

## TOP-DOWN DESIGN AND THE ALGEBRA OF COMMUNICATING PROCESSES

J.A. BERGSTRA

*Centre for Mathematics and Computer Science, 1009 AB Amsterdam, Netherlands*

J.V. TUCKER

*Department of Computer Studies, University of Leeds, Leeds LS2 9JT, United Kingdom*

Communicated by K. Apt

Received January 1984

Revised July 1984

**Abstract.** We develop an algebraic theory for the top-down design of communicating systems in which levels of abstraction are represented by algebras, and their stepwise refinements are represented by homomorphisms. Particular attention is paid to the equational specification of these levels of abstraction. A number of examples are included for illustration, most notably a top-down design for a communication protocol.

### Introduction

We present an algebraic theory for the top-down design of concurrent systems. The starting point is the *modular structure* of concurrent systems at a given *level of abstraction*: we consider systems made from certain basic systems by means of composition tools, including sequencing, alternative choice and parallel composition. This view of modularity leads to an algebraic structure for a level of abstraction. The *hierarchical structure* existing between two levels of abstraction, that characterises top-down design, is analysed using algebraic specifications and homomorphisms.

Let us review the contents of the paper in more detail.

In Section 1 we examine informally the modular structure of concurrent systems and experiment with top-down design using graph-theoretic and algebraic notations for systems. Graph substitutions and algebraic transformations are seen to model top-down refinements, and we set ourselves the task of creating a proper foundation for the algebraic formulation.

In Sections 2, 3 and 4 we develop the necessary algebraic tools to provide the syntax and semantics for the algebraic specification of levels of abstraction for concurrent systems. These tools are based on a set of algebraic laws for the behaviour of concurrent processes called ACP—axioms for communicating processes—first discussed in Bergstra and Klop [8].

In Section 5 we present an algebraic model for the top-down design of concurrent systems in which equationally specified algorithms are stepwise refined. The informal graph substitutions and transformations of equations are replaced by constructions of factor algebras and homomorphisms.

In Section 6 a communication protocol is examined in detail to exemplify our theory.

In Section 7 we discuss the origins of the theory in our work on concurrency, and on VLSI system design. We also comment on the relationship between this ACP-based theory and other algebraic approaches to concurrency, most notably those of Milner [16] and Hoare, Brookes and Roscoe [15].

The prerequisites of this paper are a knowledge of concurrency (Ben-Ari [6], Hoare [14], Milner [16]) and a knowledge of algebra, and equational specification methods in computer science (ADJ [1, 2], Goguen and Meseguer [12]).

Finally, let us note that this paper is intimately related to the paper of Bergstra, Klop and Tucker [10], in which parts of the algebraic theory presented here are generalised to account for the hierarchical structure of computer systems in general.

## 1. Prelude on algebraic and graphical notations for systems

Very informally, consider a system built from certain system primitives, and given systems, by means of certain *composition principles*. Rather abstractly, the system primitives we will call *atomic actions* and the resulting system we will call a *process*. The composition principles of interest are:

- sequential composition;
- alternative composition;
- iteration/recursion;
- parallel composition;
- encapsulation.

We will describe these operations together with a graphical notation to picture the structure of the resulting process.

**1.0. Equality.** Let  $X$  and  $Y$  be processes. The *equality* of  $X$  and  $Y$ , written  $X = Y$ , means that  $X$  and  $Y$  are considered to be computationally equivalent, although the semantics of processes is *not* specified at this stage of our discussion.

**1.1. Sequential composition.** Let  $X$  and  $Y$  be processes. The *sequential composition* of  $X$  and  $Y$  makes a process  $X \cdot Y$  that schedules  $Y$  after the completion of  $X$ . If the processes  $X$  and  $Y$  are represented by nodes then  $X \cdot Y$  is represented by the graph seen in Fig. 1.

**1.2. Alternative composition.** Let  $X$  and  $Y$  be processes. The *alternative composition*

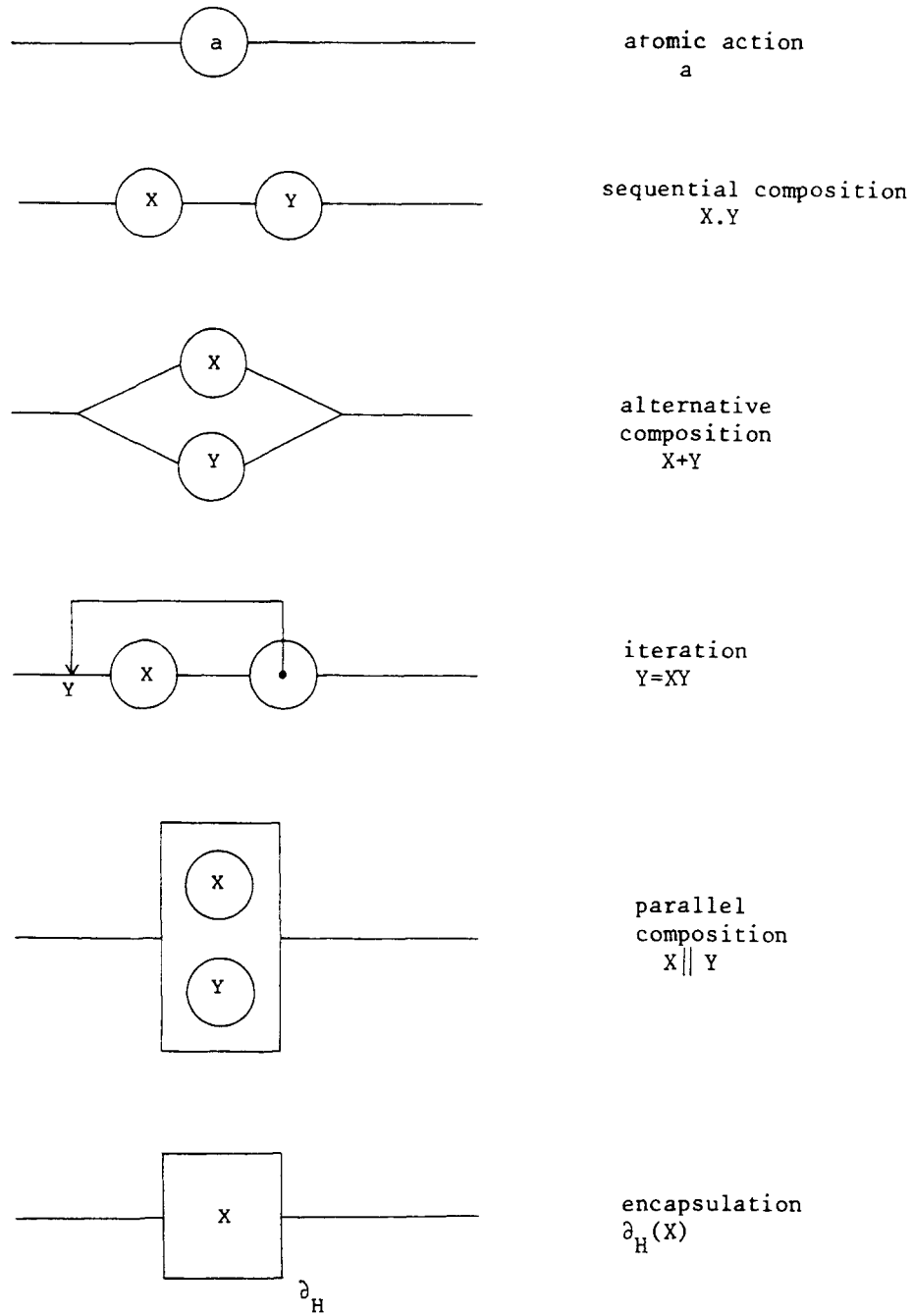


Fig. 1. Diagrams for processes.

of  $X$  and  $Y$  makes a process  $X + Y$  that schedules either  $X$  or  $Y$ , but not both; its graph is seen in Fig. 1.

**1.3. Iteration/recursion.** Let  $X$  and  $Y$  be processes. The *iteration* of  $X$  under  $Y$  specifies a process that schedules  $X$  an unspecified number of times under the control of  $Y$ . In symbols, this is expressed by an equation,

$$Y = X \cdot Y,$$

and by a graph seen in Fig. 1. Although we are especially interested in recursion as a composition principle we will not attempt a simple informal description here.

**1.4. Parallel composition.** Let  $X$  and  $Y$  be processes. The *parallel composition* of  $X$  and  $Y$  makes a process  $X \parallel Y$  that concurrently schedules the processes  $X$  and  $Y$ ; its diagram is seen in Fig. 1.

**1.5. Encapsulation.** Let  $X$  be a process and let  $H$  be a set of atomic actions. The *encapsulation* of  $X$  with respect to communications by the actions of  $H$  makes a process  $\partial_H(X)$  in which potential communications by actions in  $H$  have been prevented.

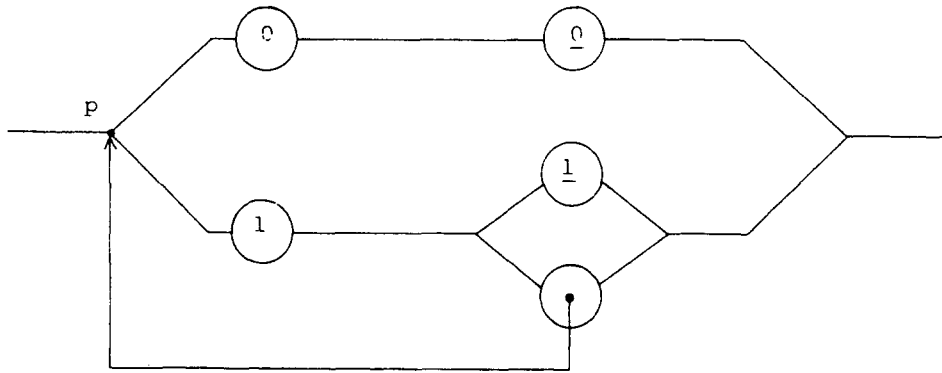
Informally, we can imagine processes built from some set of atomic actions and given processes using the above composition tools; and for such processes we have a symbolic and graphical notation. We will experiment with these notations by means of examples.

