Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.A. Bergstra, J.V. Tucker

Top-down design and the algebra of communicating processes

TOP-DOWN DESIGN AND THE ALGEBRA OF COMMUNICATING PROCESSES

J.A. BERGSTRA

Centre for Mathematics and Computer Science, Amsterdam

J.V. TUCKER

Department of Computer Studies, University of Leeds, LEEDS LS2 9JT

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.

$bg F_{11}, bg F_{12}, bg F_{32}, bg F_{43}$

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 [7].

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 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.

We thank E.R. Olderog for helpful conversations on the various approaches to concurrency. We thank Ms. Judith Thursby for typing this paper.

The structure of this paper is as follows:

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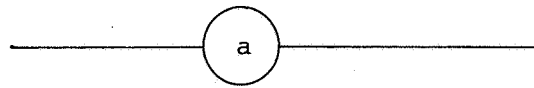
1.1 Sequential Composition Let X and Y be processes. The *sequential composition* of X and Y makes a process X.Y that schedules Y after the completion of X. If the processes X and Y are represented by nodes then X.Y is represented by the graph seen in Figure 1a.

1.2 Alternative Composition Let X and Y be processes. The *alternative composition* of X and Y makes a process X+Y that schedules either X or Y, but not both; its graph is seen in Figure 1a.

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,

$$Y = X.Y,$$

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

atomic action
a
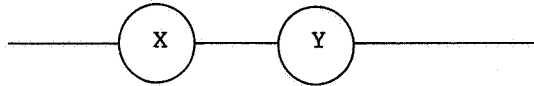
sequential composition
X.Y

alternative
composition
X+Y

iteration
Y=XY
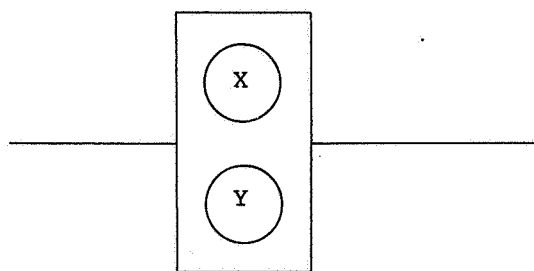
parallel
composition
$X \| Y$
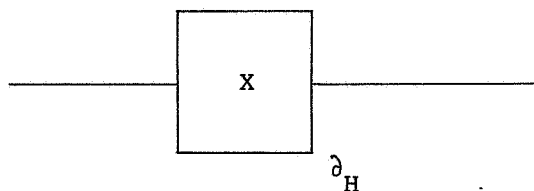
encapsulation
$\partial_H(X)$

Figure 1a : <u>Diagrams for processes</u>

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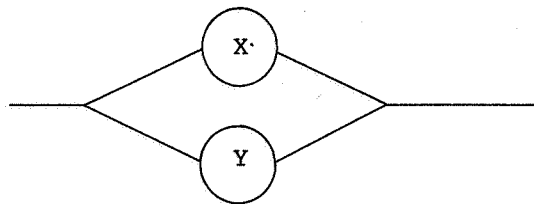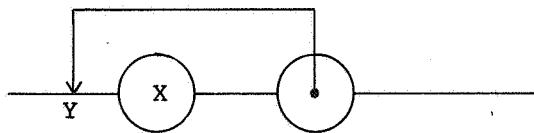the processes X and Y; its diagram is seen in Figure 1a.

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
$\underline{0}$	write 0
$\underline{1}$	write 1

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

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

1.6.2 Process $p = q.\underline{0}$

$$q = 0 + 1.\underline{1}q$$

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

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

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
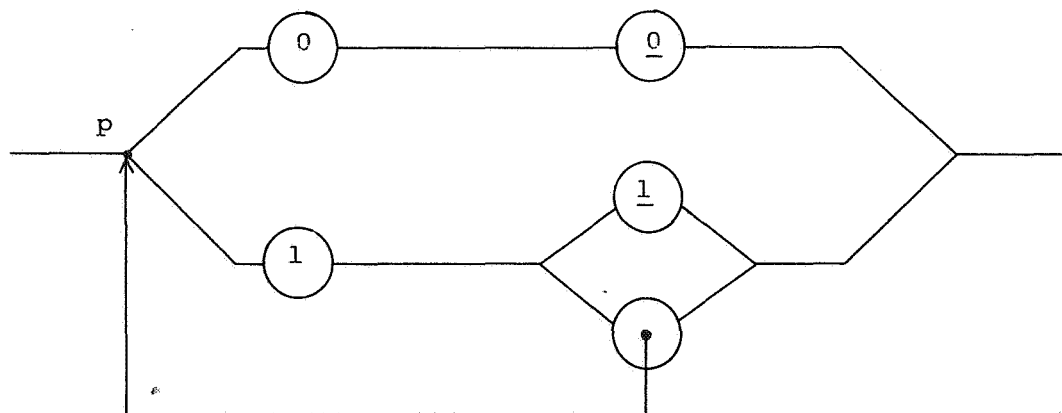
$$x \quad \text{input } x$$

