MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Computation Structures Group

Memo No. 27

The Design and Transformation of Asyncrhonous Computational Structures (Part I - Completely Deterministic Structure Graphs)

by

Fred Luconi

February 1967

Abstract

The concept of the structure graph is introduced as a means of describing asyncrhonous, multi-process computational structures. The proposal is to constrain the description of such graphs so as to guarantee output determinism and to develop means for transforming such graphs into other graphs which are to be equivalent with respect to terminal behavior.

This memo is based, in part, on a proposal for thesis research in partial fulfillment of the requirements for the degree of Doctor of Philosophy submitted in the Electrical Engineering Department on 28 February, 1967.

I Introduction

The recent trend in the design of digital equipment towards the increased use of integrated circuitry, implies that certain design techniques ought to be re-evaluated. Design languages such as those developed by H. Schorr at Princeton or J.B. Dennis at M.I.T. reflect the current practice of building computer systems basically from registers and register transfer gating. Such structures are used because they seem to provide a means of expression easily understood by both the designers and any other parties who must become involved with the complexity of the system, e.g., maintenance technicians. Moreover, as these structures are most usually organized accordin~ to function, last minute changes can usually be made on only the functions involved, with little reorganization of the remaining structure required. The Huffman$^{(2)}$ state table or state graph model for sequential machines, however, has found relatively little use in practical design because structures of reasonable size when derived from the use of such techniques usually lack this property of understandability. Therefore, one criterion for a design representation scheme appears to be its clarity of interpretation.

The advent of integrated circuitry also necessitates a re-evaluation of our design criteria. Cost considerations have changed. The cost of a 100 component circuit is not substantially higher than the cost of a 20 component circuit. Cost becomes measured, for example, in terms of the number of terminals per integrated circuit. Moreover, since an integrated circuit cannot be internally changed or repaired once constructed, only the terminal characteristics of the circuit need be specified to determine its behavior in the system being designed.

<u>Theorem</u>: A wfsg is completely functional.

Proof: For sake of simplicity, this proof will demonstrate how the execution of any whsg can be simulated by an appropriately defined MCM (see ref. 7) and therefore Van Horn's result can be used to prove functionality directly.

Let us suppose that we are given a wfsg, G, and that we wish to build a simulation model of G by defining a corresponding MCM, M. For every active node C_α, in G, there will be a corresponding clerk cell, C_α, in M. For every link, $\ell(C_{\alpha_i}, C'_{\beta_j})$, there will be a corresponding value cell $\ell(C_{\alpha_i}, C'_{\beta_j})$ in M. Since in G, active nodes can alter the values of associated links, the clerk cells of M will correspondingly be able to alter the values stored in associated value cells. Since G is well-formed, no adjacent nodes will ever try to alter the value of a connecting link simultaneously, and therefore in M no two clerk cells with access to the same value cell will every need write capability for that cell at the same time. Folloing this reasoning, every action node can be modeled by a clerk cell with the following transaction table:

This implies that as long as the terminal behavior can be derived and represented in an illustrative format (such as one of the current design languages), the actual structure of the implementation within the circuit is of little concern.*
Implementation criteria such as conceptual organization may soon be replaced by such concerns as - Is this structure topologically representable on a two-dimensional plane (indicating single layer wirability), Is this structure tolerant to non-perfect yields on the chip, etc.

Much work has been done and, I am sure, will continue to be done on specifying algorithms for the decomposition of deterministic state tables or state graphs into various structural implementations, for example, the work on single feedback shift register realizations by R. Martin or the methods of loopless decomposition developed by H. P. Zeiger, both at M.I.T. An algebraic structure theory of sequential machines has even formed the basis of a book by J. Hartmanis and R. Stearns. Algorithms will probably soon evolve enabling state graphs to be decomposed into forms satisfying criteria such as the wireability or fault tolerance mentioned above. The one problem common to all these studies, however, is that they all specify their algorithms in terms of operations on deterministic state graphs or state tables, and engineers simply do not design computing structures in terms of deterministic state graphs or state tables.

*
[For example it might be perfectly reasonable to replace a shift register specified in a system design by a behaviorally-equivalent shift matrix although such a substitution might be untolerable, cost-wise, when using discrete logic].

If only for purposes of conceptualization, it appears necessary, therefore, that a useful digital design language (digital system representation scheme) be built around the concept of specifying, in detail, the operation and interconnection of a small set of easily understood design blocks or subsystems. If networks of such blocks are to represent deterministic machines, then it should be within the scope of the representation scheme to at least guarantee the output functionality of such networks.* As with existing design language descriptions, however, unless satisfaction of the design criteria mentioned above plus criteria yet to be defined, can be easily implimented directly into the scheme of representatio it should be possible to perform restructuring transformations on these networks which yield functionally equivalent representations that satisfy the desired criteria.

Since a machine description in any existing digital design language implies some corresponding state graph, why couldn't we just allow the designers to use a convenient description format, generate the implied state graphs, and then apply the decomposition algorithm to get the desired implemented structure? One reason is that as integrated circuits develop to the point where they may contain hundreds of active elements, the state graph for the hardware implemented on a single chip may be capable of assuming over 2^{100} different states and even with the use of large computers, dealing with state tables of this size is completely impractical. Moreover the structural implementation used for the original specification of the machine has put all the constraints between state variable dependence and timing interdependence implied by this structure into the derived state graph, yielding

*[Output functionality is defined as the property of a system to have its sequence of output values be uniquely determined by its initial "state" and the sequence of input values.]

a state graph containing more information than the amount necessary just to achieve the desired terminal behavior of the subsystem being implemented. In other words, even if the "state" of a system were observable, it was not in general intended as an observable entity by the designer. Unless there is some way of identifying this overspecification, these unnecessary (and probably undesirable) constraints must be carried through the decomposition algorithm and into the final realization. For this reason it seems desirable to make the original specification of the design in a representation which constrains only the behavioral (terminal) aspects of the system.

