Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

W.P. Weijland

The algebra of synchronous processes

Computer Science/Department of Software Technology Report CS-R8807 January

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

The Algebra of Synchronous Processes

W.P. Weijland

Centre for Mathematics and Computer Science

Kruislaan 413, 1098 SJ Amsterdam

abstract: An algebraical theory called ASP is presented, describing synchronous cooperation of processes. The theory ASP was first introduced in BERGSTRA & KLOP [3] as an alternative for the theory ACP, which works with asynchronous cooperation (see also in [5]). One of the main differences between ASP, as it is presented here and the algebraical theory SCCS of MILNER [8] is the representation of parallelism, which is done by considering a computation step as a vector, each component of which represents an atomic action on the corresponding channel.
This paper concludes with an example, to give an idea how to work with ASP.

Key words and phrases: concurrency, process algebra, synchronous cooperation, direct product, hiding, abstraction, empty process.
1985 Mathematics Subject classification: 68N15, 68Q10, 68Q45, 68Q55.
1987 CR categories: D.3.1, F.1.2, F.3.2, F.4.3.

Report CS-R8807
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

contents:

1. INTRODUCTION
2. AN ALGEBRA OF SYNCHRONOUS PROCESSES
2.1 SOME REMARKS ON PARALLEL COMPOSITION
3. MODELS
3.1 THE TRANSITION MODEL
3.2 THE GRAPH MODEL
4. RECURSION
5. EXAMPLE: COMPUTER INTEGRATED MANUFACTURING

1. INTRODUCTION

In the current research on (hardware) verification one of the main goals is to find strong proof systems and tools to specify and verify designs of algorithms and architectures. For instance, in the development of integrated circuits ('chips') the important stage of testing a prototype (to save the high costs of producing defective processors) can be dealt with much more efficiently, when a strong verification tool is available. Therefore, developing a verification theory has very high priority and is subject of study at many universities and scientific institutions.

In BERGSTRA & KLOP [3] a theory called *Algebra of Communicating Processes* (ACP) is presented, which is an algebraical theory providing us with a formal description of concurrent processes. In ACP, parallelism is described as interleaving and therefore, in ACP we have *asynchronous* cooperation of parallel processes. In many cases it turns out, however, that a process can be described much easier in a clocked network instead. Therefore in [3] a variant on ACP, called *Algebra of Synchronous Processes* (ASP), was suggested in which *synchronous* cooperation could be modeled.

In this paper we will present the algebra ASP in full detail. Especially the specific language used here is quite different from the usual approaches in BERGSTRA & KLOP [3], MILNER [8] and HENNESSY [10]. Especially the fact that parallel composition is represented by taking *vectors* of atomic actions will turn out to simplify the theoretical aspects of the theory.

The idea of developing an algebraic theory for synchronous processes is not new. In fact, the algebra ASP presented in this paper is very similar to the theory SCCS of Milner. There also are some important differences between both theories:

- Since in ASP we use a vector notation to represent parallelism (instead of a new operator as in SCCS and ACP) the theory has a smaller signature. For this reason one may expect that the study of its theoretical aspects will become much easier to deal with. For example, in SCCS the operator $\times$ stands for both parallel composition and communication. In ASP we have the operator $|$ which stands for communication only.

- In SCCS we have handshaking, i.e. in any communication one has at most two participants. In ASP this restriction is not needed.
- Abstraction is dealt with in ASP by use of a general renaming operator (like in ACP) whereas SCCS has abstraction automatically which is a restriction, since we do not always want to hide all communication actions of a process. In ASP one can abstract automatically (if it is convenient) by chosing an appropriate communication function.
- As indicated before in SCCS we have an operator $\times$. Now assume in a large and complex configuration we want to evaluate an expression of the form $t=a\times b\times\bar{a}\times c\times b\times....$ then we need to find all pairs of the form $(a,\bar{a})$ such that both a and a are subterms of t. This is quite an elaborate job when the complexity of t will become larger. One could solve this problem by not considering t as a term but as a sequence of symbols with one coordinate for every symbol: $t=[(a,\bar{a}), (b,b), (c), ...]$ and then we can evaluate t in linear time. In ASP this evaluation method follows immediately from the construction of vectors of atomic actions and the definition of the communication function.

After having introduced the algebraic theory ASP we will consider two of its models, which may be looked at as an operational and a denotational semantics for ASP. Next we will introduce the notion of recursion and finally study a particular example, in order to illustrate in which way one can work with ASP in practical cases.

At this place I especially want to thank Jos Baeten, who took the time to correct several draft versions of this paper and who convincingly rejected the concept of parallel composition by considering direct products on algebras.

2. AN ALGEBRA OF SYNCHRONOUS PROCESSES

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

On intuitive grounds x(y + z) and xy + xz present different mechanisms (the moment of choice is different), and therefore an axiom x(y + z) = xy + xz is not included.

We have a special constant $\delta \in A$ denoting deadlock, the acknowledgement of a process that it cannot do anything anymore, the absence of an alternative. Axioms A6-7 give the laws for δ.

In table 1 all axioms of the *Basic Process Algebra* BPA_δ are presented (see BERGSTRA & KLOP [3]).

$x + y = y + x$	A1
$x + (y + z) = (x + y) + z$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x + \delta = x$	A6
$\delta x = \delta$	A7

table 1. Basic Process Algebra BPA_δ.

The following proposition provides us with a useful tool for induction methods.

proposition 2.1 Suppose t is a closed term in BPA_δ then t can be written in one of the following

forms:

1. t is a constant from A
2. t is of the form u + v, where u and v are closed terms of less complexity (depth) than t
3. t is of the form a·u, where a is a constant and u is a closed term of less complexity than t.

Next, suppose a port P is associated to our basic algebra BPA_δ, and suppose we have a binary function $|$ which is both commutative and associative. We may look at $a \mid b$ as a communication action at port P, which is the result of simultaneously performing a and b.
Furthermore, assume there exists a unit element $1 \in A$ such that $1 \mid x = x \mid 1 = x$ for all $x \in A$. This unit element stands for an idle action during which a process is still running but not performing any significant step. The notion of an idle action was first introduced by MILNER [8] in a different setting.
In the sequel we assume $|$ to bind stronger than + but weaker than $\cdot$. It follows immediately that:

proposition 2.2 $(A, |, 1)$ is an Abelian monoid.

A function $|$ as described so far is called a *communication function* if δ is a zero element for $|$, i.e. if for all $x \in A$ we have $\delta \mid x = \delta$. The symbol δ is chosen here because of its long tradition in process algebra, especially in ACP.
Resuming, we find that $|$ is a communication function if the following conditions are satisfied (see table 2) ($a \in A$):

$x \mid y = y \mid x$	C1
$x \mid (y \mid z) = x \mid (y \mid z)$	C2
$\delta \mid x = \delta$	C3
$1 \mid x = x$	C4

table 2. Communication function ($a \in A$).

So far, we only considered $|$ to be defined on atomic actions. This definition can be extended to processes over BPA_δ as follows. Assume two processes a·b and c·d are both performed at port P,

then | acts as a *synchronous communication merge* on both processes, i.e. $(a\cdot b) \mid (c\cdot d) = (a \mid c)\cdot(b \mid d)$ So from two BPA_δ-processes we can construct a new BPA_δ-process by 'stepwise communication'. This intuition can be formally described by adding the following axioms to our algebra for all $a,b\in A$ (see table 3):

$ax \mid b = (a \mid b)x$	SC1
$ax \mid by = (a \mid b)(x \mid y)$	SC2
$(x+y) \mid z = x \mid z + y \mid z$	SC3

table 3. Communication merge on processes ($a,b\in A$).

Another way to look at | is as follows: from axiom C2 it follows that in expressions with only |, we may leave out the brackets; thus we write $a \mid a \mid b \mid c$ instead of $((a \mid (a \mid b)) \mid c)$. Therefore, we may consider such expressions as *multisets* of atomic actions, which are all simultaneously performed at a certain channel attached to the algebra. Note that, in case one of the two processes terminates in one step (e.g. in $a \mid (b\cdot y)$), after the communication action $a \mid b$ the process continues with y, which fits into the idea of a multiset representation of actions.

Next, we introduce *renaming* operators on BPA_δ-processes (see BAETEN & BERGSTRA [2]). In fact, renamings are functions from the set of atomic actions A into A. Assume $f:A\to A$ is a function on A, a so-called *atomic renaming*. Then in table 4 the axioms of the renaming function ρ_f are presented.

$\rho_f(\delta) = \delta$	R1
$\rho_f(1) = 1$	R2
$\rho_f(a) = f(a) \quad (a\neq\delta,1)$	R3
$\rho_f(x+y) = \rho_f(x) + \rho_f(y)$	R4
$\rho_f(xy) = \rho_f(x)\cdot\rho_f(y)$	R5
$(\rho_f\circ\rho_g)(x) = \rho_{f\circ g}(x)$	R6

table 4. Renaming in ASP ($a\in A$).

A specific example of an atomic renaming is the one which renames all constants from a certain set $I \subseteq A$ into one particular constant $r \in A$, leaving all other elements from A untouched. The renaming

$r_I(\delta) = \delta$	
$r_I(1) = 1$	
$r_I(a) = r$	for every $a \in I$ $(a \neq \delta, 1)$
$r_I(a) = a$	for every $a \notin I$
$r_I(x + y) = r_I(x) + r_I(y)$	
$r_I(xy) = r_I(x) \cdot r_I(y)$	
$(r_I \circ r_J)(x) = r_{I \cup J}(x)$	

table 5. Simple renamings in ASP ($r,a \in A$, $I,J \subseteq A$).

function that results from such an atomic renamings is denoted by r_I and will be referred to as a *simple* renaming function. In table 5 the rules of table 4 are translated to simple renaming functions. Having the axioms R1-R6 we do not need to add them to our system ASP.