$$\underline{x} \quad \text{output } x$$

and is algebraically defined by

$$BAG(x) = x.(\underline{x} \| BAG(x))$$

and is pictorially defined by
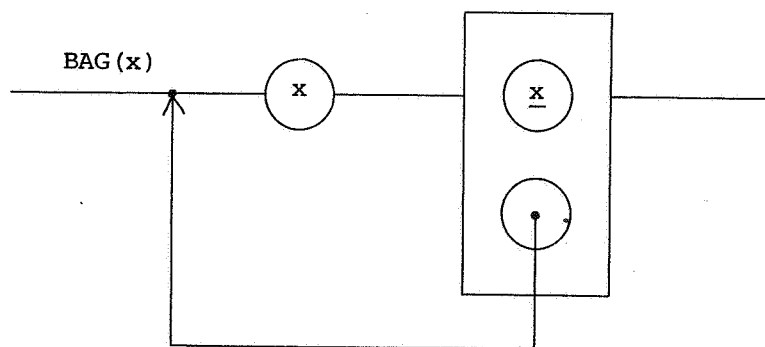
BAG(x)

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

1.6.5 <u>Two Element Bag</u> 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

$$BAG(0,1) = BAG(0) \| BAG(1)$$

The picture is simply

BAG(0)

BAG(1)
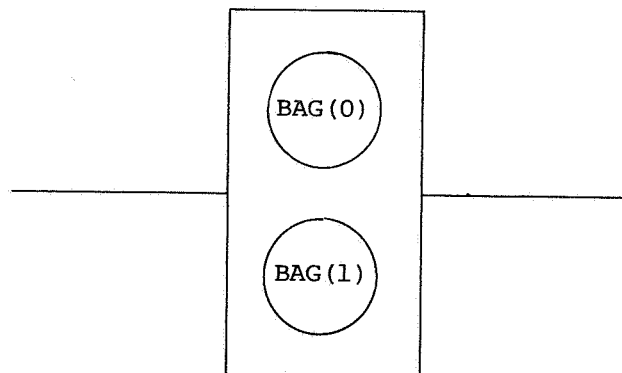
1.6.6 <u>Two Element Bag II</u>   A different specification for the two element bag over {0,1} is the fixed-point equation

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

This is pictured

BAG(0,1)
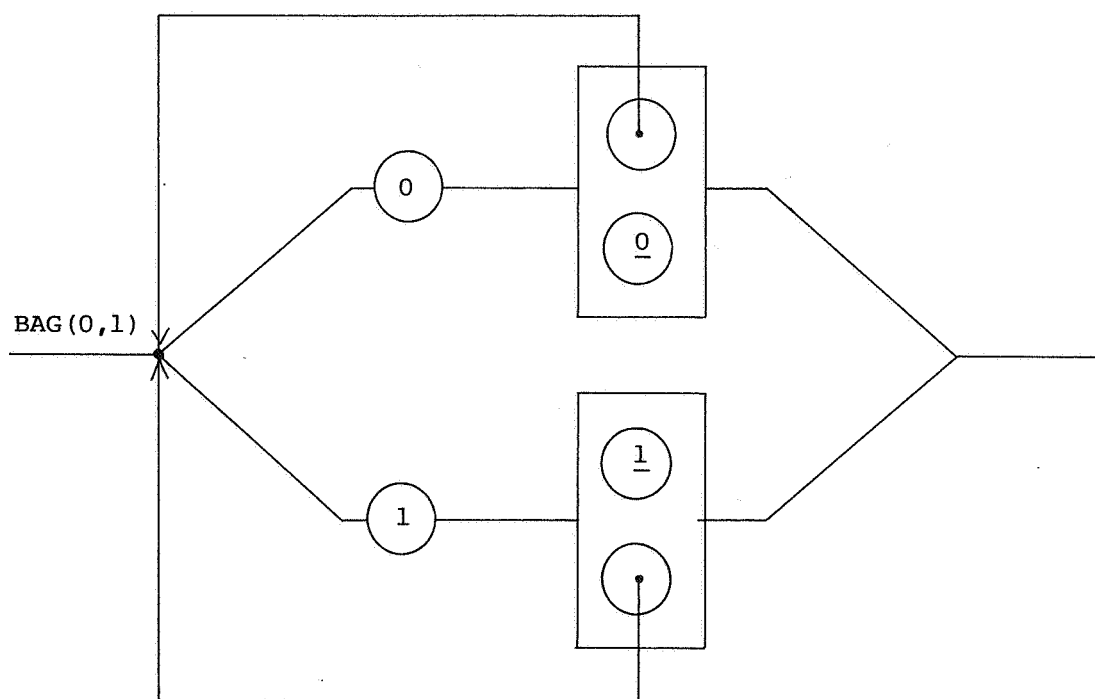
This bag has been examined in detail in Bergstra and Klop [9].

