for x as follows: write x in head normal form $\sum_{i<n} a_i \cdot x_i + \sum_{j<m} b_j$. Then, the root of the process graph for x has edges labeled a_i to nodes p_i, and edges labeled b_j to an endnode. At node p_i we continue in the same way with the head normal form of process x_i. By 1.11, the head normal forms determine the finite projections, and the principle AIP says that a process is determined by its finite projections. The proof is finished, if we remark that in $\mathbb{G}$ two graphs are bisimilar, iff they are bisimilar to any finite depth.

1.21. Corollary. *If* $\mathbf{A}$ *is an initial algebra of concrete process algebra as defined in* 1.17, *then* $\mathbf{A}$ *is a subalgebra of* $\mathbb{G}$.

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 v_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 and Olderog, (1987). To be more precise, we extend the signature with operators

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

and add axioms of Table III. 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.* The pre-abstraction operator will identify all internal actions, will abstract from their identity, but no action can be removed altogether; for that purpose we need the (full) abstraction operator τ_I. (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 2.5–2.7 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 ports (or channels) linking them. We assume that at each location a certain process is executed, and that these processes can communicate via the ports, thus obtaining a communication network. These communications will consist of the transfer of a piece of data. So suppose we have a finite set of data D (often, $D = \{0, 1\}$), and we have communication channels 1, 2, ..., k. Then we have the following atomic actions:

$r_i(d)$ = *read d along port i* ($d \in D$, $1 \leqslant i \leqslant k$);
$s_i(d)$ = *send d along port i* ($d \in D$, $1 \leqslant i \leqslant k$);
$c_i(d)$ = *communicate d along port i* ($d \in D$, $1 \leqslant i \leqslant k$),

TABLE III

$a_H(b) = b$	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

and on these atomic actions, we define the communication function as

$$\gamma(r_i(d), s_i(d)) = \gamma(s_i(d), r_i(d)) = c_i(d)(d \in D, 1 \leqslant i \leqslant k)$$
$$\gamma(a, b) = \delta \text{ in all other cases.}$$

We call this restricted communication format *read/send* communication (or read/write communication).

2.5. DEFINITION. We consider the communication network of Fig. 1 (so we represent locations by circles, with the name of its process inside, and ports by lines). The processes P, Q, B_1, B_2 are given by the 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 $f\colon D \times D \to D$ is some given function)

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

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 $H = \{r_i(d), s_i(d)\colon d \in D,\ i = 2, 3, 4\}$, 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 Section 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.

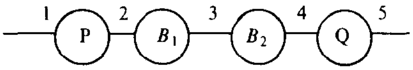

FIGURE 1

$$\partial_H(P \,\|\, B_1 \,\|\, B_2 \,\|\, Q)$$

$$= \partial_H \Bigg(\sum_{d \in D} r_1(d) \Bigg(\Bigg(\sum_{e \in D} r_1(e)\, s_2(d)\, s_2(e) \Bigg) \,\|\, B_1 \,\|\, B_2 \,\|\, Q \Bigg)$$

$$+ \sum_{d \in D} r_2(d) \Bigg(P \,\|\, \Bigg(s_3(d) \sum_{e \in D} r_2(e)\, s_3(e) \Bigg) \,\|\, B_2 \,\|\, Q \Bigg)$$

$$+ \sum_{d \in D} r_3(d) \Bigg(P \,\|\, B_1 \,\|\, \Bigg(s_4(d) \sum_{e \in D} r_3(e)\, s_4(e) \Bigg) \,\|\, Q \Bigg)$$

$$+ \sum_{d \in D} r_4(d) \Bigg(P \,\|\, B_1 \,\|\, B_2 \,\|\, \Bigg(\sum_{e \in D} r_4(e)\, s_5(f(d, e)) \Bigg) \Bigg) + \delta \Bigg)$$

$$= \sum_{d \in D} r_1(d) \cdot \partial_H \Bigg(\Bigg(\sum_{e \in D} r_1(e)\, s_2(d)\, s_2(e) \Bigg) \,\|\, B_1 \,\|\, B_2 \,\|\, Q \Bigg) + \delta + \delta + \delta$$

$$= \sum_{d \in D} r_1(d) \cdot \sum_{e \in D} r_1(e) \cdot \partial_H((s_2(d)\, s_2(e)) \,\|\, B_1 \,\|\, B_2 \,\|\, Q)$$

$$= \sum_{d \in D} r_1(d) \cdot \sum_{e \in D} r_1(e) \cdot c_2(d)$$

$$\cdot \partial_H \Bigg(s_2(e) \,\|\, \Bigg(s_3 d) \sum_{f \in D} r_2(f)\, s_3(f) \Bigg) \,\|\, B_2 \,\|\, Q \Bigg)$$

$$= \sum_{d \in D} r_1(d) \cdot \sum_{e \in D} r_1(e) \cdot c_2(d) \cdot c_3(d)$$

$$\cdot \partial_H \Bigg(s_2(e) \,\|\, \Bigg(\sum_{f \in D} r_2(f)\, s_3(f) \Bigg) \,\|\, \Bigg(s_4(d) \sum_{g \in D} r_3(g)\, s_4(g) \Bigg) \,\|\, Q \Bigg)$$

$$= \sum_{d \in D} r_1(d) \sum_{e \in D} r_1(e)\, c_2(d)\, c_3(d) \Bigg(c_2(e)$$

$$\cdot \partial_H \Bigg(s_3(e) \,\|\, \Bigg(s_4(d) \sum_{g \in D} r_3(g)\, s_4(g) \Bigg) \,\|\, Q \Bigg) + c_4(d)$$

$$\cdot \partial_H \Bigg(s_2(e) \,\|\, \Bigg(\sum_{f \in D} r_2(f)\, s_3(f) \Bigg) \,\|\, \Bigg(\sum_{g \in D} r_3(g)\, s_4(g) \Bigg) \,\|$$

$$\Bigg(\sum_{h \in D} r_4(h)\, s_5(f(d, h)) \Bigg) \Bigg) \Bigg)$$

$$= \sum_{d\in D} r_1(d) \sum_{e\in D} r_1(e)\, c_2(d)\, c_3(d) \Bigg(c_2(e)\, c_4(d)$$

$$\cdot \partial_H \Bigg(s_3(e) \,\|\, \Big(\sum_{g\in D} r_3(g)\, s_4(g) \Big) \,\|\, \Big(\sum_{h\in D} r_4(h)\, s_5(f(d,h)) \Big) \Bigg)$$

$$+ c_4(d)\, c_2(e)$$

$$\cdot \partial_H \Bigg(s_3(e) \,\|\, \Big(\sum_{g\in D} r_3(g)\, s_4(g) \Big) \,\|\, \Big(\sum_{h\in D} r_4(h)\, s_5(f(d,h)) \Big) \Bigg) \Bigg)$$

$$= \sum_{d\in D} r_1(d) \sum_{e\in D} r_1(e)\, c_2(d)\, c_3(d) (c_2(e)\, c_4(d) + c_4(d)\, c_2(e))$$

$$\cdot c_3(e) \cdot \partial_H \Bigg(s_4(e) \,\|\, \Big(\sum_{h\in D} r_4(h)\, s_5(f(d,h)) \Big) \Bigg)$$

$$= \sum_{d\in D} r_1(d) \sum_{e\in D} r_1(e)\, c_2(d)\, c_3(d) (c_2(e)\, c_4(d)$$

$$+ c_4(d)\, c_2(e)) \cdot c_3(e)\, c_4(e)\, s_5(f(d,e)).$$

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

$$t_I \circ \partial_H(P \,\|\, B_1 \,\|\, B_2 \,\|\, Q)$$

$$= \sum_{d\in D} r_1(d) \sum_{e\in D} r_1(e) \cdot t \cdot t (t \cdot t + t \cdot t) \cdot t \cdot t \cdot s_5(f(d,e))$$

$$= \sum_{d\in D} r_1(d) \sum_{e\in D} r_1(e) \cdot t^6 \cdot s_5(f(d,e)).$$

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

2.8. Example III: Localization. For sake of simplicity, we only define this operator in the case of read/send communication as described in 2.4. Let $B \subseteq A - \{\delta\}$ be such that for each port i and each $d \in D$ at most one of the atoms $r_i(d)$, $s_i(d)$, $c_i(d)$ is in B. If B satisfies this requirement, then the communication function γ has an "inverse" on $B \mid A$; i.e., if $c_i(d) \in B \mid A$, then exactly one of $r_i(d)$, $s_i(d)$ is in B. Thus, we can call the original in B $\gamma^{-1}(c_i(d))$.