$x + y = y + x$		A1	$x \mid y = y \mid x$	C1
$x + (y + z) = (x + y) + z$		A2	$(x \mid y) \mid z = x \mid (y \mid z)$	C2
$x + x = x$		A3	$\delta \mid x = \delta$	C3
$(x + y)z = xz + yz$		A4	$1 \mid x = x$	C4
$(xy)z = x(yz)$		A5		
$x + \delta = x$		A6	$ax \mid b = (a \mid b)x$	SC1
$\delta x = \delta$		A7	$ax \mid by = (a \mid b)(x \mid y)$	SC2
			$(x+y) \mid z = x \mid z + y \mid z$	SC3
$\rho_f(\delta) = \delta$		R1	$\rho_f(x + y) = \rho_I(x) + \rho_I(y)$	R4
$\rho_f(1) = 1$		R2	$\rho_f(xy) = \rho_I(x) \cdot \rho_I(y)$	R5
$\rho_f(a) = f(a)$	$(a \neq \delta, 1)$	R3	$(\rho_f \circ \rho_g)(x) = \rho_{f \circ g}(x)$	R6

table 6. ASP(A).

In table 6 all axioms, introduced so far, are presented together. The algebra which is thus constituted, will be called *Algebra of Synchronous Processes*, or ASP for short. Since the sets of atomic actions A is a parameter of ASP, we will often write ASP(A). However, if A is some arbitrary fixed set then we will write ASP for short. In fact ASP is an *axiom scheme* since we have its axioms for any pair of constants a,b$\in$ A, all sets I$\subseteq$A and all functions f:A$\to$A.

We turn the axioms of table 6 into a term rewrite system in order to be able to define normal forms in ASP. The resulting rewrite system will be called RASP.

definition 2.1 The term rewrite system RASP can be found from table 6 by omitting the axioms A1, A2, C1 and C2 and next replacing all occurences of '=' by $\to$. Since we do not have A1 and C1 as rewrite rules the axioms A6, C3, C4, SC1 and SC3 have to be included twice (see table 7).

$x+x \to x$	RA1	$\delta \mid x \to \delta$	RC1
$(x+y)z \to xz+yz$	RA2	$x \mid \delta \to \delta$	RC2
$(xy)z \to x(yz)$	RA3	$1 \mid x \to x$	RC3
$x+\delta \to x$	RA4	$x \mid 1 \to x$	RC4
$\delta+x \to x$	RA5		
$\delta x \to \delta$	RA6		
$\rho_f(\delta) \to \delta$	RR1		
$\rho_f(1) \to 1$	RR2	$ax \mid b \to (a \mid b)x$	RS1
$\rho_f(a) \to f(a)$ $(a\neq\delta,1)$	RR3	$a \mid bx \to (a \mid b)x$	RS2
$\rho_f(x+y) \to \rho_I(x)+\rho_I(y)$	RR4	$ax \mid by \to (a \mid b)(x \mid y)$	RS3
$\rho_f(xy) \to \rho_I(x)\cdot\rho_I(y)$	RR5	$(x+y) \mid z \to x \mid z+y \mid z$	RS4
$(\rho_f\circ\rho_g)(x) \to \rho_{f\circ g}(x)$	RR6	$x \mid (y+z) \to x \mid y+x \mid z$	RS5

table 7. RASP(A).

RASP is a term rewrite system on ASP-terms *modulo* commutativity and associativity of + and |, so we may consider a RASP-normal form to be built from multisets of summands and communications.

proposition 2.3 RASP is strongly terminating.

proposition 2.4 If t is a normal form with respect to RASP then it is a BPA_δ-term.

The proof of proposition 2.3 can be found by using structural induction on ASP-terms. It says that the term rewrite system RASP has no infinite reductions. Proposition 2.4 can easily be proved by structural induction on terms that are *not* a BPA_δ-term (hence containing at least one occurrence of |) and by showing that such a term is always the instantiation of the lefthand side of some rule from RASP. For further information we recommend the reader to consult [-] in which a similar proof is presented in full detail.

theorem 2.5 (elimination)
For any closed ASP-term s, there exists a closed BPA_δ-term t such that $ASP \vdash s=t$.

proof From proposition 2.3 it follows that any ASP term s has a reduction to a normal form t which is a BPA_δ-term (according to proposition 2.4). Such a reduction represents a proof in ASP and hence we find $ASP \vdash s=t$. □

So starting from any closed term with | we can find a derivation, using equations from ASP, to a closed term without these operators (i.e. a closed BPA_δ-term).

proposition 2.6 RASP is confluent.

proof Consider the rewrite system RBPA which is the restriction of RASP to the rules RA1-RA6. It is easy to prove that RBPA is confluent (a formal proof is omitted here). By propositions 2.3 and 2.4 it then follows immediately that RASP is confluent as well. □

We find that every ASP term has a normal form (proposition 2.3) which is a BPA_δ-term

(proposition 2.4) and which is unique modulo the ordering of the summands and communications (proposition 2.6). As a result we find:

theorem 2.7 ASP is a conservative extension of BPA_δ.

proof Since $BPA_\delta \subseteq ASP$, it is clear that ASP is an extension. Now assume $ASP \vdash s=t$ for two BPA_δ-terms s and t, then there exists a proof in ASP consisting of equations $s=u_1$, $u_i=u_{i+1}$, $u_k=t$ $(0<i<k)$ that are closed instances of axioms from ASP. Clearly, every equation is an instance or a context of an instance of a rule or the reverse of a rule in RASP, and therefore by propositions 2.3 and 2.6 s, u_i $(0<i<k)$ and t all have the same normal form. Since both s and t are BPA_δ-terms, any rule that is applicable to them has to be a closed instance of one of the rules RA1-RA6 (all other rules do not have BPA_δ-terms as lefthand sides). However the righthand sides of the rules RA1-RA6 are BPA_δ-terms again, so any reduction of s (or t) to a normal form is a proof in BPA_δ. Since s and t both share the same normal form we find $BPA_\delta \vdash s=t$. □

It is important to see a conceptual difference between $|$ and the $\times$-operator in SCCS, introduced by MILNER [8]. As is indicated above, $|$ should be interpreted as a *communication* function, working on a certain port P which is associated to the algebra (see also [1]). In [-], however, $\times$ is introduced as a (synchronous) parallel composition operator, which is quite different from our notion of communication. Actually, Milner requires every $a \in A$ to have an inverse element $\bar{a}$, such that for all $a \in A$: $a \times \bar{a} = 1$ (hence, $(A-\{\delta\},\times,1,^-)$ is an Abelian group). So, an expression such as $a \times b \times \bar{a} \times c$ can be evaluated to $a \times \bar{a} \times b \times c$ (using commutativity of $\times$), which is equal to $1 \times b \times c$, and thus we obtain $b \times c$ (since 1 is a unit element). This expression, which is in normal form, should be interpreted as the parallel execution of two atomic actions. Note, that in SCCS we automatically abstract from communications such as $a \times \bar{a}$.
Naturally, the question arises how parallel composition can be represented in the theory ASP. Since all atomic actions of the form $a | b$ are considered as a communication and not as the parallel execution of two atoms, we have to find a new construct in our theory.

definition 2.2 Let $\mathcal{P}$ be a set of *ports* and assume A to be a fixed set of atomic actions.
Then $A^{\mathcal{P}}$ is defined as the set of all functions from $\mathcal{P}$ into A.

Functions $\mathbf{v} \in A^{\mathcal{P}}$ are called (atomic) *vectors* and represent the simultaneous execution of the atomic actions $\mathbf{v}(P)$ at all ports $P \in \mathcal{P}$. Vectors are considered to be the new atomic actions in our algebra $ASP(A^{\mathcal{P}})$.

example

Suppose a *buffer* B consists of two ports 'left' and 'right'. Assume $A=\{r(x),s(x),1,\delta: x \in \{a,b\}\}$, where r(x) stands for receiving the value x, and s(x) stands for sending x. A possible definition of B could read as follows:

$$B = (r(a)\ 1)\cdot(1\ s(a)) + (r(b)\ 1)\cdot(1\ s(b)).$$

So, B can receive a value (either a or b) from the left port and next send it away to the right port.

Note that in the signature of both ASP(A) and $ASP(A^{\mathcal{P}})$, there exist constants δ and 1. Although it is not necessary to identify these constants with atomic vectors (they both can exist in their own right) we often chose to interpret δ as the vector with only δ's at all its components, and 1 as the vector with all 1's. The *vectors* δ and 1 are denoted by $\boldsymbol{\delta}$ and $\mathbf{1}$ respectively. So $\boldsymbol{\delta}=(\delta\ \delta\ \ldots\ \delta)$ and $\mathbf{1}=(1\ 1\ \ldots\ 1)$.