   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 redesign the one element bag discussed in 1.5.4.   Let
the behaviour of the bag be specified by
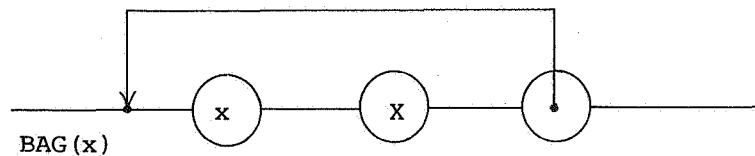
BAG(x)

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
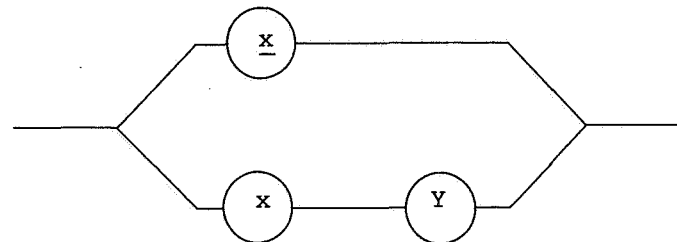
and, in symbols,

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

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

        We observe that Y=X.X.   Hence substituting in previous
specifications we obtain the graph
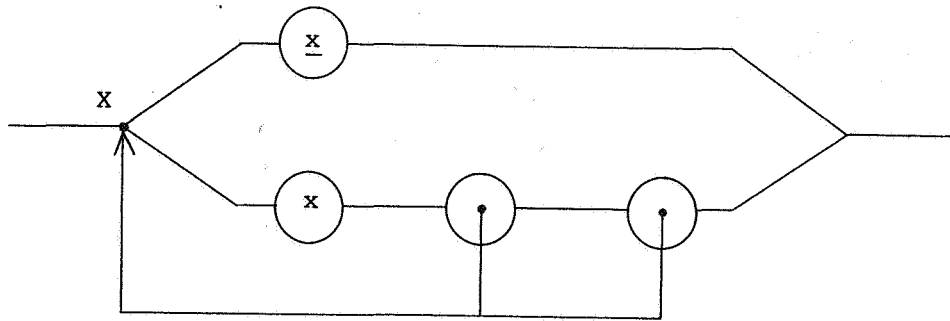
and the formula

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

Thus, the process BAG(x) is specified by substituting this graph for the node X in its top-most description, with the following result:
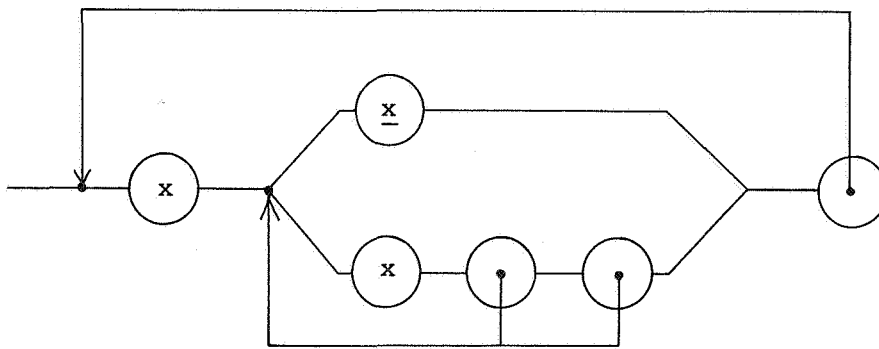
Algebraically, we are left with the following pair of algebraic equations to define the bag :

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

1.8   Observations   The following points are to be noticed :

         A level of abstraction for systems is characterised by its 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.

         The algebraic (and, indeed, graphical) notations for processes are very attractive, but they require a proper mathematical theory if they

are to be of use. The purpose of this paper is to outline that theory
(leaving the graph-theoretical theory to another occasion). We close
this section with the following correspondences :

architecture	graph	set of equations
top-down design	graph substitution	algebraic substitution

2. ALGEBRA OF COMMUNICATING PROCESSES

We will explain certain algebraic properties of the process building operations $\cdot$, $+$, $\|$ and ∂_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* over A consists of a set P equipped with the following operators

sequential composition	$x \cdot y$
alternative composition	$x + y$
parallel composition	$x \| y$
left merge	$x \parallel\!\!\!\!\llcorner y$
communication merge	$\mid$
delta	δ
encapsulation	$\partial_H(x)$

All operators are binary, except the constant δ, 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 '$\mid$' restricts as a map $\mid: A \times A \to A$.

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

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

$$x + y = y + x \qquad \text{A1}$$

$$x + (y + z) = (x + y) + z \qquad \text{A2}$$

$$x + x = x \qquad \text{A3}$$

$$(x + y) \cdot z = x \cdot z + y \cdot z \qquad \text{A4}$$

$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \qquad \text{A5}$$

$$x + \delta = x \qquad \text{A6}$$

$$\delta \cdot x = \delta \qquad \text{A7}$$

$$a \mid b = b \mid a \qquad \text{C1}$$

$$(a \mid b) \mid c = a \mid (b \mid c) \qquad \text{C2}$$

$$\delta \mid a = \delta \qquad \text{C3}$$

$$x \parallel y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x + x \mid y \qquad \text{CM1}$$

