T H E   U N I V E R S I T Y   O F   M I C H I G A N

COLLEGE OF LITERATURE, SCIENCE, AND THE ARTS

Department of Philosophy

Technical Report

COMPUTATION, BEHAVIOR, AND STRUCTURE IN FIXED AND GROWING AUTOMATA

Arthur W. Burks

under contract with:

DEPARTMENT OF THE NAVY
OFFICE OF NAVAL RESEARCH
CONTRACT NO. 1224(21)
WASHINGTON, D. C.

October 1959

engn

UMR Ø9Ø8

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION[1]

This paper is concerned mainly with deterministic automata. Most of this symposium has been devoted to probabilistic automata, so a word of explanation is in order.

It is perhaps true that deterministic theories are inadequate to account fully for growth, learning, mutation, evolution, and the actual operation of digital computers, and hence that a probabilistic theory is required for a full explanation of all these phenomena. Nevertheless, a discussion of deterministic automata is not out of place in a symposium on self-organizing systems. We cannot fully understand probabilistic automata until we know the limits of deterministic ones, for only then will we know what can be done by probabilistic automata that cannot be done by deterministic machines. Moreover, a probabilistic automaton may be regarded as a deterministic automaton to which has been added a probability measure governing the transitions between states. Just as both deterministic and probabilistic theories have been important in physical science, we may expect both kinds to be fruitful in the study of automata.

A complete theory of self-organizing systems must show exactly what role determinism plays in these systems. I accept von Neumann's suggestion that automata theory must include probabilistic as well as deterministic logics,[2] but I also believe with him that deterministic analyses of such phenomena as self-reproduction are of interest (see Section 4 below).

Various species of deterministic automata have been studied. Turing machines have been intensively investigated, both directly and through the medium of recursive functions.[3] Fixed (finite) automata have received much attention of late, and von Neumann worked some with self-reproducing automata. Little has been done, however, to relate these separate inquiries. In the present paper, I will make some suggestions directed toward a unified theory of deterministic automata.

1. This is a revised version of a paper presented May 6, 1959, at the Interdisciplinary Conference on Self-Organizing Systems, Chicago, co-sponsored by the Information Systems Branch of the Office of Naval Research and the Armour Institute of the Illinois Institute of Technology. The writing of this report was partly supported by the U. S. Signal Corps through Project MICHIGAN (Contract No. DA-36-039-sc-52654).
John Holland and Jesse Wright have made many helpful suggestions.
2. John von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," pp. 43-98 in Automata Studies (edited by C. E. Shannon and J. McCarthy), Princeton, 1956.
3. See, for example, S. C. Kleene, Introduction to Metamathematics, New York, 1952, and Martin Davis, Computability and Unsolvability, New York, 1958.

2. STRUCTURE, BEHAVIOR, AND COMPUTATION IN FIXED AUTOMATA AND GENERALIZED TURING MACHINES

I will begin with fixed automata and Turing machines and discuss later self-reproducing machines and automata generally. A fixed automaton[4] consists of a finite number of switch and delay elements interconnected so as to make a pattern or structure which is the same at every moment of time.[5,6] Figure 2.1 represents a very simple fixed automaton which deletes every other pulse (1) that is fed into the input. A digital computer with a finite tape is, of course, a fixed automaton. (A digital computer with an indefinitely expandable tape is a generalized Turing machine in the sense defined below.)

The structure of a fixed automaton is to be distinguished from its behavior. The _structure_ consists in the fixed pattern of interconnections of the elements. The structures of two automata are the same if the nodes (i.e., the junctions of the wires of the elements) of these automata can be placed in one-one correspondence so that corresponding nodes are nodes of identical elements.[7] Thus if we change the conjunction of Fig. 2.1 into a disjunction we obtain a different structure.

The behavior of an automaton consists, roughly speaking, of the relation of input stimuli to output responses. A more precise definition of behavior may be given in terms of temporal sequences of states, where the relevant times are the nonnegative integers 0,1,2,3.... Let us call an infinite temporal sequence of input states to an automaton an _input history_ and an infinite temporal sequence of output states an _output history_. A _behavior pair_ of an automaton consists of an input history paired with the output history which is produced when that input history is impressed on the inputs of the automaton. A behavior pair for the automaton of Fig. 2.1 is shown in the lower part of the figure. The _behavior_ of a fixed automaton is the set of all behavior pairs of that automaton.

4. Strictly speaking, these automata should be called _finite_ fixed automata, since there are also infinite fixed automata (fixed structures composed of infinitely many switch and delay elements). But we will say little about infinite fixed automata (though some use will be made of them in Section 5) and hence for convenience we will use "fixed automaton" to mean _finite_ fixed automaton and use "infinite fixed automaton" for the infinite case.
5. A. W. Burks and Jesse Wright, "Theory of Logical Nets," _Proceedings of the Institute of Radio Engineers_, _41_, 1357-1365 (1953).
6. A. W. Burks and Hao Wang, "The Logic of Automata," _Journal of the Association for Computing Machinery_, _4_, 193-218, 279-297 (1957).
7. Elements are taken to include their input and output wires.

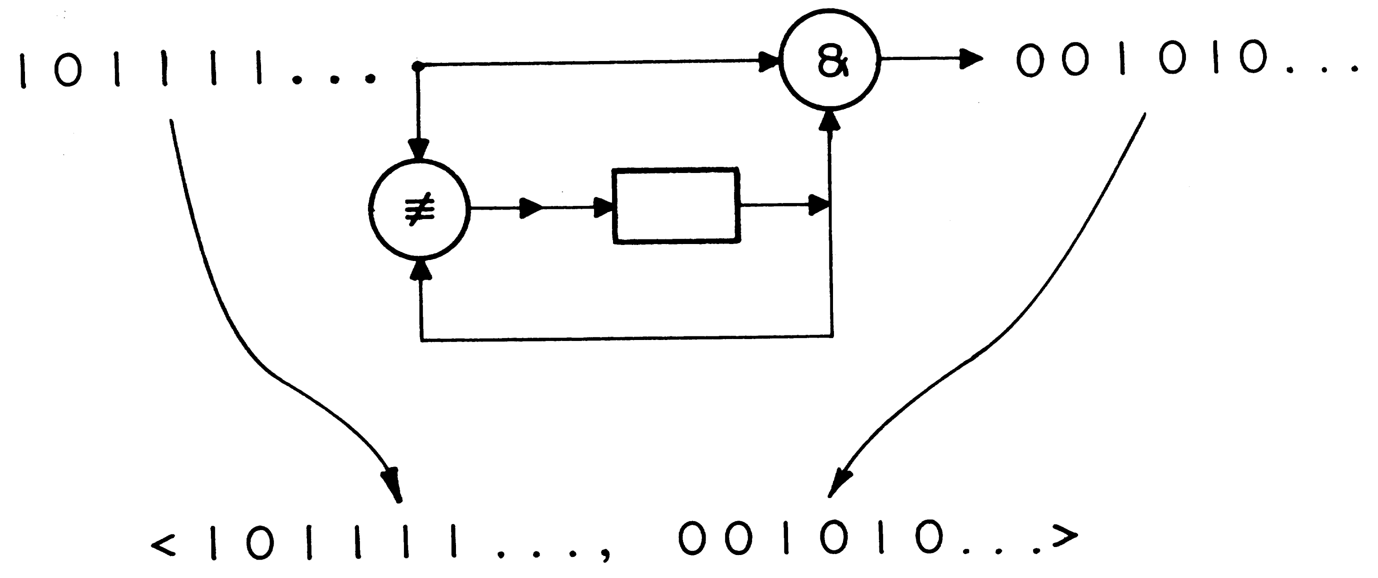

Fig. 2.1. Behavior of fixed automaton.

We turn next to Turing machines. As Turing originally conceived them, each is essentially a fixed automaton connected to a single infinite tape. The fixed automaton scans one square of the tape at any one time, and can move the tape (or move relatively to the tape) one square at a time. We will generalize Turing's concept in a number of ways to obtain what we will call a <u>generalized Turing machine</u>. To relate Turing machines to the fixed automata previously discussed, we will allow the fixed part of the machine to have inputs and outputs, as in Fig. 2.2; as a consequence, the earlier concept of <u>behavior</u> applies directly to Turing machines. Next we will allow any finite number of tapes. Finally, we will regard a tape at any given moment of time as a finite automaton and construe a change in tape length as a change in this automaton; thus for us the tape is finite at each moment of time.[8] A tape "square" can be synthesized from switch and delay elements in a number of ways, and squares can be added or subtracted to represent the growing or shrinking of the tape.[9] Hence the definition of structural equivalence given above can be extended to take account of tapes. While a generalized Turing machine has a given structure at any moment of time, its structure changes through time in a very simple fashion and under the control of the computation going on in the machine. A digital computer together with an operator who can indefinitely expand the magnetic tapes is a generalized Turing machine.

Computation requires a problem and an answer, and the answer must be presented somewhere. Turing used alternate squares of the tape for this purpose. We will stipulate that the answer of a generalized Turing machine is to appear on the outputs. We will not, however, take the output history as the answer, for a reason to be explained below. Instead, we will distinguish a <u>control output wire</u> from the remainder of the output wires, which will be called <u>answer output wires</u>. A "1" on the control output wire signifies that the state of the answer output wires at that time is part of the answer, while a "0" on the control output wire means that the state of the answer output wires at that time is to be ignored. The subsequence of the states of the answer output wires so selected is the <u>computed output sequence</u>. The sequence 3561 is the first part of the computed output sequence of Fig. 2.2. A pair consisting of an input history and the resultant computed output sequence is a <u>computation pair</u>. The <u>computation</u> of an automaton is the set of all computation pairs of that automaton.

8. There are other ways of treating Turing machines. For example, one can consider an infinitely long tape with a fixed automaton moving along it. Note, however, that though there are an infinite number of squares on this tape, at each moment of time only a finite number of them have ever changed state; it is for this reason that we can treat the tape as finite at each moment of time.
9. The synthesis of a tape "square" storing one bit (a "0" or a "1") is described in Section 4 below and also in Section 6 of "The Logic of Fixed and Growing Automata," presented at the Harvard Symposium on the Theory of Switching, 1957, to be published in the <u>Proceedings</u> of the Symposium, edited by H. Aiken. One can construct tape squares capable of storing any of n characters in a similar way.