Now we can define the localization operator.

2.9. DEFINITION. Let $B \subseteq A - \{\delta, t\}$ satisfy the requirement in 2.8. Let $B | A = \{c_1, ..., c_n\}$. We define the *localization operator* v^t_B by

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

It is easy to see that v^t_B has the properties:

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

Intuitively, we think of v^t_B 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 communication network shown in 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:

$$S = \sum_{d \in D} r_1(d)\, s_2(d)\, r_3(\text{ack})\, s_1(\text{ack}) S$$

$$R = \sum_{d \in D} r_2(d)\, s_4(d)\, s_3(\text{ack}) R.$$

Here we have a special element *ack* denoting acknowledgment (it is easiest to take ack $\notin D$), so S sends a $d \in D$ to R, receives an acknowledgment, and then sends the next d; R receives the data and sends back an acknowledgment.

S and R also communicate with the environment E; so E can send data

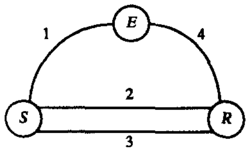

FIGURE 2

along 1, receive an acknowledgment along 1 or receive data along 4. Thus we can put

$$E=\left(\sum_{d\in D} s_1(d)+\sum_{d\in D} r_4(d)+r_1(\mathrm{ack})\right)E.$$

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 $r_1(\mathrm{ack})$, before a next $s_1(d')$ can follow. We can express this by using the localization operator. We put $H=\{s_i(d),\ r_i(d)\colon d\in D,\ i\in\{1,2,3,4\}\}$, and look at

$$v^t_{\alpha(E)}\circ\partial_H(E\parallel S\parallel R),$$

so in the process $\partial_H(E\parallel S\parallel R)$ we focus on actions from E and we localize to E. It is easily seen that $\alpha(E)=\{s_1(d),\ r_4(d)\colon d\in D\}\cup\{r_1(\mathrm{ack})\}$ and that $\alpha(E)$ satisfies the requirement of 2.8. We see

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

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, over a data set D with more than one element (see Fig. 3). An *infinite* guarded specification can be given by the equations

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

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

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

Now we look for a *finite* recursive specification of Q, in the context of handshaking communication (i.e., a finite specification E which has a solution x in some model iff the specification above has a solution x). 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:

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

That is why we want to specify Q under the condition

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

3.2. Next, we will prove two theorems:

THEOREM 1. *We cannot define Q by a finite guarded recursive specification in concrete process algebra without renaming under the condition $\alpha(Q) \cap \mathrm{ran}(\gamma) = \varnothing$.*

THEOREM 2. *We can define Q by a finite guarded recursive specification in concrete process algebra with renaming operators* (*as defined in Section* 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:

$$\mathrm{PA} = \mathrm{A1\text{–}5} + \mathrm{M1\text{–}4} + \mathrm{PR1\text{–}4}$$

(here M1 is axiom $x \| y = x \mathbin{\|\!\_} y + y \mathbin{\|\!\_} x$ and M2–4 = CM2–4) and

$$\mathrm{PA}_\delta = \mathrm{PA} + \mathrm{A6, 7}.$$

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

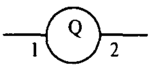

FIGURE 3

Proof. See Bergstra and Tiuryn (1987).

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 $x = s_1(x, x_2, ..., x_n)$ for some $x_2, ..., x_n$). We define a theory PA*, intermediate between PA and PA_δ. 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, r are PA*-terms, then so are $s + r$, $s \cdot r$, $s \| r$, $s \mathbin{|\!\lfloor} r$, and $s \cdot \delta$.

The *axioms of* PA* are

$$\mathrm{PA}^* = \mathrm{PA} + \{a\delta \mathbin{|\!\lfloor} x = ax\delta\}.$$

Note that rewriting a PA*-term by use of a PA*-axiom will yield a PA*-term. Note also that $\mathrm{CPA} \vdash a\delta \mathbin{|\!\lfloor} x = ax\delta$ (use 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 (i.e., we replace each (sub)term $s\delta$ by s).

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 1.12, there is an expansion s_1^n of s_1 in which all x_j are n times guarded, so that we can reduce, in PA_δ, $\pi_n(s_1^n)$ to a term in FCPE. Likewise, for E' there is an expansion s'^n_1 of s'_1, so that $\pi_n(s'^n_1)$ reduces in PA to a v'_n in FCPE. Now the reduction from $\pi_n(s'^n_1)$ to v'_n in PA can be exactly transcribed to a reduction from $\pi_n(s_1^n)$ to a $v_n \in$ FCPE in PA* (sometimes using the PA*-axiom instead of CM2).

Since x is a solution of E we have $\pi_n(x) = v_n$. But v_n is a PA*-term, and so A6 and A7 cannot be applied to v_n, so since x does not deadlock, δ cannot occur in v_n. But that means that $v_n = v'_n$, so $\pi_n(s_1^n) = v_n = v'_n = \pi_n(s'^n_1)$, 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 1.12 and the observation

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

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

Proof. Let E be a finite guarded recursive specification in $\mathrm{ACP}_\gamma + \mathrm{PR}$ defining x. Put $H = A \mid A = \mathrm{ran}\,\gamma - \{\delta\}$, 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 ($\mathrm{ACP}_\gamma + \mathrm{PR}$)-term amounts to leaving out (i.e., set equal to δ) all communication (sub)terms of the form $x \mid y$ 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 $\mathrm{ACP}_\gamma + \mathrm{PR}$ with trivial γ is the same as theory PA_δ.

3.9. COROLLARY. *Q is not definable by a finite guarded recursive specification in* $\mathrm{ACP}_\gamma + \mathrm{PR}$, *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 $\mathrm{ACP}_\gamma + \mathrm{PR}$, so it is a ($\mathrm{ACP}_\gamma + \mathrm{PR}$)-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

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

are the only non-trivial communications.