Starting from a fixed algebra ASP(A), we have to define a new communication function between the atomic vectors (apart from the axioms of table 2 there are no further constraints on the choice of such a function). From the definition of the communication function on A, we often chose to define $\mid$ on atomic vectors from $A^{\mathcal{P}}$ as follows.

definition 2.3

Suppose $\mid$ is a communication function on A, then the *natural extension* of $\mid$ is defined by:
if v and w are two functions from $A^{\mathcal{P}}$ then for all $p \in \mathcal{P}$

$$(v \mid w)(P) = \begin{cases} \delta & \text{if for any } P \in \mathcal{P} \text{ we have } (v(P) \mid w(P)) = \delta \\ v(P) \mid w(P) & \text{otherwise.} \end{cases}$$

So the natural extension of a communication function results from applying the communication function at all ports separately but with the restriction this does not yield a deadlock, not at any port. Otherwise the whole communication fails, i.e. is equal to $\boldsymbol{\delta}$.

In the same way we define a natural extension of the renaming operators. Although $ASP(A^{\mathcal{P}})$ permits us to define different renamings, it turns out to be useful to define a natural extension of renamings from ASP(A).

definition 2.4

Let $f:A\to A$ be an atomic renaming on A. Then for all $v\in A^{\mathcal{P}}$ the *natural extension* $\mathbf{f}^{\mathcal{P}}$ of f is defined by: $\mathbf{f}^{\mathcal{P}}(v)(P) = f(v(P))$.

The natural extension $\boldsymbol{\rho}_f^{\mathcal{P}}$ of ρ_f is defined as: $\boldsymbol{\rho}_I^{\mathcal{P}} := \rho_{f^{\mathcal{P}}}$, often denoted by $\boldsymbol{\rho}_I$ if no confusion arises. Similarly, the natural extension of a simple renaming r_I is denoted by $\mathbf{r}_I^{\mathcal{P}}$ or $\mathbf{r}_I$.

Note that the natural extension of a simple renaming need not be simple. In section 5 we will find an application for the definitions 2.3 and 2.4.

2.1 SOME REMARKS ON PARALLEL COMPOSITION

Using atomic vectors, we find that the behaviour of a process in a more complex network consisting of more than only one port, can be described in the setting of the same simple algebra ASP. Of course one could chose a different approach for the problem of parallel composition. For instance, like in SCCS we could have introduced a new operator $\times$. As it turns out, however, we will have great difficulty in finding appropriate axioms for the interaction of $|$ and $\times$. Such an axiom system would at least contain the following five equations.

(i) $x\times y = y\times x$

(ii) $(x\times y)\times z = x\times(y\times z)$

(iii) $(a\cdot x)\times b = (a\times b)\cdot x$

(iv) $(a\cdot x)\times(b\cdot y) = (a\times b)\cdot(x\times y)$

(v) $(x + y)\times z = x\times z + y\times z$

Here $\times$ stands for parallel composition without communication (hence for all atomic actions $a,b\in A$ we find that $a\times b$ is in normal form). Note that these are in fact the same axioms as the ones for $|$, i.e. the axioms C1, C2, SC1, SC2 and SC3.

The question now is, how do we find simple rules for a term of the form $(x\times y) | z$? Assume we would add the following rule to our system:

(vi) $(x\times y) | z = (x | z)\times y + x\times(y | z)$

Now, assume z is a process which whishes to communicate with *both* x and y then such a process cannot be represented by $(x \times y) \mid z$ since rule (vi) implies that z does communicate only with either x or y. And this might not be what we want (see the example below which is due to J.A. Bergstra).

example (Bergstra)

Set $A=\{r_1(a), r_2(a), s_1(a), s_2(a), 1, \delta\}$, where $r_i(a)$ stands for receiving the value a from channel i and $s_j(a)$ stands for sending the value a to port j. Now define:

$$x = s_1(a)$$
$$y = s_2(a)$$
$$z = r_1(a) \times r_2(a)$$

and $\quad r_i(a) \mid s_i(a) = s_i(a) \mid r_i(a) = c_i(a)$

$$r_i(a) \mid s_j(a) = s_j(a) \mid r_i(a) = \delta \qquad (i \neq j).$$

Suppose we would have accepted rule (vi) then we would have had

$$(s_1(a) \times s_2(a)) \mid (r_1(a) \times r_2(a)) = [s_1(a) \mid (r_1(a) \times r_2(a))] + [s_2(a) \mid (r_1(a) \times r_2(a))] =$$
$$= [s_1(a) \mid r_1(a)] + [s_1(a) \mid r_2(a)] + [s_2(a) \mid r_1(a)] + [s_2(a) \mid r_2(a)] = c_1(a) + c_2(a).$$

This is not quite what we wanted, however, since obviously we had in mind that $(s_1(a) \times s_2(a)) \mid (r_1(a) \times r_2(a))$ would yield $c_1(a) \times c_2(a)$.

From this example we learn that we have to be much more careful in chosing the axioms for the interaction between $\mid$ and $\times$. For instance consider the following axioms:

(vii) $\qquad (x \times y) \mid a = (x \mid a) \times y + x \times (y \mid a)$

(viii) $\qquad (x \times y) \mid (a \times z) = ((x \mid a) \times y) \mid z + (x \times (y \mid a)) \mid z.$

Here we see that the rules for the interaction between $\times$ and $\mid$ will become rather complicated. In $ASP(A^{\mathcal{P}})$ this is overcome by the notion of atomic vectors, making it immediately clear which actions should communicate and which should not by defining a proper communication function. It is clear that the axioms (i)-(v), (vii) and (viii) correspond to a rewrite system with very complex normal forms and very long reduction sequences. Therefore, from a practical point of view, we have chosen here to present a different construction for parallel composition.

Looking at the particular representation of parallelism in ASP, using atomic vectors representing the simultaneous execution of atomic actions at various ports, one might think that such an algebra could be represented by a *direct product* of the algebra ASP(A) over $\mathcal{P}$. Direct products (or

cartesian products) of algebras over a set $\mathcal{P}$ are defined as given below.

definition 2.4 Let $\mathcal{P}$ be a set and assume $\mathcal{A}$ is an algebra in a language $L(\mathcal{A})$ with universe $|\mathcal{A}|$.
Then the *direct product* $\mathcal{A}(\mathcal{P})$ of $\mathcal{A}$ *over* $\mathcal{P}$ is an algebra for $L(\mathcal{A})$, defined as follows:

1. The domain of $\mathcal{A}^{\mathcal{P}}$ consists of all functions $f:\mathcal{P}\rightarrow|\mathcal{A}|$.
2. If a is a constant symbol in $L(\mathcal{A})$, then:
   $a^{\mathcal{A}(\mathcal{P})} := \lambda P\in\mathcal{P}.\ a^{\mathcal{A}}$.
3. If f is a k-ary function symbol in $L(\mathcal{A})$ and $f_1,...,f_k$ are elements from $|\mathcal{A}_{\mathcal{P}}|$, then define:
   $f^{\mathcal{A}(\mathcal{P})}(f_1,...,f_k) := \lambda P\in\mathcal{P}.\ f^{\mathcal{A}}(f_1(P),...,f_k(P))$.

In part 2 and 3 of the definition we use the lambda notation $\lambda P\in\mathcal{P}.t(P)$ to indicate the function which gives t(P) for any given $P\in\mathcal{P}$.
The main idea behind the notion of direct products is that the product domain consists of all vectors of elements from the original domain, and function symbols can be distributed over the components of the vectors. For instance (a 1) + (b b) = ((a + b) (1 + b)) and (a b)·(c d) = (a·b c·d). Now we could think of representing parallel composition by considering the algebra $ASP(\mathcal{P})$. Note that in the product algebra we have: $ASP(\mathcal{P})\models s=t$ if and only if for all $P\in\mathcal{P}$: $ASP\models s(P)=t(P)$. Thus we can prove $ASP(\mathcal{P})\models$ (a b) + (a δ) = ((a + a) (b + δ)) = (a b).
The reason why this approach does not work is because of the following example, which is due to J.C.M. Baeten.

example (Baeten)

Consider the buffer B from a previous example which consists of two ports.
Again we have A={r(x), s(x), 1, δ: x∈{a,b}} and B is defined by:

$$B = (r(a)\ 1)\cdot(1\ s(a)) + (r(b)\ 1)\cdot(1\ s(b)).$$

Now we prove
$ASP(\mathcal{P})\models (r(a)\ 1)\cdot(1\ s(a)) + (r(b)\ 1)\cdot(1\ s(b)) =$
$= ((r(a)\cdot 1 + r(b)\cdot 1)\ (1\cdot s(a) + 1\cdot s(b)) =$ (using axiom A1)
$= ((r(a)\cdot 1 + r(b)\cdot 1)\ (1\cdot s(b) + 1\cdot s(a)) =$
$= (r(a)\ 1)\cdot(1\ s(b)) + (r(b)\ 1)\cdot(1\ s(a))$.
So, $ASP(\mathcal{P})$ is not a trace consistent model for ASP.

Similar examples can be found based on using the axioms A3 or C1. Trace inconsistency can be considered as quite a severe problem, since the model identifies processes that have different traces.

3. MODELS

In this section we will study two models for ASP both of which provide us with a clear operational and a declarative semantics for ASP.