$$a \mathbin{\lfloor\!\lfloor} x = a \cdot x \qquad \text{CM2}$$

$$(ax) \mathbin{\lfloor\!\lfloor} y = a(x \parallel y) \qquad \text{CM3}$$

$$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z \qquad \text{CM4}$$

$$(ax) \mid b = (a \mid b) \cdot x \qquad \text{CM5}$$

$$a \mid (bx) = (a \mid b) \cdot x \qquad \text{CM6}$$

$$(ax) \mid (by) = (a \mid b) \cdot (x \parallel y) \qquad \text{CM7}$$

$$(x + y) \mid z = x \mid z + y \mid z \qquad \text{CM8}$$

$$x \mid (y + z) = x \mid y + x \mid z \qquad \text{CM9}$$

$$\partial_H(a) = a \quad \text{if } a \notin H \qquad \text{D1}$$

$$\partial_H(a) = \delta \quad \text{if } a \in H \qquad \text{D2}$$

$$\partial_H(x + y) = \partial_H(x) + \partial_H(y) \qquad \text{D3}$$

$$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y) \qquad \text{D4}$$

Figure 2a *Axioms of ACP*

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 <X>. The algebra P is
said to be *generated* by a subset X if P = <X>. 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 ϕ: P→Q is a map which respects all operations and which
leaves atoms invariant. The *image* of a homomorphism ϕ : P→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 ϕ: P→Q be a homomorphism and define the relation $\equiv_\phi$ for
$x,y \in P$ by

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

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

Homomorphism Theorem   *If ϕ:P→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 ϕ:P→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.

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 in 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.

(2)   Parallel composition is a many argument operator
$$\text{cobegin}[X_1 \| \,.... \,\| \,X_k] \text{ coend}$$
that is both commutative and associative.

(3)   Communication is based on the synchronous execution of a pair of atomic actions within processes; for example,
$$c!x \,|\, c?y \;=\; x := y \;.$$

The conditions (1) and (2) are combined in the following requirements for $\|$, $\|\!\_$ 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 $x, y, z \in P$

$$(x \|\!\_ y) \|\!\_ z = x \|\!\_ (y \| z)$$

$$(x \,|\, y) \|\!\_ z = x \,|\, (y \|\!\_ z)$$

$$x \,|\, y = y \,|\, x$$ SC

$$x \| y = y \| x$$

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

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

These axioms are not independent relative to ACP, for instance the commutativity and associativity of $\|$ 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 $x, y, z \in P$

$$(x \,|\, y) \,|\, z = \delta$$ HS

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

$$X_k^i = \underset{t \epsilon I_k-\{i\}}{\|} x_t$$

$$X_k^{i,j} = \underset{t \epsilon I_k-\{i,j\}}{\|} x_t$$

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 .

<u>Theorem</u>   *Let* P *be a process algebra over* A *with standard concurrency and communication by handshaking. Then for any* $x_1,\dots,x_k \epsilon P$

$$(x_1\| \dots \| x_k) = \sum_{1 \le i \le k} x_i \|\!\!\_ X_k^i + \sum_{1 \le i < j \le k} (x_i | x_j) \|\!\!\_ X_k^{i,j}$$

<u>Proof</u>   We use induction on k starting at k=2 : by axiom CM1

$$x_1\| x_2 = x_1 \|\!\!\_ x_2 + x_2 \|\!\!\_ x_1 + x_1 | x_2$$

which is the required identity.

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

$$(x_1\| \dots \| x_\ell) = (x_1\| \dots \| x_k) \| x_\ell$$

$$= (x_1\| \dots \| x_k) \|\!\!\_ x_\ell + x_\ell \|\!\!\_ (x_1\| \dots \| x_k)$$

$$+ (x_1\| \dots \| x_k) | x_\ell \quad .$$

Let these three subterms be denoted α,β,γ respectively. Then

$$\alpha = (\sum_{1 \le i \le k} x_i \,\|\!|\, x_k^i + \sum_{1 \le i < j \le k} (x_i | x_j) \,\|\!|\, x_k^{i,j}) \,\|\!|\, x_\ell$$

by induction hypothesis;

$$= \sum_{1 \le i \le k} (x_i \,\|\!|\, x_k^i) \,\|\!|\, x_\ell + \sum_{1 \le i < j \le k} ((x_i | x_j) \,\|\!|\, x_k^{i,j}) \,\|\!|\, x_\ell$$

by CM4;

$$= \sum_{1 \le i \le k} x_i \,\|\!|\, (x_k^i \,\|\, x_\ell) + \sum_{1 \le i < j \le k} (x_i | x_j) \,\|\!|\, (x_k^{i,j} \,\|\, x_\ell)$$

by the first axiom of
standard concurrency;

$$= \sum_{1 \le i \le k} x_i \,\|\!|\, x_\ell^i + \sum_{1 \le i < j \le k} (x_i | x_j) \,\|\!|\, x_\ell^{i,j}$$

$$\beta = x_\ell \,\|\!|\, x_\ell^\ell \qquad\qquad \text{by definition of } X_j^i$$

