MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Computation Structures Group Memo
25

# ON THE EQUIVALENCE OF TWO ASYNCHRONOUS COMPUTATION DESCRIPTION SCHEMES

Fred L. Luconi

December 1966

# On the Equivalence of Two Asynchronous Computation Description Schemes

The advent of multi-processing computing facilities has given rise to the problem of how to describe programming systems consisting of several interacting sub-systems. Systematic procedures are needed to specify the necessary nature of these interactions and to determine their effect on total system performance.

Two recent doctoral dissertations at Project MAC have dealt with this problem using seemingly different approaches. Jorge Rodriguez has used a class of directed graphs called 'program graphs' to represent computational processes[1], whereas Earl Van Horn has develped an abstract automaton-like model which he calls an 'MCM', Machine for Coordinated Multiprocessing. Both of these multi-process computational description schemes establish a set of primary building blocks and rules for the interconnection and control of networks built from these blocks. The property of well-formed structures in both systems is that, given identical initial conditions, the sequence of different values appearing at any storage position (links in a Program Graph, cells in an MCM) during operation of the structure is uniquely determined and is independent of the relative computational processing times taken by any sub-computations.

## I. The Two Structures Studied

a) Program Graphs (1) represent a class of directed graphs. Corresponding to the branches of graphs are two types of links, i.e., control links or data links. Corresponding to the graph nodes are three types of program graph nodes, i.e., operators, selectors, or junctions.

An <u>operator</u> is a node which has two or more <u>input</u> <u>connectors</u> and one or more <u>output</u> <u>connectors</u>. The set of input connectors of an operator shall be considered to be ordered so that we may uniquely identify a connector by its ordinal number in the set. Similarly for the set of output connectors. Operators will be represented in a program graph by a circle: ( ○ )

A <u>selector</u> is a node which has two or more input connectors and precisely 2 output connectors. The sets of input connectors and the set of output connectors of a selector are also ordered sets. In particular one output connector of a selector will be labeled + and the other will be labeled -. Selectors will be represented in a program graph by a diamond-shaped symbol. ( ◇ ) Operators and selectors have a <u>zeroth</u> input connector which is distinguished in that it can only be the tip of a control link, furthermore it is not required for such a link to exist at all. All zeroth input connectors which remain unlinked are called free zeroth connectors.

---

1. Much of the description of program graphs is taken verbatim from reference 1.

A *junction* is a node which has two or more input connectors and one output connector. The set of input connectors of a junction is not ordered. Junctions will be represented in a program graph by a rectangle. (□)

A *link* is a directed line segment having a *root* and a *tip*. The direction specified in a link by means of an arrowhead is from the root to the tip.

In a program graph a link always connects an output connector of a node to an input connector of some other node. That is to say that the root of a link lies at some output connector while its tip lies at some input connector.

A *control link* is one which is rooted at an output connector of a selector or at the output connector of a junction whose input connectors are tips of control links.

A *data link* is one which is rooted at an output connector of an operator or at the output connector of a junction whose input connectors are tips of data links.

A *program graph* is a finite set of operators, selectors and junctions interconnected by means of control and data links according to the following rules:

a. Every input connector of an operator or selector must be the tip of one data link except for the zeroth connector which may only be the tip of one control link.

b. Every input connector of a junction is the tip of one data link or one control link. However, for any given junction all input connectors must be tips of the same type of link. We shall accordingly distinguish between data junctions and control junctions.

c. Any output connector may be the root of any number of links.

Links are capable of holding values which they receive from terminals or output connectors and give to the input connector at their tips.

The status of an input connector may assume two possible conditions which are called status 0 or status 1. When all the input connectors are in status 1, the operator or selector is applied and all input connectors are put in status 0. Such an occurance is called an A-event. When the application of the operator or selector has yielded a new set of values, a B-event occurs placing all output dependent input connectors in status 1.

A junction is applied whenever one or more of its input connectors are in status 1.

Under Rodriguez's interpretation, events may only occur at discrete time intervals as determined by an independent clock. Associated with each operator, selector and junction is an integer $t \geq 1$ specifying the number of intervals elapsed between the occurance of event A and event B for the node.