From the above considerations, it seems as though not only should a system be defined in terms of a network of interconnected subsystems, but that this network should also represent explicitly all behavioral relationships that it establishes. This implies that all functional relationships and timing constraints imposed by the network representation should be observable. For this reason it appears advisable to use descriptions of the system building blocks, the subsystem, in which the sequences of output values can be determined from the sequences of input values without regard for the time interdependence of these sequences; unless, of course, these timing constraints are explicitly expressed as part of the functional relationship. Moreover since synchronization of separate subsystems can be achieved only by the sharing of common information, e.g., a clock signal, the relative timing of subsystems which are logically disconnected must be considered as arbitrary. In other words, the relative timing of elements within a network should be assumed to be arbitrary, i.e., the network will operate asynchronously, allowing specific questions of synchronism to be answered only in terms of logical interconnection constraints. Therefore, if it is desired that systems modelled in the design language display output functionality (at least with respect to a specified set of input sequences), then procedures should exist within the representation scheme to guarantee this functionality in spite of the asynchronous operation of the network.

Still another important feature missing in existing digital representation schemes is an effective means for dealing with structures in which several different computations may be thought to be proceeding currently. Multi-processor computing systems represent present generation examples of such structures. If the locus of control for each computation is defined as a process, then it might be desirable to describe multi-computation systems in such a way that several processes can be allowed to proceed through the network representation (asynchronously with respect to one another), either independently or with allowable inter-communication, in such a manner that the output functinality of the system is assumed. In keeping with the desire to define systems in terms of only local constraints, this output functionality should be attainable through only local constraints on process activity without the need for global synchronization.

In summary, the objectives of the proposed thesis research will be to develop a scheme for the representation of finite sequential machines which meets the following objectives:

1) An engineer must be able to "easily" represent any set of finite state events within the framework of the representation scheme proposed.

2) Systems in which several asynchronously communicating processes are proceeding concurrently should be "easily" represented in the design scheme.

3) It should be possible to define a theory of equivalence for networks defined within the scheme of representation. Moreover, using this theory of equivalence, it should be possible to develop and define equivalence-preserving, restructuring transformations on such networks.

The following section contains a short description of several previously developed sequential machine representations. These will be briefly discussed in terms of how they individually satisfy the aforementioned objectives.

II Previous Work

When the Huffman(2) model is used for describing asynchronous sequential machines, proper operation is guaranteed only by observing the transient behavior of the particular excitation matrix implementation of the flow table. Whenever several state variables are allowed to change simultaneously, the resulting state must not depend on the order of these changes, i.e., all races must be non-critical. The use of this model requires that the entire "state" of the system be characterized in order to interpret the transient behavior.

As we have already mentioned, the "state" of even a single integrated circuit may soon reflect the status of so many state variables that the state table representation of a sequential machine might be intractable even with the use of large computers. Moreover the designer even now is unable to comprehend of his machine as making transitions from one "state" to another. The "state" in this context simply contains too much information for any reasonably sized sequential machine. This incomprehensibility of the Huffman "state" is magnified enormously when we begin to deal with machines in which several computations are thought to be proceeding concurrently. As the complexity and sophistication of design concepts increase, we can only expect the gap between Huffman state table analysis and practical design representations to increase.

David Muller(4) has developed a theory of logical nets that offers a solution to the problem of timing. Muller nets are completely speed independent in the sense that a block's communication with these blocks connected to it is allowed to proceed completely asynchronously. Very similar nets have been described by R. McNaughton(3) and H. Gray and J. Sims.(1) It is assumed that such asynchronous behavior can be achieved only by the use of feedback between communicating blocks. Each block can use the feedback to indicate to the blocks connected to its inputs when it is ready for a new problem and to indicate to the blocks connected to its outputs when it has an answer ready. As the timing within each block must be synchronized, such systems may be thought of as having replaced the necessity of complete synchronization by placing constraints on the information flow between disjoint sets of state variables and requiring local synchronization of the variables within these sets. Such studies have given much insight into the requirements of asynchronous design, and, in fact, the Illiac I was built at the Digital Computer Laboratory of the University of Illinois using many of Muller's ideas. Following this type of design procedure the engineer is able to satisfy the race constraints specified in terms of the state table of his system without having to refer to such a table; rather he need only be concerned with the state tables of the individual blocks separately. The results of these studies have yet to be interpreted in terms of designing structures which are able to concurrently process several interacting computations and as such must be examined more carefully in this context.