3.1 THE TRANSITION MODEL

The first model we will present here is the *transition model* (see also VAN GLABBEEK [9]). The way in which the model will be described here strongly resembles the presentation from [-].
On BPA_δ-terms, for each $a \in A-\{\delta\}$ we define binary predicates $\rightarrow^a$ and a unary predicate $\rightarrow^a \surd$. Intuitively, $x \rightarrow^a y$ means that process x can evolve into process y by executing a. $x \rightarrow^a \surd$ means that process x can terminate successfully. In table 9 the proof rules for these two predicates are presented. From now on we assume $\rightarrow^a$ and $\rightarrow^a \surd$ to be the minimal predicates that are closed under derivations from table 9.

a:	$a \rightarrow^a \surd$	$(a \neq \delta)$		
+:	$\dfrac{x \rightarrow^a x'}{x+y \rightarrow^a x'}$	$\dfrac{x \rightarrow^a \surd}{(x+y) \rightarrow^a \surd}$	$\dfrac{y \rightarrow^a y'}{x+y \rightarrow^a y'}$	$\dfrac{y \rightarrow^a \surd}{(x+y) \rightarrow^a \surd}$
·:	$\dfrac{x \rightarrow^a x'}{x \cdot y \rightarrow^a x' \cdot y}$	$\dfrac{x \rightarrow^a \surd}{x \cdot y \rightarrow^a y}$		

table 9. The transition predicates $\rightarrow^a$ and $\rightarrow^a \surd$ on BPA_δ-terms $(a \in A-\{\delta\})$.

definition 3.1 A *bisimulation* is a binary reflexive relation R on BPA_δ-terms, satisfying $(a \in A-\{\delta\})$:

1. If R(p,q) and $p \rightarrow^a p'$, then there exists q' such that $q \rightarrow^a q'$ and R(p',q');
2. If R(p,q) and $q \rightarrow^a q'$, then there exists p' such that $p \rightarrow^a p'$ and R(p',q');
3. If R(p,q), then $p \rightarrow^a \surd$ if and only if $q \rightarrow^a \surd$.

If there exists a bisimulation R between processes p and q, then we say p and q are *bisimilar*, notation: $p \leftrightarroweq q$.

theorem 3.1 $\leftrightarrows$ is a congruence relation on BPA_δ-terms.

The proof of theorem 3.1 is left to the reader. Recall that a relation is called a *congruence* if it is an equivalence relation which respects function symbols.
The key point of bisimulation equivalence is the fact that except for having the same traces, all moments of choices in the process are maintained . For instance note that $a(c + d) \not\leftrightarrows (ac + ad)$, since we have $a(c + d) \rightarrow^a (c + d)$ whereas only $(ac + ad) \rightarrow^a c$ and $(ac + ad) \rightarrow^a d$, and clearly we have neither $(c + d) \leftrightarrows c$ nor $(c + d) \leftrightarrows d$.

definition 3.2 The *transition model* $\mathbb{T}$ is the set of closed BPA_δ-terms modulo $\leftrightarrows$.

definition 3.3 A *basic term* is a closed BPA_δ-term defined inductively as follows:

1. All constants from A are basic terms.
2. If $t_0,\ldots,t_{n-1}$ are basic terms then so is $t = a_0t_0 + \cdots + a_{n-1}t_{n-1} + b_0 + \cdots + b_{m-1}$ for certain n,m with $n+m>0$, $a_i,b_j \in A$ and $a_i \neq \delta$.

A basic term is often written as $(\Sigma_{i<n}\, a_it_i + \Sigma_{j<m}\, b_j)$.

definition 3.4 The *depth* dp(t) of a basic term t is defined inductively as follows:

1. for all $a \in A$: $dp(a) = 1$
2. for all $a \in A-\{\delta\}$ and basic terms s: $dp(a\cdot s) = 1 + dp(s)$
3. for any two basis terms s and t: $dp(s + t) = \max(dp(t), dp(s))$.

proposition 3.2 For every BPA_δ-term s there exists a basic term t such that $BPA_\delta \vdash s=t$.

proposition 3.3 Let t be a basic term and $a \in A-\{\delta\}$. Then the following statements hold:

1. If $t \rightarrow^a s$, then s is a basic term and $dp(s)<dp(t)$;
2. If $t \rightarrow^a s$, then $BPA_\delta \vdash t = as + t$ (i.e. $a\cdot s$ is a *summand* of t);
3. If $t \rightarrow^a \surd$, then $BPA_\delta \vdash t = a + t$.

The proof of proposition 3.2 is easy, whereas proposition 3.3 can be proved using induction on dp(t). Both proofs are left to the reader (see also [-]). The following theorem is an important result

about the transition predicates of table 9.

theorem 3.4 For all closed BPA_δ-terms s and t we have: $BPA_\delta \vdash s=t \Rightarrow s \leftrightarrows t$.

proof We only need to prove that $\leftrightarrows$ respects all axioms of BPA_δ. For instance consider the axiom (A1) $(s + t) \leftrightarrows (t + s)$. Set $R = I \cup \{(s + t,t + s)\} \cup \{(t + s,s + t)\}$, where I is the binary identity relation. Assume we have R(p,q) then we find that either p and q are identical or $p = (s + t)$ and $q = (t + s)$. Now suppose $(s + t) \rightarrow^a u$ then this transition is an instance of one of the +-rules in table 9. Therefore it follows, that either $s \rightarrow^a u$ or $t \rightarrow^a u$ and so $(t + s) \rightarrow^a u$ (applying the +-rule again) and by definition we have R(u,u). In the same way it follows from $(s + t) \rightarrow^a \surd$ that $(t + s) \rightarrow^a \surd$. Hence we find that R is a bisimulation between $(s + t)$ and $(t + s)$.
In the same way we find that (A2) $((s + t) + u) \leftrightarrows (s + (t + u))$ and (A3) $(s + s) \leftrightarrows s$.
In order to prove (A4) $(s + t)u \leftrightarrows (su + tu)$ set $R = I \cup \{((s + t)u,su + tu)\} \cup \{(su + tu,(s + t)u)\}$, then it easily follows that R is a bisimulation between $(s + t)u$ and $(su + tu)$.
In the same way we find (A5) $(st)u \leftrightarrows s(tu)$. Note that δ nor $\delta \cdot x$ can be the left hand side of any transition, and therefore we have (A6) $s + \delta \leftrightarrows s$ and (A7) $\delta s \leftrightarrows \delta$. □

Clearly δ represents an atomic action which cannot proceed (and hence cannot terminate).
The converse of 3.4 holds as well, as is stated in the following theorem:

theorem 3.5 The transition model $\mathbb{T}$ is isomorphic to the initial algebra for BPA_δ.

proof So we have to prove $BPA_\delta \vdash s=t \Leftrightarrow s \leftrightarrows t$. By proposition 3.2 it is sufficient to prove this for basic terms s and t only (using transitivity of $\leftrightarrows$).
$\Rightarrow$ by theorem 3.4.
$\Leftarrow$ This is done by induction on dp(s) + dp(t), as follows.
If dp(s) + dp(t) = 2 it directly follows that both s,t are sums of atomic actions from A and hence $s \leftrightarrows t$ if and only if $BPA_\delta \vdash s=t$. Now assume $s \leftrightarrows t$ for BPA_δ-terms s and t and for all s', t' $dp(s') + dp(t') < dp(s) + dp(t)$ with $s' \leftrightarrows t'$ it is already proved that $BPA_\delta \vdash s'=t'$.
It is enough to prove that any summand a or $a \cdot s'$ of s is also a summand of t (and vice versa) since then it follows that both $BPA_\delta \vdash s = s + t$ and $BPA_\delta \vdash t = t + s$, which yields $BPA_\delta \vdash s=t$.
(1) Assume a is a summand of s ($a \in A$), then $s \equiv a + r$ or $s \equiv a$. Clearly $s \rightarrow^a \surd$ and hence $t \rightarrow^a \surd$

since $s \leftrightarroweq t$, and therefore by proposition 3.3 it follows that a is a summand of t.

(2) Assume $a \cdot s'$ is a summand of s, i.e. $s \equiv a \cdot s' + r$ or $s \equiv a \cdot s'$. Then $s \rightarrow^a s'$, and so $t \rightarrow^a t'$ for some t' with $s' \leftrightarroweq t'$. By proposition 3.3 it follows that t' is a basic term with $dp(t') < dp(t)$ and at' is a summand of t, i.e. $BPA_\delta \vdash t = at' + t$. Furthermore, $dp(s') < dp(s)$ so by induction we conclude that $BPA_\delta \vdash s' = t'$. Hence $BPA_\delta \vdash at' = as'$ and therefore $BPA_\delta \vdash t = as' + t$. □

Theorem 3.5 makes clear that BPA_δ is in fact a full axiomatisation of bisimulation equivalence on closed BPA_δ-terms.

In order to extend BPA_δ to the larger theory ASP, consider the additional rules in table 10:

$\mid$:	$\dfrac{x \rightarrow^a x' \,,\, y \rightarrow^b y'}{x \mid y \rightarrow^{a \mid b} x' \mid y'}$	$\dfrac{x \rightarrow^a x' \,,\, y \rightarrow^b \surd}{x \mid y \rightarrow^{a \mid b} x'}$	$(a \mid b \neq \delta)$
	$\dfrac{x \rightarrow^a \surd \,,\, y \rightarrow^b y'}{x \mid y \rightarrow^{a \mid b} y'}$	$\dfrac{x \rightarrow^a \surd \,,\, y \rightarrow^b \surd}{x \mid y \rightarrow^{a \mid b} \surd}$	$(a \mid b \neq \delta)$
ρ_f :	$\dfrac{x \rightarrow^a x'}{\rho_f(x) \rightarrow^{f(a)} \rho_f(x')}$	$\dfrac{x \rightarrow^a \surd}{\rho_f(x) \rightarrow^{f(a)} \surd}$	$(a \neq \delta, 1;\ f(a) \neq \delta)$
	$\dfrac{x \rightarrow^1 x'}{\rho_f(x) \rightarrow^1 \rho_f(x')}$	$\dfrac{x \rightarrow^1 \surd}{\rho_f(x) \rightarrow^1 \surd}$	

table 10. The additional transition predicates $\rightarrow^a$ and $\rightarrow^a \surd$ on ASP-terms ($a \in A - \{\delta\}$).