Suppose $D = \{d_1, ..., d_n\}$, 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 $H = \{l(d) \colon d \in D\}$. We claim we can define Q by

$$Q = \sum_{d \in D} r_1(d) \cdot s_{2_{\{u\}}} \circ \partial_H(l_{\{s_2\}}(Q) \parallel s_2(d) \cdot Z)$$

$$Z = \sum_{d \in D} l(d) \cdot Z.$$

To show this specification is correct, define R_σ, for $\sigma \in D^*$, inductively:

$$R_\varepsilon = Q \qquad \text{(as given above)}$$

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

First we need the following observation:

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

We prove this by RSP, showing that both sides satisfy the same equations: we have, on the one hand,

$$\begin{aligned} R_\varepsilon &= \sum_{d \in D} r_1(d) \cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\varepsilon) \parallel s_2(d) Z) \\ &= \sum_{d \in D} r_1(d) \cdot R_d, \end{aligned}$$

and

$$\begin{aligned} R_{\sigma * d} &= s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \parallel s_2(d) Z) \\ &= s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \mathbin{\lfloor\!\lfloor} s_2(d) Z) + s_{2\{u\}}\, \partial_H(s_2(d) Z \mathbin{\lfloor\!\lfloor} l_{\{s_2\}}(R_\sigma)) \\ &\qquad \text{(since } s_2(d) \text{ does not communicate)} \\ &= s_{2\{u\}} \circ \partial_H(l_{\{s_2\}} \left(\sum_{e \in D} r_1(e) \right) [l_{\{s_2\}}(R_{e * \sigma}) \parallel s_2(d) Z] \\ &\qquad \text{(by induction)} \\ &\qquad + l_{\{s_2\}}(s_2(f))[l_{\{s_2\}}(R_{\sigma' * f}) \parallel s_2(d) Z]) \\ &\qquad \text{(if } \sigma = \sigma' * f\text{; if } \sigma = \varepsilon, \text{ this term does not appear)} \\ &\qquad + s_2(d) \cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \parallel Z) \\ &= \sum_{e \in D} r_1(e) \cdot s_{2\{u\}}(\partial_H(l_{\{s_2\}}(R_{e * \sigma}) \parallel s_2(d) Z) + \delta) \\ &\qquad + s_2(d) \cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \parallel Z) \\ &= \sum_{e \in D} r_1(e) \cdot R_{e * \sigma * d} + s_2(d) \cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \parallel Z). \end{aligned}$$

On the other hand, we have

$$\begin{aligned}
&s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\varepsilon) \| Z)\\
&\quad = s_{2\{u\}} \circ \partial_H \Bigg(\sum_{d \in D} r_1(d) \cdot [l_{\{s_2\}}(s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(Q) \| s_2(d)Z)) \| Z]\\
&\qquad + l(d) \cdot [l_{\{s_2\}}(R_\varepsilon) \| Z] \Bigg)\\
&\quad = \sum_{d \in D} r_1(d) \cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_d) \| Z),
\end{aligned}$$

and

$$\begin{aligned}
&s_{2\{u\}} \circ \partial_H(l_{\{s2\}}(R_{\sigma * d}) \| Z)\\
&\quad = s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma)) \| s_2(d)Z) \| Z)\\
&\quad = s_{2\{u\}} \circ \partial_H \Bigg(\Bigg[l_{\{s_2\}} \Bigg(\sum_{e \in D} r_1(e)\, s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_{e * \sigma}) \| s_2(d)Z)] \Bigg)\\
&\qquad + l_{\{s_2\}}(s_2(d)[s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \| Z)]) \Bigg] \| Z \Bigg)\\
&\quad = s_{2\{u\}} \circ \partial_H \Bigg(\sum_{e \in D} r_1(e)(l_{\{s_2\}}(R_{e * \sigma * d}) \| Z) \Bigg)\\
&\qquad + s_{2\{u\}} \circ \partial_H(u(d)(s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \| Z) \| Z))\\
&\quad = \sum_{e \in D} r_1(e) \cdot s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_{e * \sigma * d}) \| Z)\\
&\qquad + s_2(d) \cdot s_{2\{u\}} \circ \partial_H(s_{2\{u\}} \circ \partial_H(l_{\{s_2\}}(R_\sigma) \| Z) \| Z).
\end{aligned}$$

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

$$\begin{aligned}
R_\varepsilon &= \sum_{d \in D} r_1(d) \cdot R_d\\
R_{\sigma * d} &= \sum_{e \in D} r_1(e) \cdot R_{e * \sigma * d} + s_2(d) \cdot R_\sigma.
\end{aligned}$$

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, $R_\varepsilon = Q_\varepsilon$, so the equations above indeed define the queue Q. Finally, $\alpha(Q) \cap \mathrm{ran}(\gamma) = \varnothing$ 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 and 2.11.

4.1. *Note.* The principle we will use very often in this section is the principle of head normal forms (HNF), in particular the following consequence of HNF: each projection must be a finite term (1.11).

4.2. *Deadlock Behaviour.* We define a predicate DL in concrete process algebra as follows: on FCPE, we define DL inductively by

1. $\mathrm{DL}(\delta)$

2. $\mathrm{DL}(x) \Rightarrow \mathrm{DL}(ax+y)$ $(a \in A,\ x, y \in \mathrm{FCPE})$; and in general, we define

3. $\mathrm{DL}(x) \Leftrightarrow \exists n\ \mathrm{DL}(\pi_n(x))$.

4.3. Lemma. *The following hold:*

1. $\neg\ \mathrm{DL}(a)$ *if* $a \in A - \{\delta\}$,
2. $\neg\ \mathrm{DL}(x) \Rightarrow \neg\ \mathrm{DL}(ax)$ $(a \in A - \{\delta\})$,
3. $\neg\ \mathrm{DL}(x)$ *and* $\neg\ \mathrm{DL}(y) \Rightarrow \neg\ \mathrm{DL}(x+y)$.

Proof. Easy.

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

Proof. $\Rightarrow$ Suppose $\neg\ \mathrm{DL}(x)$. Take $n \geqslant 1$. Then $\neg\ \mathrm{DL}(\pi_n(x))$. By 1.11, there must be an $s \in \mathrm{FCPE}$ such that $\pi_n(x) = s$. Then $\neg\ \mathrm{DL}(s)$, and by applying 4.3, we find $\neg\ \mathrm{DL}(a_H(s))$. Then $\neg\ \mathrm{DL}(a_H(\pi_n(x)))$, and since by limit rule we can easily show $a_H \circ \pi_n = \pi_n \circ a_H$, we have $\neg\ \mathrm{DL}(\pi_n(a_H(x)))$. Since n was chosen arbitrarily, we have $\neg\ \mathrm{DL}(a_H(x))$.

$\Leftarrow$ 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

$$\mathrm{DL}(a\delta+b) \text{ but } \neg\, \mathrm{DL}(\partial_{\{a\}}(a\delta+b));$$

2.2. if $a \in A-\{\delta\}$, then $\neg\, \mathrm{DL}(a)$ but $\mathrm{DL}(\partial_{\{a\}}(a))$.

Thus, *neither* of the implications

$$\mathrm{DL}(x) \Rightarrow \mathrm{DL}(\partial_H(x))$$

$$\mathrm{DL}(\partial_H(x)) \Rightarrow \mathrm{DL}(x)$$

holds, in general.

3. By 4.2.3., the predicate DL is semi-recursive. In general, DL will, however, not be decidable.

4.6. *Note.* The following statements are easily proved:

1. $\mathrm{DL}(x \| y) \Rightarrow \mathrm{DL}(x) \vee \mathrm{DL}(y)$
2. $\mathrm{DL}(x \cdot y) \Rightarrow \mathrm{DL}(x) \vee \mathrm{DL}(y)$.

(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 and Klop (1986).

4.7. DEFINITION. The set of *subprocesses* (or *states*) of x is the set of all processes obtained by executing a number of steps from x. We have the following inductive definition:

1. $x \in \mathrm{Sub}(x)$
2. $ay+z \in \mathrm{Sub}(x) \Rightarrow y \in \mathrm{Sub}(x)$.