Earl Van Horn(7) has recently completed a doctoral dissertation at Project MAC on the development of machines for coordinated multiprocessing, or MCM's. Although primarily intended as a design concept for the development of a computing system or a programming language system, the class of abstract machines proposed offer some interesting constraints on any sequential machine in which several inter-communicating but arbitrarily timed processes are allowed to proceed concurrently. The computational state of an MCM is defined as the static contents of all the memory elements in the system, i.e., the cells and the control matrix. On the basis of the information in such a state, a "scheduler" will allow a certain set of processes to perform a single operation (the choice of this set may be completely arbitrary within the constraint of a few selection rules). When these operations conclude, a new computation state exists. A valid MCM, although it need not progress through the same sequence of computational states for a given set of initial conditions, can always guarantee the same sequence of different values through each cell. This property is called complete functionality. Since the output of the system is assumed to be the sequence of different values appearing in a particular subset of these cells, the output streams are completely determined by the initial conditions. Such an automaton lends itself much more directly to interpretation as a multi-computation asynchronous processor than the Huffman state table model. Another interesting aspect of this model is the possibility of communication in either direction between any pair of cells. The MCM does not provide an adequate model for all our objectives, however. The possibility of complete local control and scheduling of computations through processors is restricted by the necessity of a central control, the scheduler. Although any MCM built according to the rules must have complete functionality, the scheduler is required to resolve conflicts in attempts to change the control matrix. Moreover, such an automaton is much too constrained to be used as a behavioral description of a sequential machine. In particular, given a set of initial conditions, the sequence of values appearing in every cell is determined whereas, for a behavioral description all that need be determined is the sequence of values through those cells designated as outputs, output functionality.

C. Petri(5) has developed a theory of logic nets based on certain physical principles. Petri-structures may be explained by considering a set of memory elements, possibly overlapping subsets of which are connected through processors. Each processor continually monitors the values of the associated subset of memory elements. The processor's actions are defined only in terms of certain configurations of these values. When such a configuration appears, the processor makes a deterministic transaction which replaces the observed configuration with a new set of specified values. Since each processor acts independently, such networks exhibit completely local control. Moreover there is the possibility of bilateral communication between any pair of processors having access to common storage elements, indicating the ease with such structures might be used for multiprocessor machine description. As no timing information is assigned to any processor, the processing of such a network must, by definition, be asynchronous. Although Petri gives examples of how a very limited set of processors might be interconnected in a particular way to realize a universal Turing Machine, no general results have been stated which indicate how the class of allowable processors might be defined. Also because a general Petri-structure need not be output functional, either a set of rules for general processor interconnection or a class of processors allowing arbitrary interconnection must be defined to insure the output functionality of these networks.

III. The Model

The model used for the description of computational structures will be a class of undirected graphs called structure graphs. The vertices of such graphs will consists of a set of active nodes and terminal points which are interconnected by a series of links or branches.

A. Nodes

There are two type of nodes used in defining a structure graph: active nodes, and terminal points.

An active node is represented in a structure graph by a circle with an indicated, ordered set of connection points, (1, 2, ... , n). For an example of an active node see Figure 1. The node illustrated has associated

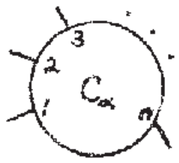


Figure 1

with it a name and a classification represented by:

$$C_\alpha$$

where C is the classification name or node type, and α is the proper name of the particular node illustrated. More will be said about the operation of active nodes and the interpretation of the classification name in section C.

Passive nodes or terminal points are represented in a structure graph by a shaded circle with a single indicated connection point (see Figure 2). These

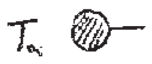


Figure 2

nodes are associated with a proper name and the special classification 'Terminal', represented by:

$$T_\alpha$$

where T is the classification name reserved for terminal points, and α is the proper name of the particular node illustrated.

B. Links

A link is represented in a structure graph as an undirected line and has associated with it a name, a value, and possibly two nodes. Links are used as the branches in a structure graph to connect a connection point of some node to a connection point of another node, or a link may connect two points on the same active node. The locations of a link's terminations are specified explicitly in its name. A link name will be represented as:

$$\ell(C_{a_i}, C'_{b_j})$$

where ℓ is the proper name of the particular link illustrated and C, a, i and C', b, j are, respectively, the classification, proper name and connection point of the associated nodes upon which the link terminates. The set (C_{a_i}, C'_{b_j}) is considered to be an unordered pair as no directiveness is to be assigned to a link and therefore $\ell(C_{a_i}, C'_{b_j})$ designates the same link as $\ell(C'_{b_j}, C_{a_i})$. Moreover either member of this set can be the free symbol (-) if a link extremity is not associated with a node. If both components of the set are specified as '-', e.g., $\ell(-, -)$, then the link is said to be <u>unconnected</u>.

Every link in a structure graph has associated with it a <u>value</u>. This value can be any symbol $s \in \tilde{V}$, where $\tilde{V}$ is the set defined as the alphabet of the structure graph. As will be explained in the next section, the value associated with a link can be set as an initial condition and then changed by any of the connected active nodes. To determine the value associated with a link at any given time we will define the value function $\nu(\ )$, which when supplied with the name of a link will yield the current value associated with that link.

C. Structure Graphs

A structure graph is a set of active nodes and/or terminal points interconnected by links. The naming conventions for links and their associated nodes as described above are assumed to be followed.

Every active node type used in a structure graph must have associated with it a set of transformations or productions represented by:

$$\begin{aligned} \bar{C}: \quad & \bar{s}^1 \rightarrow \bar{t}^1 \\ & \bar{s}^2 \rightarrow \bar{t}^2 \\ & \quad \vdots \\ & \bar{s}^m \rightarrow \bar{t}^m \end{aligned}$$

where, if C nodes have n connection points, $\bar{s}^i$ is an ordered n-tuple

$$\bar{s}^i = (s_1^i,\ s_2^i,\ \ldots,\ s_n^i)$$

and similarly

$$\bar{t}^i = (t_1^i,\ t_2^i,\ \ldots,\ t_n^i)$$

for $s_j^i,\ t_k^i \in \bar{V} \cup \{-\}$

The local configuration $\bar{\rho}_\alpha$ associated with a node C_α having n connection points is defined as the vectors:

$$\bar{\rho}_\alpha = (\rho_1,\ \rho_2,\ \ldots,\ \rho_n)$$

where each ρ_i is the value currently associated with the link connected to connection point i of node C_α.