A program graph is deterministic if any two applications of the graph to the same set of input values yields execution sequences such that the same sequence of operators and/or selectors generates the values appearing on any specified link regardless of the time interval assignments to nodes of the program graph. Since all nodes are assumed to be deterministic, i.e., they always yield the same output values for a given set of input values, this means that the sequence of different values appearing on every data link is specified completely by the initial conditions.

For a thorough discussion of the properties necessary to insure the determinism of a graph, the reader is referred to reference 1. For the model, we will develop it is sufficient to say that every junction must have mutually exclusive inputs i.e., no two input connectors may ever be in status 1 simultaneously.

b) Machines for Coordinated Multiprocessing, MCM's, are the automata-like structures developed by Earl Van Horn to study the problem of multiprocess determinism. There are three basic unit types to an MCM structure, the cell, the control matrix, and the scheduler.

The cell represents the basic storage unit of the MCM. It corresponds to the memory cell, segment, hand register, etc. of a computer. However an MCM cell may be either a value cell or a clerk cell. When a cell is a value cell, its contents represent data which may be used or changed by qualified clerk cells. When a cell is a clerk cell, its contents specify a particular entry in the transaction table associated with each cell.

There must be one such entry in the transaction table for every possible value that can be held in the clerk cell. Two types of transaction entrys are

<u>get</u> i <u>replace</u> <u>with</u> f(·)

<u>put</u> i <u>with</u> v <u>replace</u> <u>with</u> f(·)

When activated a clerk cell specifying a <u>get</u> will fetch the contents of cell i and its contents will be changed according to f(·) which can have as arguments the current contents of the clerk cell and the fetched value. One thing this replacement can be used for is to sequence the actions of a clerk cell by loading it with the value necessary to specify the next instruction to be used. When a clerk cell specifying a <u>put</u> is activated, the value v will be placed in cell i and the contents of the clerk cell will be altered according to f(·).

The control matrix is Van Horn's means for controlling communication between processes. If there are n cells in a particular MCM, then the matrix will be n x n. If cell i has read-capability for cell j, i.e., cell i will be allowed to do a <u>get</u> j, then matrix element (i,j) must contain a number greater than or equal to 1. If cell i has write capability for cell j, i.e., cell i will be allowed to do a <u>put</u> j, then element (i,j) must be the only element in column j whose contents are greater than 0. Because a clerk cell must read its own value to specify a transaction and must write into itself because of the replacement function, a clerk cell is defined as a cell which has both read and write capabilities for itself. The three matrix altering instructions which may appear in a transaction table are:

```
send i to e         replace with w
done i              replace with w
bye to e            replace with w
```

The _send_ will add one to matrix element (e,i), thus giving cell e read capability for cell i. The _done_ performed by clerk cell x will subtract one from matrix element (x,i), representing the relinquishing of read capability for cell i by clerk cell x. The _bye_ performed by clerk cell x subtracts one from matrix element (x,x) and adds one to(e,x), representing clerk cell x yielding write capability for itself to cell e.

The scedhuler's job is to activate (by sending go-pulses to) those clerk cells which will be allowed to make their next transaction. Only enabled clerks receive go-pulses, where by enabled we mean a clerk that has all the read, write capabilities needed for the next transaction. Of the enabled clerks the scheduler selects a choice set, in which no two clerks will try to alter the same control matrix element if activated. The number of enabled, non-conflicting clerks the scheduler activates is completely at the choice of the scheduler and may even be chosen randomly if desired,as long as each enabled cell eventually does receive a go-pulse. Once activated a clerk is allowed to make exactly one transaction. After all activated clerks have completed their single transaction, a computational state exists and the scheduler selects the next choice set.

Van Horn proves that a properly constructed MCM displays complete functionality. By complete functionality is meant that given identical initial conditions (initial values in all cells and the control matrix), the ordered history of sucessive values obtained by each cell without regard to time, must be the same (where defined) for all executions of the MCM regardless of the relative processing times assigned to various transactions.

The next sections of this paper will show how an MCM can be modeled by a program graph and vice-versa. After this correspondence has been established, the remainder of the paper will discuss some properties of interacting computational processes which appear in both MCM's and Program Graphs and seem necessary to provide determinism.