**1.6. Examples.** In most of the following six examples we will use the atomic actions

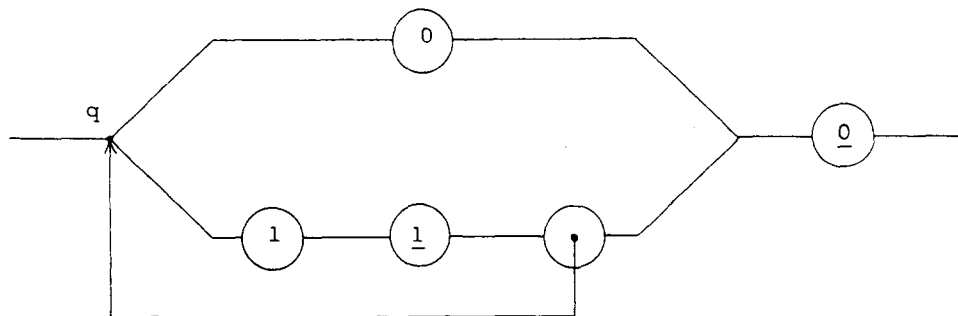
|          |       |   |
|----------|-------|---|
| 0        | read  | 0 |
| 1        | read  | 1 |
| <u>0</u> | write | 0 |
| <u>1</u> | write | 1 |

The first two examples merely illustrate the relationship between formulae and pictures.

**1.6.1. Process.**  $p = 0 \cdot \underline{0} + 1(1 + p)$



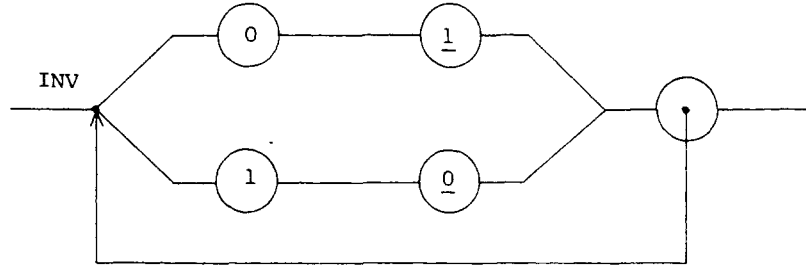
**1.6.2. Process.**  $p = q \cdot \underline{0}, q = 0 + 1 \cdot \underline{1}q$



**1.6.3. Inverter.** An inverter can be modelled as a process  $INV$  specified by the equation

$$INV = (0 \cdot \underline{1} + 1 \cdot \underline{0}) INV$$

and by the graph



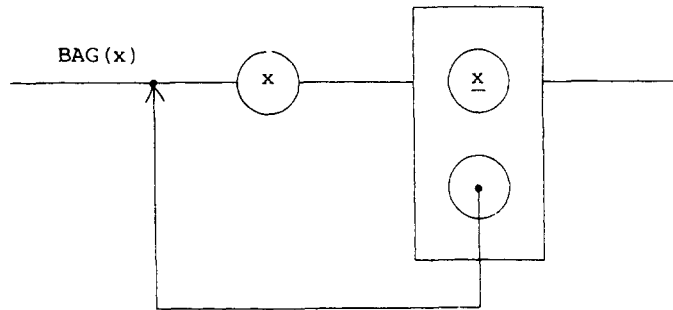
**1.6.4. One element bag.** A bag over the set  $\{x\}$  is a multiset that can input and output  $x$ . It can be modelled as a process  $BAG(x)$  over the atomic actions

$x$  input  $x$   
 $\underline{x}$  output  $x$

and is algebraically defined by

$$BAG(x) = x \cdot (\underline{x} \parallel BAG(x))$$

and is pictorially defined by

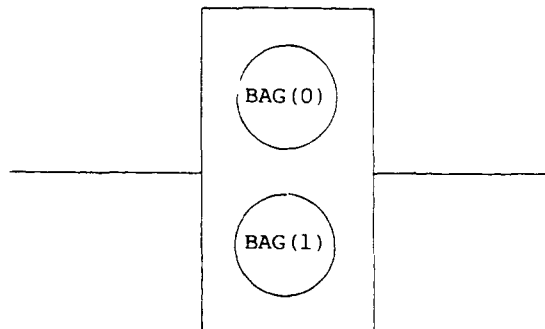


This bag has been examined in Bergstra and Klop [9].

**1.6.5. Two element bag.** A bag over the set  $\{0, 1\}$  can be modelled as a process  $BAG(0, 1)$  made from the one element bag  $BAG(x)$  by setting

$$BAG(0, 1) = BAG(0) \parallel BAG(1).$$

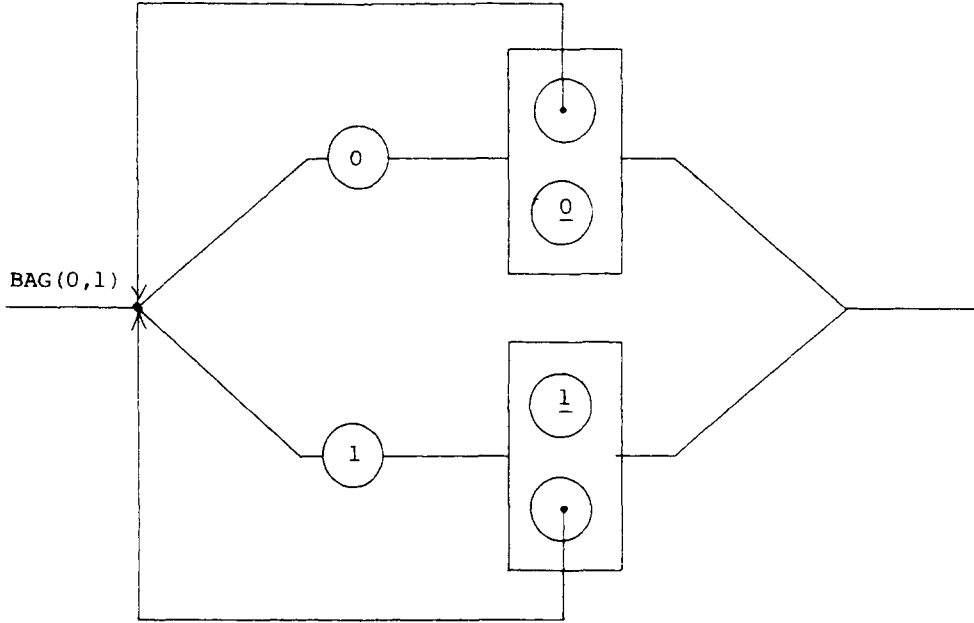
The picture is simply



**1.6.6. Two element bag II.** A different specification for the two element bag over  $\{0, 1\}$  is the fixed-point equation

$$BAG(0, 1) = 0(0 \parallel BAG(0, 1)) + 1(\underline{1} \parallel BAG(0, 1)).$$

This is pictured

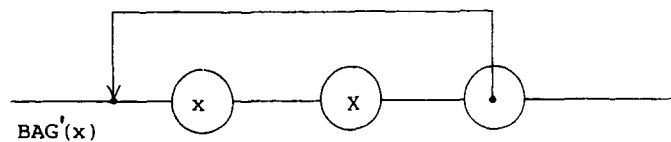


This bag has been examined in detail in Bergstra and Klop [9].

We now come to the principal problem of interest in this paper: the transformation of systems in top-down design.

**1.7. Top-down design and graph substitution.** Top-down design refers to the refinement of system specifications, starting from general descriptions and ending at specific definitions. Consider the refinement of a specification of a system  $S$  in which a component  $X$  of  $S$  is implemented by a system  $P$  and yields a new specification of a system  $S'$ . The graphical notation for systems allows us to visualise this refinement in terms of the substitution of the graph for  $P$  at the node for  $X$  in the graph of  $S$ . The result of the substitution is the graph for  $S'$ . We will illustrate this idea and, in particular, examine the meaning of the refinement for the algebraic formalism.

We will design another model of the one element bag discussed in 1.6.4. Let the behaviour of the new bag  $BAG'(x)$  be specified by

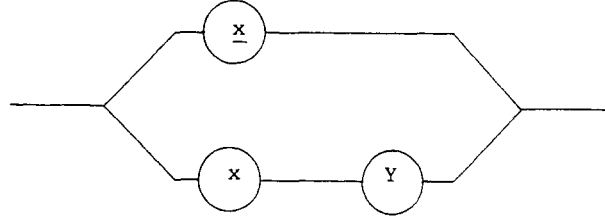


and by the formula

$$BAG'(x) = xXBAG'(x).$$

The parameter  $X$  denotes a process that is the bag that contains a single  $x$  and terminates when empty.

Now we take  $X$  to be specified by

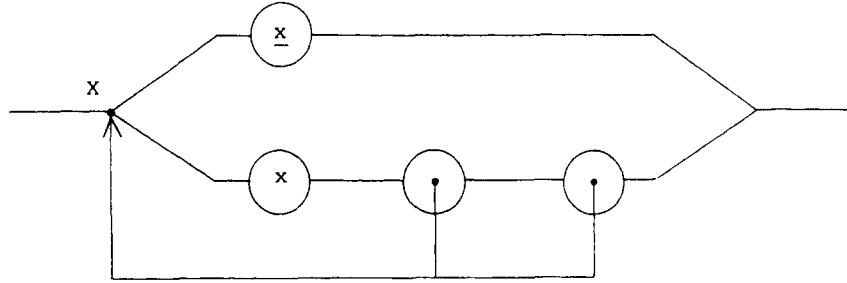


and, in symbols,

$$X = \underline{x} + xY$$

where  $Y$  is the bag that contains two elements and terminates when empty.