Node C_α is said to be adjacent to node C' if in the structure graph there exists a link $\ell\ (C_\alpha, C'_\beta)$.

Two vectors $\bar{\rho}$ and $\bar{\eta}$ are said to be equivalent, $\bar{\rho} \equiv \bar{\eta}$, iff

1)both are vectors of the same order, n, and

2) $\forall_i\ 1 \leq i \leq n \quad \forall \rho_i\ \forall \eta_i \quad (\rho_i \neq - \cap\ \eta_i \neq -) \Rightarrow (\rho_i = \eta_i)$

If there is at least one production, $\bar{s}^i \to \bar{t}^i$, in the set of transformations associated with an active node C_α with local configuration $\bar{\rho}_\alpha$ such that $\bar{\rho}_\alpha \equiv \bar{s}^i$ then the node C_α is said to be applicable.

If an applicable node C_α applys itself to its local configuration, it replaces the value set comprising that local configuration with a new set as specified by the applicable production, i.e., if $\bar{\rho}_\alpha \equiv \bar{s}^i$ and $(\bar{s}^i \to \bar{t}^i) \varepsilon\ \bar{C}$ then the values of links connected to C_α are replaced so that $\bar{\rho}_\alpha \equiv \bar{t}^i$. *

A structure graph is executed as follows:

1) An initial system configuration is established, i.e., an initial value is associated with every link in the graph.

2) Any time a node is applicable, it may apply itself, revising local configurations in the process. Because of this rule the active nodes may be considered as proceeding asynchronously with respect to one another.

*If t^i_j = -, then the value of the link connected to C_{α_j} is left unaltered.

The progress of a structure graph in execution will be referred to as its history. The history of a graph at time t will be defined as the set of all of its link histories. A link history is defined as the ordered sequence of different values associated with that link from initial conditions until time t.

It should be noted now that the terminal classification, T, has no associated set of transformations, and therefore since it can alter no link value, a terminal point may be thought of as a passive or inactive node type. With only a single connection point, a terminal point may be associated with only a single link. Those links which terminate on at least one terminal point are referred to as terminal links, and the link histories of these links represent, by convention, the only information that can be exchanged between the given structure graph and its environment.

In accord with the definitions given in the first part of this paper, a structure graph is said to be functional or deterministic if its history is uniquely determined by the initial conditions. On the other hand, the graph is said to be output functional or terminal deterministic if the link histories of all terminal links are uniquely determined by the initial conditions of the graph.

D. Well-Formed Strucure Graphs

Structure graphs as just defined provide a powerful tool for describing discrete computational structures. However not all such graphs are output functional, and, in fact, if two adjacent nodes can be applicable at the same time during execution, then the graph history may be ambiguous. For these, among other reasons, it is advantageous to define a well-formed structure graph, wfsg, as:

1) wfsg's satisfy the rules of naming described above, and in addition all links, nodes, and classifications have unique names. Therefore, given the set of classifications and associated transformations in addition to the set of nodes and links, the structure graph is determined uniquely.

2) wfsg's have unambiguous sets of transformations associated with each node type. This means that for a given local configuration at most one production in the classification may apply.

3) wfsg's are conflict-free. A structure graph is <u>conflict-free</u> if for any pair of adjacent nodes that can become simultaneously applicable, the value associated with any connecting link is altered by neither application.

4) wfsg's are transformation lossless. A structure graph is said to be <u>transformation lossless</u> only if the application of any applicable node is never able to prevent the application of any other simultaneously applicable node, i.e., an applicable node is able to lose its applicability only through its own action.

<u>Theorem</u>: A wfsg is completely functional.

Proof: For sake of simplicity, this proof will demonstrate how the execution of any whsg can be simulated by an appropriately defined MCM (see ref. 7) and therefore Van Horn's result can be used to prove functionality directly.

Let us suppose that we are given a wfsg, G, and that we wish to build a simulation model of G by defining a corresponding MCM, M. For every active node C_α, in G, there will be a corresponding clerk cell, C_α, in M. For every link, $\ell(C_{\alpha_i}, C'_{\beta_j})$, there will be a corresponding value cell $\ell(C_{\alpha_i}, C'_{\beta_j})$ in M. Since in G, active nodes can alter the values of associated links, the clerk cells of M will correspondingly be able to alter the values stored in associated value cells. Since G is well-formed, no adjacent nodes will ever try to alter the value of a connecting link simultaneously, and therefore in M no two clerk cells with access to the same value cell will every need write capability for that cell at the same time. Folloing this reasoning, every action node can be modeled by a clerk cell with the following transaction table:

```
$C_\alpha$:  get  $\ell(C_{\alpha_1}, *)$                    replace with next transaction
      get  $\ell(C_{\alpha_2}, *)$                    replace with next transaction
             ⋮
      get  $\ell(C_{\alpha_n}, *)$                    replace with next transaction
    { calculation -                  }
    { recognition of local           }   replace with first transaction of put set $\bar{\rho}$
    { configuration $\bar{\rho}$                 }

        { put  $v_i$, $\ell(C_{\alpha_i}, *)$            replace with next transaction
        {          ⋮
        { put  $v_j$, $\ell(C_{\alpha_j}, *)$            replace with next transaction
        { send  $\ell(C_{\alpha_p}, *)$, *          replace with next transaction
put set {          ⋮
  $\bar{\rho}$     { send  $\ell(C_{\alpha_q}, *)$, *          replace with next transaction
        { done  $\ell(C_{\alpha_p}, *)$             replace with next transaction
        {          ⋮
        { done  $\ell(C_{\alpha_q}, *)$             replace with first trancaction

put set { same form  as put set $\bar{\rho}$ but with implementation of production
  $\bar{\eta}$     { $\bar{\eta} \to \bar{\eta}$.
        {          ⋮
          etc. with one put set for every production in the classification $\bar{C}$
```