## II. An MCM Model of a Particular Deterministic Program Graph, PG 1

To establish a time correspondence between the two representations, we will say that both the program graph and the MCM scheduler have access to a clock with period T, where T >> operation time of any clerk cell transaction.

a) All links, control or data, will be modeled as a single value cell having the same name as the modeled link. No difference will be made between data value cells and control value cells. They will be differieniated only by what kind of clerk cells can read and write their values..

b) All operators as shown in figure 1 will be modeled as two MCM cells, n.A and n.B, with the following properties.

i) cell n.A has the following transaction table.[1]

|  |  |  |
|---|---|---|
| <u>get</u> n.1 | <u>replace</u> <u>with</u> next transaction and retain the value of n.1 |  |
| <u>put</u> ns.1 | replace with next transaction |  |
| <u>send</u> ns.1 <u>to</u> n.B | " | " |
| <u>done</u> ms.1 | " | " |
| <u>get</u> n.2 | " | " |
| <u>put</u> ns.2 | " | " |
| <u>send</u> ns.2 <u>to</u> n.B | " | " |
| ⋮ |  |  |
| <u>get</u> n.P | " | " |
| <u>put</u> ns.P | " | " |
| <u>send</u> ns.P <u>to</u> n.B | " | " |

1) The <u>puts</u> are here assumed to store the value obtained by the last <u>get</u>

```
    done ns. P          "          "
    done n.1            "          "
    done n.2            "          "
      ⋮
    done n.P    replace with first transaction
```

The timing must be such that if all read-write capabilities are available, all of the above actions can take place within one T-interval. The scheduler will allow the clerk containing * to proceed only at a T-instant.

ii) cell n.B has the following transaction table

```
    get ns.P     replace with next transaction
```

$\left\{ \text{Application of node-n's function using the values in cells ns.1} \rightarrow \text{ns.P. This function must always be a pure procedure in that it always yields the same results for identical input values. When it is finished, it puts the output values on the output cell } \alpha.i_\alpha, \beta.i_\beta, \ldots, p.i_p \right\}$

```
    send  ns.1 to n.A          "          "
    done  ns.1                 "          "
    send  ns.2 to n.A          "          "
    done  ns.2                 "          "
      ⋮
    done  ns.p                 "          "
  * send  α.i_α to α           "          "
    send  β.i_β to β           "          "
      ⋮
    send  p.i_p to p  replace with first transaction
```

The scheduler will allow clerk containing * to proceed only at a T-instant.

We can now examine how the above clerk cells can simulate the action of operator n in PG 1. As values are presented to operator n, so cell n.A receives read capability for the cells n.1 thru n.p. This cell n.A having read capability for all n.i corresponds to input corrector i of n in PG 1 being in status 1.

At the first T-interval when n.A has received all necessary read-capabilities, it enables cell n.B (by giving it the read-capability necessary for its first transaction) and relinguishes its read capabilities for cells n.1 then n.p. This action corresponds to the A-event in PG 1, where, at the T-interval when operator n first has all status 1 inputs, it sets all these inputs to status 0 and proceeds to apply the function fn to the values received.

Once enabled, cell n.B must eventually apply fn to the values in ns.1 then ns.p. Note that this intermediate storage allows the function to be sequentially calculated if necessary. As various output values are obtained, they are placed into the cells corresponding tothe output links of operator n. Cell n.B will have write capability for these cells at this time because PG 1 is a deterministic program graph which implies that the dependent input connectors of all output links of an active operator must be in status 0 any time that operator can be applied and its outputs are needed. At the first interval when all output values have been calculated and placed, n.B gives read capability of the output link cells to all dependent operator, selector and junction representatives. This action, of course, corresponds to an event B in PG 1 where at the first T-interval that fn has all output values, these value are placed on the output links and corresponding input connectors are set to status 1.

c) All selectors as shown in figure 1 will be modeled by two MCM cells, $\beta$.A and $\beta$.B, with the same set of properties as an operator model. The only difference would be that $\beta$.B will write into two control value cells.

d) All junctions as shown in figure 1 will be modeled as three MCM cells J.A1, J.A2 and J.B with the following properties.

i) cell J.A1 has the following transaction table.

```
     get  J.1 replace with next transaction
     put  JS  replace with next transaction
*    done J.1       "              "
     send JS to J.B replace with first transaction
```

ii) similarly cell J.A2 has the following transaction table

```
     get J.2  replace with next transaction
     put JS   replace with next transaction
*'   done J.2       "              "
     send JS to J.B replace with first transaction
```

iii) Cell J.B has the following transaction table

```
     get JS   replace with next transaction
     done JS  replace with next transaction
     put $\alpha.i_\alpha$       "              "
*"   send $\alpha.i_\alpha$ to $\alpha$ replace with first transaction
```

The scheduler will enable clerk cells containing *, *', or *" only at a T-interval.