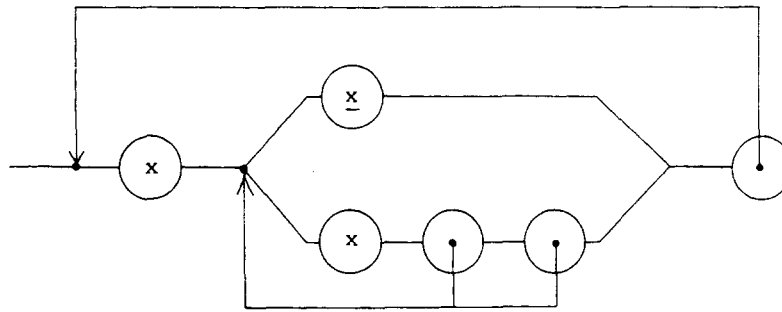
We observe that  $Y = X \cdot X$ . Hence substituting in previous specifications we obtain the graph



and the formula

$$X = \underline{x} + xXX.$$

Thus, the process  $BAG'(x)$  is specified by substituting this graph for the node  $X$  in its top-most description, with the following result:



Algebraically, we are left with the following pair of algebraic equations to define the bag:

$$BAG'(x) = xXBAG'(x), \quad X = \underline{x} + xXX.$$

*1.8. Observations.* Actually, we have done little more in this prelude than describe an algebraic signature and a graph-theoretic notation for system architectures, together with some transformations of equations and graphs, and we have deliber-

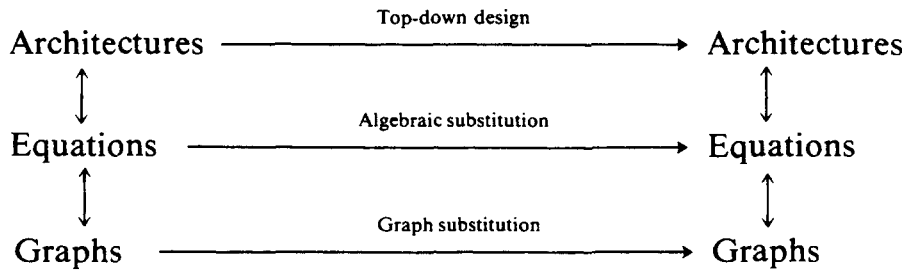
ately avoided the subject of the semantics of these notations. The algebraic and graphical notations for processes are attractive, but they require a proper mathematical theory if they are to be of use.

The purpose of this paper is to outline such a theory for the algebraic approach (leaving the graph-theoretical theory to another occasion.) Rather than constructing and contemplating various semantical models for these notations we will *begin* by giving axioms, legislating for their properties.

The conclusions of this prelude are:

- A level of abstraction for systems is characterised by atomic actions, given systems and composition tools.
- System architectures can be specified by formulae or by systems of equations.
- Top-down design can be represented by the process of transforming formulae and sets of equations by means of substitutions.

Our project is motivated by the ‘commutativity’ of the top-half of the following diagram:



## 2. Algebra of communicating processes

We will explain certain algebraic properties of the process building operations  $\cdot$ ,  $+$ ,  $\parallel$  and  $\partial_H$ , when used with a finite set of atomic actions with a given pattern of communication. Our basic tool is a set ACP of algebraic *axioms for communicating processes*, introduced in Bergstra and Klop [7].

### 2.1. ACP-algebras

Let  $A$  be a finite set, called the set of *atomic actions*. An ACP-*algebra* or *process algebra* over  $A$  consists of a set  $P$  equipped with the following operators:

|                         |                              |
|-------------------------|------------------------------|
| sequential composition  | $x \cdot y$                  |
| alternative composition | $x + y$                      |
| parallel composition    | $x \parallel y$              |
| left merge              | $x \parallel\!\!\! \sqcup y$ |
| communication merge     | $ $                          |
| deadlock/failure        | $\delta$                     |
| encapsulation           | $\partial_H(x)$              |

All operators are binary, except the constant  $\delta$ , which is a distinguished atomic action, and the unary  $H$ -projection ( $H \subseteq A$ )  $\partial_H$ . The set  $P$  contains  $A$  as a subset on which communication ‘ $|$ ’ restricts as a map  $| : A \times A \rightarrow A$ .



These operations satisfy the equational axioms in Fig. 2, where  $a, b, c$  vary over  $A$  and  $x, y, z$  over  $P$ . Often we will write instead of  $x \cdot y$  just  $xy$ .

|   |     |
|---|-----|
| $x + y = y + x$   | A1  |
| $x + (y + z) = (x + y) + z$                                 | A2  |
| $x + x = x$   | A3  |
| $(x + y) \cdot z = x \cdot z + y \cdot z$                   | A4  |
| $(x \cdot y) \cdot z = x \cdot (y \cdot z)$                 | A5  |
| $x + \delta = x$  | A6  |
| $\delta \cdot x = \delta$                                   | A7  |
| $a   b = b   a$   | C1  |
| $(a   b)   c = a   (b   c)$                                 | C2  |
| $\delta   a = \delta$                                       | C3  |
| $x \parallel y = x \parallel y + y \parallel x + x   y$     | CM1 |
| $a \parallel x = a \cdot x$                                 | CM2 |
| $(ax) \parallel y = a(x \parallel y)$                       | CM3 |
| $(x + y) \parallel z = x \parallel z + y \parallel z$       | CM4 |
| $(ax)   b = (a   b) \cdot x$                                | CM5 |
| $a   (bx) = (a   b) \cdot x$                                | CM6 |
| $(ax)   (by) = (a   b) \cdot (x \parallel y)$               | CM7 |
| $(x + y)   z = x   z + y   z$                               | CM8 |
| $x   (y + z) = x   y + x   z$                               | CM9 |
| $\partial_H(a) = a$ if $a \notin H$                         | D1  |
| $\partial_H(a) = \delta$ if $a \in H$                       | D2  |
| $\partial_H(x + y) = \partial_H(x) + \partial_H(y)$         | D3  |
| $\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$ | D4  |

Fig. 2. Axioms of ACP.

## 2.2. Commentary on axioms

On intuitive grounds  $x \cdot (y + z)$  and  $x \cdot y + x \cdot z$  present different mechanisms and an axiom  $x \cdot (y + z) = x \cdot y + x \cdot z$  is not included in ACP.

The constant  $\delta$  is to be interpreted as deadlock or failure, i.e. as an action which cannot be performed, hence  $\delta x = \delta$ ; the law  $x + \delta = x$  postulates that in the context of an alternative it will never be chosen.

The source of intuition for the  $\parallel$ -operation axioms is the arbitrary interleaving semantics of parallelism. The operations  $\parallel$ , left-merge, and  $|$ , communication merge, are auxiliary operations helpful in obtaining a finitary specification of  $\parallel$ . The essential algebraic properties of  $\parallel$  and  $|$  are the *linearity laws* CM4, CM8, CM9. Intuitively,  $x \parallel y$  is  $x \parallel y$  but takes its initial step from  $x$ ; and  $x | y$  is  $x \parallel y$  but takes its initial step as a communication of an initial action of  $x$  and an initial action of  $y$ .

We discuss the origins of these laws, and their relation with those of Milner, in the concluding remarks (Section 7).

Considered in toto, the axioms can be viewed as an equational specification for a certain algebra of processes generated from  $A$  by the operations of  $\cdot$  and  $+$ , with  $\parallel, \sqcup, |$  defined by simultaneous recursion. Also, the equations define a well-balanced left-right term rewriting system (see [8] for a proof of the confluency and termination properties of the system).

### 2.3. Generators

Let  $P$  be a process algebra over  $A$ . A *subalgebra*  $Q$  of  $P$  is a subset  $Q$  of  $P$  containing  $A$  and closed under all the operations.

Let  $X = \{x_i \mid i \in I\}$  be a subset of a process algebra  $P$  over  $A$ . The smallest subalgebra of  $P$  containing  $X$  is denoted by  $\langle X \rangle$ . The algebra  $P$  is said to be *generated* by a subset  $X$  if  $P = \langle X \rangle$ . The subalgebra  $M(P)$  of  $P$  generated by  $A$  is called the *minimal subalgebra* of  $P$ .

### 2.4. Role of generators

Let  $P$  be a process algebra over  $A$  generated by  $X$ . Then one thinks of  $P$  as the class of all concurrent systems that can be manufactured from the system primitives of  $A$ , and the systems of  $X$ , by means of the composition tools represented by the operations of  $P$ . The algebra  $P$  represents a *level of system abstraction* determined by  $A$ ,  $X$  and the composition tools.

### 2.5. Homomorphisms

Let  $P$  and  $Q$  be process algebras over  $A$ . A *homomorphism*  $\phi : P \rightarrow Q$  is a map which respects all operations and which leaves atoms invariant. The *image* of a homomorphism  $\phi : P \rightarrow Q$  is an  $A$ -subalgebra of  $Q$ , denoted by  $\phi(P)$ .

A *congruence*  $\equiv$  on process algebra  $P$  is an equivalence relation respected by the operations of  $P$ ; the factor algebra we denote  $P/\equiv$ .

The usual relationship between homomorphisms and congruences obtains: let  $\phi : P \rightarrow Q$  be a homomorphism and define the relation  $\equiv_\phi$  for  $x, y \in P$  by

$$x \equiv_\phi y \text{ if and only if } \phi(x) = \phi(y).$$

Now  $\equiv_\phi$  is a congruence and we have the following:

**Homomorphism Theorem.** *If  $\phi : P \rightarrow Q$  is a homomorphism then  $P/\equiv_\phi$  is isomorphic to  $\phi(P)$ .*

### 2.6. Role of homomorphisms

Let  $P$  and  $Q$  be process algebras over  $A$  modelling two classes of concurrent systems. Then a homomorphism  $\phi : P \rightarrow Q$  models the *realisation of the systems of  $P$  as systems of  $Q$*  in such a way that the system primitives in  $A$  and the composition tools are preserved.