* is used to represent the name of the "adjacent" clerk cell (if any)

The simulation of G is interpreted as follows: The initial configuration of G is stored into the appropriate value cells of A. All clerk cells are given read-write capability for themselves and are loaded with the first instruction in their transaction tables. Each clerk cell also receives read-capability for every value cell (corresponding to a connected link) whose contents would not restrict the corresponding node from being applicable. Because the modelled graph is well-formed, every clerk cell corresponding to an initially applicable node will have all the necessary read capabilities and, moreover, will have read-write capability for those cells whose values will be altered. When clerk cell C_α is loaded with the calculation phase of its transaction table, this corresponds to active node C_α becoming applicable. The "puts" of the transaction table alter the contents of the corresponding value cells in the same way as would the application of the corresponding active node alter the contents of the associated links. Of that set of value cells whose contents were altered, send's of read capability are made to "adjacent" clerk cells for that subset of value cells whose altered contents prevent further applicability of C_α. C_α then does a series of done's for this same subset of associated value cells.

With a little thought, it can be seen that an MCM such as the one prepared can simulate any vfsg in the sense that the history of the graph G in execution will be identical, where defined, to the history of the sequence of values stored in the respective value cells of M. Since the MCM found is properly constructed in Van Horn's sense, we can use Van Horn's result directly and say that the history of the cells in M (and therefore the history of the links in G) is uniquely determined by the initial conditions. Therefore M (and G) is completely functional. QED

It could be noted here that structure graphs offer two distinct advantages over MCM's as a parallel process computational model. First of all, structure graphs are quite free in format and structure allowing them to be used more [illegible] in modelling a wider range of computational structures than the MCM. In addition, whereas the MCM (like the Rodriquez program graph) requires global [illegible] on operation to guarantee functionality, the strictly conflict-free [illegible] been shown to be functional if only a local condition on adjacent node [illegible] is satisfied.

[illegible] now established a local graph constraint (wherein only the production [illegible] active nodes are restricted), we have at our disposal a local [illegible] sufficient to guarantee complete functionality. One might now begin to [illegible] there might not exist a necessary and sufficient local constraint. The [illegible] a necessary local condition for complete determinacy cannot exist [illegible] with specified initial conditions. Before attempting to substantiate [illegible] a few definitions are in order.

A node α is said to be a <u>distance D</u> from node β in a structure graph if the shortest path from α to β in the graph consists of a chain of D links, i.e., $\exists$ {set of D-1 nodes named $i_1, i_2, \ldots, i_{D-1}$ } $\ni$ the condition {α <u>adjacent to</u> i_1 & i_1 <u>adjacent to</u> i_2 & ... & i_{D-1} <u>adjacent to</u> β} is satisfied and D-1 is the smallest number of nodes for which the condition can be satisfied.

A <u>D-Distance determinism test</u> is defined as being a set of constraints on structure graph definition which, for each active node in the graph, restricts the allowable production sets for that node as the function of that part of the structure graph including nodes a distance D or less from the given node, for some fixed number D. When satisfied, the test constraints guarantee the complete functionality of the structure graph.

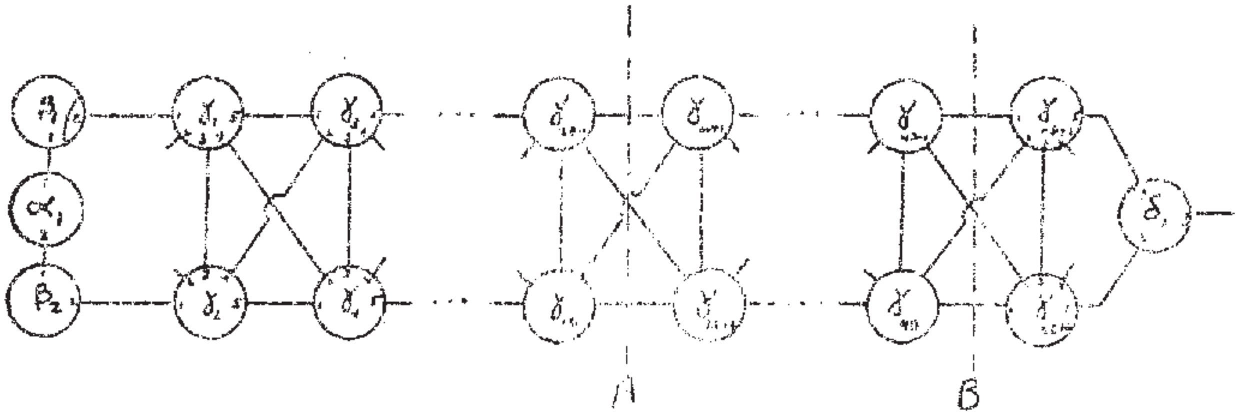
Theorem: There can exist no general D-distance determinism test such that a structure graph with specified initial conditions is guaranteed to be completely functional if and only if the test is satisfied.