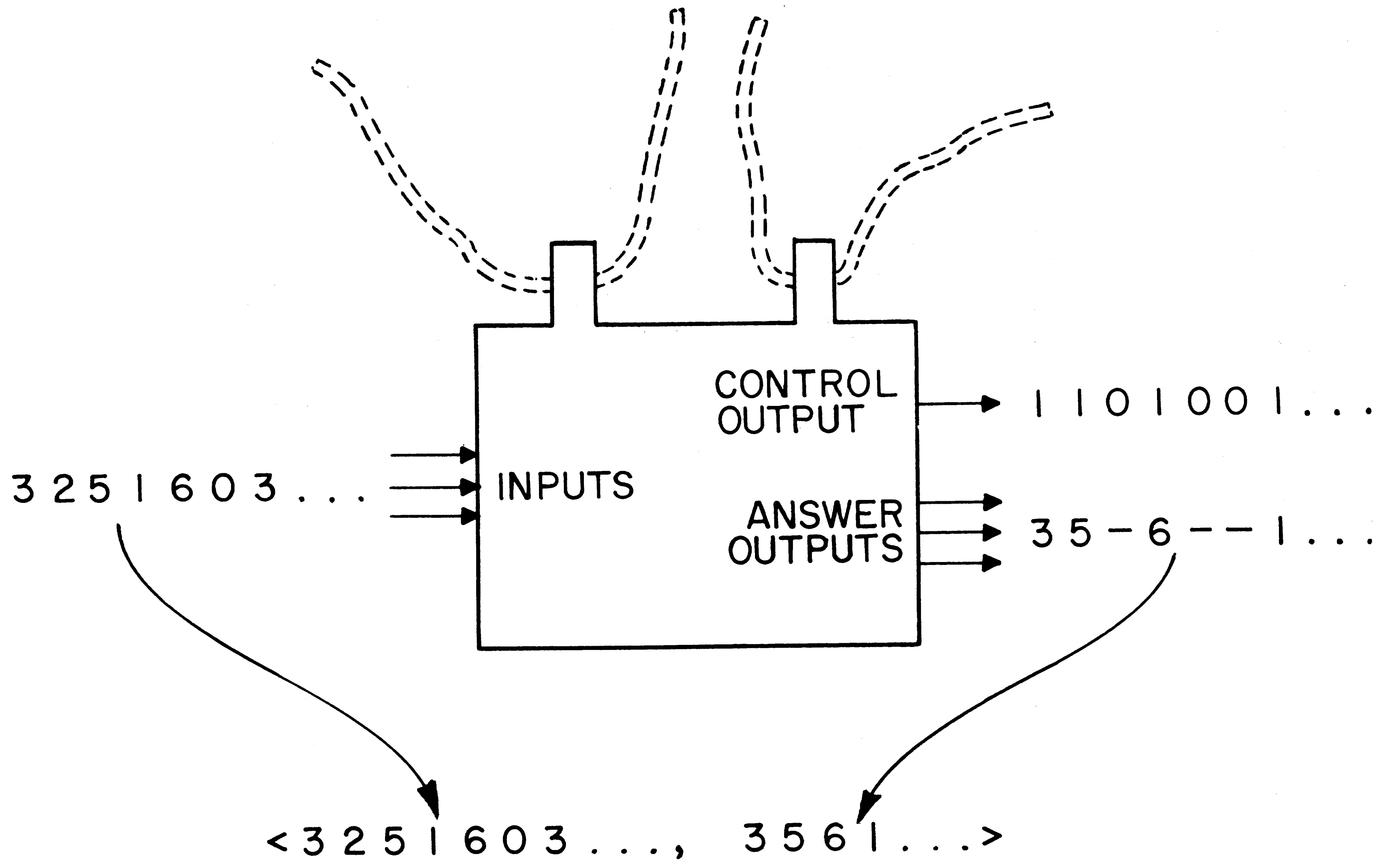

Fig. 2.2. Computation of generalized Turing machine.

There is a relational analogy between behavior and computation on the one hand and "real time" simulation and "nonreal time" simulation on the other hand. In real time simulation the answer is produced at a rate determined by the actual time taken by the process which the computer is simulating, while in a "nonreal time" simulation, the answer is produced at a rate independent of actual time. Similarly, the behavior of an automaton includes the outputs at every moment of time (including the control output), while the computation includes only outputs selected by the computer itself, and in general these answer states do not appear at a uniform rate.

A computed output sequence may be null, finite, or infinite. A computation is said to be <u>infinite</u> if every computed output sequence (i.e., the second element of each computation pair) is infinite, <u>finite</u> if every computed output sequence is finite or null, and <u>mixed</u> otherwise.

We can apply the concept of computation (and the related concepts just defined) to fixed automata by distinguishing a control output of a fixed automaton from the other (or answer) outputs. Hence all three concepts—structure, behavior, and computation—apply to both fixed automata and generalized Turing machines. These three concepts will also apply to the growing automata defined later (Section 4).

It is easy to show that the class of infinite computations of generalized Turing machines includes the class of behaviors of such machines as a proper part; the same holds true for fixed automata. Consider the behavior of an automaton A; this is the computation of an automaton A' made by taking the outputs of A as the answer outputs of A' and making the control output of A' always active (always in state 1). But there are infinite computations which are not deterministic in character and hence which are not the behavior of any deterministic automaton. An example is the computation I(0), I(1), I(2), ...; I(1), I(2), I(3), ... [where I(t) is the input state at t], which is realized by a very simple fixed automaton: the output state at t is the input state at t, and the control output node C has the behavior $C(0) = 0$, $C(t+1) = 1$ for all t. Hence (for either fixed automata or generalized Turing machines) the class of infinite computations is broader than but inclusive of the class of behaviors. For this reason we did not take the whole output history of a machine as the answer computed by this machine—to do this would be to identify behavior and computation and hence to restrict unduly the domain of the computable.

3. ANALYSIS AND SYNTHESIS OF AUTOMATA

Our discussions of automata in Sections 4 and 5 will presuppose some knowledge of fixed automata and generalized Turing machines, and so we will mention briefly some results about these special cases of automata and will note some of the unsolved problems. Since the concepts of behavior, structure, and computation apply to all automata (see Sections 4 and 5), many of the results and problems concerning fixed automata and generalized Turing machines which we will discuss in the present section may be extended to cover automata generally.

Much is known about the behavior of fixed automata. For example, each fixed automaton has a finite number of delay output states, which fact has important consequences for the behavior of these devices. In the case of a fixed automaton without inputs it means that the behavior of the automaton is periodic.[10] In the case of an automaton with inputs, the finitude of states means that the device can detect only "regular events,"[11] that if the number of delay states is known, then its behavior can be computed from the results of a finitely long behavioral test, and that there is a decision procedure for behavioral equivalence.[12,13] Since the notion of computation is a new one for fixed automata, little is known about this. The same argument that shows that the behavior of a fixed automaton without inputs is periodic shows that the computation of such an automaton is also periodic. There is an algorithm for deciding whether the computation of a fixed automaton (with inputs) is infinite, finite, or mixed,[14] but the problem of the existence of an algorithm for deciding whether two fixed automata have the same computation is unsolved.

A great deal is known about Turing computability and the closely related notions of algorithm, general recursiveness, and partial recursiveness. We do not have space to explain these concepts here,[15] but it is worth noting that these results can be expressed in terms of our notion of a generalized Turing machine, and they contain answers to most questions about the computation of such machines. As applied to generalized Turing machines, these results on recursiveness, algorithms, and computability need interpretation, however, because

10. Burks and Wright, _op_. _cit_., Theorem I. Note that in this case the behavior is a single output history.
11. I. M. Copi, C. C. Elgot, and J. B. Wright, "Realization of Events by Logical Nets," _Journal of the Association for Computing Machinery_, _5_, 181-196 (1958).
12. E. F. Moore, "Gedanken-Experiments on Sequential Machines," pp. 129-153 of _Automata Studies_, _op_. _cit_.
13. Burks and Wang, _op_. _cit_., sec. 2.2.
14. Unpublished results of the author and Jesse Wright.
15. See Kleene, _op_. _cit_., and Davis, _op_. _cit_.

there are many ways of relating a generalized Turing machine to these notions. Thus there are alternative methods of using a generalized Turing machine to compute a recursive function. For example, for a given nonrelative recursive function, one can design a machine without inputs whose computation will be an enumeration of the values of that function. Alternatively, one can design a generalized Turing machine with inputs so that whenever arguments (problems) are presented on the input, the computed functional values (answers) will appear as part of the computation. The distinction between finite or mixed computation on the one hand and infinite computation on the other (see Section 2) can be made to correspond to the distinction between partial recursiveness and general recursiveness.

We will relate a few of the things known about computability and recursiveness to our concept of a generalized Turing machine. Every computable number (in Turing's sense) can be produced as the computed output sequence of a generalized Turing machine with no inputs and one tape. The values of a function which is general recursive relative to a given function can be produced as the computation of a generalized Turing machine which receives the values of the given function as inputs. There is no algorithm for deciding whether the computation of an arbitrary generalized Turing machine is finite, mixed, or infinite, but there is an algorithm for deciding whether the structure of a machine can ever change. There is no algorithm for deciding whether two generalized Turing machines produce the same computation.[16]

Since the notion of behavior as applied to a Turing machine is new, there are many open questions in this area. For example: is the class of behaviors of generalized Turing machines with n+1 tapes larger than the class of behaviors for machines with n tapes? Is there an algorithm for deciding whether two generalized Turing machines have the same behavior?