Viewing  $P$  and  $Q$  as levels of abstraction, a homomorphism  $\phi : P \rightarrow Q$  realising  $P$ -systems as  $Q$ -systems may be considered in one of two ways:

- (i) *Top-down*: The systems of  $P$  are *implemented, specialised or refined* by the systems of  $Q$ ;
- (ii) *Bottom-up*: The systems of  $P$  are *abstractions or modularisations* of the systems of  $Q$ .

And it is important to note that the *homomorphism property guarantees the compatibility of system architectures and applies usefully to both the design processes of top-down implementation and bottom-up modularisation*. In this paper we will be concerned exclusively with the first subject, of course.

Often an algebra homomorphism  $\phi: P \rightarrow Q$  is associated with the idea of an abstraction being made, for

$$P / \equiv_{\phi} \equiv \text{im}(\phi)$$

and hence there is an identification of systems in  $P$  to make classes of systems represented by the systems of  $\text{im}(\phi)$ . This notion of an abstraction is *not* to be confused with the notions mentioned above which are dependent on the nature of the elements of their algebras and their computational purpose.

### 3. Standard concurrency and handshaking

We will use the laws of ACP as a foundation for the algebraic analysis of system construction by means of the composition tools described in Section 1. Now, an important source of intuition about concurrent systems is the language CSP, first described in Hoare [14]. This draws attention to three further ideas about concurrent processes:

- (1) Processes are configurations of atomic actions. In particular, a process *begins* with an atomic action, and this supports the intuition behind the operations  $\parallel$  and  $|$  which depends upon processes having initial atomic actions.
- (2) Parallel composition is a many argument operator

$$\text{cobegin}[X_1 \parallel \dots \parallel X_k] \text{coend}$$

that is both commutative and associative.

- (3) Communication is based on the synchronous execution of a pair of atomic actions within processes; for example,

$$c!x | c?y = y := x.$$

The conditions (1) and (2) are combined in the following requirements for  $\parallel$ ,  $\parallel$  and  $|$ , which we call the *axioms of standard concurrency*.

#### 3.1. Standard concurrency

A process algebra  $P$  has *standard concurrency* if it satisfies the following properties for  $x, y, z \in P$ :

$$\begin{aligned}
(x \parallel y) \parallel z &= x \parallel (y \parallel z), \\
(x | y) \parallel z &= x | (y \parallel z), \\
x | y &= y | x, \\
x \parallel y &= y \parallel x, \\
x | (y | z) &= (x | y) | z, \\
x \parallel (y \parallel z) &= (x \parallel y) \parallel z.
\end{aligned} \tag{SC}$$

These axioms are not independent relative to ACP, for instance the commutativity and associativity of  $\parallel$  are derivable from the other axioms.

Condition (3) can be recognised in the following definition of handshaking which entails that all communications are binary.

### 3.2. Handshaking

A process algebra  $P$  is said to have *communication by means of handshaking* if it satisfies for  $x, y, z \in P$

$$(x | y) | z = \delta. \tag{HS}$$

### 3.3. Expansion theorem

An important axiom of ACP is axiom CM1 which defines the parallel operator  $\parallel$  in terms of  $\parallel$ ,  $|$  and  $+$ . In the presence of associativity for  $\parallel$ , it is natural to examine the generalisation of the property CM1 from the binary parallel merge  $x \parallel y$  to the  $k$ -ary parallel merge  $x_1 \parallel \cdots \parallel x_k$  in process algebra  $P$ . This generalisation is called an *expansion theorem* after an analogous theorem in Milner [16] which eliminates  $\parallel$ . Here is some notation: let  $I_k = \{1, \dots, k\}$  and let  $x_1, \dots, x_k \in P$  then we write

$$X_k^i = \parallel_{t \in I_k - \{i\}} x_t, \quad X_k^{i,j} = \parallel_{t \in I_k - \{i, j\}} x_t.$$

That is,  $X_k^i$  is obtained by merging  $x_1, \dots, x_k$  except  $x_i$  and  $X_k^{i,j}$  is obtained by merging  $x_1, \dots, x_k$  except  $x_i, x_j$ .

**Theorem 1.** *Let  $P$  be a process algebra over  $A$  with standard concurrency and communication by handshaking. Then for any  $x_1, \dots, x_k \in P$*

$$(x_1 \parallel \cdots \parallel x_k) = \sum_{1 \leq i \leq k} x_i \parallel X_k^i + \sum_{1 \leq i < j \leq k} (x_i | x_j) \parallel X_k^{i,j}.$$

**Proof.** We use induction on  $k$  starting at  $k = 2$ : by axiom CM1

$$x_1 \parallel x_2 = x_1 \parallel x_2 + x_2 \parallel x_1 + x_1 | x_2$$

which is the required identity.

Suppose the identity is true for  $k$  and consider the case  $l = k + 1$ :

$$\begin{aligned} (x_1 \parallel \cdots \parallel x_l) &= (x_1 \parallel \cdots \parallel x_k) \parallel x_l \\ &= (x_1 \parallel \cdots \parallel x_k) \parallel x_l + x_l \parallel (x_1 \parallel \cdots \parallel x_k) + (x_1 \parallel \cdots \parallel x_k) | x_l. \end{aligned}$$

Let these three subterms be denoted  $\alpha, \beta, \gamma$  respectively. Then

$$\alpha = \left( \sum_{1 \leq i \leq k} x_i \parallel X_k^i + \sum_{1 \leq i < j \leq k} (x_i | x_j) \parallel X_k^{i,j} \right) \parallel x_l$$

by induction hypothesis;

$$= \sum_{1 \leq i \leq k} (x_i \parallel X_k^i) \parallel x_l + \sum_{1 \leq i < j \leq k} ((x_i | x_j) \parallel X_k^{i,j}) \parallel x_l$$

by CM4;

$$= \sum_{1 \leq i \leq k} x_i \parallel (X_k^i \parallel x_l) + \sum_{1 \leq i < j \leq k} (x_i | x_j) \parallel (X_k^{i,j} \parallel x_l)$$

by the first axiom of standard concurrency;

$$= \sum_{1 \leq i \leq k} x_i \parallel X_l^i + \sum_{1 \leq i < j \leq k} (x_i | x_j) \parallel X_l^{i,j}$$

$$\beta = x_l \parallel X_l^l$$

by definition of  $X_j^i$

$$\gamma = \left( \sum_{1 \leq i \leq k} x_i \parallel X_k^i + \sum_{1 \leq i < j \leq k} (x_i | x_j) \parallel X_k^{i,j} \right) | x_l$$

by induction hypothesis;

$$= \sum_{1 \leq i \leq k} (x_i \parallel X_k^i) | x_l + \sum_{1 \leq i < j \leq k} ((x_i | x_j) \parallel X_k^{i,j}) | x_l$$

by CM8;

$$= \sum_{1 \leq i \leq k} (x_i | x_l) \parallel X_k^i + \sum_{1 \leq i < j \leq k} (x_i | x_j | x_l) \parallel X_k^{i,j}$$

by second and third axioms of standard concurrency;

$$= \sum_{1 \leq i \leq k} (x_i | x_l) \parallel X_l^{i,l}$$

by handshaking and the definition of  $X_n^{i,j}$ .

In conclusion,

$$\begin{aligned} \alpha + \beta + \gamma &= \left( \sum_{1 \leq i \leq k} x_i \parallel X_l^i + x_l \parallel X_l^l \right) \\ &\quad + \left( \sum_{1 \leq i < j \leq k} (x_i | x_j) \parallel X_l^{i,j} + \sum_{1 \leq i \leq k} (x_i | x_l) \parallel X_l^{i,l} \right) \end{aligned}$$

which is the identity required in view of the definition of the  $X_n^i$  and  $X_n^{i,j}$ .  $\square$

#### 4. Models of ACP

In this section we will discuss five important kinds of ACP-algebra, namely:

|                  |  |
|------------------|--|
| $A_\omega$       | algebra of finite processes;                             |
| $A_\omega[X]$    | free algebra on set $X = \{X_1, \dots, X_n\}$ ;          |
| $A_\omega(X, E)$ | factor algebra of $A_\omega[X]$ by set $E$ of equations; |
| $A_n$            | $n$ th approximation algebra $A_\omega \bmod n$ ;        |
| $A^\infty$       | algebra of infinite processes;                           |

and their roles in solving recursion equations.

##### 4.1. Initial algebra $A_\omega$

Let  $A = \{a_1, \dots, a_n, \delta\}$  be an alphabet and let  $\gamma: A \times A \rightarrow A$  be a map that satisfies axioms C1–C3 and hence may serve as a communication function. Let  $ACP(A, \gamma)$  be the set of axioms obtained by adding to the ACP axioms over  $A$  the axioms

$$a \mid b = \gamma(a, b)$$

for each  $a, b \in A$ . Now  $ACP(A, \gamma)$  is a set of equations whose class  $MOD(ACP(A, \gamma))$  of models is the class of all ACP-algebras over  $A$  whose communication function extends  $\gamma$ . In particular, the class of models has an *initial algebra* which we denote  $A_\omega^\gamma$ , or simply  $A_\omega$ , where  $\gamma$  is understood.

**Example.** Let  $A = \{a, b, s!, s?, t!, t?, i, \delta\}$  and define communication  $\mid$  on  $A$  by setting

$$s! \mid s? = i \quad \text{and} \quad t! \mid t? = i$$

and all other communication values  $\delta$ . Let  $H = \{s!, s?, t!, t?\}$ . In  $A_\omega$  we have identities such as:

$$\begin{aligned}
 a \parallel b &= a \cdot b + b \cdot a, \\
 s! \parallel s? &= s! \cdot s? + s? \cdot s! + i \\
 \partial_H(s! \parallel s?) &= i \\
 \partial_H(a \cdot s! \cdot a \cdot t? \parallel b \cdot s? \cdot b \cdot t!) &= abi(abi + bai) + bai(abi + bai) \\
 &= (ab + ba)i(ab + ba)i \\
 &= \partial_H(a \parallel b)i\partial_H(a \parallel b)i.
 \end{aligned}$$

The algebra  $A_\omega$  will be used in further examples in this section.