Proof: The conjecture will be proven by assuming that such a test exists with some fixed value for D and then exhibiting a structure graph for which the test fails. Having assumed the existence of such a test, the structure graph of figure 3 is examined. Let the test now be applied to each node of the network. In particular notice that test applied to the nodes which lie to the left of line A, must yield a consistent result whether production set $\bar{\alpha}$ or $\bar{\alpha}'$ were used, since in both cases the structure graph to the left of the dashed line, B, is completely deterministic and equivalent* for the given initial conditions. Now realize that if the test indicates the entire graph to be completely deterministic, production set $\bar{\alpha}$ may be used in which

*Equivalent in the sense that the sequence of values for every link will be identical in both cases.

case the graph is actually non-deterministic because the final value of link $\delta(\theta_{i_j}, -)$ is determined by the relative timing of nodes β_1 and β_2. On the other hand, if the determinism test had indicated the graph to be non-deterministic, production set $\tilde{\alpha}'$ could be used, in which case, the structure graph would instead be completely deterministic. It follows that a D-local determinism test must be unable to determine the functionality of such a structure graph

QED



<u>Initial conditions</u>: Every link initially has value of '0' associated with it.

<u>Production sets</u>:

$\tilde{\alpha}$: 00 → 11

$\hat{\alpha}'$: 00 → 10

20 → 21

$\tilde{\beta}$: 10 → 21

$\tilde{\gamma}$: 100-0 → -11-1

10110 → -1--1

$\tilde{\delta}$: 100 → --1

010 → --2

<u>Operation</u>: Depending on whether link $\ell(\beta_{1_2}, \gamma_{1_1})$ or link $\ell(\beta_{2_2}, \gamma_{2})$ attains a value of 1 first, the structure of γ-nodes guarantees respectively that link $\ell(\gamma_{4D+2_5}, \delta_{1_2})$ or link $\ell(\gamma_{4D+2_5}, \delta_{1_2})$ will attain a value of 1 first.

Figure 3

IV Problems for Further Study

The preceeding section demonstrated how the complete functionality of a structure graph could be guaranteed by satisfaction of only a simple local constraint. In the first section it was discussed as to why a output functional design might be preferred. Of course, if a structure graph is completely functional it is trivially output functional. The problem is to loosen the restrictions and devise a set of reasonably-local design constraints which guarantee output functionality.

As an example of a structure graph which displays output functionality, the reader is referred to the Appendix where a model of an asynchronous, multi-process linear-bounded automaton is illustrated. Although it will not be proved in this paper, it can be shown that the graph illustrated is output function but not completely functional. Through investigation of such structures as this, it is hoped that constraints upon model formulation can be developed which assure output functionality of the structure.

As is shown in part II of the Appendix, additional, redundant nodes can be added to the structure graph of part I. These nodes allow more than one process interrogate a ring node modeling identical state, symbol pairs. More important for our purposes, is the fact that the terminal behavior of the structure has been unaltered, and for this reason the two structures can be thought of as equivalent with respect to terminal behavior. Further work will be directed towards establishing a theory of equivalence for structure graphs which would enable us determine the equivalence of such structures as in the example given. Even more

Bibliography

(1) Gray, H.J. and Sims, J.C. Jr., "Design Criteria for Autosynchronous Circuits", Proc. Eastern Joint Computer Conf., Dec., 1958, pp 94-99.

(2) Huffman, D.A., "The Synthesis of Sequential Switching Circuits," Sequential Machines: Selected Papers, E.F. Moore (Ed), Addison-Wesley Publishing Co. Inc., 1964, pp 3-62.

(3) McNaughton, R., Badly Timed Elements and Well Timed Nets, Technical Report No. 65-02 at the Moore School of Electrical Engineering of the University of Pennsylvania, June 1964.

(4) Muller, D., "Asynchronous Logics and Applications to Information Processing," Lockheed Symposium paper.

(5) Petri, C.A., Kommunikation mit Automaten, Schriften des Reinisch-West falischen Inst. Instrumentelle Math, and der Universitat Bonn, Nr. 2, Bonn, 1962

(6) Rodriquez, J.E., "Analysis and Transformation of Computational Processes," Project MAC, M.I.T., MAC Memorandum MAC-M-301, March 1966.

(7) Van Horn, E.C., "Computer Design for Asynchronously Reproducible Multiprocessin Ph.D. dissertation, Mass. Inst. of Technology, August 1966.

Appendix

Part I

A linear-bounded automaton, LBA, is defined to be a Turing machine under the restriction that the length of the tape presented to the machine is no longer than a fixed multiple of the length of the input string (see figure 4). Such restricted Turing machines have received attention because of their relation to context-free languages, non-contracting context-sensitive languages, and to the field of structural linguistics in general.

The LBA consists of a finite-state control unit to which is coupled a reading head. The head scans a single, finite length tape, which can be shifted in either direction. Input data are presented to the LBA in the form of a pattern of symbols written on the tape. At each step of the computation, the LBA may replace the currently scanned symbol and then shift the tape or halt. For the purposes of this presentation, the output of the LBA will be defined as the history of the tape contents.

The LBA is defined by the structure of its control unit. The LBA is always assumed to be started in a specified starting state, q_0. The control unit is defined in terms of a finite set of 2-tuple to 3-tuple transformations as follows:

$$(q_i, s_k) \rightarrow (q_j, s_\ell, d)$$

where $q_i, q_j \in \hat{Q}$ defined as the state set

$s_k, s_\ell \in \hat{S}$ defined as the alphabet

$d \in \{L, R, H\}$

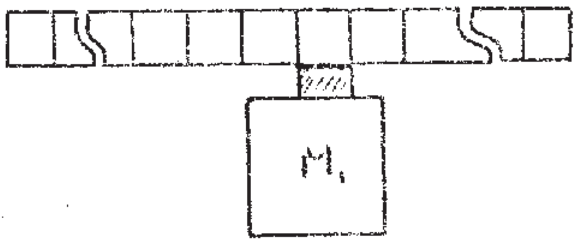


Figure 4

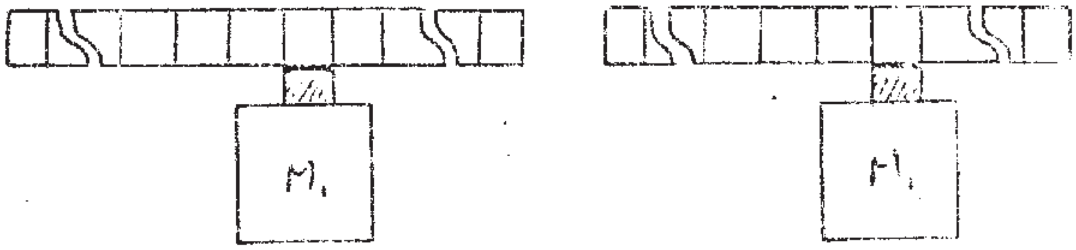


Figure 5

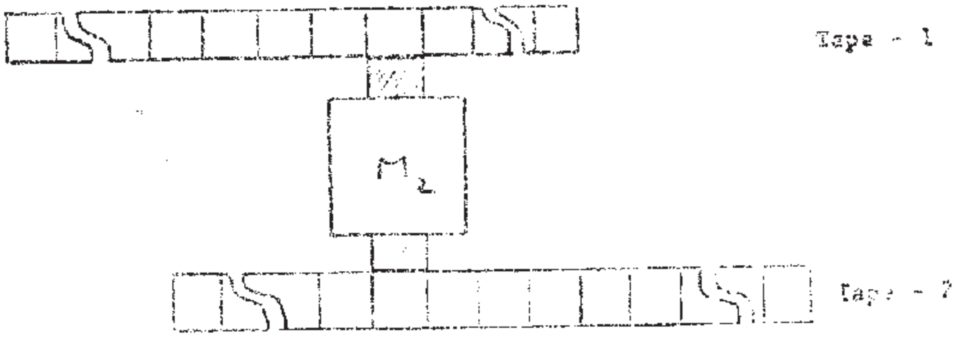
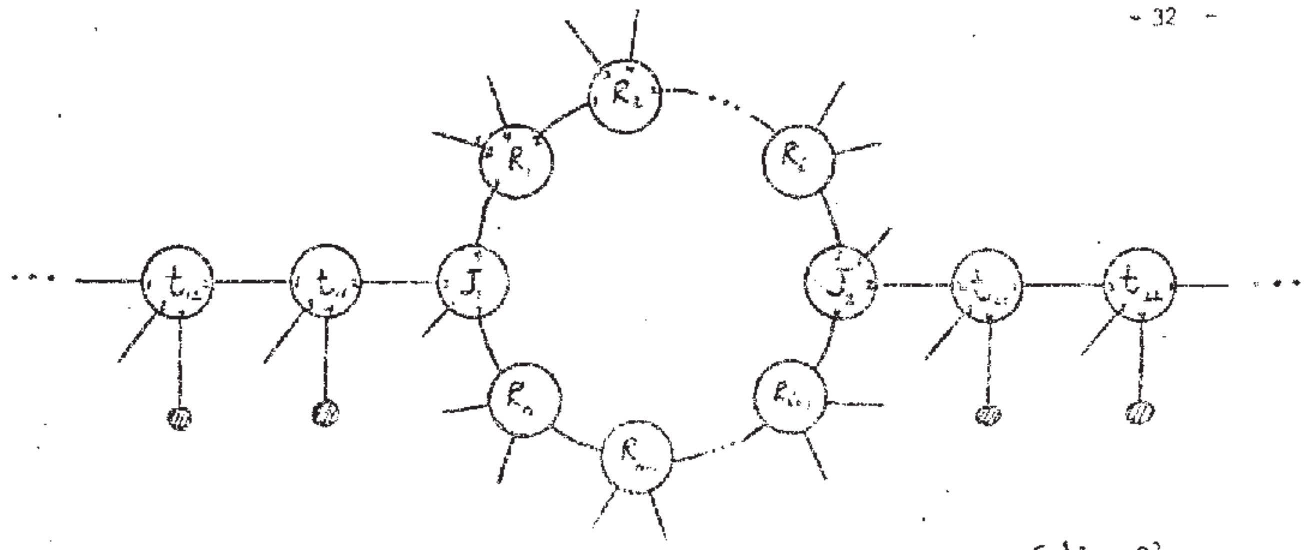
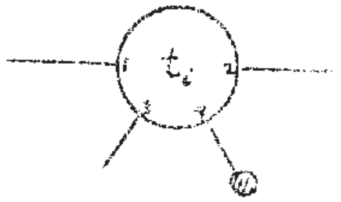


Figure 6

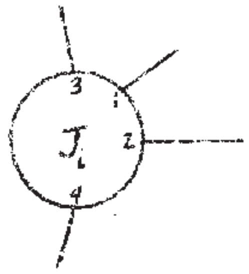


$x \in \{1\ ,\ 2\}$
$d \in \{L,\ R,\ H\}$
$S_i\ ,\ S_j \in \bar{S}$
$q,\ q_i,\ q_j,\ q_m \in \bar{Q}$