$$\gamma = (\sum_{1 \le i \le k} x_i \,\|\!|\, x_k^i + \sum_{1 \le i < j \le k} (x_i | x_j) \,\|\!|\, x_k^{i,j}) \,|\, x_\ell$$

by induction hypothesis;

$$= \sum_{1 \le i \le k} (x_i \,\|\!|\, x_k^i) \,|\, x_\ell + \sum_{1 \le i < j \le k} ((x_i | x_j) \,\|\!|\, x_k^{i,j}) \,|\, x_\ell$$

by CM8;

$$= \sum_{1 \le i \le k} (x_i | x_\ell) \amalg x_r^i + \sum_{1 \le i < j \le k} (x_i | x_j | x_\ell) \amalg x_k^{i,j}$$

by second and third axioms of standard concurrency;

$$= \sum_{1 \le i \le k} (x_i | x_\ell) \amalg x_\ell^{i,\ell}$$

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

In conclusion,

$$\alpha + \beta + \gamma = (\sum_{1 \le i \le k} x_i \amalg x_\ell^i + x_\ell \amalg x_\ell^\ell)$$

$$+ (\sum_{1 \le i < j \le k} (x_i | x_j) \amalg x_\ell^{i,j} + \sum_{1 \le i \le k} (x_i | x_\ell) \amalg x_\ell^{i,\ell}))$$

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

4. MODELS OF ACP

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

A_ω *algebra of finite processes;*

$A_\omega[X]$ *free algebra on set* $X = \{X_1, \ldots, X_n\}$

$A_\omega(X,E)$ *factor algebra of* $A_\omega[X]$ *by set* E *of equations*

A_n *n-th approximation algebra* A_ω *mod n;*

A^∞ *algebra of infinite processes;*

and their roles in solving recursion equations.

4.1   <u>Initial Algebra A_ω</u>   Let $A = \{a_1, \ldots, a_n, \delta\}$ be an alphabet and let $\gamma : A \times A \to A$ be a map that satisfies axioms C1–C3 and hence may serve as a communication function. Let ACP(A,γ) be the set of axioms obtained by adding to the ACP axioms over A the axioms

$$a|b \;=\; \gamma(a,b)$$

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

4.2   <u>Terms</u>   Let T_A be the term algebra over the signature Σ consisting of names for the atomic acts in A and for the operations $\cdot$, $+$, $\|$, $\|\!\!\!\rfloor$, $|$, ∂_H for $H \subseteq A$. There exists a surjective homomorphism

$$v : T_A \to A_\omega$$

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,γ). Notice that the elements of A_ω are finite processes made exclusively from atomic actions.

The initial algebra A_ω is characterised by the following fact :

<u>Completeness Lemma</u> *For any* $t_1, t_2 \in T_A$

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

Let Σ_o be the subsignature of Σ containing names for atomic acts in A and for the operations $+$, $\cdot$ only. Let T_A^o be the subset of T_A containing the terms of Σ_o. Let AS be the set of axioms of ACP that refer to sequential and alternative composition only, i.e. axioms A1-A7. Since AS is a set of equations the class $\mathrm{MOD}(\Sigma_o, \mathrm{AS})$ of its models has an initial object A_ω^o. As before we know the following :

<u>Completeness Lemma</u> *For any* $t_1, t_2 \in T_A^o$

$$\mathrm{AS} \vdash t_1 = t_2 \text{ } \textit{if, and only if,} \text{ } A_\omega^o \models t_1 = t_2 \text{ .}$$

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

<u>Elimination Theorem</u> *For each* $t \in T_A$ *there exists* $t^o \in T_A^o$ *such that*

$$\mathrm{ACP}(A, \gamma) \vdash t = t^o$$

or, equivalently,

$$A_\omega \models t = t^o$$

<u>Conservative Extension Theorem</u> *The axiom system* $\mathrm{ACP}(A, \gamma)$ *is a conservative extension of the system* AS *in the following equivalent senses : for any* $t_1, t_2 \in T_A^o$ *, proof theoretically conservative*

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

and semantically conservative,

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

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

<u>Theorem</u> *The algebra* A_ω *satisfies the axioms of standard concurrency. Moreover, if for all atomic acts* $a, b, c \in A$

$$a | b | c = \delta$$

then A_ω *satisfies the handshaking axiom.*

__Theorem__ *The algebra A_ω is a computable algebra.*

4.3 __The Free Algebra $A_\omega[X]$ and its Quotients__ Let $X = \{X_1, \ldots, X_n\}$ and $A_\omega[X] = A_\omega[X_1, \ldots, X_n]$ 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 :

$$t_1(X_1, \ldots, X_n) = t_1'(X_1, \ldots, X_n)$$
$$\cdot$$
$$\cdot$$
$$\cdot$$
$$t_n(X_1, \ldots, X_n) = t_n'(X_1, \ldots, X_n)$$

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.

4.4 __The Approximation Algebra A_n__ We will define a family of algebras $\{A_n : n \in \omega\}$ such that A_n approximates A_ω up to stage n.