##### 4.2. Terms

Let  $T_A$  be the term algebra over the signature  $\Sigma$  consisting of names for the atomic acts in  $A$  and for the operations  $\cdot, +, \parallel, \llbracket, \mid, \partial_H$  for  $H \subseteq A$ . There exists a surjective

homomorphism

$$v: T_A \rightarrow A_\omega$$

that semantically evaluates the syntax in  $T_A$ . By the Homomorphism Theorem in 2.5,

$$T_A / \equiv_v \cong A_\omega.$$

Of course  $\equiv_v$  is axiomatised by the set of equations  $ACP(A, \gamma)$ . Notice that the elements of  $A_\omega$  are finite processes made exclusively from atomic actions.

The initial algebra  $A_\omega$  is characterised by the following fact:

**Completeness Lemma.** *For any  $t_1, t_2 \in T_A$*

$$ACP(A, \gamma) \vdash t_1 = t_2 \text{ if and only if } A_\omega \models t_1 = t_2.$$

Let  $\Sigma_0$  be the subsignature of  $\Sigma$  containing names for atomic acts in  $A$  and for the operations  $+$ ,  $\cdot$  only. Let  $T_A^0$  be the subset of  $T_A$  containing the terms of  $\Sigma_0$ . Let  $BPA$  (=Basic Process Algebra) be the set of axioms of ACP that refer to sequential and alternative composition only, i.e. axioms A1–A7. Since  $BPA$  is a set of equations the class  $MOD(\Sigma_0, AS)$  of its models has an initial object  $A_\omega^0$ . As before we know the following:

**Completeness Lemma.** *For any  $t_1, t_2 \in T_A^0$*

$$BPA \vdash t_1 = t_2 \text{ if and only if } A_\omega^0 \models t_1 = t_2.$$

We can connect these two sets of laws by means of the following result eliminating parallelism:

**Elimination Theorem.** *For each  $t \in T_A$  there exists  $t^0 \in T_A^0$  such that*

$$ACP(A, \gamma) \vdash t = t^0$$

*or, equivalently,*

$$A_\omega \models t = t^0.$$

**Conservative Extension Theorem.** *The axiom system  $ACP(A, \gamma)$  is a conservative extension of the system  $AS$  in the following equivalent senses: for any  $t_1, t_2 \in T_A^0$ , proof theoretically conservative*

$$ACP(A, \gamma) \vdash t_1 = t_2 \text{ if and only if } AS \vdash t_1 = t_2$$

*and semantically conservative,*

$$A_\omega \models t_1 = t_2 \text{ if and only if } A_\omega^0 \models t_1 = t_2.$$

The proofs of these results can be extracted from Bergstra and Klop [8] as, indeed, can the proofs of the following facts:

**Theorem 2.** *The algebra  $A_\omega$  satisfies the axioms of standard concurrency. Moreover, if for all atomic acts  $a, b, c \in A$*

$$a \mid b \mid c = \delta$$

*then  $A_\omega$  satisfies the handshaking axiom.*

**Theorem 3.** *The algebra  $A_\omega$  is a computable algebra.*

#### 4.3. The free algebra $A_\omega[X]$ and its quotients

Let  $X = \{X_1, \dots, X_n\}$  and  $A_\omega[X] = A_\omega[X_1, \dots, X_n]$  be the free algebra on  $X$  in the equational class of process algebras defined by  $ACP(A, \gamma)$ . Let  $E$  be any system of equations:

$$t_1(X_1, \dots, X_n) = t'_1(X_1, \dots, X_n)$$

$$\vdots$$

$$t_n(X_1, \dots, X_n) = t'_n(X_1, \dots, X_n).$$

Then  $E$  defines a factor structure  $A_\omega(X, E)$  defined by

$$A_\omega(X, E) = A_\omega[X] / \equiv_E$$

where  $\equiv_E$  is the congruence generated by  $E$ . We will return to this type of model in 4.6.

**Illustration.** Let  $X = \{X_1, X_2\}$  and  $E$  contain only the pair

$$X_1 = as!X_1, \quad X_2 = bs?X_2.$$

Then in the algebra  $A_\omega(X, E)$  we have the following identity:

$$\partial_H(X_1 \parallel X_2) = (a \parallel b) \cdot i \cdot \partial_H(X_1 \parallel X_2).$$

#### 4.4. The approximation algebra $A_n$

We will define a family of algebras  $\{A_n: n \in \omega, n \geq 1\}$  such that  $A_n$  approximates  $A_\omega$  up to stage  $n$ .

On  $A_\omega$  one defines *projection operators*  $(\cdot)_n: A_\omega \rightarrow A_\omega$  as follows:

$$(a)_n = a, \quad (ax)_1 = a,$$

$$(ax)_{n+1} = a(x)_n, \quad (x+y)_n = (x)_n + (y)_n.$$

Here  $(x)_n$  is an  $n$ th approximation of the finite process  $x \in A_\omega$ .

For each  $n \geq 1$  a congruence relation  $\equiv_n$  on  $A_\omega$  is obtained by

$$x \equiv_n y \Leftrightarrow (x)_n = (y)_n.$$



The algebras  $A_\omega / \equiv_n$  are again  $ACP(A, \gamma)$ -algebras. We write  $A_n$  or  $A_\omega \bmod n$  for  $A_\omega / \equiv_n$ . Clearly,  $(\cdot)_n$  induces a homomorphism

$$(\cdot)_n : A_{n+1} \rightarrow A_n.$$

**Illustration.** In  $A_4$  we have the identities

$$as! \cdot as! = as! \cdot (as! \cdot as!), \quad (aaaa) \parallel (aaaa) = aaaa.$$

#### 4.5. The algebra of infinite processes $A^\infty$

The approximation mappings create a chain

$$A_1 \xleftarrow{(\cdot)_1} A_2 \xleftarrow{(\cdot)_2} A_3 \leftarrow \cdots$$

from which we define the *projective or inverse limit*  $A^\infty$  of the family  $\{A_n : n \in \omega\}$ . The algebra of finite processes  $A_\omega$  also embeds in  $A^\infty$ . For the details of this construction see Bergstra and Klop [7].

**Theorem 4.** *The algebra  $A^\infty$  satisfies the axioms of standard concurrency. Moreover, if for all atomic acts  $a, b, c \in A$*

$$a \mid b \mid c = \delta$$

*then  $A_\infty$  satisfies the handshaking axiom.*

**Illustration.** An element  $p$  of  $A^\infty$  can be represented as a sequence  $p_n \in A_n$  with  $p_n = \phi_n(p_{n+1})$  for all  $n$ . A typical example:

$$p_{2n} = (ab)^n, \quad p_{2n+1} = (ab)^n \cdot a.$$

For each  $n$ ,  $A_n \models p_n = a \cdot b \cdot p_n$ . Therefore, by definition,  $p = a \cdot b \cdot p$  in  $A^\infty$ .

#### 4.6. Equations

With reference to 1.7 and 4.3, systems of equations of the special form  $(X, E)$  where

$$X = \{X_1, \dots, X_m\} \quad \text{and} \quad E = \{X_i = t_i(X_1, \dots, X_m) : 1 \leq i \leq m\}$$

are an invaluable tool for the specification of processes; such equations are termed *guarded* or *Greibach equations* when each occurrence of a variable in a right-hand side is preceded or ‘guarded’ by an atomic action. Let us define them carefully, by induction; first the *unguarded terms*:

- (i) a variable is an unguarded term;
- (ii) if  $t$  is an unguarded term and  $t'$  is any term then

$$\begin{array}{lll} t \cdot t' & t + t' & t' + t \\ t \parallel t' & t' \parallel t & t \sqcup t' \\ t \mid t' & t' \mid t & \partial_H(t) \end{array}$$

are unguarded terms;

(iii) nothing else is an unguarded term.

Now, a  $t$  is *guarded* if and only if it is not unguarded; and an equation  $X = t$  is *guarded* if  $t$  is guarded. But we must consider their use with care.

Consider the equation, with  $a \in A$ ,

$$X = aX.$$

This fails to have a solution in  $A_\omega$ , whereas intuitively it serves to specify the infinite process

$$a^\omega = aaa \dots a \dots$$

The example illustrates the fact that  $A_\omega$  is not useful for defining the semantics of equational specifications of processes, and this leads to our interest in the algebras  $A_n$  and  $A^\infty$ . Clearly, in  $A_n$  the equation  $X = aX$  has a solution  $a^n$ . Indeed we have the following important result.

**Theorem 5.** *Let  $(X, E)$  be a system of guarded or Greibach equations. Then  $E$  has a unique solution in  $A_n$  for every  $n$ .*

In consequence,  $(X, E)$  has a unique solution in  $A^\infty$ . Thus guarded equations may be successfully employed for process specification using  $A_n$  and  $A^\infty$  for semantics. We have the following important algebraic reformulation of the above theorem:

**Consistency Lemma.** *Let  $(X, E)$  be a system of guarded equations. Then for every  $n$  there exists a unique homomorphism*

$$\phi_n : A_\omega(X, E) \rightarrow A_n.$$

Information on this subject can be found in Bergstra and Klop [7].

Let us observe that the guarded equations are solved in the presence of standard concurrency and handshaking in  $A_n$  and  $A^\infty$ . Algebraically, this point can be made explicit in the following reformulation:

**Consistency Lemma.** *Let  $(X, E)$  be a system of guarded equations. Then for every  $n$  there exists a unique homomorphism*

$$\phi_n : A_\omega(X, SC \cup HS \cup E) \rightarrow A_n$$

wherein  $SC$  and  $HS$  denote the sets of equations for standard concurrency and handshaking respectively.

