using some memory of its own, to produce output on one or all of its output lines. It is assumed that:

i) Communication lines are the only way by which computing stations may communicate.

ii) A communication line transmits information within an unpredictable but finite amount of time.

Restrictions are imposed on the behaviour of computing stations:

iii) At any given time, a computing station is either computing or waiting for information on one of its input lines.

iv) Each computing station follows a sequential program. (We call here sequential program what is usually called a program elsewhere.)

Remarks: first, since several computing stations may be computing simultaneously, this model indeed exhibits some form of parallelism. Second, restriction iii) means that a computing station cannot be waiting on data coming from one or another of its input lines, or alternately that no two computing stations are allowed to send data on the same channel. Third, we do not restrict the computing stations to have a finite memory.

The reader who is mathematically inclined can think of a set of Turing machines connected via one-way tapes, where each machine can use its own working tape.

Example:
The process $f$ of program $F$ is associated to the continuous function $f$ in $N^2 \times N^2 = N^2$ defined recursively by:

$$ f((x_1, y_1), (x_2, y_2)) = A((f(x_1, y_1), f(x_2, y_2))) $$

The process $g$ is associated to two functions, one per output line, defined recursively by:

$$ g_1(x) = A((f(x), g_2(x))) $$

$$ g_2(x) = A((f(x), g_1(x))) $$

Similarly, the function $h$ maps $N^2$ into $N^2$:

$$ h(x, y) = A((x, y)) $$

where $x$ is in $N$ and the notation

(a) means the unit length sequence whose first element is $x$.

In these examples, not much computation is actually performed on the inputs, but an arbitrary amount of computation would be performed by a computing station, requiring possibly an unbounded amount of memory. The restriction of the interpretation of nodes to continuous functions can be understood in concrete terms:

- Monotonicity means that receiving more input at a computing station can only provoke it to send more output. Indeed this is a crucial property since it allows parallel operation: a machine need not have

Property:

The minimal solution of $Y_1(x), Y_2(x), \ldots, Y_n(x)$ of the
(N.B. : this is a way to ensure that the parallel recursive programs are syntactically well formed; it is sufficient for our purposes although it may give several labels to an edge). We construct now a set of fixpoint equations that contain variables in two types: sequence domains, and continuous mappings between sequence domains.

Example:
To the schema of fig.8, we associate the system:
\[
\sigma = P(1) = g_2(f(1, X))
\]
\[
X = g_2(f(1, X))
\]
where X and F are respectively an unknown sequence and an unknown continuous mapping between sequence domains. The continuous mappings from a c.p.o into a c.p.o constitute also a c.p.o. with the ordering \( f \leq g \) if \( \forall x \in X \) \( f(x) \leq g(x) \).

The existence of a solution (for functional) relations

6. DISCUSSION AND CONCLUSION
The kind of parallel programming we have studied in this paper is severely limited: it can produce only determinate programs. We argue however that:

1) Large parts of operating systems are written so as to be determinate. The method of monitors advocated by Hoare narrows down the possible locations of nondeterminism.

2) The primitives unit and send are on a level that we studied are not too far from reality as exemplified by [1], [2], [3], [4].

3) We do not think it is possible to extend the theory to non-determinate parallel programs, although how to satisfactorily do so is far from obvious.

4) The programming language we have introduced can be extended by adding new primitive processes (i.e. that cannot be programmed as processes with unit and send). A typical such process is

REFERENCES:
[4] IBM system 360 Operating system, Control program services, Form C28-6544-0.

1. Introduction

Many algorithms are naturally organized as systems of independent processes which interact, without one another. In this paper, we present a structure to the programming of such systems. This approach is embodied in a programming language which subordinates control to structure, relieving the programmer of the burden of control management and permitting the process systems to be executed either sequentially or concurrently with the same result. The language was designed to reflect the clear semantic conception of process interaction presented in [1], with the result that programs are relatively easy to verify.

1.2 Alternative approaches to concurrency

A different approach to concurrency, typified by the SIMULA control primitives call, goto, and return, is fairly widespread. [1]-[6] The SIMULA primitives can be used to implement Conway's style of concurrency, where control transfers are hidden in the input/output commands, but they also allow many other types of interaction which often result in intricate control relationships. Use of the return command in particular leads to obscure control structures, because it resembles a go to command with a moving target. For the sake of program reliability and verification one needs to impose discipline on the use of these primitives, and when this is done [10-18] it leads to the structuring of process interaction along the lines of Conway's original proposal.

1.3 Related ideas

The evaluation mechanism used in our system has its roots in theoretical work [19-20] on "call-by-
2.4 Reconfiguration

While a process program is running, it may be visualized as a directed graph where nodes represent processes and edges stand for communication channels. During computation, this graph may evolve in a top-down fashion: a node may be replaced by a subgraph, provided this subgraph can be appropriately specified into the incoming/outgoing edges (i.e., channels) of the original node. A reconfiguration instruction has the form:

```
repl {body} {name} {parameter-list}
```

and its body specifies a transformation of this sort. The keyword `repl` stands for "do concurrently" or "do in parallel." "Do in parallel" indicates that the selection of the processes to be taken over is based on some programmatic criteria, for example, based on the current status of the involved processes.

**Remark:** In Algol, a procedure call lumps together three distinct operations: the creation of new procedures, the binding of formal to actual parameters and control transfer. In SIMP (28) or PIP (29), some of these actions may be performed separately. Here, procedures are bound to their arguments as soon as they are created, but control is transferred in an entirely separate manner.

Conway's original coroutine scheme requires the coroutine network to be acyclic. The theory tells us that this restriction is unnecessary. From a pragmatic point of view, the extra cost incurred at execution time is minimal and feedback loops in coroutine programs are in fact quite useful. Another constraint, ensuring a unique entry point, is relaxed.

2.6 Functional notation

The constructs explained so far are sufficient for all programming. However, we can write much more elegant programs in a functional notation. Most processes have a single output line so that they are functions from streams to streams. (22) Thus in the way that Algol 60 permits functions along with procedures, processes may be declared functional and used to build stream expressions. In process calls occurring in reconfiguration instructions, such expressions may be provided as arguments where input channels are expected. For example, the program in fig.3 will now look like:
Process SIFT \( \langle x \leq x \rangle \) QN;
Print QN;
repeat

communication channels. Processes are represented
by data structures containing local access environ-
ments and control continuations. A channel

there is no need to deactivate the producer process; it may
continue to run in anticipation of further demands
for the channel. In this way, \( SIF(L) \) is a subsequen-

tic of \( L \).

Lemma 2: for any sequence L, SIF(L) is a subsequen-
tic of L.