Later we will discuss von Neumann's universal constructing machine, so Turing's universal computing machine is of particular interest here. Let us call a generalized Turing machine with no inputs, one tape, one control output node, and one answer output node a special purpose machine. Translated into our terms, Turing's result is that there is a one tape generalized Turing machine with inputs, called a universal Turing machine, with this property: for each special purpose machine M there is a finitely long program or input sequence S such that when S is supplied to the inputs of the universal Turing machine the computed output sequence of M and the universal Turing machine are identical; this is illustrated in Fig. 3.1. It should be noted that in general the universal Turing machine and the special purpose machine do not produce the same output histories, so that the equivalence between them has to do with computation and not behavior. It should be noted also that the finitely long input sequence S can be supplied to the universal Turing machine by connecting a cycle free sequence of delays without inputs to each of the inputs of the universal machine; the input to these delays is fed by a contradiction (an output that is always false), and the program is stored in the initial states of the delays. After

16. Unpublished result of Jesse Wright.

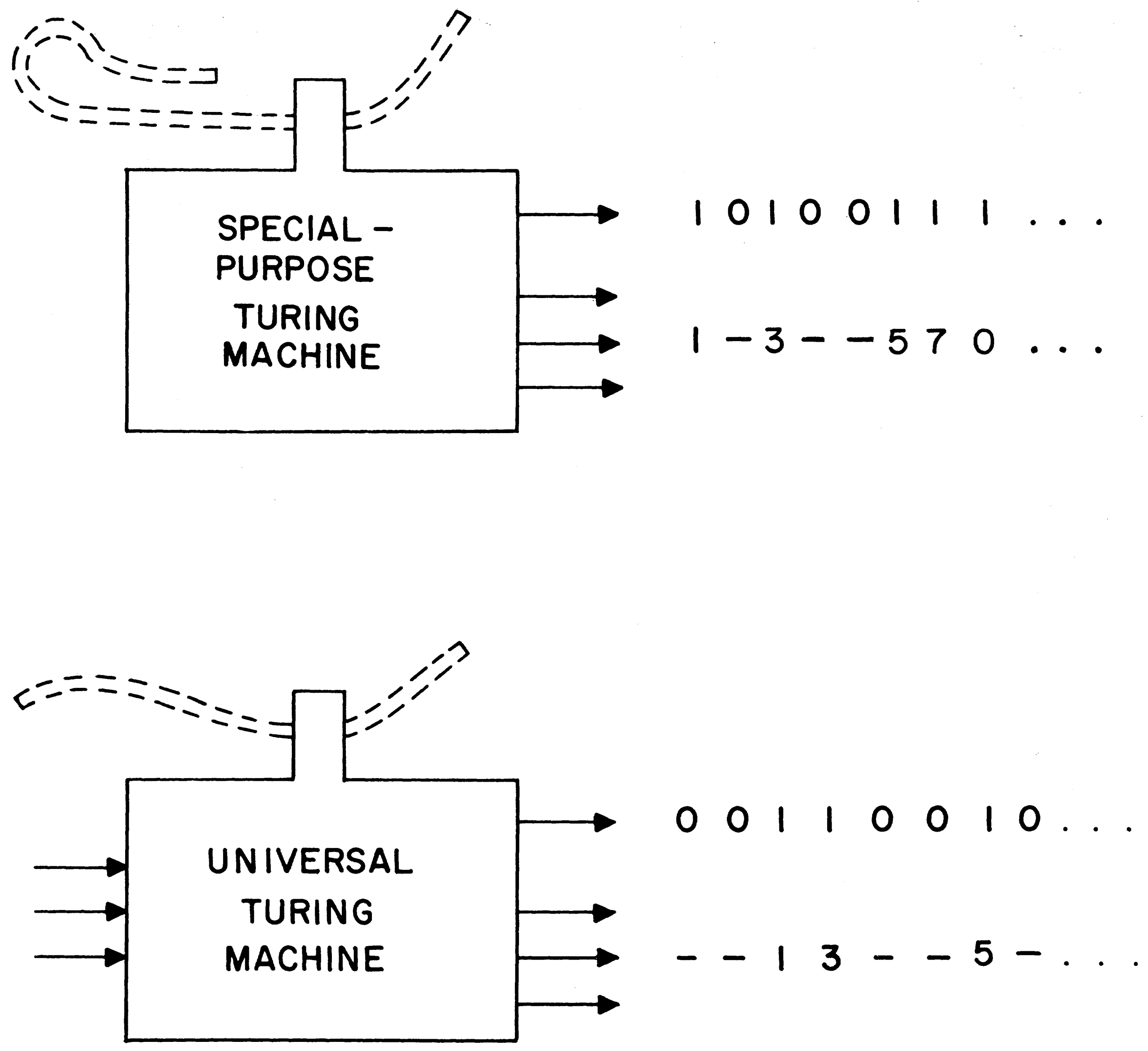

Fig. 3.1. Universal Turing machine.

a period of time equal to the length of this sequence of delays, the input to the universal machine is always zero.

It is very likely that a similar result holds when inputs are added to the special purpose machine and more than one tape are allowed; that is, for each set of generalized Turing machines with n inputs and m outputs, there is a universal Turing machine. Does an analogous result hold for fixed automata? Computer engineers often speak of trading time for hardware and of the choice between constructing ("wiring in") an operation and instructing (programming) that operation, and this might seem to imply the existence of a universal fixed automaton. But in fact there are no universal fixed automata. We will prove that there is no universal fixed automaton for the class of input-free fixed automata with m binary outputs. To prove this we first show that any fixed automaton A (with inputs) when supplied by a finite input sequence S produces a periodic output of period no greater than n, where n is the number of delay states of A. To say that S is finite is to say that there will come a time when the input states of A are always the same thereafter. The automaton A will then behave as a noninput device with n delay states, and so the computed output sequence of A is periodic with period n or less. Hence A cannot produce a computed output sequence of, for example, period 2n, and so A is not a universal machine. Thus there is no fixed automaton which is universal for the class of noninput fixed automata with m outputs, and consequently there is no fixed automaton for the class of fixed automata with n inputs and m outputs.

We will next mention some reduction problems. Does adding more tapes increase the class of computations produced by generalized Turing machines? A generalized Turing machine can have only one multi-channel reading and writing head per tape. In contrast, the growing automata introduced in the next section can have tapes with more than one multi-channel read-write head attached to them.[17] What effect, if any, does this have on the class of behaviors or the class of computations? In some cases one can use several single-headed tapes alternately to do the work of one many-headed tape. This can be done in the special case of an automaton which records information on two tapes and compares these tapes, using one head for recording and another head on the same tape for reading and comparing. To accomplish this result with one-head tapes, we use two such tapes for each of the two-head tapes on the original machine. At each moment one of these tapes is used to record the input information while the other is used for the comparison; when the latter tape is emptied, it is used for recording and the former tape is used for comparing. It is worth noting that we have here an abstract buffering and scheduling problem.

No discussion of automata at a symposium on self-organizing systems would be complete without reference to machines which synthesize other machines. This synthesis process can be conceived in many ways. One method which has been

17. This can be done by constructing an indefinite series of storage squares, and moving the reading heads along them. Motion of a reading "head" is accomplished by "destroying" the head and constructing another head in the place where it is wanted.

studied some of late[18-21] is illustrated in Fig. 3.2. Suppose we have a formal language L in which a human can conveniently and precisely state some behavioral conditions C which he would like an automaton to satisfy. We would like to know if there is a synthesis machine S of which we can prove mathematically the following result: when S is given any behavioral condition C, it will always produce either the design of some automaton A which satisfies this condition, or it will tell us that there is no such automaton.[22] The condition given at the top of Fig. 3.2 is that for all times other than zero the output of the desired automaton is to be active (in state 1) if the input has been active at some previous time. (That is, B is active at time t+1 if and only if there is a time x no larger than t at which A is active.) There are fixed automata satisfying this condition, and so if a synthesis machine for a language containing this condition exists, this machine would produce as output a representation of one of these automata.

These is a large area of research here because there are many problems of the kind just described. The synthesis process represented in Fig. 3.2 starts with a particular language L and results in the synthesis of a fixed automaton satisfying a condition on its behavior. There are many different languages L which should be investigated and it is of interest to synthesize generalized Turing machines as well as fixed automata. In both cases we can allow the condition to be a condition on the computation of the desired automaton as well as a condition on its behavior. For each specification of the the language L, the nature of the condition C, and the nature of the desired automaton A, the question arises: is there a synthesis machine and if so, what is its structure? In some cases there is a machine and in some cases there is not. It is important to realize that, if there is no synthesis machine of this kind, there is no method of using computers to design computers which is guaranteed to produce an answer in all cases. Hence the broad problem is this: to determine the theoretical limits of mechanizability of the synthesis process by deterministic machines of this kind.

---

18. J. R. Büchi, C. C. Elgot, and J. B. Wright, "The Non-Existence of Certain Algorithms of Finite Automata Theory," Abstract, Notices of the American Mathematical Society, 5, 98 (February, 1958).
19. A. Church, "Application of Recursive Arithmetic in the Theory of Computers and Automata," notes from summer session course in Advanced Theory of the Logical Design of Digital Computers, The University of Michigan, 1958.
20. C. C. Elgot and J. R. Büchi, "Decision Problems of Weak Second Order Arithmetics and Finite Automata, Part I," Abstract, Notices of the American Mathematical Society, 5, 834 (December, 1958).
21. C. C. Elgot, "Decision Problems of Weak Second Order Arithmetics and Finite Automata, Part II," Abstract, Notices of the American Mathematical Society, 6, 48 (February, 1959).
22. Other problems in this area are formulated by Büchi, Elgot, and Wright, op. cit. What we have called a synthesis problem they call a combined solvability and synthesis problem.