To clarify what is meant by saying that the MCM just defined simulates the behavior of PG 1, we must define how the two computational representatives are to be compared. Given corresponding initial conditions (corresponding constants and input values are equivalent), the sequence of values found in any link n.i of PG 1 at the times when its dependent input connector is in status 1, will be exactly the same sequence of values found in the corresponding value cell n.i of MCM when no clerk cell has write capability for n.i.

## III. A Deterministic Program Graph Model of a Particular MCM

To establish a time correspondence between the two representatives, we will say that the clock rate of the program graph synchronizer is so great compared to the functioning time of an MCM element that for all purposes we can think of the program graph as operating continuously.

Some of the modules might seem unnecessarily complex but this results from the seemingly more complex elements of an MCM. For example a cell may either a clerk cell or a value cell. Whereas a clerk cell can fetch any values it needs and has read capabilities for, an operator must have all input connectors in status 1 before its function becomes applicable. Whereas a value cell can be referenced as many times as possible by as many clerk cells as have read capability for it, a link can give its value to only one node one time; because after the value is accepted the connector is returned to status zero.

Keeping the properties of a deterministic program graph in mind, the models shown in figure 2 were developed. We will examine in detail how, the control matrix, the cells, and the scheduler are represented in this model.

a) The CONTROL MATRIX

The control matrix is represented implicitly in this model. The status of the control matrix, can, of course, be represented by a single number which we will call C. The current value of the contents of any cell M will correspondingly be represented by a number v.M.

When J.A1 or J.A2 receives and gets read-capability for a value, it gives this value to J.B through JS and relinquishes its read capability for J.1 or J.2 at the next t-interval. This corresponds to the operation in PG 1 where junction J upon receipt of a value over either J.1 or J.2 takes that value and sets the status of the input connector to 0.

When J.B gets a value through JS, it puts value on $\alpha.i_\alpha$ and gives read capability for this value to $\alpha$ at the next T-interval. This, of course, corresponds to a B-event in junction B, where the value received at the input is placed on the output links and the status of corresponding input connector is set to 1.

Note that there can be no conflict as to who has read-write capability for JS. Since we are modeling a deterministic program graph, J.A1 can never get read capability for J.1 at the same time that J.A2 gets read capability for J.2.

e) Constants and initial input values are represented simply as value cells for which all dependent clerks intially have read capability.

f) Initial conditions are represented by having all constants and initial inputs values placed in their respective value cells for which all dependent clerks have read capability. All clerks are enabled ordisabled, i.e., all cells modelling a node have read, write-capability for themselves. Finally, all event-B modelling clerk cells (cells whose name ends in .B) have write capability for all output link value cells, and all event-A modelling clerk cells have write capabilities for intermediate storage cells, ns., JS.

b) The SCHEDULER

The scheduler is modeled primarily by the operater, SCHEDULE. There is one input link to the scheduler from every cell of the modeled MCM. The value from cell M can be thought of as a pair, $\{v.m, C\}$ containing the current contents of cell m and the control matrix as seen by cell M. When all links, cell 2.i, are in status 1, the scheduler has an A-event and proceeds to apply its rather complex function to this representation of a computational state. Because of an MCM's structure the control matrix indicates whether a cell is a value cell or a clerk cell, and the contents of a clerk cell uniquely specifies the action it is to perform. With the available information the scheduler performs the following functions:

1) By comparing all the matrix representing values returned from the cells against the matrix value at the last computation state, MATRIX, SCHEDULE can decide if any of the clerk cells had performed an operation which changed the contents of the control matrix. If there were no changes, then the matrix remains unaltered and the value of C' given to all CELL1,i outputs and MATRIX will be the same as before. If some cells have indicated non-conflicting changes of the matrix, then the value of C' given to all CELL1.i outputs and MATRIX will reflect the composite effect of all these changes. If two clerks have attempted to alter the same matrix cell, then SCHEDULE allows only one change to become effective (using the same selection

algorithm as the modeled MCM uses in its choicing of a choice set). So that the effect of the unselected clerk cell's request is not lost, its CELL1.i link will receive, besides v.i and c' a note about the matrix change it desires and a KEEP.i control will be given to retain the information until the next computation state. As soon as the scheduler can allow the necessary matrix change by this cell, its action will be effected and the cell will again become enabled, i.e., it will receive a go.i control.

2) If $\{v.M, C\}$ indicates that cell M is a value cell that can only be read, the scheduler gives $\{v.M, C'\}$ to link CELL1.M and a control signal to KEEP.M so that the present contents of the cell will be retained until the next computation state.

3) If $\{v.. M, C\}$ indicates that value cell m can be written into, the scheduler looks at the contents of all clerk cells and determines if the one with write capability is indeed going to write into cell M (this lookup is deterministic because of the static information in transaction tables). If m is going to receive a new value, the scheduler merely puts $\{v.M, C'\}$ on CELL1,M. If the contents of M are not to be changed before the next computation state, $\{v.M, C'\}$ is put on link CELL1.M and a control signal is given to KEEP.M.

4) If $\{v.M, C\}$ indicates that cell m is an enabled clerk cell, then the scheduler puts $\{v.M, C'\}$ on CELL1.M and a control signal is given to go.M; if it is disabled then KEEP.M is activated instead of go.M. Moreover if v.M indicates that cell M will do a get k and m has read-capability for k, then the scheduler puts v.k on line get.k (these lines just receive some unused null value otherwise). At first glance this policy might seem more restricted than the possible random set of enabled clerk cells which Van Horn allows the scheduler to select. But since the rules for an MCM are such that an enabled clerk cell must eventually receive a go-pulse and since the contents of any cells which an enabled clerk cell may wish to read-out of or write-into cannot be altered at least until that clerk cell receives a go-pulse, nothing can be gained or lost by not holding up the sending of a go-pulse to this cell.

When all the aforementioned decisions have been made, SCHEDULE has B-event and a transition is begun to the next computational state.

c) THE CELL

One of three things may happen to a cell,m, in the transitions between computational states.

1) If it is a value cell that is not to be changed during this transition or a not-to-be-activated clerk cell, it will receive a KEEP.M control which will shunt the value on CELL1.M to link CELL2.M.

2) If it is a value cell whose value is to be altered during this transition, then some cell i will send a new value via link p.iM, which will then appear on link CELL2.M as M's new content value.

3) If a cell is activated clerk cell, it will receive a go.M control from the scheduler and $\{$v.M, C$\}$ will appear as an input to f.M. f.M will cause the same effects as if the corresponding cell in the modeled MCM contained v.M. Since every function performed by a clerk cell always replaces the cell contents according to some replacement function, the new contents v'.M will be placed on the link to J.M. If cell M would effect a change in the matrix value (from C to C'), then f.M sends $\{$v'.M, C'$\}$ to CELL2.M If cell M would do a put k, then put.MK would be set to the value to be placed. If cell m would do a get i, then f.M will receive the necessary data through link GET.IM. Whenever a get is not performed, some null value, $\phi$ , is given to f.M which is accepted but ignored. The fact that f.M would be activated only if C indicated that M had read-capability for i guarantee that CELL2.i = CELL1.i and f.M will receive a unique value of v.i regardless of the relative timing of the elements making up cells M and i.

The operation of the links, operators, and junctions will allow each program graph modelled cell to return a $\{$v.M, C$\}$ pair to the scheduler via links CELL2. which will correspond to the next computational state of the mocelled MCM. Thus the correspondence of values between the MCM and its Program Graph model is that the sequence of values passing through any cell M of the MCM will be the same as the sequence of values attained by v.M in link CELL1.M.