4.8. DEFINITIONS. Let x be any process:

1. x is *root nondeterministic* if there is an $a \in A-\{\delta\}$ and processes x_1, x_2, y such that $x_1 \neq x_2$ and

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

2. x is *nondeterministic* if there is a $y \in \mathrm{Sub}(x)$ which is root nondeterministic.

3. $\mathrm{DET}(x) \Leftrightarrow x$ is not nondeterministic.

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

$$\mathrm{DET}(x) \Rightarrow \mathrm{DET}(a_H(x))$$
$$\mathrm{DET}(a_H(x)) \Rightarrow \mathrm{DET}(x)$$

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. $\mathrm{DET}(aa + b\delta)$ but $\neg\, \mathrm{DET}(a_{\{b\}}(aa + b\delta))$;

1.2. $\neg\, \mathrm{DET}(aa + ab)$ but $\mathrm{DET}(a_{\{b\}}(aa + ab))$.

2. In case $a = \delta$, the implication

$$\mathrm{DET}(x) \Rightarrow \mathrm{DET}(\partial_H(x))$$

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

$$\mathrm{DET}(\partial_H(x)) \Rightarrow \mathrm{DET}(x)$$

does *not*, as the following counterexample shows.

If $a \in A - \{\delta\}$, then

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

4.10. It is easily seen that $\mathrm{DET}(x)$ and $\mathrm{DET}(y)$ is not a sufficient condition to conclude to $\mathrm{DET}(x \parallel y)$ (take $x = aa$, $y = ab$), so we need some extra condition(s). When we deal with large systems, we do want to be able to conclude the determinacy of the whole system from the determinacy of the parts. The following proposition (whose proof we omit) lists sufficient conditions for this.

PROPOSITION.

$$\frac{\begin{array}{c}\mathrm{DET}(x) \qquad \mathrm{DET}(y)\\ \alpha(x) \cap \alpha(y) \subseteq H\\ (\alpha(x) \mid \alpha(y)) \cap (\alpha(x) \cup \alpha(y)) \subseteq H\end{array}}{\mathrm{DET}(\partial_H(x \parallel y))}.$$

4.11. THEOREM. *Let x be any process in concrete process algebra. Then*

$$\mathrm{DET}(x) \Leftrightarrow \quad \textit{for all} \quad n \geqslant 1 \qquad \mathrm{DET}(\pi_n(x)).$$

Proof. $\Rightarrow$ Suppose $n \geqslant 1$ is such that $\neg\, \mathrm{DET}(\pi_n(x))$, so there is a $y \in \mathrm{Sub}(\pi_n(x))$ and $a \in A - \{\delta\}$, processes x_1, x_2, x_3 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

$y \in \mathrm{Sub}(\pi_n(x))$, there is a $y' \in \mathrm{Sub}(x)$ 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 $\neg\, \mathrm{DET}(x)$.

$\Leftarrow$ Suppose $\neg\, \mathrm{DET}(x)$, so there is a $y \in \mathrm{Sub}(x)$ and $a \in A - \{\delta\}$, and processes x_1, x_2, x_3 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 $\neg\, \mathrm{DET}(\pi_{n+m+1}(x))$.

4.12. *Note.* By 4.11, the predicate DET is co-semi-recursive. In general, DET will, however, not be decidable.

4.13. Definition. Now, we will define the *trace set* of a process in concrete process algebra. A trace set is a set of words from $A - \{\delta\}$. On FCPE, we define *tr* inductively:

1. $\mathrm{tr}(a) = \{\varepsilon, a\}$, if $a \in A - \{\delta\}$,
2. $\mathrm{tr}(\delta) = \{\varepsilon\}$,
3. $\mathrm{tr}(ax) = \{\varepsilon\} \cup \{a^*\sigma \colon \sigma \in \mathrm{tr}(x)\}$, if $a \in A - \{\delta\}$,
4. $\mathrm{tr}(x + y) = \mathrm{tr}(x) \cup \mathrm{tr}(y)$;

and we extend this definition to all processes by

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

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* $\mathscr{T}$ as the set of all prefix closed subsets of $(A - \{\delta\})^*$. Note that $\mathscr{T} = \{\varnothing\} \cup \mathrm{ran}(\mathrm{tr})$.

4.15. Definitions. On $\mathscr{T}$, we define three operations:

1. if $Z \in \mathscr{T}$, $(\partial/\partial a)(Z) = \{\sigma \colon a^*\sigma \in Z\}$, so $(\partial/\partial a) \colon \mathscr{T} \to \mathscr{T}$ for a $a \in A - \{\delta\}$.

2. If $Z \in \mathscr{T}$, $\mathrm{first}(Z) = \{a \colon \exists \sigma \in (A - \{\delta\})^* \quad a^*\sigma \in Z\}$, so $\mathrm{first} \colon \mathscr{T} \to \mathscr{A}(= \mathrm{Pow}(A - \{\delta\}))$.

3. If $Z \in \mathscr{T}$ and $a \in A - \{\delta\}$, $a^*Z = \{\varepsilon\} \cup \{a^*\sigma \colon \sigma \in Z\}$, so $^* \colon (A - \{\delta\}) \times \mathscr{T} \to \mathscr{T}$.

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

4.16. Lemma. *For all $Z \in \mathscr{T} - \{\varnothing\}$ $Z = \bigcup_{a \in A - \{\delta\}} a^*(\partial/\partial a)(Z)$.*

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 (1985), in the setting of CCS, (Milner, 1980) or CSP (Brookes, Hoare, and Roscoe, 1984). 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 be any processes. If* $\mathrm{DET}(x)$ *and* $\mathrm{DET}(y)$ *and* $\neg\mathrm{DL}(x)$ *and* $\neg\,\mathrm{DL}(y)$ *and* $\mathrm{tr}(x) = \mathrm{tr}(y)$, *then* $x = y$.

Proof. We prove this for $x, y \in \mathrm{FCPE}$. The general case then follows from AIP. We use induction, but in a little different form as in the definition of FCPE. We will prove the statement: Suppose $\mathrm{DET}(x)$ and $\mathrm{DET}(y)$ and $\neg\,\mathrm{DL}(x)$ and $\neg\,\mathrm{DL}(y)$ and $\mathrm{tr}(x) = \mathrm{tr}(y)$,

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

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

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 $\mathrm{DET}(x)$ and $\mathrm{DET}(y)$, we can assume all the a_i are distinct, and all the c_k are distinct. Since $\neg\,\mathrm{DL}(x)$ and $\neg\,\mathrm{DL}(y)$, we can assume that $x_i \neq \delta$ (if $1 \leqslant i \leqslant n$) and $y_k \neq \delta$ (if $1 \leqslant k \leqslant r$). Using Definition 4.13, we see that this implies that there is a $\sigma \in \mathrm{tr}(x_i)$ and a $\sigma \in \mathrm{tr}(y_k)$ with $\sigma \neq \varepsilon$. Now $\{a_i\colon \leqslant 1 \leqslant i \leqslant n\} \cup \{b_j\colon 1 \leqslant j \leqslant m\} = \mathrm{first}(\mathrm{tr}(x)) = \mathrm{first}(\mathrm{tr}(y)) = \{c_k\colon 1 \leqslant k \leqslant r\} \cup \{d_l\colon 1 \leqslant l \leqslant s\}$. If $1 \leqslant i \leqslant n$, then there is a $\sigma \in \mathrm{tr}(x_i)$ with $\sigma \neq \varepsilon$. Then $a_i^* \sigma \in \mathrm{tr}(x)$, so $a_i^* \sigma \in \mathrm{tr}(y)$. It follows that there must be a k $(1 \leqslant k \leqslant r)$ with $a_i = c_k$ and $\sigma \in \mathrm{tr}(y_k)$. Thus $\{a_i\colon 1 \leqslant i \leqslant n\} = \{c_k\colon 1 \leqslant k \leqslant r\}$ and also $\{b_j\colon 1 \leqslant j \leqslant m\} = \{d_l\colon 1 \leqslant l \leqslant s\}$, 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 $1 \leqslant i \leqslant n$, then $\mathrm{tr}(x_i) = (\partial/\partial a_i)(\mathrm{tr}(x)) = (\partial/\partial a_i)(\mathrm{tr}(y)) = \mathrm{tr}(y_i)$ (since all the a_i are distinct). Since $x_i \in \mathrm{Sub}(x)$, $y_i \in \mathrm{Sub}(y)$, we have $\mathrm{DET}(x_i)$ and $\mathrm{DET}(y_i)$ immediately from Definition 4.8; we have $\neg\,\mathrm{DL}(x_i)$ and $\neg\,\mathrm{DL}(y_i)$ 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 axioms. We formulate this in 4.18.