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

$$(a)_n = a$$
$$(ax)_1 = a$$
$$(ax)_{n+1} = a(x)_n$$
$$(x+y)_n = (x)_n + (y)_n \quad .$$

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_ω is obtained by

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

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

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

4.5 <u>The Algebra of Infinite Processes A^∞</u>   The approximation mappings create a chain

$$A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow \quad \ldots$$
$$(.)_1 \quad (.)_2$$

from which we define the *projective or inverse limit* A^∞ of the family $\{A_n : n \in \omega\}$.   The algebra of finite processes A_ω also embeds in A^∞.

<u>Theorem</u> *The algebra A^∞ satisfies the axioms of standard concurrency. Moreover, if for all atomic acts $a,b,c \in A$*

$$a|b|c = \delta$$

then A_ω satisfies the handshaking axiom.

4.6 <u>Equations</u> With reference to 1.7 and 4.3, systems of equations of the special form (X,E) where

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

are an invaluable tool for the specification of processes; such equations we term *guarded* or *Greibach equations.* 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_ω, whereas intuitively it serves to specify the infinite process

$$a* = aaa\ldots a\ldots$$

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

<u>Theorem</u>   *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^∞. Thus guarded equations may be successfully employed for process specification using A_n and A^∞ 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*

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

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^∞. 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*

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

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

5.   ALGEBRA OF TOP-DOWN DESIGN BY STEPWISE-REFINEMENT

With the algebraic equipment described in Section 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_ω be the initial process algebra over
a given set A of atomic actions, as defined in 4.1.   The algebra A_ω
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 $A_\omega[X] = A_\omega[X_1,\ldots,X_k]$, obtained by adjoining parameters
$X_1,\ldots,X_k$ to A_ω 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
$t(X_1,\ldots,X_k) \in A_\omega[X_1,\ldots,X_k]$, together with a collection $E = \{e_1,\ldots,e_\ell\}$
of equations over the algebra $A_\omega[X_1,\ldots,X_k]$, that we call a set of
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_ω 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

$$\phi : D_A(X,E) \to P$$

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.

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 $\{A_n : n\epsilon\omega\}$ of time-bounded processes over A we define this useful criterion for consistency.

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

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

5.4   <u>Top-down Design</u>   Consider a design project that proceeds in stages

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

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

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

where the sets of parameters and design characteristics satisfy

$$X_i \subset X_{i+1}$$

$$E_i \subset E_{i+1}$$

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 $Y = \{Y_1, \ldots, Y_m\}$ 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 between stage i to stage i+1 maps X to $\tau(Y)$.

5.5   **E x a m p l e**   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

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

The third stage S_3 is

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

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

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

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
$D' \to A_n$ for all n. Now the consistency of S_3 can be derived by means of a
homomorphism $D_3 \to D'$ mapping $Y \to XX$.

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.

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 Figure 6a.
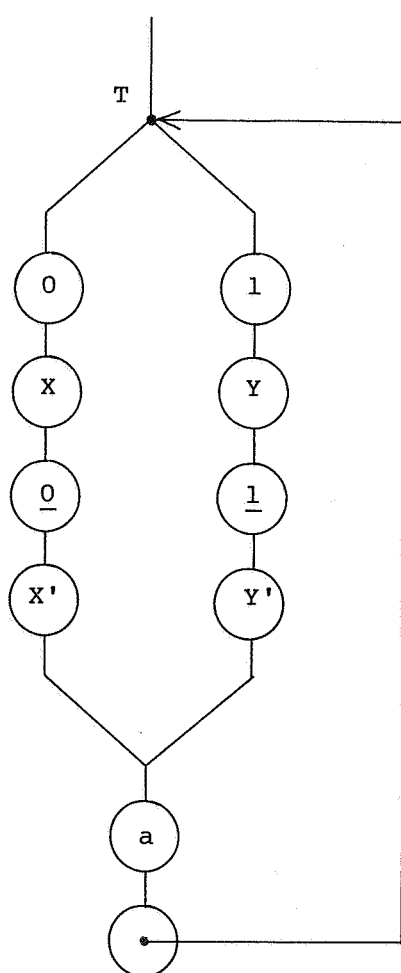
Figure 6a

and, in symbols, as follows :

$$T = (0X\underline{0}X' + 1Y\underline{1}Y')aT \qquad (e_T)$$

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

$$0a\underline{1}\underline{1}a0a\underline{1}\underline{1}a0a \ldots.$$

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 birectional 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
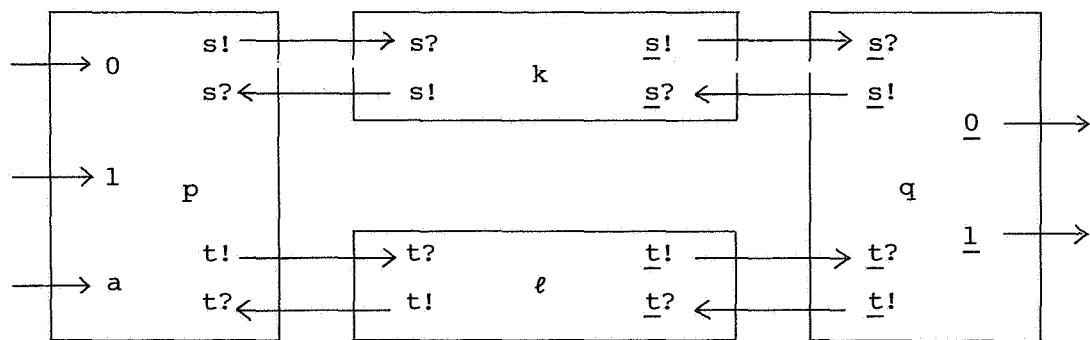
$$s! \,|\, s? = s$$

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 Figure 6b.