Again, we will assume these transition predicates to be the minimal interpretation which is closed under the rules of table 9 and table 10.

definition 3.5 The *transition model with communication* $\mathbb{TC}$ is the set of ASP-terms modulo bisimulation equivalence.

theorem 3.6 $\mathbb{TC}$ is an initial algebra for ASP.

proof It is straightforward to prove that $ASP \vdash s=t \Rightarrow s \leftrightarrows t$ (*).

So assume $s \leftrightarrows t$ for some ASP-terms s and t. By theorem 2.5 it follows that for some BPA_δ-terms s' and t' we have that $ASP \vdash s=s'$ and $ASP \vdash t=t'$. Now using (*) it follows that $s \leftrightarrows s'$ and $t \leftrightarrows t'$. Since $ASP \vdash s'=t'$ and both s' and t' are BPA_δ-terms we find by theorem 2.7 that $BPA_\delta \vdash s'=t'$. By theorem 3.5 it follows that $s' \leftrightarrows t'$ and therefore $s \leftrightarrows s' \leftrightarrows t' \leftrightarrows t$, hence $s \leftrightarrows t$. □

Later on we will return to the subject of transitions, and consider the transition predicates in models that have a larger domain.

3.2 THE GRAPH MODEL

In this section we consider another model for ASP which consists of equivalence classes of process graphs (see BAETEN, BERGSTRA & KLOP [1]).

definition 3.6 A *process graph* is a labeled, rooted, finitely branching, directed multigraph.

An edge goes from a node to another (or the same) node, and is labeled with an element from A. We consider only finitely branching process graphs, so every node has only finitely many outgoing edges. Although a process graph may have infinitely many nodes we must be able to reach any node in only finitely many steps. A graph which has finitely many nodes will be called *regular*.
An edge from node s to node s', with label a, will be denoted as $s \rightarrow^a s'$. The nodes in a process graph can be looked at as *states*. $s \rightarrow^a s'$ is called an *a-step* from s to s'.

definition 3.7 A *simulation* from a graph g to a graph h, notation R: $g \underline{\rightarrow} h$, is a relation R between nodes of g and nodes of h such that:

1. The roots of g and h are related by R;
2. If R(s,t) and from s we can do an a-step to a node s', i.e. we have $s \rightarrow^a s'$ with label $a \in A-\{\delta\}$ (so $a \neq \delta$) then in h we can do a-step from t to a node t' with R(s,t').
3. If R(s,t) and s is an end point in g then t is an end point in h.

A *bisimulation* between two graphs g and h, notation $g \underline{\leftrightarrow} h$, is a relation R such that both R: $g \underline{\rightarrow} h$ and R: $h \underline{\rightarrow} g$. Furthermore, we write $g \underline{\rightarrow} h$ if there exists an R such that R: $g \underline{\rightarrow} h$ and similarly we write $g \underline{\leftrightarrow} h$ if there is an R such that R: $g \underline{\leftrightarrow} h$.

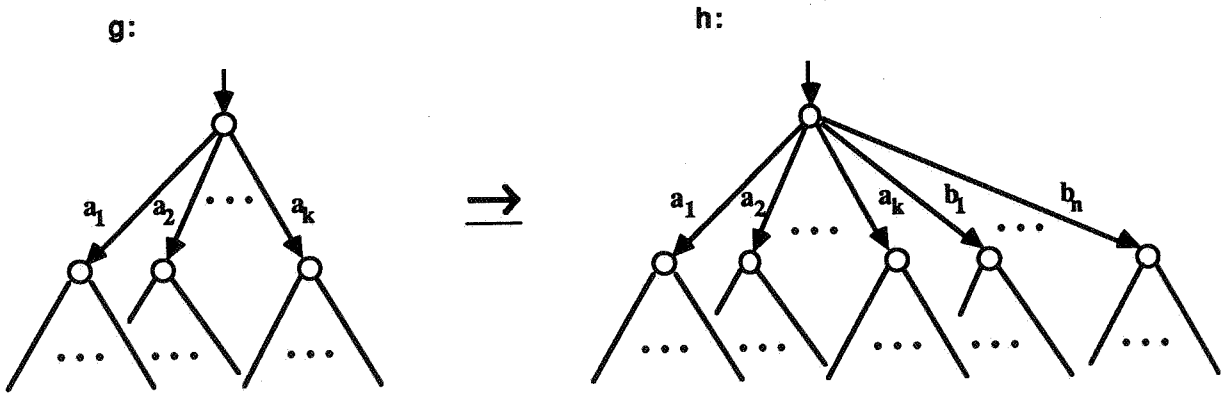
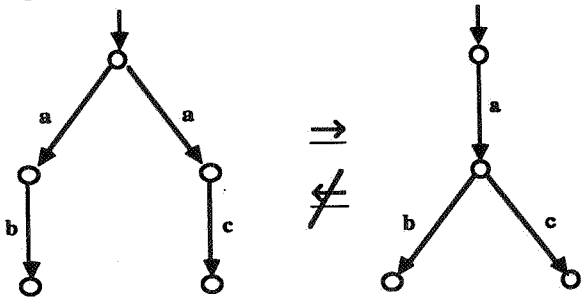


figure 1. R: $g \underline{\rightarrow} h$.

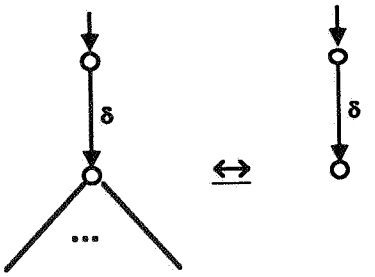
The notion of bisimulation was originally due to PARK [11]. For more information, see MILNER [7], BAETEN, BERGSTRA & KLOP [1].

examples

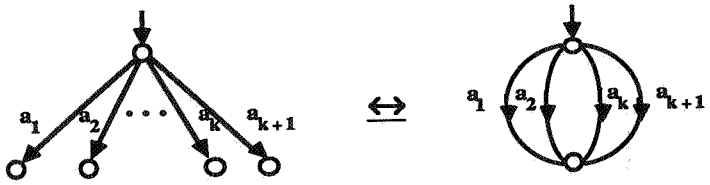
1.



2.



3.

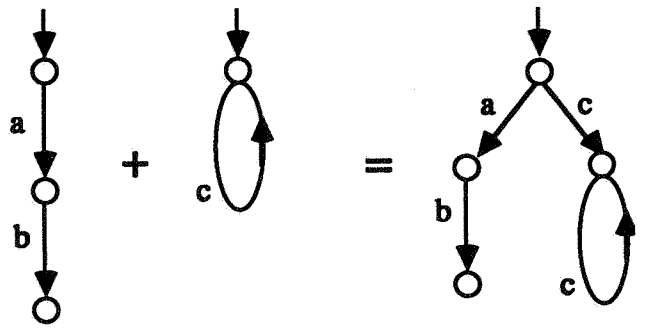


proposition 3.7 $\underline{\leftrightarrow}$ is an equivalence relation on the set of process graphs.

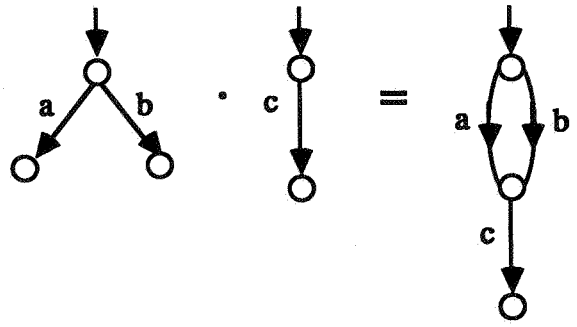
proof $\underline{\leftrightarrow}$ is reflexive (i.e. for all process graphs g we have $g\underline{\leftrightarrow}g$) since the identity relation on nodes of g (relating any node precisely with itself) is a bisimulation.
$\underline{\leftrightarrow}$ is commutative (i.e. for all g and h: $g\underline{\leftrightarrow}h \Rightarrow h\underline{\leftrightarrow}g$) directly from the definition.
$\underline{\leftrightarrow}$ is transitive (i.e. for all f, g and h: $f\underline{\leftrightarrow}g$ and $g\underline{\leftrightarrow}h \Rightarrow f\underline{\leftrightarrow}h$), which is proved as follows:
Suppose R: $f\underline{\leftrightarrow}g$ and S: $g\underline{\leftrightarrow}h$. Define the relation T between f and h, such that for any two nodes r in f and t in h: T(r,t) iff there exists a node s in g such that R(r,s) and S(s,t).
1. Now clearly the roots of f and h are related by T.
2. Next, assume T(r,t) and suppose from r we can do an a-step to a node r', so: $r \rightarrow^a r'$ ($a \neq \delta$). Let s be a node in g such that R(r,s) and S(s,t). Since R is a simulation from f to g, we can do an a-step from s to a node s', i.e. $s \rightarrow^a s'$, such that R(r',s'). Furthermore, since S is a simulation from g to h, we have edges $t \rightarrow^a t'$ in h such that S(s',t'). Directly we find that T(r',t') which satifies the second condition in definition 3.7.
3. Finally, assume for some end point r in f we have T(r,t). Let s be such that R(r,s) and S(s,t) then it directly follows that s is an end point in g hence so is t in h.
Thus we find that T: $f\underline{\rightarrow}h$. For reasons of symmetry we conclude T: $h\underline{\rightarrow}f$, hence T: $f\underline{\leftrightarrow}h$. □

Note that δ-edges are not mentioned in the definition of bisimulation. As a consequence we find that starting from a δ-edge, there is no restriction whatsoever on its subgraph (example 2).
Next we will introduce the ASP-operators +, $\cdot$ and $\mid$ on process graphs in order to turn the set of graphs (modulo $\underline{\leftrightarrow}$) into an algebra for ASP.