4.18. Definition. Extend signature Σ with operators

$$\nabla_Z\colon P \to P \qquad \text{for each} \quad Z \in \mathscr{T}$$

($\mathscr{T}$, defined in 4.14, is an algebra with functions *first*, $\partial/\partial a$, *, defined in 4.15), and extend $\mathrm{CPA}_\gamma + \mathrm{RDP}$ with axioms

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

4.19. Lemma. *Let x be a process and $Z \in \mathscr{T}$. Then*:

1. $\mathrm{tr}(\nabla_Z(x)) \subseteq Z$
2. $\mathrm{tr}(x) \subseteq Z \Rightarrow \nabla_Z(x) = x$.

Proof. Use induction on x.

4.20. Definition. Let $I \subseteq A - \{\delta\}$. For a word σ in $(A - \{\delta\})^*$, let $\varepsilon_I(\sigma)$ be the word obtained from σ by leaving out all elements from I. Then, if x is a process, we define $\mathrm{tr}_I(x)$, the *set of I-abstracted traces* of x, by

$$\mathrm{tr}_I(x) = \{\varepsilon_I(\sigma) \colon \sigma \in \mathrm{tr}(x)\}.$$

Often, when we study a large system, it becomes very difficult to calculate its external behaviour because of the great number of states of the system (many times an infinite number). In such cases, the trace set of the system, or the trace set of some components of the system, may be much easier to calculate. The following theorem gives us a way to use such information to simplify a component in the context of the system. This makes it possible to reduce the number of states before the interaction of the components is calculated. Since this theorem allows us to use trace information in a process algebra verification, it constitutes, in our view, an important interface between trace theory and process algebra. Indeed, this technique has been used extensively in the papers Vaandrager (1986) and Groenveld (1987).

4.21. Theorem. *Let p, q be two processes in concrete process algebra.*

Suppose $\alpha(p)$ *satisfies* 2.8, *so that* $v^t_{\alpha(p)}$ *is defined. Then* $Z \supseteq \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(p)} \circ \partial_H(p \parallel q)$ *implies* $\partial_H(p \parallel q) = \partial_H(\nabla_Z(p) \parallel q)$.

Proof. Let p, q be given, and suppose

$$Z \supseteq \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(p)} \circ \partial_H(p \parallel q).$$

It suffices to prove the theorem for $p, q \in \mathrm{FCPE}$ (by limit rule). We will use simultaneous induction on p and q to prove the following five statements:

1. $\partial_H(p \parallel q) = \partial_H(\nabla_Z(p) \parallel q)$.
2. $\partial_H(p \mathbin{\lfloor\!\lfloor} q) = \partial_H(\nabla_Z(p) \mathbin{\lfloor\!\lfloor} q)$.
3. $\partial_H(q \mathbin{\lfloor\!\lfloor} p) = \partial_H(q \mathbin{\lfloor\!\lfloor} \nabla_Z(p))$.
4. $\partial_H(p \mid q) = \partial_H(\nabla_Z(p) \mid q)$.
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 $a \notin \mathrm{first}(Z)$. If we would have $a \notin H$, then $a \in \mathrm{first}(\mathrm{tr}_{\{t\}} \circ v^t_{\{a\}} \circ \partial_H(a)) \subseteq \mathrm{first}(\mathrm{tr}_{\{t\}} \circ v^t_{\{a\}} \circ \partial_H(a \mathbin{\lfloor\!\lfloor} q)) \subseteq \mathrm{first}(\mathrm{tr}_{\{t\}} \circ v^t_{\{a\}} \circ \partial_H(a \parallel q)) \subseteq \mathrm{first}(Z)$, which is a contradiction. Therefore $a \in H$. Now we use induction on q.

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

Case 1.2. $q \equiv bx$, $b \in A$. Again $\partial_H(a \mid b) = \delta$, so $\partial_H(a \parallel bx) = \partial_H(a)\, \partial_H(bx) + \partial_H(b)\, \partial_H(a \parallel x) + \partial_H(a \mid b)\, \partial_H(x) = \delta \partial_H(bx) + \partial_H(b)\, \partial_H(\delta \parallel x) + \delta \partial_H(x)$ (use induction for the middle term) $= \partial_H(\delta \parallel bx)$. Statements 2–5 are easier.

Case 1.3. $q \equiv x + y$. Then $\partial_H(a \parallel (x + y)) = \partial_H(a)\, \partial_H(x + y) + \partial_H(x \mathbin{\lfloor\!\lfloor} a) + \partial_H(y \mathbin{\lfloor\!\lfloor} a) + \partial_H(a \mid x) + \partial_H(a \mid y) = \delta \partial_H(x + y) + \partial_H(x \mathbin{\lfloor\!\lfloor} \delta) + \partial_H(y \mathbin{\lfloor\!\lfloor} \delta) + \partial_H(\delta \mid x) + \partial_H(\delta \mid y)$ (induction on terms 2–5) $= \partial_H(\delta \parallel (x + y))$, and again, 2–5 are easier.

This finishes Case 1.

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

Claim. $(\partial/\partial a)(Z) \supseteq \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \parallel q)$.

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

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

so by definition $\sigma \in (\partial/\partial a)(Z)$.

Now we use induction on q.

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

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

and 2–5 are easier.

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

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

and 2–5 are easier.

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

$$\begin{aligned} \partial_H(ax \| (y+z)) &= \partial_H(a)\, \partial_H(x \| (y+z)) + \partial_H(y \mathbin{|\!\!\lfloor} ax) + \partial_H(z \mathbin{|\!\!\lfloor} ax) \\ &\quad + \partial_H(ax \mid y) + \partial_H(ax \mid z) \\ &= \partial_H(a)\, \partial_H(\nabla_{(\partial/\partial a)Z}(x) \| (y+z)) + \partial_H(y \mathbin{|\!\!\lfloor} \nabla_Z(ax)) \\ &\quad + \partial_H(z \mathbin{|\!\!\lfloor} \nabla_Z(ax)) + \partial_H(\nabla_Z(ax) \mid y) + \partial_H(\nabla_Z(ax) \mid z) \\ &= \partial_H(\nabla_Z(ax) \| (y+z)), \end{aligned}$$

and 2–5 are easier.

This finishes case 2.

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

CLAIM. $Z \supseteq \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \| q) \cup \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(y)} \circ \partial_H(y \| q)$.