To see this second Consistency Lemma, notice that  $A_n \models SC \cup HS$  and, hence, the following commutative diagram of unique homomorphisms exists:

$$\begin{array}{ccc}
 A_\omega(X, E) & \xrightarrow{\quad} & A_\omega(X, E) / \equiv_{SC \cup HS} \cong A_\omega(X, SC \cup HS \cup E) \\
 & \searrow \quad \swarrow & \\
 & A_n &
 \end{array}$$

## 5. Algebra of top-down design by stepwise-refinement

With the algebraic equipment described in Sections 2, 3 and 4 we can construct, concisely and quickly, a formal algebraic theory for the top-down design of concurrent systems.

### 5.1. Algorithmic notations

Let  $A_\omega$  be the initial process algebra over a given set  $A$  of atomic actions, as defined in 4.1. The algebra  $A_\omega$  can be thought of as an algebra, unique up to isomorphism, of *algorithmic notations* for concurrent processes over  $A$  subject to the system constraints axiomatised by ACP.

The algebra  $A_\omega[X] = A_\omega[X_1, \dots, X_k]$ , obtained by adjoining parameters  $X_1, \dots, X_k$  to  $A_\omega$  as defined in 4.3, can be thought of as an algebra, unique up to isomorphism, of *parameterised algorithm notations*.

### 5.2. Design

The aim of a design is an algorithmic notation  $t(X_1, \dots, X_k) \in A_\omega[X_1, \dots, X_k]$ , together with a collection  $E = \{e_1, \dots, e_l\}$  of equations over the algebra  $A_\omega[X_1, \dots, X_k]$ ; these equations we call a set of *design objectives*, *design characteristics* or *design constraints*. The role of these equations is to specify or constrain the properties of the parameters. Specifically, we attach to a specification  $(X, E)$  the *design algebra*.

$$D_A(X, E) \cong A_\omega(X, E).$$

The algebras of algorithmic notations  $A_\omega$  and  $A_\omega[X]$  and the design algebras  $D_A(X, E)$  are considered as syntactic objects tailored to the formal specification of *systems satisfying the laws of ACP*. These algebras have their semantics defined by homomorphisms into process algebras over  $A$ ; for example of the form

$$\phi : D_A(X, E) \rightarrow P.$$

### 5.3. Role of the design algebra and its semantics

The design algebra characterises a stage in the top-down design of a system; indeed, it mathematically determines a *level of system abstraction* for the stage of the design. This mathematical definition assists in assessing two fundamental concerns in design:

**Validity:** By calculating in  $D_A(X, E)$ , identities concerning the parameterised algorithm at hand can be derived; these identities are a principal tool for validating the design of the algorithm. An example of such a derivation will be given in 6.4.

**Consistency:** The consistency of the various specifications making up a design stage can be defined and established by means of homomorphisms from  $D_A(X, E)$  into an appropriate class of target algebras. In particular, using the class  $\{A_n: n \in \omega, n \geq 1\}$  of time-bounded processes over  $A$  we define this useful criterion for consistency.

**Definition.** The design stage  $D_A(X, E)$  is said to be *consistent* if for all  $n$  there exists a homomorphism

$$\phi_n: D_A(X, E) \rightarrow A_n.$$

Consider the design stage defined by

$$X = \{x\}, \quad E = \{x \cdot x = a\};$$

then  $D_A(X, E)$  is *inconsistent* because there does not exist a  $\phi_2: D_A(X, E) \rightarrow A_2$ .

On the other hand, for

$$X = \{x\}, \quad E = \{x = x \cdot a + b\}$$

the design stage  $D_A(X, E)$  is consistent as  $\phi_n: D_A(X, E) \rightarrow A_n$  is generated by

$$\phi_n(x) = b + (b \cdot a) + (b \cdot a^2) + \cdots + (b \cdot a^{n-1}).$$

#### 5.4. Top-down design

Consider a design project that proceeds in stages

$$S_{\text{initial}} \rightarrow \cdots \rightarrow S_i \rightarrow S_{i+1} \rightarrow \cdots \rightarrow S_{\text{final}}$$

and consider, in particular, the transition of one stage  $S_i$  to the next stage  $S_{i+1}$  which in top-down design is called a *refinement*. Algebraically, this step is represented by a homomorphism

$$\phi_{i,i+1}: D_A(X_i, E_i) \rightarrow D_A(X_{i+1}, E_{i+1})$$

where the sets of parameters and design characteristics satisfy

$$X_i \subset X_{i+1}, \quad E_i \subset E_{i+1}.$$

Typically, new variables arise as follows. Let  $X$  be a parameter in the set  $X_i$ . Then  $X$  represents a module specified at stage  $i$  to be refined in the transition to stage  $i+1$ . This refinement amounts to the substitution of an algorithm  $\tau(Y)$  with new module parameters  $Y = \{Y_1, \dots, Y_m\}$  for the module  $X$ . The design of this algorithm  $\tau(Y)$  is governed by old and new design characteristics. The homomorphism  $\phi_{i,i+1}$  that represents the refinement from stage  $i$  to stage  $i+1$  maps  $X$  to  $\tau(Y)$ .

### 5.5. Example

Let us reconsider the bag designed in 1.6. The atomic acts are in  $A = \{x, \underline{x}\}$  and the design consists of three stages  $S_1$ ,  $S_2$  and  $S_3$ .

The initial stage  $S_1$  is

$$D_1 = D_A(\{X, BAG(x)\}, \{BAG(x) = xXBAG(x)\}).$$

The second stage  $S_2$  is

$$D_2 = D_A(\{X, BAG(x), Y\}, \{BAG(x) = xXBAG(x), X = \underline{x} + xY\})$$

and the homomorphism  $\phi_{1,2}$  maps

$$X \rightarrow X \quad \text{and} \quad BAG(x) \rightarrow BAG(x).$$

The third stage  $S_3$  is

$$D_3 = D_A(\{X, BAG(x), Y\}, \{BAG(x) = xXBAG(x), X = \underline{x} + xY, Y = X \cdot X\})$$

and the homomorphism  $\phi_{2,3}$  maps

$$X \rightarrow X, \quad BAG(x) \rightarrow BAG(x), \quad Y \rightarrow Y.$$

This  $S_3$  algebra is the final stage of the design and we must check its consistency. Remembering the definition of consistency in 5.3, consider the algebra

$$D' = A_\omega(\{X, BAG(x)\}; \{BAG(x) = xXBAG(x), X = \underline{x} + xXX\}).$$

We observe that this algebra is based on guarded equations and hence the first Consistency Lemma in 4.6 can be applied to create homomorphisms  $D' \rightarrow A_n$  for all  $n$ . Now the consistency of  $S_3$  can be derived by means of a homomorphism  $D_3 \rightarrow D'$  mapping  $Y \rightarrow XX$ .

### 5.6. Final design stage

A stage in a top-down design is called a *final design stage* if for each of the variables in  $X$  in its design algebra  $D_A(X, E)$  there is a defining equation in  $E$ . At this stage no more substitutions can be made without jeopardizing the consistency of the design. An example is  $S_3$  in 5.5.

## 6. Top-down design of a communication protocol

In order to illustrate our algebraic machinery we consider the design of a toy communication protocol  $T$ . First we will give an informal specification of  $T$ .

### 6.1. Protocol behaviour

The protocol  $T$  is to allow the transmission of values 0 and 1 from a location  $P$  to a location  $Q$ , returning an acknowledgement  $a$  to  $P$  whenever a value has arrived at  $Q$ . A high-level specification of  $T$  as a process can be given graphically as in

Fig. 3, and, in symbols, as follows:

$$T = (0X0X' + 1Y1Y')aT. \quad (e_T).$$

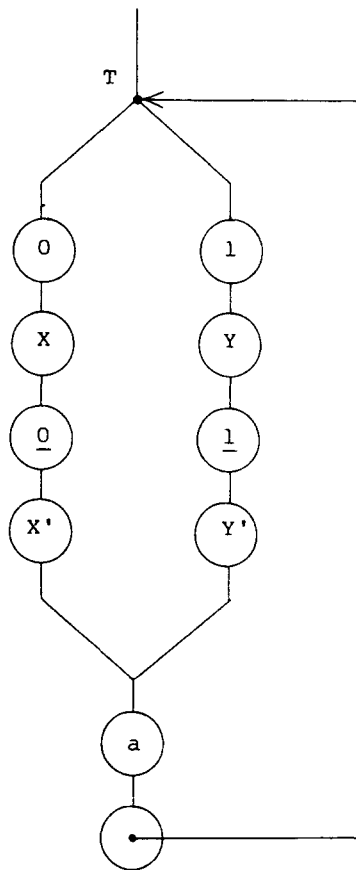


Fig. 3.

Here  $X$ ,  $Y$ ,  $X'$ ,  $Y'$  are undefined processes and the set

$$E = \{0, 1, \underline{0}, \underline{1}, a\}$$

of *external events* associated with  $T$  represents the behaviour of  $T$  by means of the denotations

- 0     receive 0 at  $\underline{P}$
- 1     receive 1 at  $\underline{P}$
- $\underline{0}$    deliver 0 at  $Q$
- $\underline{1}$    deliver 1 at  $Q$
- $a$     acknowledge at  $P$ .