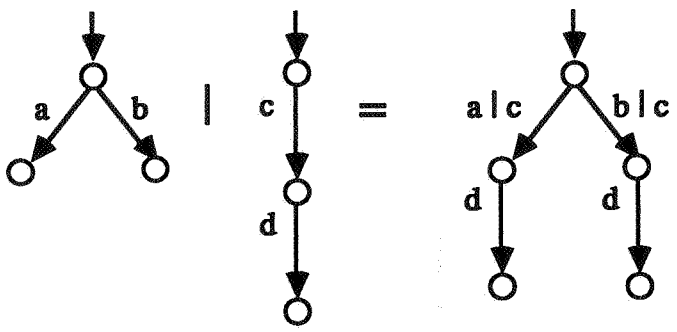
definition 3.8 The binary functions +, $\cdot$ and $\mid$ are defined on process graphs as follows. Assume g and h are two such graphs, then:
1. (g + h) is obtained as follows: start from a new root node r, and add a new edge $r \rightarrow^a s'$ for each edge $r_g \rightarrow^a s'$ in g which starts from the root node r_g of g ($a \in A$); similarly, add a new edge $r_h \rightarrow^a t'$ for each $t \rightarrow^a t'$ in h which starts from the root node r_h ($a \in A$). Finally, remove all nodes which have become inaccessible from r.



2. (g·h) is obtained by identifying all end points of g with the root of h. If g has no end points, this is just g. The root of g·h is the root of g.



3. (g | h) is defined inductively as follows. Now (g | h) is a subgraph of the cartesian product of g and h defined as follows: (1) the root of (g | h) is the pair roots of g and h.
(2) if (s,t) is a node in (g | h) then: (i) if $s \rightarrow^{a} s'$, $t \rightarrow^{b} t'$ are edges in g and h respectively, then $(s,t) \rightarrow^{(a \mid b)} (s',t')$ is an edge in (g | h); (ii) if s is an end point in g and $t \rightarrow^{b} t'$ is an edge in h then $(s,t) \rightarrow^{b} (s,t')$ is an edge in (g | h), and (iii) if $s \rightarrow^{a} s'$ is an edge in g and t is end point in h then $(s,t) \rightarrow^{a} (s',t)$ is an edge in (g | h).



definition 3.9 The unary functions ρ_f are defined on process graphs by simply replacing all labels a ($\neq\delta,1$) by f(a).

theorem 3.8 $\leftrightarrow$ is a congruence on process graphs with respect to +, ·, | and ρ_f.

proof By proposition 3.7 $\leftrightarrow$ is an equivalence relation. Now assume R: $u \leftrightarrow u'$ and S: $v \leftrightarrow v'$. Set $T = R \cup S$ and $T^+ = T \cup \{(r(u + v),r(u' + v')),(r(u' + v'),r(u + v))\}$ where r(g) stands for the *root* node of the graph g, and define R | S on the cartesian product of nodes from u and v such that: (R | S)((s,t),(s',t')) if and only if R(s,s') and S(t,t').

1. T^+: $(u + v) \leftrightarrow (u' + v')$, which follows directly from the definitions 3.7 and 3.8-1.

2. T: $(u \cdot v) \leftrightarrow (u' \cdot v')$: the roots of $(u \cdot v)$ and $(u' \cdot v')$ are related. Next, assume in $(u \cdot v)$ we have T(s,t) and $s \rightarrow^a s'$, then clearly s and s' both originate from either u or either v. Therefore in either u' or v' (so in $u' \cdot v'$) we have $t \rightarrow^a t'$ such that T(s',t').
Assume T(s,t) and s is an end point in $u \cdot v$ then it easily follows that t is an end point in $u' \cdot v'$.

3. (R | S): $(u \mid v) \leftrightarrow (u' \mid v')$. The roots of $(u \mid v)$ and $(u' \mid v')$ are related by (R | S). Next, assume ir $(u \mid v)$ we have $(s,t) \rightarrow^a (s',t')$ and (R | S)((s,t),(p,q)); by definition we have $s \rightarrow^b s'$ and $t \rightarrow^c t'$ in p and q respectively, such that $(b \mid c) = a$. Since R and S are simulations and since R(s,p) and S(t,q), there exist $p \rightarrow^b p'$ and $q \rightarrow^c q'$ in p' and q' respectively such that R(s',p') and S(t',q'). Hence we have $(p,q) \rightarrow^{(b \mid c)} (p',q')$ in $(u' \mid v')$ and clearly (R | S)((s',t'),(p',q')).
Finally, assume that R((s,t),(p,q)) for any end point (s,t) in $(u \mid v)$ then it follows easily that (p,q) is an end point in $(u' \mid v')$.

4. R: $\rho_f(u) \leftrightarrow \rho_f(u')$. Suppose in $\rho_f(u)$ there is an edge $s \rightarrow^a s'$ and suppose we have R(s,t). Then in u there exists and edge $s \rightarrow^b s'$ such that either $b \in \{\delta,1\}$ or f(b)=a. Now there are two cases: (i) if $b \in \{\delta,1\}$ then a=b, and since R is a bisimulation between u and u' there is an edge $t \rightarrow^b t'$ in u' with R(s',t'). Therefore in $\rho_f(u')$ there exists an edge $t \rightarrow^b t'$, i.e. an edge $t \rightarrow^a t'$ such that R(s',t'); (ii) if $b \neq \delta,1$ then a=f(b). In u' there exists an edge $t \rightarrow^b t'$ with R(s',t') and clearly we find that in $\rho_f(u')$ there exists an edge $t \rightarrow^a t'$ such that R(s',t'). Hence R: $\rho_f(u) \leftrightarrow \rho_f(u')$. □

definition 3.10 The *graph model* $\mathbb{G}$ is the algebra of all process graphs modulo $\leftrightarrow$.
In this algebra the constants $a \in A$ are interpreted as two-node graphs with one edge in between, labeled with a. The function symbols are defined as in the definitions 3.8 and 3.9.

theorem 3.9 $\mathbb{G}$ is a model for ASP.

proof The axioms A1-A3 follow almost immediately from the definition of + and bisimulation. In this proof we assume u, v and w to be arbitrary process graphs.

Construct u' from u by taking together all end nodes into one (new) end node. Consider the relation R, which is the identity on u except for the end nodes in u that are related with the new end node in u'. Clearly R is an bisimulation and so $u \leftrightarrows u'$. From this construction and the definition of $\cdot$ we immediately find A4: $(u + v)\cdot w \leftrightarrows u\cdot w + v\cdot w$.

Axiom A5 follows immediately from the definition of $\cdot$. Moreover, the identity relation on u is an bisimulation between u and $(u + \delta)$, so A6: $u \leftrightarrows u + \delta$. The relation which only relates the root nodes of δ and $\delta\cdot u$ is an bisimulation between δ and $\delta\cdot u$, so: A7: $\delta \leftrightarrows \delta\cdot u$.

The axioms C1-C4 and SC1-3 simply follow from definition 3.8. The axioms R1-R6 immediately follow from the definition of renaming on graphs (definition 3.9). □

4. RECURSION

In the previous section we have defined the graph model $\mathbb{G}$, which turned out to be a model for ASP in which we have graphs representing infinite processes. In the following we will investigate a way in which infinite processes can be described algebraically.

definition 4.1 A *recursive specification* over ASP is a set of equations $E = \{x = t_x : x \pounds V\}$, where V is a set of variables and t_x are ASP-terms only containing variables from V.

definition 4.2 The *Recursive Definition Principle* (**RDP**) is the rule that says that every recursive specification has a solution, i.e. is satisfied in any model.

We will write $\mathcal{A} \models$ **RDP** if the recursive definition principle **RDP** holds in the algebra $\mathcal{A}$.
Recursive specifications are used to specify processes. If a recursive specification E is satisfied in a model and $x \in V$, then $\langle x \mid E \rangle$ will denote the x-component of some solution of E. So if E has more than only one solution, $\langle x \mid E \rangle$ will denote some kind of quantified variable ranging over all E's witnesses (see VAN GLABBEEK [9]). If E has no solution, then $\langle x \mid E \rangle$ remains undefined. Finally, $\langle t \mid E \rangle$ denotes the term t in which each occurrence of a variable $x \in V$ is replaced by $\langle x \mid E \rangle$. The fact that $\langle x \mid E \rangle$ is a solution of E can simply be expressed by: $\langle x \mid E \rangle = \langle t_x \mid E \rangle$.

recursion:	$\dfrac{\langle t_x \mid E \rangle \xrightarrow{a} y}{\langle x \mid E \rangle \xrightarrow{a} y}$	$\dfrac{\langle t_x \mid E \rangle \xrightarrow{a} \surd}{\langle x \mid E \rangle \xrightarrow{a} \surd}$

table 11. Additional transitions for recursion.

definition 4.3 Let t be an ASP-term and x a variable from t. The occurrence of x in t is called *guarded* if x is preceded by an atomic action from A, i.e. if t has a subterm of the form $a \cdot s$ with $a \in A$, and this x occurs in s. If not, the occurrence of x is called *unguarded.*
A recursive specification is called guarded if each occurrence of a variable is guarded.

definition 4.4 The *Recursive Specification Principle* (**RSP**) says that every guarded recursive specification has at most one solution.