It appears as though our program graph model of the MCM could be used to prove Van Horn's result of the complete functionality of MCM structures. To do this we must first show that the program graph implied by our modelling procedure is always deterministic. Although it will not be proved in detail, this is indeed the case. One can convince himself that all of Rodriguez's rules for determinism are satisfied: all nodes are deterministic, all junctions have mutually exclusive inputs, and all cycle structure constraints are satisfied. Since a deterministic program graph implies a unique sequence of values through each link for a given initial configuration, and since to each modelled MCM cell, n, there corresponds a link CELL2.n which obtains the same values as the cell, it must be true that the sequence of values appearing in every cell of the MCM is determined uniquely by the initial conditions. This, of course, implies that complete functionality of the MCM.

## IV. Comments on Deterministic Structures

In both representations, the computational elements of processors and data are displayed explicitly. In the program graph, each processor (node) is able to receive and transmit data along a limited set of directed links to other processors. The MCM allows each cell to be either a processor or a repository of information for other processor cells. The status of each cell may change from clerk- to value-cell and back again many times during any run. The control matrix structure allows potentially any cell to communicate with any other cell. Moreover, unlike the directed link, a value cell used as a communicating link between clerk cells can allow information flow in any direction between these cells according to the status of the control matrix.

Two controls of the intercommunication between processors appear necessary to assure determinism: 1) A process must wait until all values necessary for the computation are available, 2) Once a processor decides to give a value to another processor that value cannot be altered until the receiving processor has decided for itself that it no longer needs this value. There is also the problem of concurrence, i.e., if there is ever a choice, between two or more processors, about which should be allowed to effect a change of the computational status, then this choice must be deterministic in that regardless of system timing: the same choice will always be made, the result of the choice must not affect the functionality of the system, or such a choice must never occur. Each of these rules for determinism is satisfied in the two representatives examined.

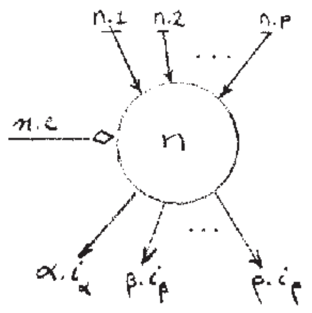
The operation of the control matrix and the corresponding enabling rules properly satisfy the rules of intercommunication. The only concurrence problem that arises is the possibility of simultaneous modification of a matrix element; this Van Horn prevents by having the scheduler restrict the activation of certain clerk cells until their actions no longer imply conflict. As the above mentioned communication restraints imply a necessity of feedback from receiving to sending processor, a general program graph is not guaranteed to be deterministic as the input connector status manipulations allow only feed forward of information from sending processor to receiving processor. The determinism of a program graph can therefore be established only if its particular structure is such that the communication rules are satisfied. A sufficient set of conditions on the structure has, as we have stated, been developed by Rodriguez.

Other schemes for representing asynchronous computation have also been developed for example, by D. Muller, R. McNaughton, and C. Petri. Each of these structures can satisfy the rules of communication and could be restricted to conflict free realizations. Of these the MCM model, alone, seems to allow possible communication in either direction to be established between any pair of processors, and, in fact, handles communication as a special kind of process in which the processor is in a particular mode, i.e., it does not have write capability for itself and so the transaction table may be thought of as either being disabled or as supplying an identity operator and replacement function.

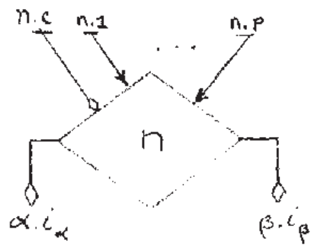
Work will continue to establish a unifying notion of deterministic asynchronous operation. The problem of concurrency and the generalization of the communication process particularly lend themselves to continued study.