$$B(t+1) \equiv (\exists x)\left[(x \leq t) \;\&\; A(x)\right]$$

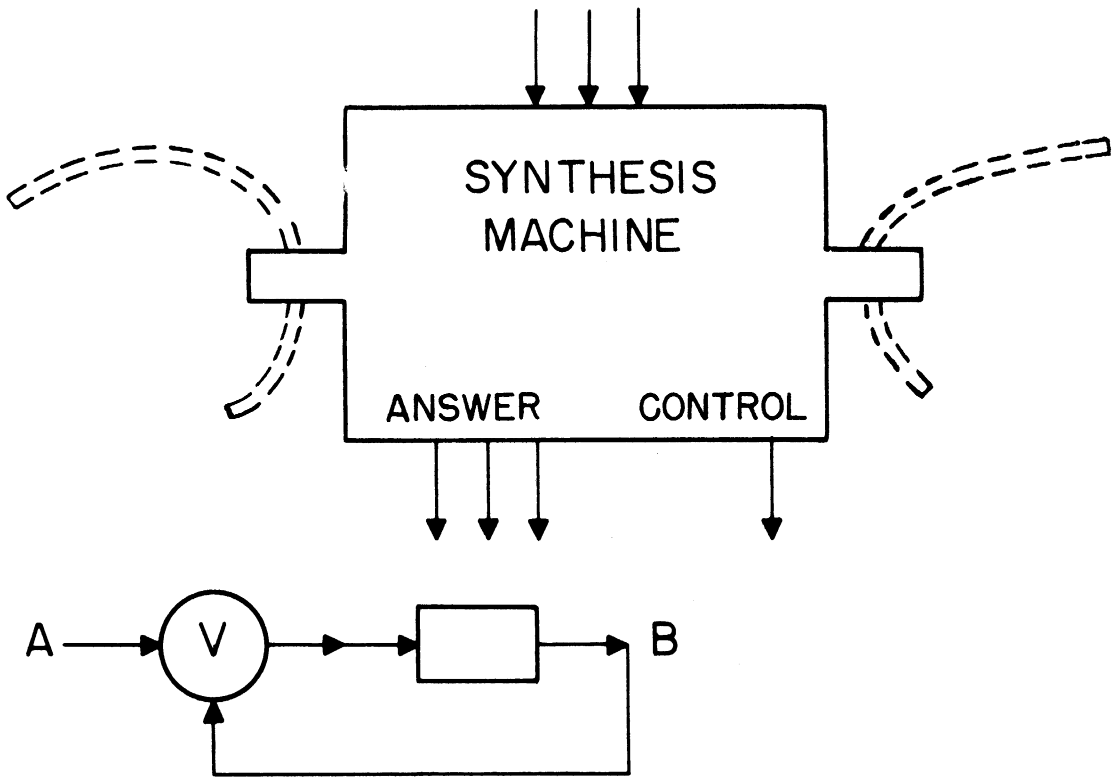

Fig. 3.2. Synthesis machine.

4. GROWING AUTOMATA

John von Neumann has shown how to design universal constructing machines and self-reproducing automata within the framework of a general definition of automata.[23-25] The details of von Neumann's definition are contained in an unpublished manuscript, and so I have worked out an alternative definition. Von Neumann's model is much closer to biological phenomena than the definition presented here; for example, each of von Neumann's cells has a unit delay,[26] while many of our cells have no delays. Von Neumann's constructions are also based on very weak primitives (his cells are capable of only 29 different states),[27] while we have assumed very strong primitives (each cell may have any of about 28,000 structures and each structure is capable of several states). While the definition presented here is much less realistic and economical than von Neumann's, it does make it easy to construct and present various kinds of growing automata.

A comment about our use of the word "automaton" is needed here. The concept we will define is that of _automaton_, without qualification. The _fixed automata_ of Section 2 will be special cases of automata, namely, those automata consisting of fixed structures of computing elements. Automata whose structure changes in time are called _growing automata_. But although our definition and some of our subsequent remarks apply to automata generally, we are particularly interested in growing automata.

The basis of each automaton is an infinite two-dimensional array of discrete cells.[28] A particular automaton is defined by specifying the structure of a finite number of cells (i.e., stating what elements are in them) at time zero; the rest of the cells are empty (structureless). The structure at time t+1 is determined by the structure at t and the complete state of the device at t, according to rules to be explained. These rules do not allow more than a finite number of cells to be structured at any particular time.

23. J. von Neumann, "The General and Logical Theory of Automata," pp. 1-31 of _Cerebral Mechanisms in Behavior - The Hixon Symposium_ (edited by L. A. Jeffress), New York, 1951. This is reprinted in _The World of Mathematics_ (edited by James R. Newman), Vol. 4, pp. 2070-2098.
24. John G. Kemeny, "Man Viewed as a Machine," _Scientific American_, _192_, 58-67 (April, 1955).
25. C. E. Shannon, "Von Neumann's Contributions to Automata Theory," _Bulletin of the American Mathematical Society_, _64_, 123-129 (May, 1958).
26. Shannon, _op. cit._, p. 127.
27. _Ibid._
28. Von Neumann considered the two-dimensional case, but his general procedure applies to any number of dimensions.

There are four types of elements which may be created in a cell. Each element is binary in the sense that each wire or node of an element is capable of two states. (1) A computing element may be a switch, a delay, a combined delay and switch, or just a plain wire. Each edge of a cell has at most one computing wire (shown in solid in the figures) impinging on it from within the cell. Specifically, the computing primitives are as follows; see Fig. 4.1. (a) Any switch with three inputs and one output, e.g., cell A1 of Fig. 4.1. There are 2^{10} of these, taking into account the different orientations of their outputs (north, south, east, and west), and allowing all 256 possible three-variable truth functions. Note that some of these truth functions will be independent of some of the inputs, so that this category includes as special cases two-input, one-input, and zero-input switches. This category also includes certain wires, such as the wires of cells A1, B1, and C1 of Fig. 4.5. (b) There are unit delay elements with one output and one, two, or three inputs. These inputs come from one, two, or three edges through a disjunction. There are 28 of these; see cell B1 of Fig. 4.1. (c) There are unit delays with two outputs and one or two inputs (e.g., cell C1 of Fig. 4.1). There are 18 of these. (d) There are unit delays with one input and three outputs (A2 of Fig. 4.1); there are four of these since the input can come from one of four different cell edges. (e) There are arrangements of wires in cells not covered by case (a), governed by the rule that incoming wires may not merge and at most one wire contacts a side. There are 28 possibilities; see cells B2, C2, A3, B3, and C3 of Fig. 4.1, and B4 of Fig. 4.5. Altogether there are 1102 different computing elements.

(2) There are scaffolding elements, which have one input wire and either one or two output wires. Scaffolding wires are shown dashed; see cells B1, C1, B2, and C2 of Fig. 4.2. If we count primitives with different orientations as different primitives there may be any of 24 different scaffolding structures in a cell. The scaffolding primitives will conduct construction signals. A branched scaffolding element (e.g., cell B2 of Fig. 4.2) has an implicit switch associated with it. By convention this switch has an initial setting; the switch setting may be changed by using switching sequences which will be introduced later.

(3) Any scaffolding element may coexist with any computing element in the same cell. See, for example, cell A2 of Fig. 4.2 and cell A5 of Fig. 4.4. There are 24 x 1102 such primitives.

(4) The last element is the key to the construction process. It is called a constructing element and is shown in cell A1 of Fig. 4.2. A constructing element receives construction signals from a computing structure and routes these into a scaffolding structure; the construction signals travel through the scaffolding and eventually cause a cell to be structured in a way to be explained. A constructing element may receive its input from any cell edge and transmit its output to any other edge of that cell; hence there are 12 different constructing elements.

A cell may have an element of any of these four types or it may be structureless. Altogether there are 27,587 (which is less than 2^{15}) possibilities,

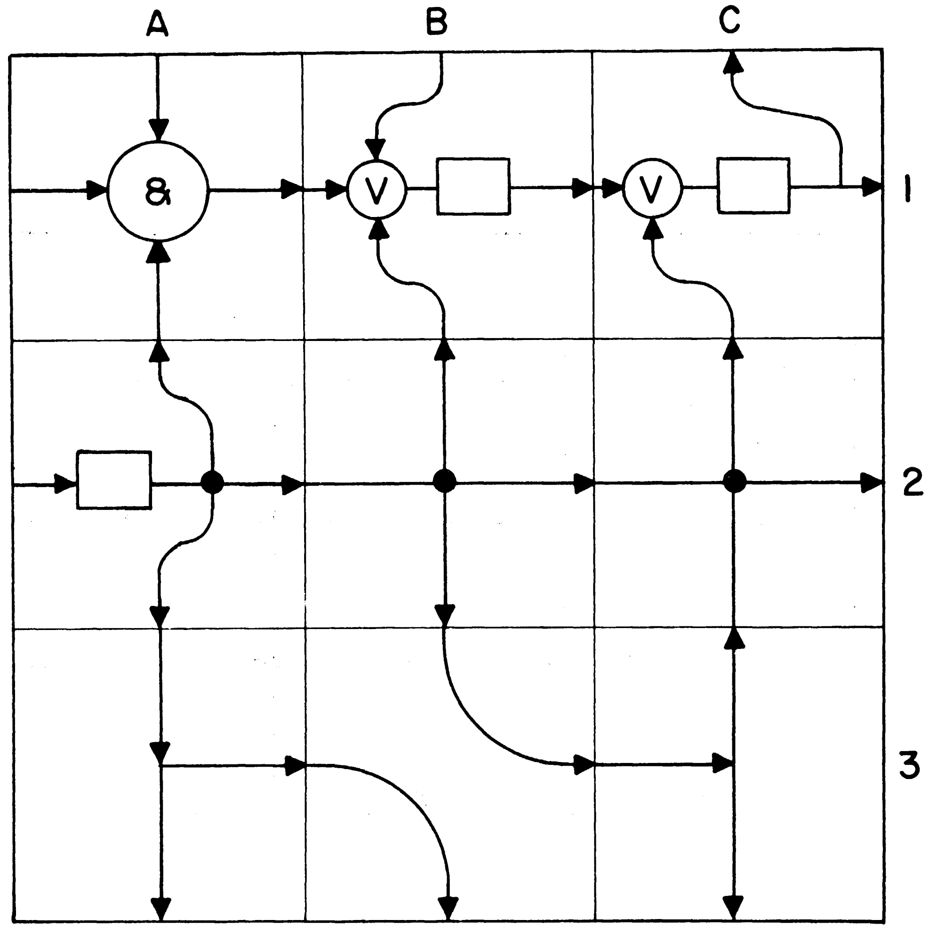

Fig. 4.1. Computing primitives.