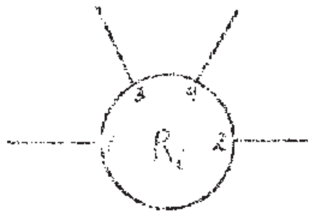


	$\underline{1}$	$\underline{2}$	$\underline{3}$	$\underline{4}$		$\underline{1}$	$\underline{2}$	$\underline{3}$	$\underline{4}$
$\bar{t}$:	0	(x,S_i,d,q)	0	-	$\rightarrow$	(x,S_i,d,q)	0	0	-
	0	(x,S_i,R,q)	1	S_j	$\rightarrow$	0	$(x,0,R,q)$	0	S_i
	$(x,0,R,q)$	0	0	S_i	$\rightarrow$	0	$(x,S_i,0,q)$	1	S_i
	0	(x,S_i,L,q)	1	S_j	$\rightarrow$	$(x,0,L,q)$	0	0	S_i
	0	$(x,0,L,q)$	0	S_i	$\rightarrow$	0	$(x,S_i,0,q)$	1	S_i
	$(x,S_i,0,q)$	0	-	-	$\rightarrow$	0	$(x,S_i,0,q)$	-	-

Figure 7c



	$\underline{1}$	$\underline{2}$	$\underline{3}$	$\underline{4}$		$\underline{1}$	$\underline{2}$	$\underline{3}$	$\underline{4}$
$\tilde{J}$:	1	-	(2,-,0,-)	0	$\rightarrow$	-	-	0	(2,-,0,-)
	2	-	(1,-,0,-)	0	$\rightarrow$	-	-	0	(1,-,0,-)
	1	-	(2,-,d,-)	0	$\rightarrow$	-	-	0	(2,-,d,-)
	2	-	(1,-,d,-)	0	$\rightarrow$	-	-	0	(1,-,d,-)
	$\bar{x}$	(x,-,0,-)	-	0	$\rightarrow$	-	0	-	(x,-,0,-)
	x	0	(x,S_i,d,q)	-	$\rightarrow$	-	(x,S_i,d,q)	0	-



	$\underline{1}$	$\underline{2}$	$\underline{3}$	$\underline{4}$		$\underline{1}$	$\underline{2}$	$\underline{3}$	$\underline{4}$
$\tilde{R}$:	$(x,S_i,0,q_j)$	0	(S_i,q_j)	-	$\rightarrow$	0	$(x,S_i,0,q_i)$	-	-
	$(x,S_i,0,q_i)$	0	(S_i,q_j)	-	$\rightarrow$	0	$(x,S_i,0,q_i)$	-	-
	$(x,S_i,0,q_i)$	0	(S_i,q_i)	-	$\rightarrow$	0	$(x,S_i,0,q_i)$	-	-
	$(x,S_i,0,q_j)$	0	(S_i,q_k)	(S_i,q_m,d)	$\rightarrow$	0	(x,S_i,d,q_m)	-	-
	(x,S_i,d,q_k)	0	-	-	$\rightarrow$	0	(x,S_i,d,q_k)	-	-

Figure 7b

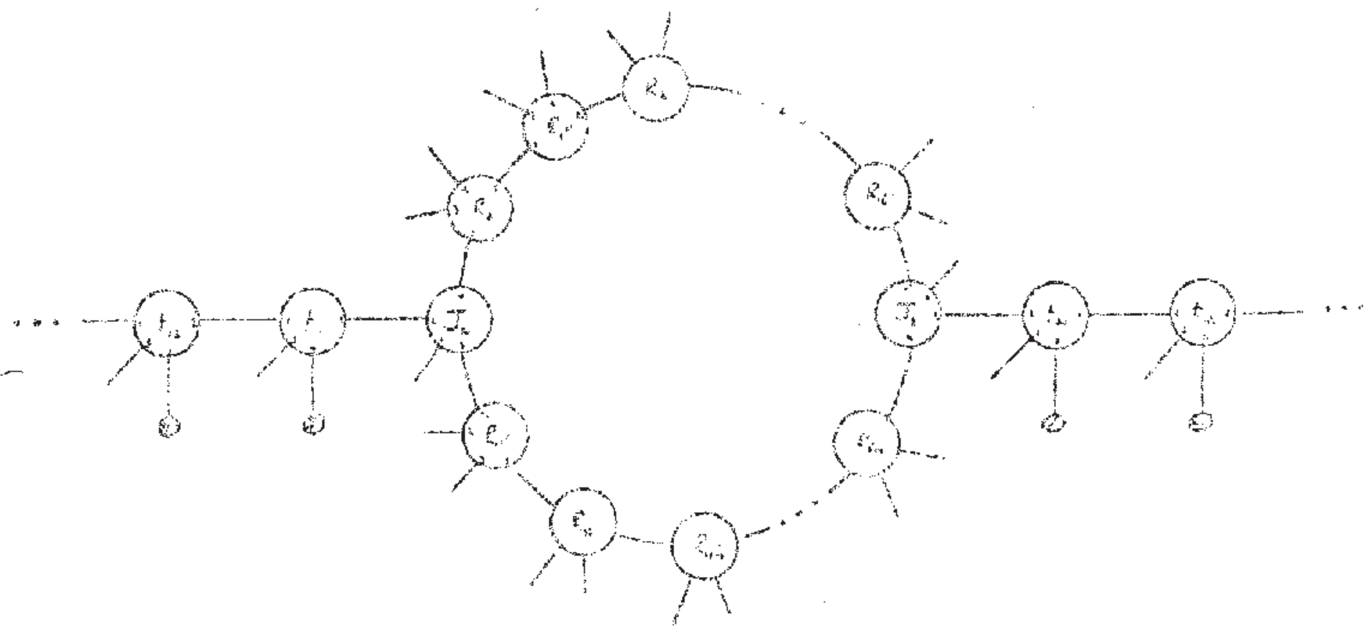


Figure 8