## Program Graph Node Types

Operator



Selector



Junction

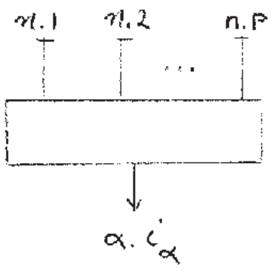
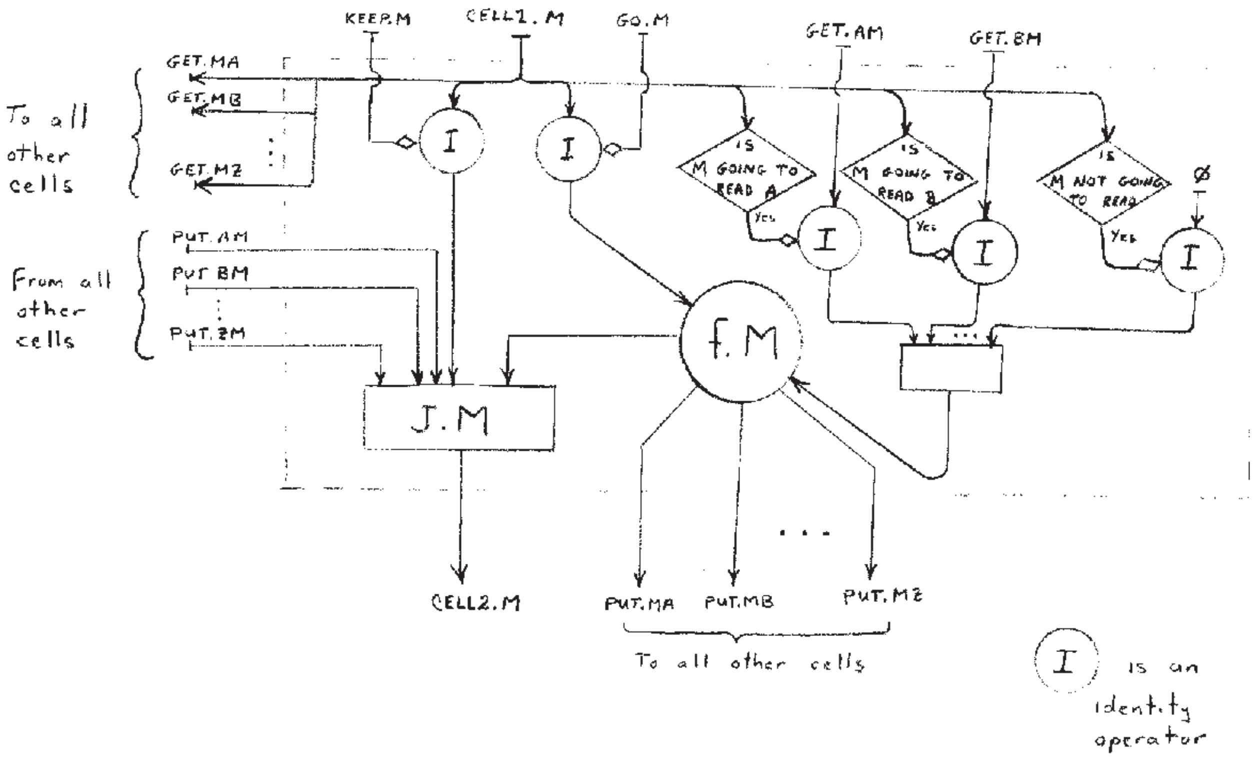


figure 1

# Model of Cell M



## Model of Scheduler

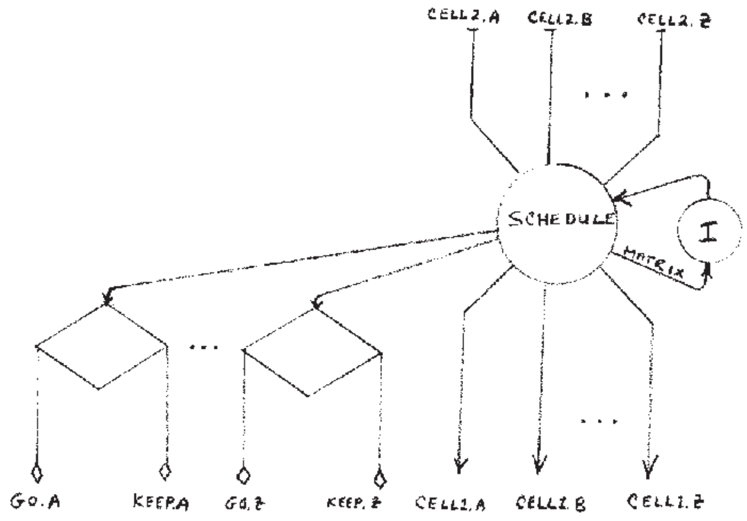


figure 2