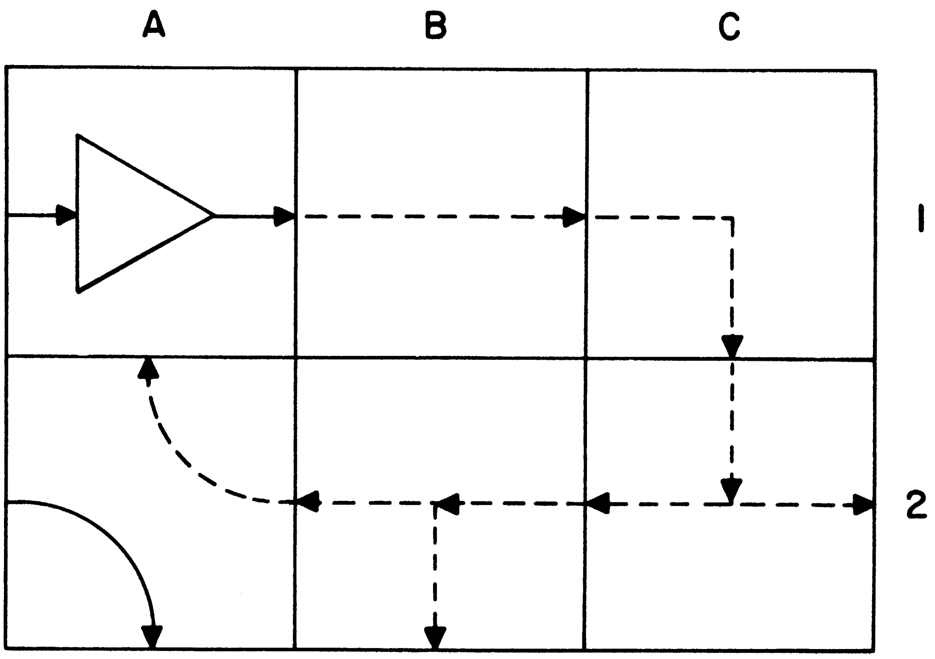

Fig. 4.2. Scaffolding and constructing primitives.

each of which will be represented by a binary sequence called a constructing sequence. These sequences will be formed or stored in a computer structure and fed into the constructing element. There is no upper limit to the length of computer wires or scaffold wires, and we will make the idealized assumption that signals will travel down the wires instantaneously.

The mode of operation of the system and the formulation of further rules can best be presented by means of an example. We will show how a tape may be lengthened under the control of a machine, thereby relating growing automata to generalized Turing machines. Figure 4.3 shows two "squares" of the tape, each square storing a single bit. The switching functions (a), (b), and (c) are the following. Let "N," "S," "W," and "E" stand for the north, south, west, and east edges of a cell, respectively. The output of switch (a) is E and is given by the equation

$$E(t) \equiv [\{\overline{S(t)} \text{ and } N(t)\} \vee \{S(t) \text{ and } W(t)\}] \quad .$$

The output of switch (b) is E and is given by the equation

$$E(t) = [\{\overline{S(t)} \text{ and } W(t)\} \vee \{S(t) \text{ and } N(t)\}] \quad .$$

The output of switch (c) is N and is given by the equation

$$N(t) = [\{\overline{E(t)} \text{ and } W(t)\} \vee \{E(t) \text{ and } S(t)\}] \quad .$$

You are to imagine that a fixed computer which is associated with the tape is connected to the inputs and outputs of Fig. 4.3. This computer may "write" on the left-hand square and "read" what is stored on that square. It may cause the information on the tape to be shifted to the right by stimulating the "right shift" signal, and in a similar way it may cause the information to be shifted to the left. By feeding signals into the constructing element, it can cause the creation, destruction, or alteration of new tape squares to the right in a way to be indicated. A similar arrangement exists for modifying the tape to the left.

We will next describe the construction of a square of tape. Cell A6 of Fig. 4.4 needs a scaffold with a branch. The construction sequence representing this element is produced by the fixed computer and fed into the constructing element, which produces the desired element in the cell touched by its output. The next constructing sequence will pass through the constructing element into the scaffold, through the vertical output of the scaffold, and will produce a new element in the cell (A5) which is touched by the output of the scaffold. In this case the structure produced is a horizontal computing wire and a vertical scaffolding wire. This process is repeated to give a structure to the next two cells (A4 and A3); the scaffolding of these cells is not shown because of pictorial difficulties.

After the rest of column A is structured, construction activities need to be shifted to column B of Fig. 4.5. This rerouting is done by a binary switching

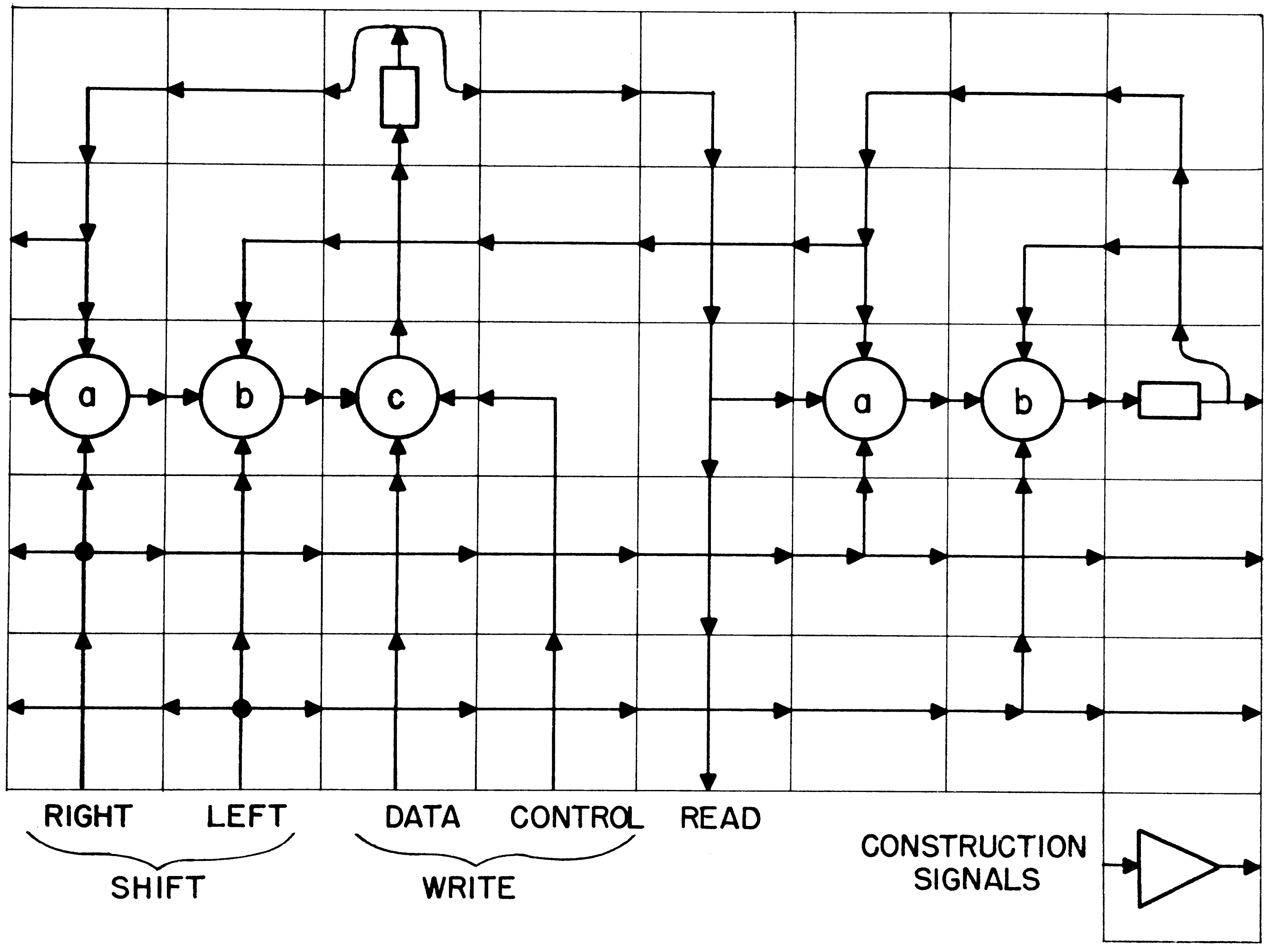

Fig. 4.3. Two "squares" of tape.