Proof. Suppose $\sigma \in \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \| q)$. We can suppose $\sigma \neq \varepsilon$. Then

$$\begin{aligned}\sigma &\in \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \mathbin{\|\!\!\lfloor} q) \cup \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(q \mathbin{\|\!\!\lfloor} x)\\ &\cup \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \mid q).\end{aligned}$$

If

$$\sigma \in \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \mathbin{\|\!\!\lfloor} q),$$

then

$$\begin{aligned}\sigma &\in \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H((x + y) \mathbin{\|\!\!\lfloor} q) \subseteq \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(p \| q)\\ &\subseteq \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(p)} \circ \partial_H(p \| q) \subseteq Z.\end{aligned}$$

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

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

with

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

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

In either case, we can conclude by using an induction argument that $\tau' \in \mathrm{tr} \circ v^t_{\alpha(x)} \circ \partial_H((x + y) \| z)$, whence, in the first case

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

so $\sigma = \varepsilon_{\{t\}}(\tau) \in \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(p \| q) \subseteq Z$.

The other case is similar.

Lastly, if $\sigma \in \mathrm{tr}_{\{t\}} \circ v^t_{\alpha(x)} \circ \partial_H(x \mid q)$, then

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

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

Then we prove Case 3 by induction on q.

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

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

and 2–5 are easier.

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

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

and 2–5 are easier.

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

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

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 $F = \sum_{d \in D} s_1(d)\, r_4(d)\, r_1(\mathrm{ack}) F$ (meaning that F is the unique solution of this guarded recursive specification), then 2.11 shows that in the context

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

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

1. for all $d \in D$

$$\varepsilon,\, s_1(d),\, s_1(d)\, r_4(d),\, s_1(d)\, r_4(d)\, r_1(\mathrm{ack}) \in Z;$$

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

CLAIM 1. $Z = \mathrm{tr}_{\{t\}} \circ v'_{\alpha(E)} \circ \partial_H(E \| S \| R)$.

Proof. Using 2.11, we get

$$\begin{aligned}
&\mathrm{tr}_{\{t\}} \circ v'_{\alpha(E)} \circ \partial_H(E \| S \| R)\\
&\quad \mathrm{tr}_{\{t\}}\left(\sum_{d \in D} s_1(d)\, t\, r_4(d)\, t\, r_1(\mathrm{ack}) \cdot v'_{\alpha(E)} \circ \partial_H(E \| S \| R)\right)\\
&\quad = \bigcup_{d \in D} (s_1(d)^*\; \mathrm{tr}_{\{t\}}(t\, r_4(d)\, t\, r_1(\mathrm{ack}) \cdot v'_{\alpha(E)} \circ \partial_H(E \| S \| R))\\
&\quad = \bigcup_{d \in D} (s_1(d)^*\; (r_4(d)^*\; (r_1(\mathrm{ack})^* \mathrm{tr}_{\{t\}} \circ v'_{\alpha(E)} \circ \partial_H(E \| S \| R)))).
\end{aligned}$$

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

$$\frac{\partial}{\partial s_1(d)} Z = r_4(d)^*\, (r_1(\mathrm{ack})^*\, Z),\; \frac{\partial}{\partial r_4(d)}\left(\frac{\partial}{\partial s_1(d)} Z\right) = r_1(\mathrm{ack})^*\, Z$$

and

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

Then

$$\begin{aligned}
\nabla_Z(E) &= \nabla_Z\left(\left(\sum_{d \in D} s_1(d) + \sum_{d \in D} r_4(d) + r_1(\mathrm{ack})\right) E\right)\\
&= \sum_{d \in D} \nabla_Z(s_1(d) E) + \sum_{d \in D} \nabla_Z(r_4(d) E) + \nabla_Z(r_1(\mathrm{ack}) E)
\end{aligned}$$

$$= \sum_{d \in D} \partial_{A - \{s_1(d):\, d \in D\}}(s_1(d))\, \nabla_{(\partial/\partial s_1(d))Z}(E) + \delta + \delta$$

$$= \sum_{d \in D} s_1(d)\, \nabla_{r_4(d)^* (r_1(\text{ack})^* Z)}(E)$$

$$= \sum_{d \in D} s_1(d)\, \partial_{A - \{r_4(d)\}}(r_4(d))\, \nabla_{r_1(\text{ack})^* Z}(E) + \delta$$

$$= \sum_{d \in D} s_1(d)\, r_4(d)\, \partial_{A - \{r_1(\text{ack})\}}(r_1(\text{ack}))\, \nabla_Z(E) + \delta$$

$$= \sum_{d \in D} s_1(d)\, r_4(d)\, r_1(\text{ack})\, \nabla_Z(E).$$

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. Thus, executing process $\lambda_s(x)$ means that we start the execution of 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

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

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 and Tucker (1985), where asynchronous communication was described as a mechanism with effect on the state.

5.2. Definition. Let M and S be two given sets (with M finite), 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

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

Now we extend the signature of concrete process algebra with operators

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

and extend the set of axioms by

$$\lambda_s^m(a) = a(m, s) \qquad l1$$

$$\lambda_s^m(ax) = a(m, s)\, \lambda_{s(m,\,a)}^m(x) \qquad l2$$

$$\lambda_s^m(x + y) = \lambda_s^m(x) + \lambda_s^m(y) \qquad l3$$

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 $b_H = \lambda_s^m$ follows.

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

5.5. Theorem. *If there is no communication (i.e., γ is trivial), and $\alpha(x) \cap \alpha(m_2) = \alpha(y) \cap \alpha(m_1) = \varnothing$, then $\lambda_{s_1}^{m_1} \circ \lambda_{s_2}^{m_2}(x \parallel y) = \lambda_{s_1}^{m_1}(x) \parallel \lambda_{s_2}^{m_2}(y)$.*

Proof. In fact, we do not need to assume that γ is trivial but it is enough if the following statement is satisfied:

$$\alpha(x) \mid \alpha(y) = \varnothing \text{ and } \{a(m_1, s) \colon a \in \alpha(x), s \in S\} \mid \{a(m_2, s) \colon a \in \alpha(y), s \in S\} = \varnothing.$$

The proof consists of a simultaneous induction on x and y to prove the statement above simultaneously with two similar statements, with a left-merge (resp. communication merge) instead of the merge operator. The proof follows the lines of the proof of 4.21 (only simpler), 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 $\alpha(x) \cap \alpha(m_2) \neq \emptyset$ or $\alpha(y) \cap \alpha(m_1) \neq \emptyset$, we have so-called *shared variables*, a situation that is also described in GPL (in the absence of communication), see Owicki and Gries (1976).

5.7. Example I. Suppose we have a serial switch as depicted in Fig. 4. (For the formulation of this example, we are grateful to Jos Vrancken.) The switches A and B are given by

$$A = aA$$

$$B = bB$$

(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 $\langle i, j\rangle$ 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*. We do this by listing all the relevant instances of axiom *I*2.

$$\lambda^m_{\langle i,j\rangle}(ax) = \begin{cases} \text{on}(a) \cdot \lambda^m_{\langle 1-i,j\rangle}(x) & \text{if } \ i \neq j \\ \text{off}(a) \cdot \lambda^m_{\langle 1-i,j\rangle}(x) & \text{if } \ i = j \end{cases}$$

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

$$\lambda^m_{\langle i,j\rangle}(bx) = \begin{cases} \text{on}(b) \cdot \lambda^m_{\langle i,1-j\rangle}(x) & \text{if } \ i \neq j \\ \text{off}(b) \cdot \lambda^m_{\langle i,1-j\rangle}(x) & \text{if } \ i = j. \end{cases}$$

We assume there is no communication, so $\gamma(a, b) = \delta$. Now suppose we start in state $\langle 0, 1\rangle$ (so the lamp is off), then we have process $P = \lambda^m_{\langle 0,1\rangle}(A \parallel B)$.