Thus an example of a conversation at  $P$  is the process

$$0a1a1a0a1a1a0a\dots$$

### 6.2. Protocol architecture

We further specify that  $T$  is a system consisting of a *sender*  $p$  and a *receiver*  $q$  which communicate through *media*  $k$  and  $l$ . The *internal events* of  $T$  are bidirectional communications between these devices, namely: let  $s$  and  $\underline{s}$  denote communications between  $p, k$  and  $k, q$ , respectively; and let  $t$  and  $\underline{t}$  denote communications between  $p, l$  and  $l, q$  respectively. Let the set of internal events be

$$I = \{s, t, \underline{s}, \underline{t}\}.$$

Consider further these internal events at the interfaces or *ports* between the media; for example, consider the bidirectional communication  $s$  between  $p, k$ . The communication  $s$  is factorised into events  $s!$  and  $s?$  and one thinks of  $s!$  as the *act of offering* the signal  $s$  at the port and of  $s?$  as the *act of expecting* the signal  $s$  at the port. Both  $p$  and  $k$  can offer and expect  $s$ . The synchronisation of these events  $s!$  and  $s?$  results in the communication  $s$ ; in symbols

$$s! | s? = s.$$

By dividing the other communications similarly the set of hidden internal events

$$H = \{s!, s?, t!, t?, \underline{s}!, \underline{s}?, \underline{t}!, \underline{t}?\}$$

is obtained.

The entire pattern of events in the protocol  $T$  is depicted in Fig. 4.

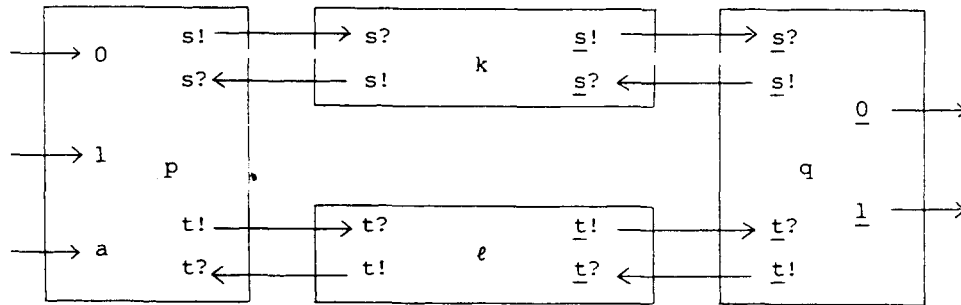


Fig. 4. Event structure of  $T$ .

Moreover, the media  $k, l$  can now be specified graphically by Fig. 5, and, in symbols, by

$$k = (s? \underline{s}! + \underline{s}? s!)k, \quad (e_k)$$

$$l = (t? \underline{t}! + \underline{t}? t!)l. \quad (e_l)$$

Thus,  $p$  and  $q$  are undefined programs made from the events of  $\{0, 1, a, s!, s?, t!, t?\}$  and  $\{0, 1, \underline{s}!, \underline{s}?, \underline{t}!, \underline{t}?\}$  respectively.

### 6.3. Algebraic specification

We will now formulate the design problem for  $T$  in algebraic terms.

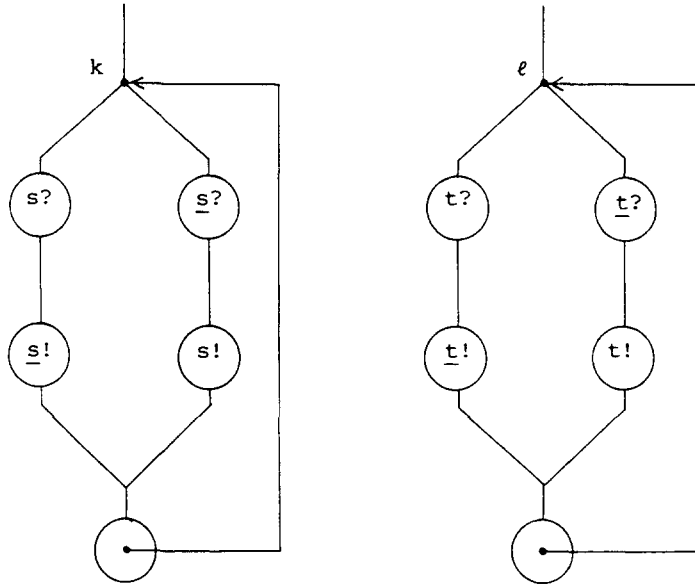


Fig. 5.

The set of atomic actions of interest is formed from the external, internal and hidden events, namely:

$$A = E \cup I \cup H \cup \{\delta\}.$$

The communication function  $\gamma: A \times A \rightarrow A$  is defined by

$$\gamma(\alpha, \beta) = \begin{cases} z & \text{if } \alpha = z! \text{ and } \beta = z? \text{ for } z \in \{s, t, \underline{s}, \underline{t}\}, \\ z & \text{if } \alpha = z? \text{ and } \alpha = z! \text{ for } z \in \{s, t, \underline{s}, \underline{t}\}, \\ \delta & \text{otherwise.} \end{cases}$$

The specifications result in a system of equations:

$$\text{Design objective:} \quad T = (0X0X' + 1Y1Y')aT, \quad (e_T)$$

$$\text{Design characteristics:} \quad T = \partial_H(p \parallel q \parallel k \parallel l), \quad (e'_T)$$

$$k = (s?s! + \underline{s}?s!)k, \quad (e_k)$$

$$l = (t?t! + \underline{t}?t!)l. \quad (e_l)$$

The *design problem* is to determine algorithms  $p, q$  (in terms of the appropriate atomic actions) that satisfy the above equations, and the system implementation constraints represented by the laws of ACP and the postulates of standard concurrency and handshaking.

Algebraically, the initial stage is represented by an algebra constructed as follows:

Let  $W = \{T, p, q, k, l, X, X', Y, Y'\}$ . Let  $G_0$  be the set containing the laws of ACP, together with the graph of  $\gamma$  on  $A$ , and the laws of standard concurrency and handshaking. Let  $G_1$  be the union of  $G_0$  and the set  $\{e_T, e'_T, e_k, e_l\}$  of design equations. Then the algebra required is

$$D_A(W, G_1) = A_\omega[W]/\equiv_{G_1}.$$



Interestingly, it is not clear that design equations can be satisfied and that the design problem is *consistent* (which we defined to mean that there exists a homomorphism

$$\psi_n : D_A(W, G_1) \rightarrow A_n$$

for each  $n$ : see 5.3). This is a standard problem for all top-down design projects of course!

#### 6.4. First refinement

We substitute algorithms for  $p$  and  $q$  over hidden actions as specified by Fig. 6, and by equations

$$p = (0 \ s! \ s? + 1 \ t! \ t?)ap, \quad (e_p)$$

$$q = (\underline{s}? \ 0 \ \underline{s}! + \underline{t}? \ 1 \ \underline{t}!)q. \quad (e_q)$$

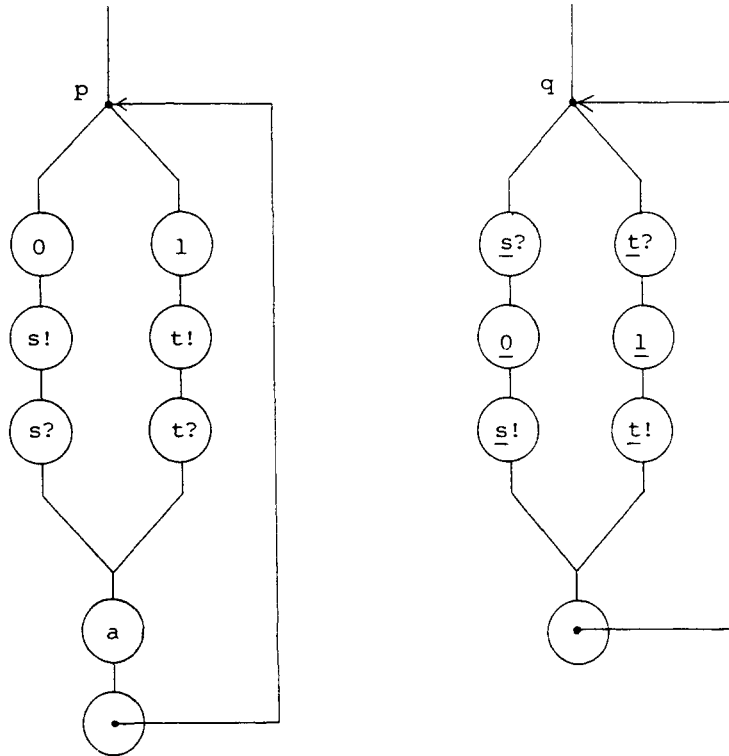


Fig. 6.

This leads to a new design algebra

$$D_A(W, G_2)$$

wherein  $G_2 = G_1 \cup \{e_p, e_q\}$ .

**Lemma.** In  $D_A(W, G_2)$  the following identity holds:

$$T = (0 \ \underline{s} \ 0 \ \underline{s} + 1 \ \underline{t} \ 1 \ \underline{t})a \ T.$$

**Proof.** The proof is a lengthy mechanical calculation using the Expansion Theorem

3.3 and the equations of  $G_2$ . Now

$$T = \partial_H(p \| q \| k \| l) = \partial_H(p \sqcup (q \| k \| l))$$

by applying the Expansion Theorem and observing that only the first term is not  $\delta$ . By the equation  $(e_p)$  for  $p$  and properties of  $\sqcup$ ,

$$T = 0 \partial_H((s!s?ap) \| q \| k \| l) + 1 \partial_H((t!t?ap) \| q \| k \| l).$$