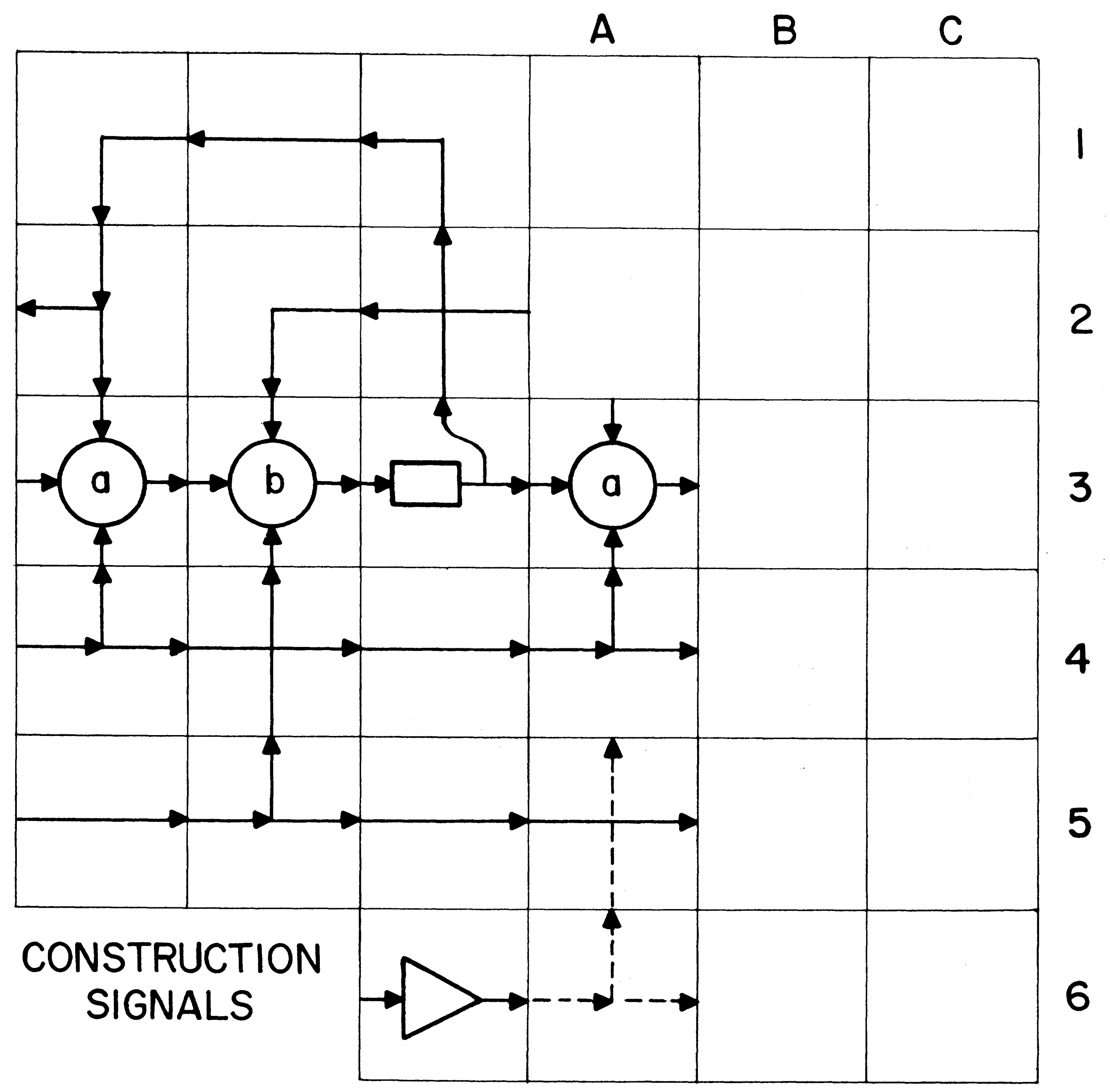

Fig. 4.4. Construction process.

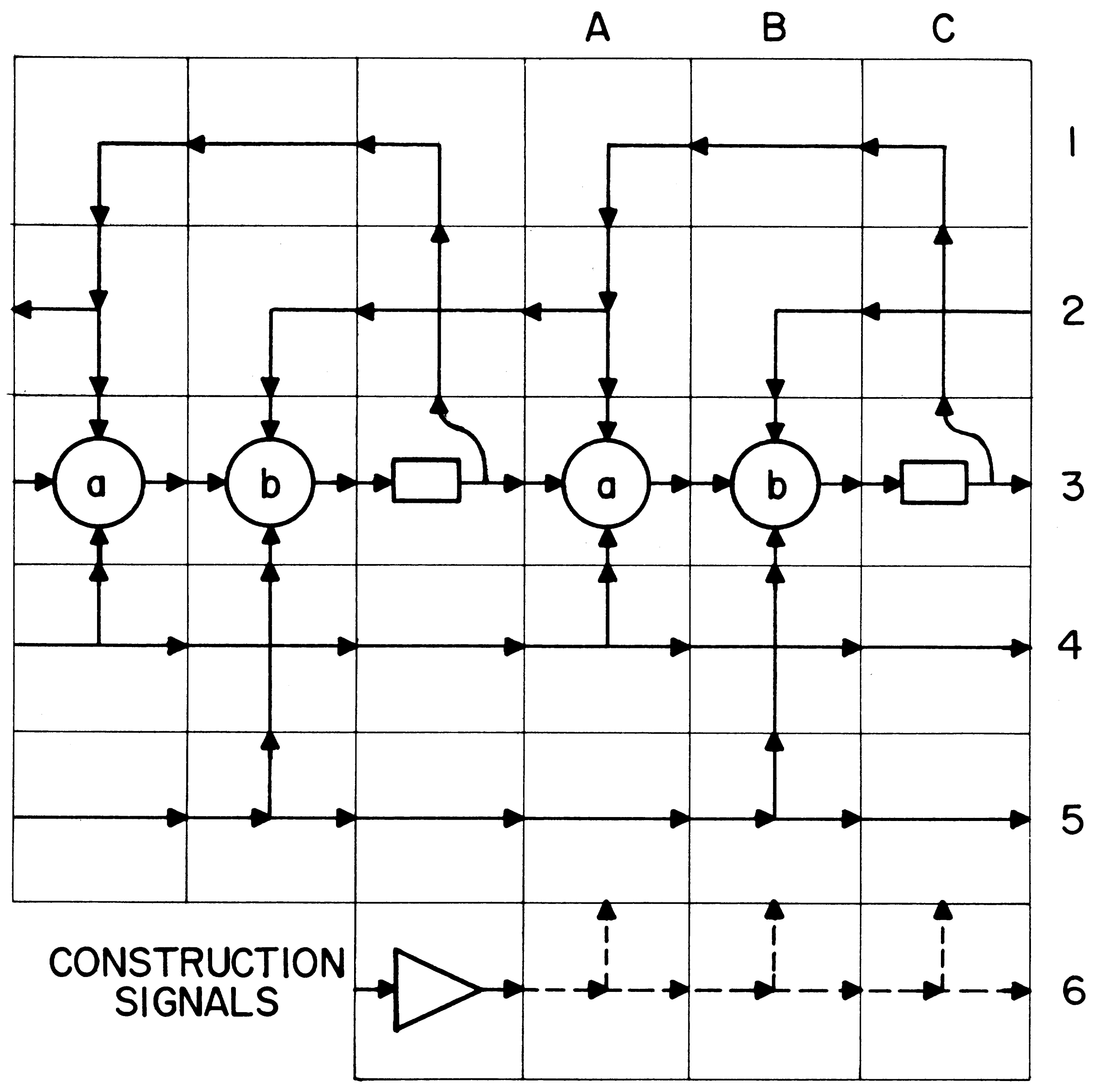

Fig. 4.5. Completion of new tape "square."

sequence which goes through the constructing element into the scaffold and changes the switch setting of the scaffold of cell A6 so that subsequent constructing sequences are routed into cell B6. After this we can structure column B by means of six constructing sequences. Another switching sequence resets the switches in cells A6 and A5 and permits the structuring of column C. This switching sequence works as follows. It would contain two switching signals in coded form. The first signal would be received by A6, and in this case would leave the switch set as it was. The second switch signal would be passed on to B6, and would cause its switch setting to be changed so that the next constructing sequence would effect cell C6 (rather than traveling through the B column). In general, a switching sequence will contain as many switching signals as there are switches which must be passed through to get to the last switch to be modified.

The central computer may want to contract the tape as well as to expand it, and so it needs the capacity to wipe out the structure of a cell. This is provided for by binary destroying sequences. A destroying sequence passes through the constructing element and down the scaffolding in the same way as a constructing sequence. However, a destroying sequence affects the cell which contains the terminus of this scaffolding channel rather than the next cell; it causes the element of this cell to be destroyed. There is a sequence of 18 destroying sequences and 2 switching sequences which will destroy the tape square just created.

We may also use a switching sequence followed by three destroying sequences to erase the elements of the bottom row (6) and then use 3 constructing sequences to create the scaffolding shown in Fig. 4.6. Thus with a master sequence of 21 constructing sequences, 3 switching sequences, and 3 destroying sequences we end up with a new tape square and with the scaffold arranged so that each repetition of this master sequence will produce a new square to the right. By iterating this master sequence we can produce a tape of any length.

In this way we can construct a generalized Turing machine except for the fact that the process of construction just described takes many units of time per tape square and hence is not as fast as that stipulated for a generalized Turing machine in Section 2. Thus the generalized Turing machine is almost but not quite a special case of our definition of automaton. We could make it so in either of two ways. First, we could use a growing automaton to simulate a generalized Turing machine by associating one major cycle (i.e., the time required for the construction of a tape square) of the growing automaton just described with one minor cycle (i.e., a single unit of time) of the generalized Turing machine. Second, we could enrich our base of construction by adding elements and wires capable of more than two states. It would not be necessary to introduce computing elements with more than two states. It would suffice, for example, to let the scaffolding elements have many states and to have constructing elements which convert binary signals into signals properly coded for the scaffolding elements; note that this method would not involve any increase in the number of computing primitives or the number of scaffolding primitives, and not a very great increase in the number of constructing primitives.