Claim. $P = (\text{on}(a) + \text{on}(b))(\text{off}(a) + \text{off}(b))P$.

Proof. We use RSP to show

$$\lambda^m_{\langle 0,1\rangle}(A \parallel B) = \lambda^m_{\langle 1,0\rangle}(A \parallel B)$$

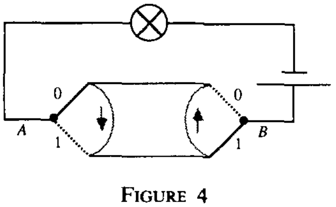

Figure 4

and

$$\lambda^m_{\langle 0,0\rangle}(A \parallel B) = \lambda^m_{\langle 1,1\rangle}(A \parallel B).$$

Then

$$\begin{aligned} P &= \lambda^m_{\langle 0,1\rangle}(A \parallel B) \\ &= \lambda^m_{\langle 0,1\rangle}((a+b)(A \parallel B)) \\ &= \mathrm{on}(a) \cdot \lambda^m_{\langle 1,1\rangle}(A \parallel B) + \mathrm{on}(b) \cdot \lambda^m_{\langle 0,0\rangle}(A \parallel B) \\ &= (\mathrm{on}(a) + \mathrm{on}(b))\, \lambda^m_{\langle 0,0\rangle}(A \parallel B) \\ &= (\mathrm{on}(a) + \mathrm{on}(b))(\mathrm{off}(a)\, \lambda^m_{\langle 1,0\rangle}(A \parallel B) \\ &\quad + \mathrm{off}(b)\, \lambda^m_{\langle 0,1\rangle}(A \parallel B)) \\ &= (\mathrm{on}(a) + \mathrm{on}(b))(\mathrm{off}(a) + \mathrm{off}(b))\, \lambda^m_{\langle 0,1\rangle}(A \parallel B) \\ &= (\mathrm{on}(a) + \mathrm{on}(b))(\mathrm{off}(a) + \mathrm{off}(b))\, P. \end{aligned}$$

5.8. Example II. Random walk. 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

$$\begin{aligned} A &= (l_A + r_A)A + h_A \\ B &= (l_B + r_B)B + h_B \end{aligned}$$

(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 $S = \{0, 1, 2, 3, 4\} \times \{0, 1, 2, 3, 4\}$. We list the relevant instances of axiom $l2$:

$$\lambda_{\langle i,j\rangle}(l_A x) = \begin{cases} \delta & \text{if } \; i = 0 \text{ or } j = i - 1 \\ l_A \cdot \lambda_{\langle i-1,j\rangle}(x) & \text{otherwise,} \end{cases}$$

$$\lambda_{\langle i,j\rangle}(r_A x) = \begin{cases} \delta & \text{if } \; i = 4 \text{ or } j = i + 1 \\ r_A \cdot \lambda_{\langle i+1,j\rangle}(x) & \text{otherwise,} \end{cases}$$

$$\lambda_{\langle i,j\rangle}(h_A x) = h_A(i) \cdot \lambda_{\langle i,j\rangle}(x) \qquad (A \text{ halts at } i);$$

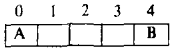

Figure 5

and we have similar equations for B:

$$\lambda_{\langle i,j\rangle}(l_B x)=\begin{cases}\delta & \text{if } j=0 \text{ or } i=j-1\\ l_B\cdot\lambda_{\langle i,j-1\rangle}(x) & \text{otherwise;}\end{cases}$$

$$\lambda_{\langle i,j\rangle}(r_B x)=\begin{cases}\delta & \text{if } j=4 \text{ or } i=j+1\\ r_B\cdot\lambda_{\langle i,j+1\rangle}(x) & \text{otherwise;}\end{cases}$$

$$\lambda_{\langle i,j\rangle}(h_B x)=h_B(j)\cdot\lambda_{\langle i,j\rangle}(x).$$

Then the situation pictured in Fig. 5 is described by

$$\lambda_{\langle 0,4\rangle}(A \parallel B).$$

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

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

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

The main tool used in proving this is Koomen's fair abstraction rule (KFAR, see Baeten, Bergstra and Klop, 1987b). We will not give the proof here, since it is outside the scope of this paper.

5.9. The following two statements list conditions which enable us to interchange two state operators, or a state operator and an encapsulation operator. We omit the proofs.

1. If $H\cap\alpha(m)=\emptyset$ 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 $\alpha(x)\cap\alpha(m_1)\cap\alpha(m_2)=\emptyset$ 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: $A\times M\times S\rightarrow \text{Pow}(A)$. The state following will depend on the alternative chosen, so eff: $A\times A\times M\times S\rightarrow S$, 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

$$\Lambda_s^m(a) = \sum_{b \in a(m,s)} b \qquad \text{L1}$$

$$\Lambda_s^m(ax) = \sum_{b \in a(m,s)} b \cdot \Lambda_{s(m,a,b)}^m(x) \qquad \text{L2}$$

$$\Lambda_s^m(x+y) = \Lambda_s^m(x) + \Lambda_s^m(y) \qquad \text{L3}$$

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 1978). 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 $c?d \mid c!\,d = c\#d$ (d is *communicated along channel* c).

We implement this as follows: Take $M = X$, $S = D$, then we define *act* and *eff* so that

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

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

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 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: 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:

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

and for any $d \in \mathbb{Z}_n$, and any process q:

$$\begin{aligned}
\Lambda^x_d(\text{read}(x)q) &= \sum_{e \in \mathbb{Z}_n} r(e)\, \Lambda^x_e(q)\\
\Lambda^y_d((y := 0)q) &= (y := 0) \cdot \Lambda^y_0(q)\\
\Lambda^y_d(\text{write}(y)q) &= w(d) \cdot \Lambda^y_d(q)\\
\Lambda^x_0(x \neq 0 \cdot q) &= \delta\\
\Lambda^x_d((x := ss(y))q) &= (y := ss(y)) \cdot \Lambda^y_{s(s(d))}(q)\\
\Lambda^x_d((x := p(x))q) &= (x := p(x)) \cdot \Lambda^x_{p(d)}(q)\\
\Lambda^x_d((x = 0)q) &= \delta \qquad \text{if} \quad d \neq 0
\end{aligned}$$

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. $t_I \circ \Lambda^x_0 \circ \Lambda^y_0(P) = \sum_{e \in \mathbb{Z}_n} r(e)\, t^{2+3e} w(2e(\text{mod}\, n))$. (Note. If the program contains statements of the form $x := y$, we have to use an operator $\Lambda^{\langle x, y\rangle}_{\langle d, e\rangle}$ instead of $\Lambda^x_d \circ \Lambda^y_e$.) We will sketch the proof of the claim by taking $n = 2$, so $\mathbb{Z}_n = \{0, 1\}$. Then

$$\begin{aligned}
&t_I \circ \Lambda^x_0 \circ \Lambda^y_0(P)\\
&\quad = t_I \circ \Lambda^x_0(\text{read}(x) \cdot \Lambda^y_0((y := 0)\, Z\, \text{write}(y))\\
&\quad = t_I(r(0) \cdot \Lambda^x_0((y := 0) \cdot \Lambda^y_0(Z\, \text{write}(y)))\\
&\qquad + r(1) \cdot \Lambda^x_1((y := 0) \cdot \Lambda^y_0(Z\, \text{write}(y))))\\
&\quad = r(0)\, t \cdot t_I(\delta + (x = 0) \cdot \Lambda^x_0 \circ \Lambda^y_0(\text{write}(y)))\\
&\qquad + r(1)\, t \cdot t_I((x \neq 0) \cdot \Lambda^x_1 \circ \Lambda^y_0((y := ss(y) \cdot (x := p(x)) \cdot Z \cdot \text{write}(y)))\\
&\quad = r(0) \cdot t \cdot t \cdot w(0) + r(1) \cdot t \cdot t \cdot t_I((y := ss(y))\\
&\qquad \cdot (x := p(x))\, \Lambda^x_0 \circ \Lambda^y_0(Z \cdot \text{write}(y)))\\
&\quad = r(0)\, ttw(0) + r(1)\, tttt t_I(\delta + (x = 0)\, \Lambda^x_0 \circ \Lambda^y_0(\text{write}(y)))\\
&\quad = r(0)\, ttw(0) + r(1)\, tttttw(0).
\end{aligned}$$