So, in a model with both **RDP** and **RSP** every guarded recursive specification has precisely one solution.
Let us extend the signature of ASP with unary operators π_n ($n \in \omega$) called *projection functions*, with the axioms of table 12 below ($a \in A$):

$\pi_n(a) = a$	PR1
$\pi_1(a \cdot x) = a$	PR2
$\pi_{n+1}(a \cdot x) = a \cdot \pi_n(x)$	PR3
$\pi_n(x + y) = \pi_n(x) + \pi_n(y)$	PR4

table 12. The projection functions π_n for $n \geq 1$.

The operator π_n cuts off the process after it has executed n atomic steps. It is easy to extend the models $\mathbb{T}$ and $\mathbb{G}$ with these operators. The extension of the theory ASP with the axioms of table 12 will be denoted by ASP + PR.

definition 4.5 The *Approximation Induction Principle* (**AIP**) is the rule that reads:
$(\forall n \geq 1: \pi_n(x) = \pi_n(y)) \Rightarrow x = y$.

proposition 4.1 $\mathbb{TC} \models$ **RSP, AIP.**

proposition 4.2 $\mathbb{G} \models$ **RSP, AIP.**

The proofs of propositions 4.1 and 4.2 are left to the reader.
In order to prove **AIP** in the graph model it turns out to be necessary that process graphs are finitely branching. If not, consider the infinitely branching graphs informally denoted by:

$$g = (\Sigma_{n \in \omega} a^n) \text{ and } h = (\Sigma_{n \in \omega} a^n) + a^\omega,$$

where a^ω denotes the infinite repetition of a-steps. Clearly, for all $n \in \omega$ we have $\pi_n(g) = \pi_n(h)$ but we do not have $g \leftrightarrow h$.
Clearly $\mathbb{T} \not\models$ **RDP** since no closed term has infinitely many transitions, whereas a process satisfying $x = a \cdot x$ can do infinitely many a-steps. We also find that $\mathbb{G} \not\models$ **RDP**, since the recursive

specification $\{x = a + xa\}$ has no solution in the model of finitely branching process graphs.

5. EXAMPLE: COMPUTER INTEGRATED MANUFACTURING

In this section we present an example of an application of ASP which is taken from MAUW [6]. In *Computer Integrated Manufacturing* (CIM), computers are integrated in the overall production process of some industrial product. From a high level of view, a plant can be seen as constructed from several concurrently operating *workcells*. Every workcell represents a well-defined part of the manufacturing process and a master control is needed to make the components cooperate correctly (see MAUW [6]).

In the following we will present a strong simplification of the CIM-architecture in [6]. It is not our aim to study the theoretical aspects of CIM-architectures in general, but merely to give an illustration of the way ASP applies to practical problems.

Consider the configuration as pictured in figure 2 below. This workcell has three components:

(WA) The workstation WA, which receives a 'semiproduct' product p from port 4 which is passed through to port 5.

(WB) The workstation WB accepts a product p from port 5, and produces a new product prod(p) which is sent away via port 6.

(WC) The workcell controller receives a certain number n at port 1, which is sent to WA and WB. After the number is accepted WA and WB will both repeat n times (independently) after which a message r ('ready') is sent to the workcell controller.

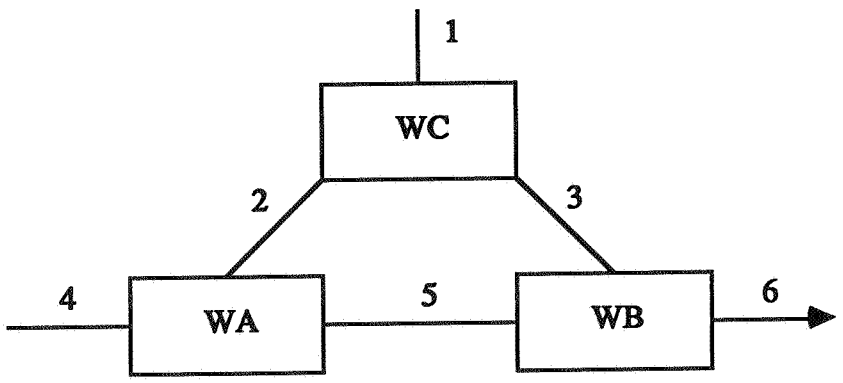


Figure 2. A workcell configuration

Imagine a factory in which unfinished semiproducts p have to be turned into commercial products prod(p). Now, the complete configuration should work as follows: from port 1, WC receives a message to generate n products of the form prod(p). So, WC will send the instruction to WA to 'pick up' n products from port 4 and pass it through to WB. Moreover, WC will send the instruction to WB to pick up n products from port 5 and produce n products of the form prod(p).

The configuration of figure 2 consists of three components interconnected by 6 ports. So we set:

$$\mathcal{P} = \{1, 2, ..., 6\}.$$

Along ports 1, 2 and 3 a positive integer n can be sent. We will assume that $n \leq N$ for some fixed $N \in \omega$. A ready message 'r' can be sent from WA and WB to WC and products of the form p or prod(p) can be sent through ports 4, 5 and 6. So we have a *data set* D defined by:

$$D = \{n: 1 \leq n \leq N\} \cup \{r\} \cup \{p, prod(p)\}.$$

In order to fix the alphabet of ASP, we define the set A of atomic actions as follows:

$$A = \{r(x), s(x), c(x): x \in D\} \cup \{1, \delta\}.$$

On atomic actions from A, the *communication function* $|$ is defined as follows:

$$r(x) \mid s(x) = s(x) \mid r(x) = c(x)$$

whereas all other communications on A not containing 1's are equal to δ.
Next we are ready to present a proper specification of the workcell. From now we will work within the algebra $ASP(A^{\mathcal{P}})$ with the natural extensions of the communication function $|$ (definition 2.3) and the renaming operators (definition 2.4). The constants of $ASP(A^{\mathcal{P}})$ consist of all 6-dimensional vectors $(a_1\ a_2 \ldots a_6)$ where $a_i \in A$ are atomic actions from ASP(A).
At this point we will introduce some shorthand notations that are very useful to avoid the elaborate vector notations.

definition 5.1 Assume we have a set of ports $\mathcal{P}$ and let $a \in A$ be some atomic action from ASP then for all $Q \in \mathcal{P}$ we define $a_Q \in A^{\mathcal{P}}$ by:

$$a_Q(P) = \begin{cases} a & \text{if } P=Q \\ 1 & \text{if } P \neq Q. \end{cases}$$

So in our setting we have that $r(p)_2 = (1\ r(p)\ 1\ 1\ 1\ 1)$ and $s(r)_5 = (1\ 1\ 1\ 1\ s(r)\ 1)$. Because of a strong tradition in process algebra we will write $r_2(p)$ instead of $r(p)_2$ and $s_5(p)$ instead of $s(p)_5$. Now let us give a precise definition of the three components of the workcell.

$WA = \mathbf{1} \cdot \Sigma_{1 \leq n \leq N} r_2(n) \cdot WA(n)$
$WA(0) = s_2(r)$
$WA(n+1) = r_4(p) \cdot s_5(p) \cdot WA(n)$

$WB = \mathbf{1} \cdot \Sigma_{1 \leq n \leq N} r_3(n) \cdot \mathbf{1} \cdot WB(n)$
$WB(0) = s_3(r)$
$WB(n+1) = r_5(p) \cdot s_6(prod(p)) \cdot WB(n)$

$WC = \Sigma_{1 \leq n \leq N} r_1(n) \cdot WC(n)$
$WC(n) = (s_2(n) \mid s_3(n)) \cdot \mathbf{2n} \cdot r_2(r) \cdot r_3(r) \cdot s_1(r)$

---

table 13. A formal specification of the workcell.

In the equation for WC(n) we have used the abbreviation $\mathbf{n}$ for $\mathbf{1}^n$. Formally we could have defined $\mathbf{1}^n$ by use of the inductive definition $\{\mathbf{1}^1 = \mathbf{1},\ \mathbf{1}^{n+1} = \mathbf{1} \cdot \mathbf{1}^n\}$. In the same way we define t^n for ASP-terms t and $n \geq 1$.
Now define:

$$I = \{r(p), c(x): x \in D\}$$

$$WORKCELL = \mathbf{1}_I(WA \mid WB \mid WC).$$

We abstract from actions like r(p) and thus from an unlimited supply of goods available at port 4. It turns out that we can prove the following theorem:

theorem 5.1 $\mathrm{WORKCELL} = \Sigma_{1\leq n\leq N}\, r_1(n)\cdot \mathbf{3}\cdot(\, s_6(\mathrm{prod}(p))\cdot \mathbf{1}\,)^n \cdot s_1(r)$.

proof The proof of theorem 5.1 can be given by use of induction on n:

Induction hypothesis For all $1\leq n\leq k$ we have:

(i) $1_I(\,\mathrm{WA}(n) \mid \mathbf{1}\cdot\mathrm{WB}(n) \mid \mathbf{2n}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r)\,) = \mathbf{2}\cdot (s_6(\mathrm{prod}(p)\cdot\mathbf{1})^n \cdot s_1(r)$

(ii) $1_I(\,\mathrm{WA}(n) \mid s_6(\mathrm{prod}(p))\cdot\mathrm{WB}(n) \mid \mathbf{2n}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r))\,) = (s_6(\mathrm{prod}(p))\cdot\mathbf{1})^{n+1} \cdot s_1(r)$.

k=1:

(i) $1_I(\,\mathrm{WA}(1) \mid \mathbf{1}\cdot\mathrm{WB}(1) \mid \mathbf{2}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r))\,) =$
$= 1_I(\, r_4(p)\cdot s_5(p)\cdot s_2(r) \mid \mathbf{1}\cdot r_5(p)\cdot s_6(\mathrm{prod}(p))\cdot s_3(r) \mid \mathbf{2}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r))\,) =$
$= 1_I(\,(r_4(p) \mid 1 \mid 1)\cdot(s_5(p) \mid r_5(p) \mid 1)\cdot(s_2(r) \mid s_6(\mathrm{prod}(p)) \mid r_2(r))\cdot(s_3(r) \mid r_3(r))\cdot s_1(r)\,) =$
$= 1_I(\, r_4(p)\cdot c_5(p)\cdot(c_2(r) \mid s_6(\mathrm{prod}(p))\cdot c_3(r)\cdot s_1(r)\,) =$
$= \mathbf{2}\cdot s_6(\mathrm{prod}(p))\cdot\mathbf{1}\cdot s_1(r)$.