Figure 6b : Event structure of T

Moreover, the media k,ℓ can now be specified graphically by Figure 6c.

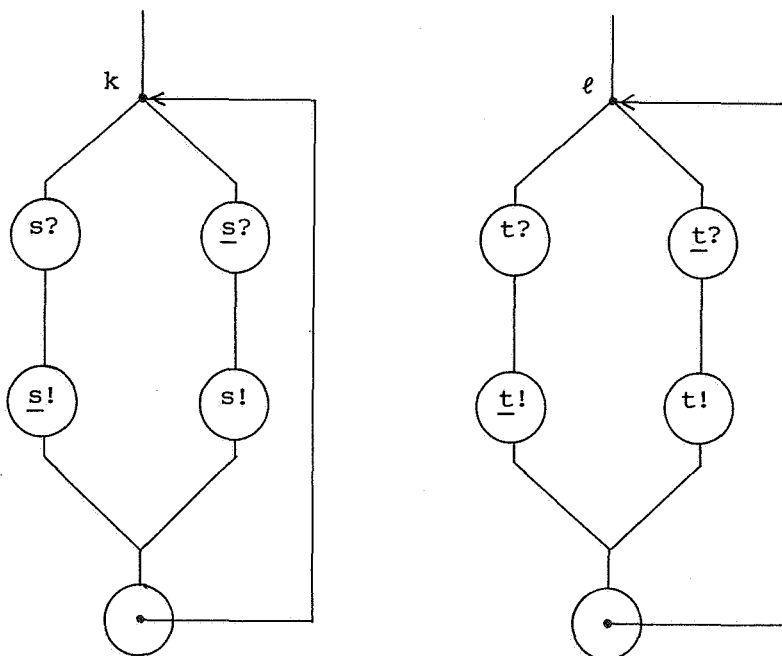Figure 6c

and, in symbols, by

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

$$\ell = (t?\underline{t}!+\underline{t}?t!\ )\ell \qquad (e_\ell)$$

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

6.3 Algebraic Specification

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

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 $\gamma : A \times A \to A$ is defined by

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

The specifications result in a system of equations :

Design objective :	$T = (0X\underline{0}X' + 1Y\underline{1}Y')aT$	(e_T)
Design characteristics :	$T = \partial_H(p\|q\|k\|\ell)$	(e_T')
	$k = (s?\underline{s}! + \underline{s}?s!)k$	(e_k)
	$\ell = (t?\underline{t}! + \underline{t}?t!)\ell$	(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 $W = \{T,p,q,k,\ell,X,X',Y,Y'\}$. Let G_0 be the set containing the laws of ACP, together with the graph of γ on A, and the laws of standard concurrency and handshaking. Let G_1 be the union of G_0 and the set $\{e_T,e_T',e_k,e_\ell\}$ of

design equations. Then the algebra required is

$$D_A(W,G_1) = A_\omega[W]/G_1$$

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

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

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

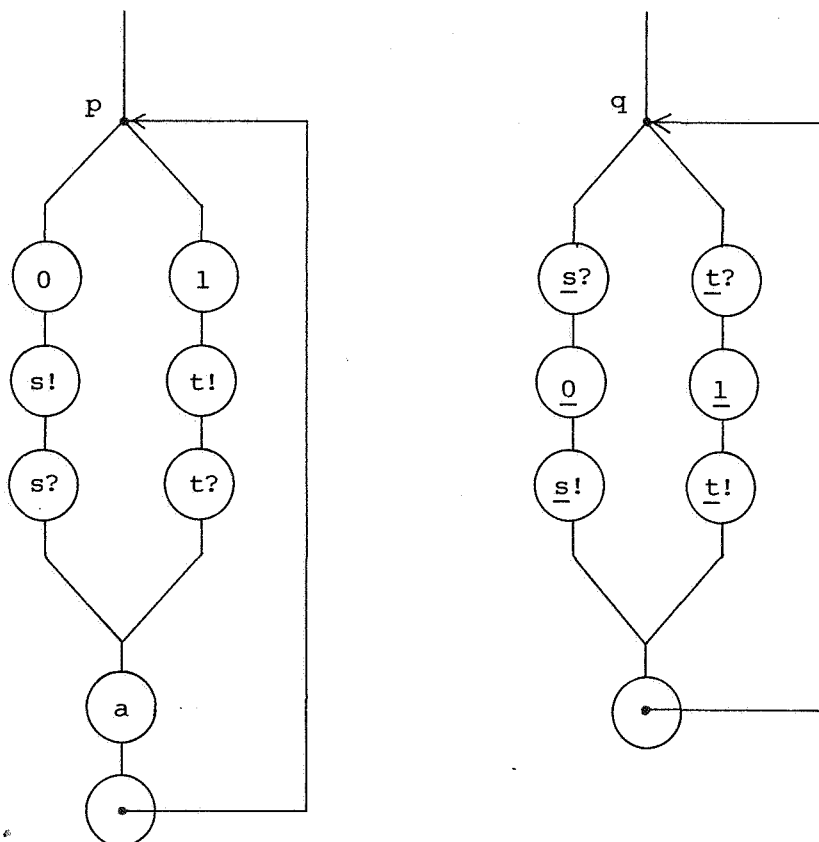6.4   <u>First Refinement</u>   We substitute algorithms for p and q over hidden actions as specified by Figure 6d