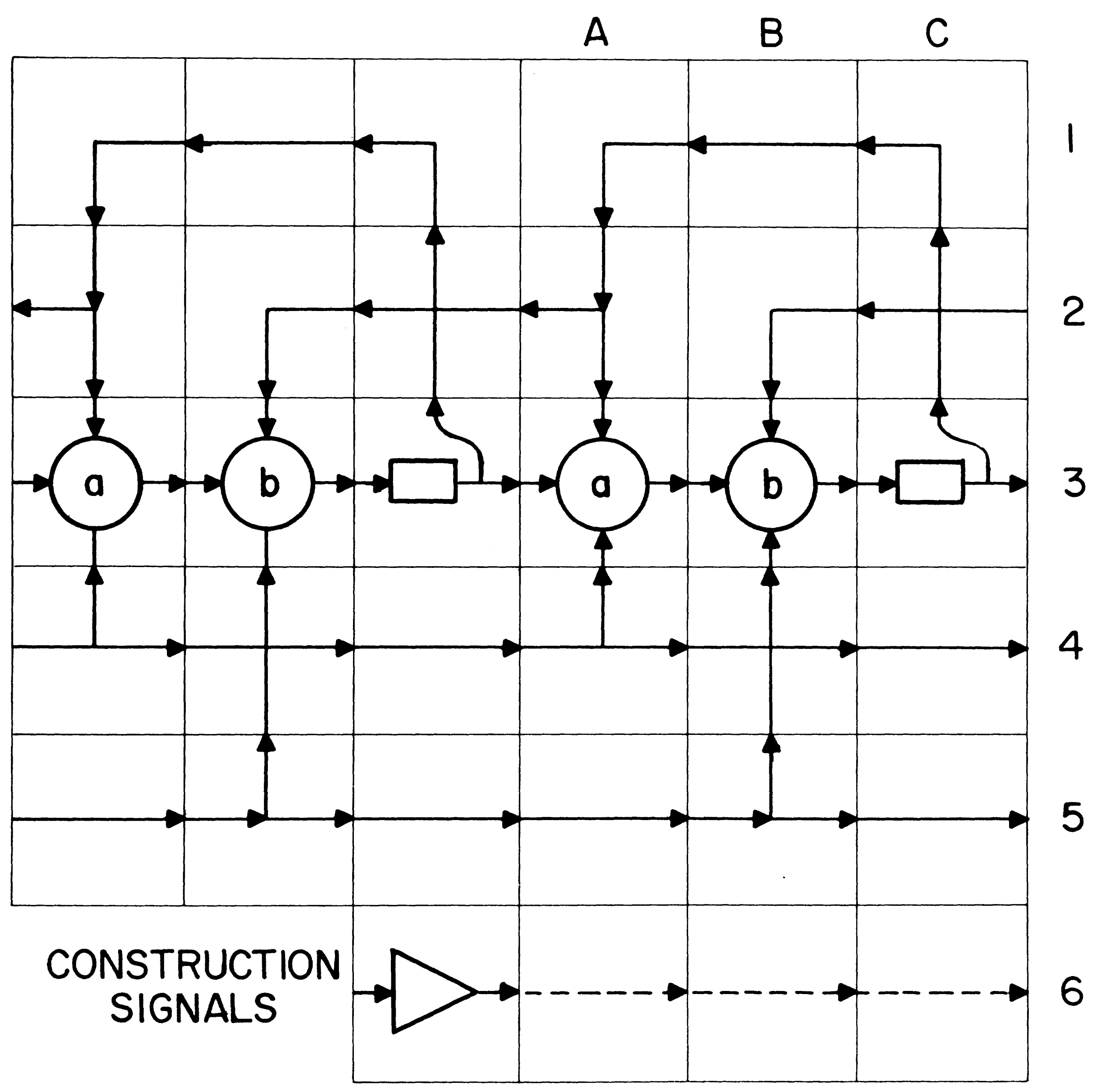

Fig. 4.6. Preparation for next construction.

We have said enough about the definition of "automata" to make its main features clear. Various details need to be specified before the definition is complete. We will assume that a primitive element is active as soon as it is created. This means that part of a computer which has been constructed may start to operate before the computer is finished. This fact needs to be taken into account in the design of the computer, as it often does in the design of actual computers, since these computers may have to be cleared to a standard initial state before they are ready to operate. Two constructing sequences, coming from two different constructing elements, might arrive at a cell at the same time; a priority rule is needed to determine which one governs the construction. There are other minor details which need to be specified but we will not bother with these here.[29]

Let us consider next what automata can be specified as automata in the sense of the present definition. Every fixed automaton can be put into a normal form,[30] and this normal form can be constructed of computing elements in cells; so each fixed automaton is an automaton in the sense defined here. We have shown in some detail how an expanding and contracting tape may be constructed; hence, except for a matter of speed, every generalized Turing machine is also an automaton.

One of the growing automata which can be constructed is von Neumann's universal constructor, of which a block diagram is shown in Fig. 4.7.[31] The <u>universal constructor</u> C is a finite structure to which may be attached a tape containing the specifications of any number of automata which are to be synthesized. In general, each specification consists of two parts. The first part of the specification is a master sequence of constructing sequences, switching sequences, and destroying sequences, which will cause a fixed part of the desired automata to be constructed. This master sequence is the description $D(\alpha)$. The second part of the specification contains the contents $T(\alpha)$ of a tape which is to be constructed and attached to automaton α.

The universal constructor operates under the direction of its control as follows. First, the tape reader reads $D(\alpha)$ and transmits it to the constructor; concurrently the constructor constructs α. Next, the reader reads $T(\alpha)$; concurrently the reproducer produces a tape, attaches it to α, and records $T(\alpha)$ on it. This process is then repeated, so that at the end of the construction process $T(\alpha)$ is recorded twice on the tape. (The reason for this duplication of

29. It is perhaps worth noting that the definition does not prevent the construction of automata which are not well-formed (e.g., have switch cycles), nor is there an effective criterion for deciding whether a given growing automaton will ever "develop" switch cycles. Restrictions could be added to guarantee that every automaton is well-formed. The simplest such restriction would be to incorporate a delay in every switch, but this tremendously complicates the automata and limits their behavior. A less severe restriction is to require every switch or wire with an output on the south edge to contain a delay. We will not take time here to investigate the various possibilities.
30. Burks and Wang, <u>op. cit.</u>, Section 2.3.
31. Von Neumann, <u>op. cit.</u>, pp. 2096-2097.

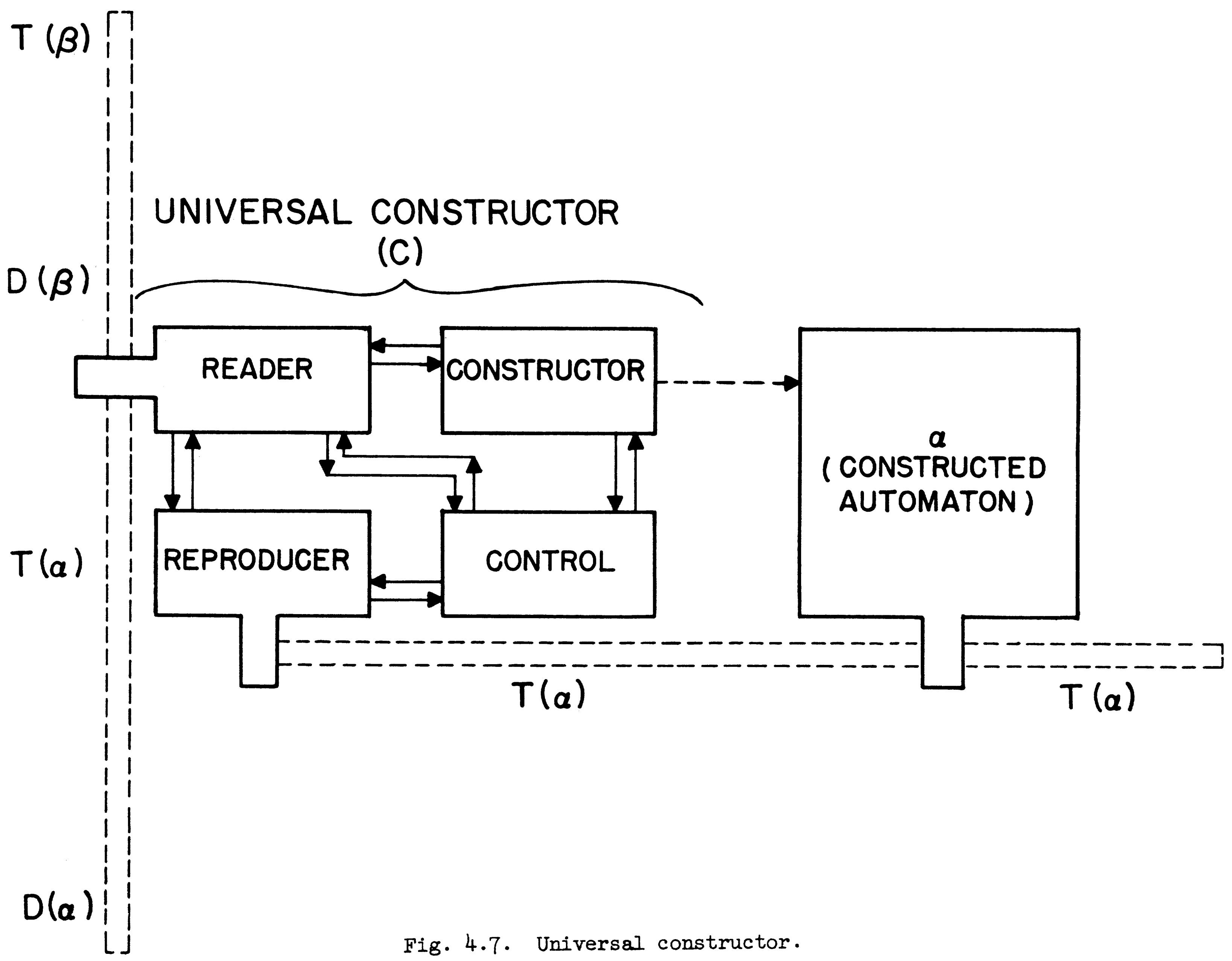

Fig. 4.7. Universal constructor.

tape information will become apparent in a moment.) In general, for each description $D(\alpha)$ of an automaton and each tape content $T(\alpha)$, the universal constructor will construct α and record $T(\alpha)$, $T(\alpha)$ on the tape attached to α.

As von Neumann showed, the universal constructor C together with the proper tape will reproduce itself; see Fig. 4.8. Let D(C) be a sequence of constructing sequences, switching sequences, and destroying sequences which completely describe C (including its scaffolding). Let the universal constructor C contain a tape storing D(C) twice when it begins action. The universal constructor will first read D(C) and produce C. It will then read D(C) twice and produce a tape recording D(C) each time. [This is the reason for the earlier requirement that $T(\alpha)$ be copied twice.] Thus the universal constructor C when supplied with a tape containing D(C), D(C) produces a universal constructor plus a tape containing D(C), D(C). Hence C with a tape storing D(C), D(C) reproduces itself. The new combination of C and the tape D(C), D(C) can then reproduce itself, and this process can be repeated _ad infinitum_.

If we combine a universal constructor and a universal computer (i.e., a universal generalized Turing machine), and supply this composite machine with a tape containing its description twice, this whole automaton will reproduce itself; see Fig. 4.9. Thus a universal constructor-computer can reproduce itself.[32]

32. _Ibid._, p. 2098.