(ii) $1_I(\,\mathrm{WA}(1) \mid s_6(\mathrm{prod}(p))\cdot\mathrm{WB}(1) \mid \mathbf{2}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r))\,) =$
$= 1_I(r_4(p)\cdot s_5(p)\cdot s_2(r) \mid s_6(\mathrm{prod}(p))\cdot r_5(p)\cdot s_6(\mathrm{prod}(p))\cdot s_3(r) \mid \mathbf{2}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r)\,) =$
$= 1_I(\,(r_4(p) \mid s_6(\mathrm{prod}(p)) \mid 1)\cdot(s_5(p) \mid r_5(p) \mid 1)\cdot(s_2(r) \mid s_6(\mathrm{prod}(p)) \mid r_2(r))\cdot(s_3(r) \mid r_3(r))\cdot s_1(r)\,) =$
$= 1_I((r_4(p) \mid s_6(\mathrm{prod}(p)))\cdot c_5(p)\cdot(c_2(r) \mid s_6(\mathrm{prod}(p)))\cdot c_3(r)\cdot s_1(r)\,) =$
$= (1 \mid s_6(\mathrm{prod}(p)))\cdot\mathbf{1}\cdot(1 \mid s_6(\mathrm{prod}(p)))\cdot\mathbf{1}\cdot s_1(r) =$
$= (s_6(\mathrm{prod}(p))\cdot\mathbf{1})^2 \cdot s_1(r)$.

k+1:

(i) $1_I(\,\mathrm{WA}(k+1) \mid \mathbf{1}\cdot\mathrm{WB}(k+1) \mid \mathbf{2(k+1)}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r))\,) =$
$= 1_I(\, r_4(r)\cdot s_5(p)\cdot\mathrm{WA}(k) \mid \mathbf{1}\cdot r_5(p)\cdot s_6(\mathrm{prod}(p))\cdot\mathrm{WB}(k) \mid \mathbf{1}\cdot\mathbf{1}\cdot\mathbf{2k}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r))\,) =$
$= 1_I(\,(r_4(p) \mid 1 \mid 1)\cdot(c_5(p) \mid 1)\cdot(\mathrm{WA}(k) \mid s_6(\mathrm{prod}(p))\cdot\mathrm{WB}(k) \mid \mathbf{2k}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r))\,) =$
$= \mathbf{2}\cdot 1_I(\,\mathrm{WA}(k) \mid s_6(\mathrm{prod}(p))\cdot\mathrm{WB}(k) \mid \mathbf{2k}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r)\,) =$
$= \mathbf{2}\cdot (s_6(\mathrm{prod}(p))\cdot\mathbf{1})^{k+1} \cdot s_1(r)$ (use (ii) with n=k).

(ii) $\tau_I(WA(k+1) \mid s_6(prod(p))\cdot WB(k+1) \mid \mathbf{2(k+1)}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r))) =$
$= \tau_I(r_4(p)\cdot s_5(p)\cdot WA(k) \mid s_6(prod(p))\cdot r_5(p)\cdot s_6(prod(p))\cdot WB(k) \mid \mathbf{1\cdot 1\cdot 2k}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r))) =$
$= \tau_I((r_4(p) \mid s_6(prod(p)) \mid 1)\cdot(c_5(p) \mid 1)\cdot(WA(k) \mid s_6(prod(p))\cdot WB(k) \mid \mathbf{2k}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r)))$
$= (1 \mid s_6(prod(p)) \mid 1)\cdot 1\cdot(s_6(prod(p))\cdot 1)^{k+1}\cdot s_1(r)$ (again, use (ii) with n=k)
$= (s_6(prod(p))\cdot 1)^{k+2}\cdot s_1(r).$

So we have proved the induction hypothesis to be true for all k. Then it easily follows that:

$WORKCELL = \tau_I(WA \mid WB \mid WC) =$
$= \tau_I(\Sigma_{1\le n\le N} (1 \mid 1 \mid r_1(n))\cdot(r_2(n)\cdot WA(n) \mid r_3(n)\cdot 1\cdot WB(n) \mid WC(n))) =$
$= \Sigma_{1\le n\le N} r_1(n)\cdot \tau_I((r_2(n) \mid r_3(n) \mid s_2(n) \mid s_3(n))\cdot(WA(n) \mid 1\cdot WB(n) \mid \mathbf{2n}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r)))) =$
$= \Sigma_{1\le n\le N} r_1(n)\cdot \tau_I((c_2(n) \mid c_3(n))\cdot(WA(n) \mid 1\cdot WB(n) \mid \mathbf{2n}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r)))) =$
$= \Sigma_{1\le n\le N} r_1(n)\cdot 1\cdot \tau_I(WA(n) \mid 1\cdot WB(n) \mid \mathbf{2n}\cdot r_2(r)\cdot r_3(r)\cdot s_1(r))) =$
$= \Sigma_{1\le n\le N} r_1(n)\cdot \mathbf{3}\cdot(s_6(prod(p))\cdot 1)^n\cdot s_1(r)$ (using the induction hypothesis (i)) □

By theorem 5.1 we have formally proved that after having received a certain value n, WORKCELL will produce n products of the form prod(p) and then return a message that it is ready. Of course we could have considered much more complicated examples than the one presented here.

In the above example we have chosen to use the natural extensions of both operators $\mid$ and ρ_f. In quite a few applications however these extensions do not give us precisely what we want and we are forced to introduce different renaming functions. For instance, the natural extension of r_I in our example, will rename the atomic action r(p) into 1, no matter at which port it occurs. But what to do then, if we wish to abstract from the occurences of r(p) at one particular port only, and encapsulate all r(p)'s occurring at other ports?

Define the following atomic renamings:

definition 5.2 For all $v \in A^{\mathcal{P}}$ f(v) is defined as follows:
for all $i \in \mathcal{P}$, $x \in D$:
if {(i=4 **and** v(i)=r(p)) **or** v(i) = c(x)} **then** f(v)(i) = 1 **else** f(v)(i) = v(i).

definition 5.3 For all $v \in A^{\mathcal{P}}$ g(v) is defined as follows:

if for some $i \in \mathcal{P}$, $x \in D$: $\{v(i) = s(x)$ **and** $i \neq 6$ **and** $i \neq 1\}$ **or** $\{v(i) = r(x)$ **and** $i \neq 1\}$

then $g(v) = \delta$ **else** $g(v) = v$.

Using both definitions, and using the axioms of renamings (see table 4) we can derive the following theorem:

theorem 5.2 $\rho_f \circ \rho_g(WA \mid WB \mid WC) = \Sigma_{1 \leq n \leq N} r_1(n) \cdot 3 \cdot (\, s_6(\text{prod}(p)) \cdot 1 \,)^n \cdot s_1(r)$

The proof of theorem 5.2 follows easily from the proof of theorem 5.1 and is left to the reader.

REFERENCES

[1] Baeten, J.C.M., Bergstra, J.A. and Klop, J.W., *On the consistency of Koomen's fair abstraction rule*, TCS 51 (1/2), pp. 129-176, 1987.

[2] Baeten, J.C.M. Bergstra, J.A., *Global renaming operators in concrete process algebra*, report CS-R8521, Centre of Math and Comp. Sci., Amsterdam 1985.

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

[4] Bergstra, J.A. and Klop, J.W., *The algebra of recursively defined processes and the algebra of regular processes*, proc. 11th ICALP, ed. J. Paredaens, pp. 82-95, Antwerpen, Springer LNCS 172, 1984.

[5] Bergstra, J.A. and Klop, J.W., *The algebra of communicating processes with abstraction*, TCS 37(1), blz. 77-121, 1985.

[6] Mauw, S, *Process algebra as a tool for the specification and verification of CIM-architectures*, report P8708, Programming Research Group, University of Amsterdam, 1987.

[7] Milner, R., *A calculus of communicating systems*, Springer LNCS 92, 1980.

[8] Milner, R., *Calculi for synchrony and asynchrony*, TCS 25, pp. 267-310, 1983.

[9] Glabbeek, R.J. van, *Bounded nondeterminism and the approximation induction principle in process algebra*, proc. STACS, 1987.

[10] Hennessy, M., *Synchronous and Asynchronous Experiments on Processes*, Inf.&Control 59, pp. 36-83 (1983).

[11] Park, D.M., *Concurrency and automata on infinite sequences*, Proc. 5th GI Conf., Springer LNCS 104, 1981.