<u>Figure 6d</u>

and by equations

$$p = (0 \text{ s! s?} + 1t! \text{ t?})ap \qquad\qquad (e_p)$$

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

This leads to a new design algebra

$$D_A(W, G_2)$$

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

<u>Lemma</u>   In $D_A(W, G_2)$ the following identity holds :

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

<u>Proof</u>   The proof is a lengthy mechanical calculation using the Expansion Theorem 3.3 and the equations of G_2. Now

$$T = \partial_H(p \| q \| k \| \ell) = \partial_H(p \underline{\|} (q \| k \| \ell))$$

by applying the Expansion Theorem and observing that only the first term is not δ. By the equation (e_p) for p and properties of $\underline{\|}$,

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

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

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

And similarly it can be shown that

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

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

$$X = s \; \underline{s} \qquad\qquad (e_X)$$

$$X' = \underline{s} \; s \qquad\qquad (e_Y)$$

$$Y = t \; \underline{t} \qquad\qquad (e'_X)$$

$$Y' = \underline{t} \; t \qquad\qquad (e'_Y)$$

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 appealling 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
operational semantics of systems; for that task a weakening of the
axioms is required. For 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].

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 A1-5 and the idea of the expansion theorem; Milner's restriction operator is here called the encapsulation operator. (In [9] Milner's τ-laws have been incorporated in the algebraic framework.) From Hennessy [13] we have adopted laws C1 and C2.

The left-merge ‖ and projective limit A^∞ 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^∞ 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 hierarchiy 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.

Another source of practical work on the problem is the programming

of VLSI-oriented algorithms in the concurrent language OCCAM [17].

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

REFERENCES

1. ADJ (GOGUEN, J.A., J.W. THATCHER, E.G. WAGNER & J.B. WRIGHT),
    Initial algebra semantics and continuous algebras, JACM 29 (1977)
    68-95.

2. ADJ (GOGUEN, J.A., J.W. THATCHER & 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 (1978), 80-149.

3. DE BAKKER, J.W. & J.I. ZUCKER, *Denotational semantics of
    concurrency*, Proc. 14th ACM Symp. on Theory of Computing (1982),
    153-158.

4. DE BAKKER, J.W. & J.I. ZUCKER, *Processes and the denotational
    semantics of concurrency*, Report IW 209/82, Mathematisch Centrum
    1982, to appear in Information & Control.

5. BELL, C.G. & A. NEWELL, *Computer structures : Readings and Examples*,
    McGraw-Hill 1971.

6. BEN-ARI, M., *Principles of concurrent programming*, Prentice-Hall
    International, 1982.

7. BERGSTRA, J.A. & J.W. KLOP, *Fixed point semantics in process
    algebras*, Report IW 206/82, Mathematisch Centrum, 1982.

8. BERGSTRA, J.A. & J.W. KLOP, *Process algebra for communication and
    mutual exclusion*, Report IW 218/83, Mathematisch Centrum, 1983.

9. BERGSTRA, J.A. & J.W. KLOP, *An abstraction mechanism for process
    algebras*, Report IW 231/83, Mathematisch Centrum, 1983.

10. BERGSTRA, J.A., J.W. KLOP & J.V. TUCKER, *Algebraic tools for
    system construction*, in D. Kozen and E. Clarke (editors)
    Logics of Programs, CMU 1983, Springer LNCS, to appear.

11. DEW, P.M. & J.V. TUCKER, *An experimental study of a timing assumption
    in VLSI complexity theory*, University of Leeds, Department of
    Computer Studies, Report 168.

12. GOGUEN, J.A. & J. MESEGUER, *An initiality primer*, in preparation.

13. HENNESSY, M., *A term model for synchronous processes*, Information
    and Control, 51, (1981) 58-75.

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

15. HOARE, C.A.R., S.D. BROOKES & A.W. ROSCOE, *A theory of
    communicating sequential processes*, to appear in JACM.

16. MILNER, R., *A Calculus for Communicating Systems*, Springer LNCS 92,
    1980.

17. OCCAM, *The OCCAM programming manual*, INMOS, Bristol 1982.