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

$$\text{queue} = \text{read}^{\omega} \parallel \text{write}^{\omega},$$

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 $M = \{\langle 1, 2\rangle\}$ (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*:

$$A_{\sigma}^{\langle 1,2\rangle}(\text{read} \cdot q) = \sum_{d \in D} r_1(d) \cdot A_{\sigma * d}^{\langle 1,2\rangle}(q)$$

$$A_{\varepsilon}^{\langle 1,2\rangle}(\text{write} \cdot q) = \delta$$

$$A_{\sigma * d}^{\langle 1,2\rangle}(\text{write} \cdot q) = s_2(d) \cdot A_{\sigma}^{\langle 1,2\rangle}(q).$$

Claim. $Q = A_{\varepsilon}^{\langle 1,2\rangle}(\text{read}^{\omega} \parallel \text{write}^{\omega})$.

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

$$R_{\sigma} = A_{\sigma}^{\langle 1,2\rangle}(\text{read}^{\omega} \parallel \text{write}^{\omega}).$$

Then:

1. $R_{\varepsilon} = A_{\varepsilon}^{\langle 1,2\rangle}(\text{read}(\text{read}^{\omega} \parallel \text{write}^{\omega}) + \text{write}(\text{read}^{\omega} \parallel \text{write}^{\omega}))$
$= \sum_{d \in D} r_1(d) \cdot A_{d}^{\langle 1,2\rangle}(\text{read}^{\omega} \parallel \text{write}^{\omega}) + \delta = \sum_{d \in D} r_1(d)\, R_d$.
2. $R_{\sigma * d} = A_{\sigma * d}^{\langle 1,2\rangle}(\text{read}(\text{read}^{\omega} \parallel \text{write}^{\omega}) + \text{write}(\text{read}^{\omega} \parallel \text{write}^{\omega}))$
$= \sum_{e \in D} r_1(e) \cdot A_{\sigma * d * e}^{\langle 1,2\rangle}(\text{read}^{\omega} \parallel \text{write}^{\omega}) + s_2(d)\, A_{\sigma}^{\langle 1,2\rangle}(\text{read}^{\omega} \parallel \text{write}^{\omega})$
$= \sum_{e \in D} r_1(e)\, R_{\sigma * d * e} + s_2(d)\, R_{\sigma}$.

Therefore, the R_{σ} satisfy the equations for the Q_{σ} in 3.1, so by RSP $R_{\sigma} = Q_{\sigma}$.

Received May 1986; accepted December 14, 1987

References

Baeten, J. C. M., and Bergstra, J. A. (1985), "Global Renaming Operators in Concrete Process Algebra," Report CS-R8521, Centre for Mathematics and Computer Science, Amsterdam.

Baeten, J. C. M., Bergstra, J. A., and Klop, J. W. (1987a), Conditional axioms and α/β calculus in process algebra, *in* "Proceedings, IFIP Conf. on Formal Description of Programming Concepts-III Ebberup, 1986," (M. Wirsing, Ed.), pp. 53–75, North-Holland, Amsterdam.

BAETEN, J. C. M., BERGSTRA, J. A., AND KLOP, J. W. (1987b), On the consistency of Koomen's fair abstraction rule, *Theoret. Comput. Sci.* **51** (1/2) 129–176.

BAETEN, J. C. M., AND VAN GLABBEEK, R. J. (1987), Another look at abstraction in process algebra, *in* "Proceedings, 14th ICALP, Karlsruhe" (Th. Ottmann, Ed.), Lecture Notes in Comput. Sci. Vol. 267, pp. 84–94, Springer-Verlag, New York/Berlin.

DE BAKKER, J. W., AND ZUCKER, J. I. (1982), Processes and the denotational semantics of concurrency, *Inform. and Control* **54** (1/2) 70–120.

BERGSTRA, J. A., AND KLOP, J. W. (1986), Algebra of communicating processes, *in* "Proceedings, CWI Symp. Math. and Comput. Sci. (J. W. de Bakker, M. Hazewinkel, and J. K. Lenstra, Eds.) pp. 89–138, North-Holland, Amsterdam, 1986.

BERGSTRA, J. A., AND KLOP, J. W. (1985). Algebra of communicating processes with abstraction, *Theoret. Comput. Sci.* **37** (1) 77–121.

BERGSTRA, J. A., AND KLOP, J. W. (1984), Process algebra for synchronous communication, *Inform. and Control* **60** (1/3) 109–137.

BERGSTRA, J. A., KLOP, J. W., AND OLDEROG, E.-R. (1987), Readies and failures in the algebra of communicating processes, Report CS-R8748, Centre for Mathematics and Computer Science, Amsterdam; *SIAM J. Comput.*, in press.

BERGSTRA, J. A., KLOP, J. W., AND TUCKER, J. V. (1985), Process algebra with asynchronous communication mechanisms, *in* "Proceedings, Seminar on Concurrency" (S. D. Brookes, A. W. Roscoe, G. Winskel, Eds.), pp. 76–95, Lecture Notes in Comput. Sci. Vol. 197, Springer-Verlag, New York/Berlin.

BERGSTRA, J. A., AND TIURYN, J. (1987), Process algebra semantics for queues, *Fund. Inform.* **10** 213–244.

BERGSTRA, J. A., AND TUCKER, J. V. (1985), Top-down design and the algebra of communicating processes, *Sci. Comput. Programming* **5** (2) 171–199.

BROOKES, S. D., HOARE, C. A. R., AND ROSCOE, A. W. (1984), A theory of communicating sequential processes, *J. Assoc. Comput. Mach.* **31** (3) 560–599.

ENGELFRIET, J. (1985), Determinacy → (observation equivalence = trace equivalence), *Theoret. Comput. Sci.* **36** (1) 21–25.

GROENVELD, R. A. (1987), "Verification of a Sliding Window Protocol by Means of Process Algebra," Report P8701, Programming Research Group, University of Amsterdam.

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

MILNER, R. (1980), "A Calculus of Communicating Systems," Lecture Notes in Comput. Sci. Vol. 92, Springer-Verlag, New York/Berlin.

OWICKI, S., AND GRIES, D. (1983), An axiomatic proof technique for parallel programs, *Acta Inform.* **6** 319–340.

REM, M. (1983), Partially ordered computations, with applications to VLSI design, *in* "Proceedings, 4th Advanced Course on Found. of Comp. Sci., part 2" (J. W. de Bakker and J. van Leeuwen, Eds.), MC Tract 159, pp. 1–44, Math. Centre, Amsterdam.

VAANDRAGER, F. W. (1986), "Verification of Two Communication Protocols by Means of Process Algebra," Report CS-R8608, Centre for Math. and Comput. Sci., Amsterdam.