Let us write  $T = t_1 + t_2$  and consider the term  $t_1$  separately

$$\begin{aligned} t_1 &= 0 \partial_H((s!s?ap) \| q \| k \| l) \\ &= 0 \partial_H((s!s?ap | k) \sqcup (p \| l)) && \text{by Expansion Theorem and } (e_k) \\ &= 0 s \partial_H((s?ap) \| (s!k) \| q \| l) && \text{by } (e_k) \\ &= 0 s \partial_H((s!k | q) \sqcup ((s?ap) \| l)) && \text{by Expansion Theorem and } (e_q) \\ &= 0 s \underline{s} \partial_H(k \| (0 \underline{s}! q) \| (s?ap) \| l) && \text{by } (e_q) \\ &= 0 s \underline{s} \partial_H((0 \underline{s}! q) \sqcup (k \| (s?ap) \| l)) && \text{by Expansion Theorem} \\ &= 0 s \underline{s} 0 \partial_H((\underline{s}!q) \| k \| (s?ap) \| l) && \text{by definition of } \sqcup \\ &= 0 s \underline{s} 0 \underline{s} \partial_H(q \| (s!k) \| (s?ap) \| l) && \text{by Expansion Theorem and } (e_k) \\ &= 0 s \underline{s} 0 \underline{s} \partial_H(((s!k) | (s?ap)) \sqcup (q \| l)) && \text{by Expansion Theorem} \\ &= 0 s \underline{s} 0 \underline{s} s \partial_H(k \| ap \| q \| l) && \text{by definition of } | \\ &= 0 s \underline{s} 0 \underline{s} s \partial_H(ap \sqcup (k \| q \| l)) && \text{by Expansion Theorem} \\ &= 0 s \underline{s} 0 \underline{s} s a \partial_H(p \| k \| q \| l) && \text{by definition of } \sqcup \\ &= 0 s \underline{s} 0 \underline{s} s a T && \text{by } (e_T). \end{aligned}$$

And similarly it can be shown that

$$t_2 = 1 t \underline{t} \underline{1} \underline{t} t a T$$

and hence that the required identity holds.  $\square$

### 6.5. Second refinement

We substitute algorithms for  $X, Y, X', Y'$  over internal actions as specified by the equations

$$X = s \underline{s}, \tag{e_X}$$

$$X' = \underline{s} s, \tag{e_Y}$$

$$Y = t \underline{t}, \tag{e'_X}$$

$$Y' = \underline{t} t. \tag{e'_Y}$$

This leads to the last design algebra

$$D_A(W, G_3)$$

wherein  $G_3 = G_2 \cup \{e_X, e_Y, e'_X, e'_Y\}$ . These substitutions are authorised by the Lemma in 6.4. Since there is now an equation for each variable,  $D_A(W, G_3)$  represents the final stage of the design. The consistency of the design is immediately established by observing that  $W, G_3$  constitute a guarded system of equations over ACP, augmented by standard concurrency and handshaking, and appealing to the second Consistency Lemma in 4.6.

## 7. Concluding remarks

The subject of this paper arises naturally from two areas of research:

- (i) the theory of concurrent processes; and
- (ii) the theory of the design of VLSI systems.

We will comment on both subjects.

### 7.1. Theory of concurrent processes

This paper is intended to contribute to an exclusively algebraic theory of concurrency. We view the axioms of ACP as a kernel of properties of the functional or behavioural semantics of concurrent systems. As demonstrated in Section 3, new axioms, consistent with ACP, may be added to analyse functional semantics. ACP is not intended as a tool to study the truly parallel operational semantics of systems; for that task a weakening of the axioms is required. To see this, observe that in ACP, for atomic actions  $a, b$

$$a \parallel b \equiv a \cdot b + b \cdot a + a | b$$

and that this equation fails to capture the idea that in actual operation  $a$  and  $b$  may be executed simultaneously, without properly communicating. It follows that CM1 does *not* capture the intuition of parallel execution, and must be replaced if the truly parallel operational semantics of  $\parallel$  is to be investigated. ACP is designed to handle the arbitrary interleaving semantics of parallelism which is supposed to be compatible (at the level of functional semantics) with *any* operational semantics of parallelism.

Composition operators for concurrent processes have been the subject of long standing research by R. Milner, an introduction to which is Milner [16]. An important idea is that of a *calculus*, called CCS, for the composition operators which describes their effects by means of *laws*. In recent studies of CSP the modular structure of programs is established by means of operators and their laws; see Hoare, Brookes and Roscoe [15], and Olderog and Hoare [18].

However, hierarchical aspects of system construction are treated only through special algebraic operators akin to the encapsulation operator in ACP which allows

an interconnected set of processes to be regarded as a single process with various hidden components. The theory of levels of abstraction for concurrent systems based on homomorphisms is new and its principal ideas, as explained in this paper, can be applied to other axiomatic approaches to concurrency.

To conclude these remarks on concurrency we will catalogue the principal influences on ACP. In addition to work on calculi for concurrency, from Milner's CCS we have adopted the laws A1–A5 and the idea of the expansion theorem; Milner's restriction operator is here called the encapsulation operator. From Hennessy [13] we have adopted laws C1 and C2. On the other hand,  $.$  represents full sequential composition and not just prefix multiplication as in CCS.

Also CCS has a fixed communication function

$$a \mid \bar{a} = \tau$$

where  $\tau$  is a new constant having its special laws, not present in ACP, and where it is supposed that atomic actions, other than  $\tau$ , exist in pairs:

$$A = \Delta \cup \bar{\Delta} \cup \{\tau\}$$

where  $\bar{a} \in \bar{\Delta}$  corresponds with  $a \in \Delta$ . (In [9] Milner's  $\tau$ -law have been incorporated in this algebraic framework.) Finally,  $\delta$  does not appear in CCS where part of its role is played by NIL; and  $\parallel$  and  $|$  are not present in CCS.

The left-merge  $\parallel$  and projective limit  $A^\infty$  first appeared in [7]. The full system ACP, including  $|$ , was introduced in [8]. Our work on ACP arose from a question in De Bakker and Zucker [3] about the existence of solutions for non-guarded fixed point equations in their topological model of processes ( $A^\infty$  is equivalent to their space of uniform processes); see also De Bakker and Zucker [4].

## 7.2. Theory of the design of VLSI systems

A VLSI system is a system specially implemented in silicon using a VLSI technology. The need for custom VLSI leads to the problem of programming into silicon wherein system descriptions are compiled into circuits. Thus, the following scientific problem is encountered:

**VLSI System Hierarchy Problem.** To analyse and structure VLSI computation as a hierarchy of levels of computation; and to develop formal many-level specification languages which have regard for verifying system behaviour and predicting system performance.

The problem asks for a generalisation of the von Neumann machine-language hierarchy (Bell and Newell [5]); and its answers may be as complex in their organisation. The subject of the design of concurrent systems, using composition tools, is important for this problem; and especially that of the hierarchical structure of concurrent systems, which determines top-down design.

The VLSI Hierarchy Problem is of interest to us: in Dew and Tucker [11] timing problems of the sequential operator are considered from a theoretical and experimental point of view. Calculations for a circuit at the functional unit level and at the level of a pass transistor logic implementation are made and shown to be inconsistent; and an experiment is described to investigate this discrepancy. In the language of this paper, the discrepancy means that homomorphisms do not necessarily preserve system timing.

The general algebraic theory of hierarchical computer systems described in Bergstra, Klop and Tucker [10] also addresses this problem.

## Acknowledgment

We thank E.R. Olderog for helpful conversations on the various approaches to concurrency. We thank our referees for their comments on an earlier version of the article. We thank Ms. Judith Thursby for typing this paper.

## References

- [1] ADJ (J.A. Goguen, J.W. Thatcher, E.G. Wagner and J.B. Wright), Initial algebra semantics and continuous algebras, *J. ACM* **29** (1977) 68–95.
- [2] ADJ (J.A. Goguen, J.W. Thatcher and E.G. Wagner), An initial algebra approach to the specification, correctness and implementation of abstract data types, in: R.T. Yeh, Ed., *Current Trends in Programming Methodology IV, Data Structuring* (Prentice-Hall, Englewood Cliffs, NJ, 1978) 80–149.
- [3] J.W. de Bakker and J.I. Zucker, Denotational semantics of concurrency, *Proc. 14th ACM Symposium on Theory of Computing* (1982) 153–158.
- [4] J.W. de Bakker and J.I. Zucker, Processes and the denotational semantics of concurrency, *Information and Control* **54** (1982) 70–120.
- [5] C.G. Bell and A. Newell, *Computer Structures: Readings and Examples* (McGraw-Hill, New York, 1971).
- [6] M. Ben-Ari, *Principles of Concurrent Programming* (Prentice-Hall, Englewood Cliffs, NJ, 1982).
- [7] J.A. Bergstra and J.W. Klop, *Algebra of Communicating Process*, CWI, Monograph Series 1 (North-Holland, Amsterdam) to appear.
- [8] J.W. Bergstra and J.W. Klop, Process algebra for synchronous communication, *Information and Control* **60** (1984) 109–137.
- [9] J.A. Bergstra and J.W. Klop, The algebra of regular processes and the algebra of recursively defined processes, *Proc. ICALP '84, Lecture Notes in Computer Science* **172** (Springer, Berlin, 1984).
- [10] J.A. Bergstra, J.W. Klop and J.V. Tucker, Algebraic tools for system construction, in: D. Kozen and E. Clarke, Eds., *Logics of Programs, CMU 1983, Lecture Notes in Computer Science*, **164** (Springer, Berlin, 1984).
- [11] P.M. Dew and J.V. Tucker, An experimental study of a timing assumption in VLSI complexity theory, University of Leeds, Department of Computer Studies, Report 168.
- [12] J.A. Goguen and J. Meseguer, An initiality primer, in preparation.
- [13] M. Hennessy, A term model for synchronous processes, *Information and Control* **51** (1981) 58–75.
- [14] C.A.R. Hoare, Communicating sequential processes, *Comm. ACM* **21** (1978) 666–677.
- [15] C.A.R. Hoare, S.D. Brookes and A.W. Roscoe, A theory of communicating sequential processes, *J. ACM*, to appear.
- [16] R. Milner, *A Calculus for Communicating Systems*, Lecture Notes in Computer Science **92** (Springer, Berlin, 1980).
- [17] Occam, *The Occam Programming Manual* (INMOS, Bristol, 1982).
- [18] E.-R. Olderog and C.A.R. Hoare, Specification-oriented semantics for communicating processes, Programming Research Group TM 37, Oxford (1984).