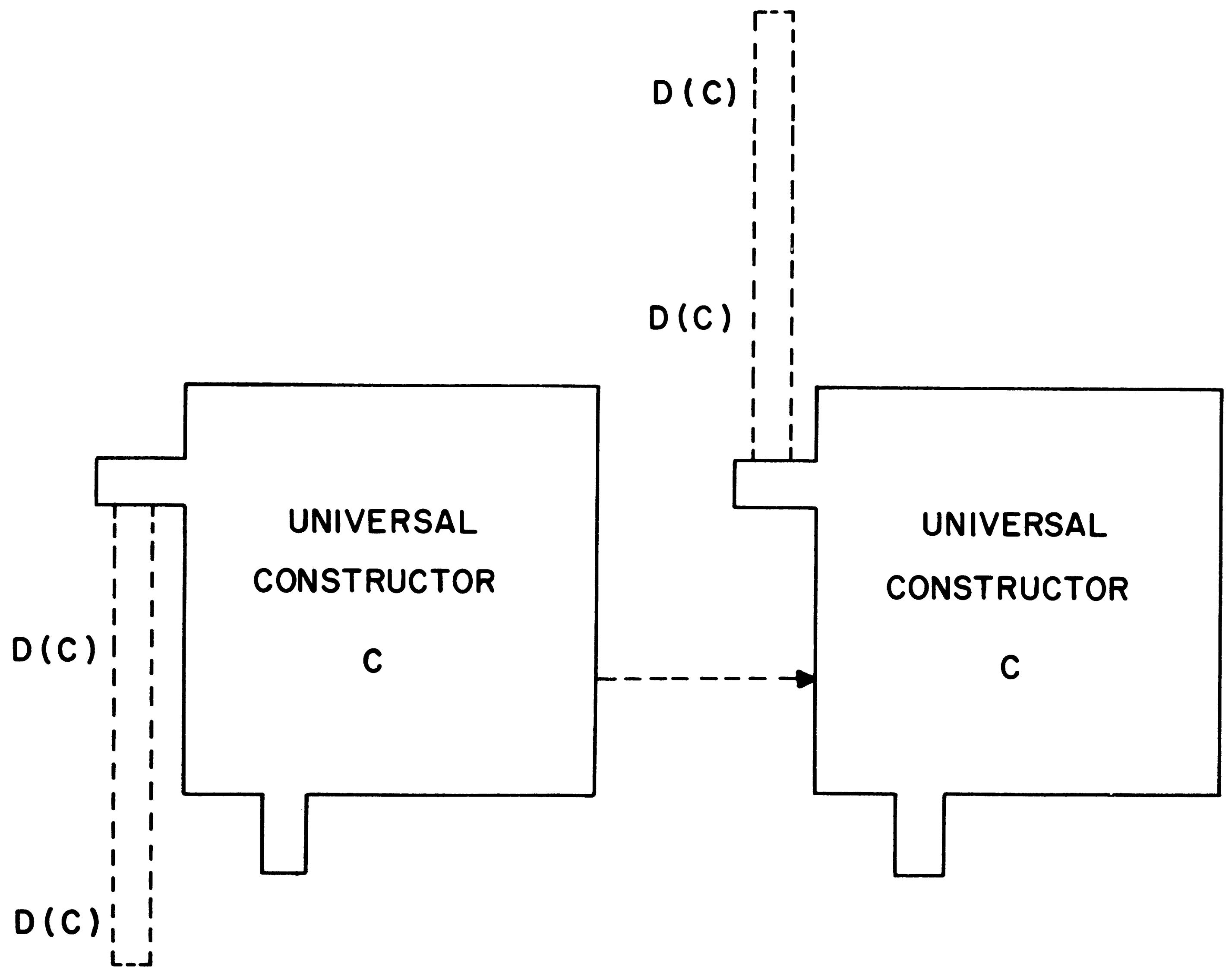

Fig. 4.8. Self-reproduction.

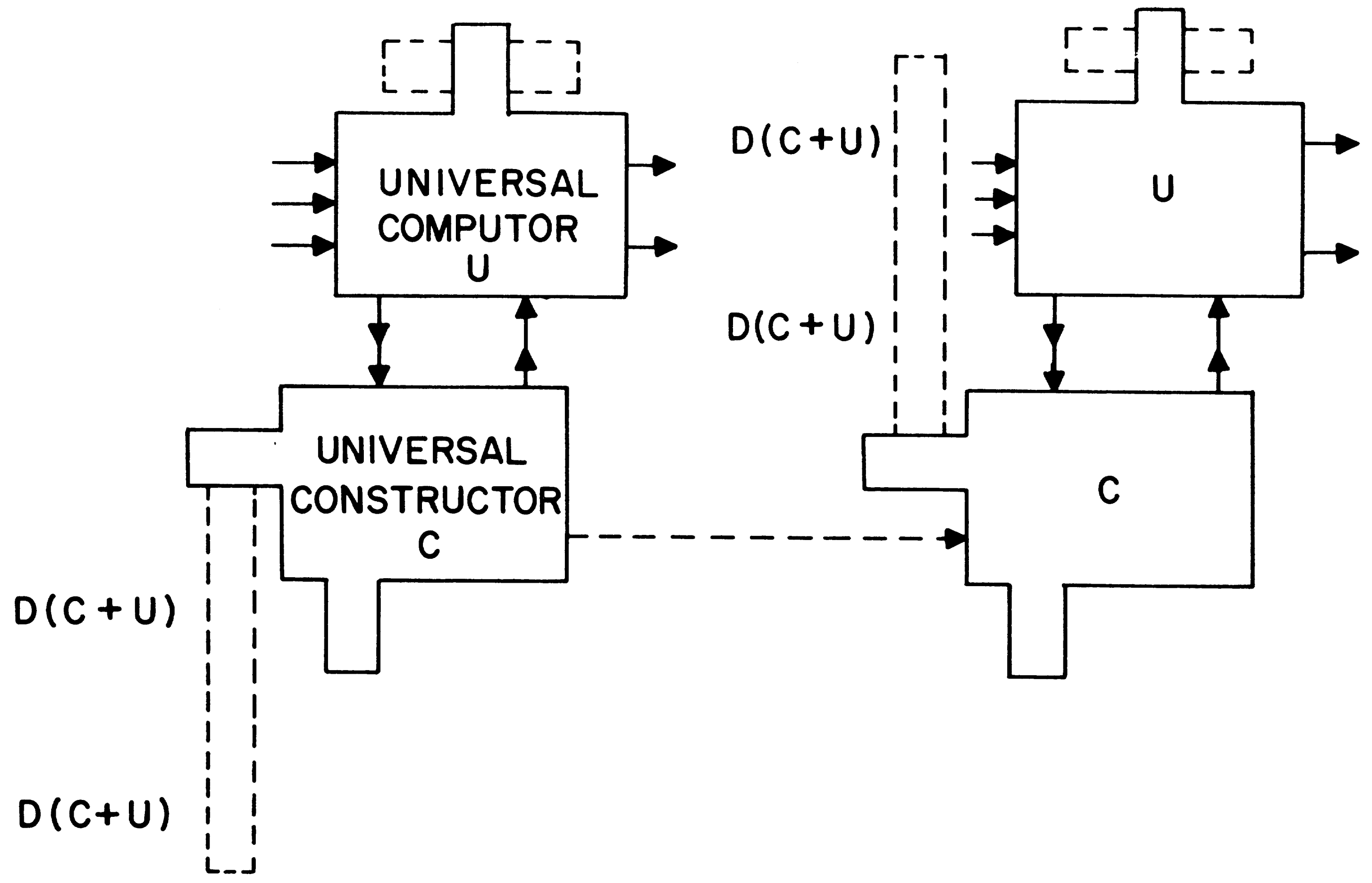

Fig. 4.9. Universal constructor-and-computer.

5. SOME PROBLEMS ABOUT GROWING AUTOMATA

In Section 3 we mentioned some problems and results about fixed automata and generalized Turing machines and remarked that these may be extended to cover automata in general (as defined in Section 4) and hence growing automata in particular. There is another type of problem, quite different from any of the problems discussed before, which is applicable to automata of all kinds; this concerns the relation of the cyclic complexity of the structure of an automaton to its behavior. Since our earlier discussion of a generalized Turing machine did not make explicit whatever cycles are implicit in the tapes, this problem had to be postponed to the present section.

Let us consider the problem first for the case of a fixed automaton. We can uniquely decompose the structure of a fixed automaton into _maximal cycles_. The _degree_ of a maximal cycle is the number of unit delay elements in it, and the degree of a fixed automaton is the maximum of the degrees of its maximal cycles.[33] Figure 5.1 consists of a single maximal cycle of degree 2; the output nodes are marked by stars. Figure 5.2 contains six maximal cycles, each of degree 1, so Fig. 5.2 is of degree 1; its output nodes are also marked by stars.

How, now, is the degree of an automaton related to its behavior? Is there some critical degree n, such that all fixed automata behaviors can be realized by structures of that degree (or less)? It has been conjectured that there is no such degree. The general problem remains open, but John Holland has shown that no automaton of degree 1 can realize the behavior of Fig. 5.1, which is a ternary counter.[34]

The same problem can be raised about the computation of a fixed automaton: is there some critical degree m, such that all fixed automata computations can be realized by structures of that degree or less? Both problems can be extended to generalized Turing machines and to growing automata. Of course the structure of any growing automaton changes with time, and hence the degree of the structure changes with time as well. But we can define the degree of a growing automaton to be the maximum degree of any of its structures at any time for any input history, if the maximum exists; otherwise the degree is infinite.

We can now ask these questions about automata generally: (1) Is there some finite degree n such that all behaviors can be realized by automata of this de-

33. These concepts are explained in detail in Sec. 4.1 of Burks and Wang, _op. cit_
34. John H. Holland, _Cycles in Logical Nets_, Univ. of Mich. Ph.D. dissertation, 1959. This thesis contains many other results in the same general area.

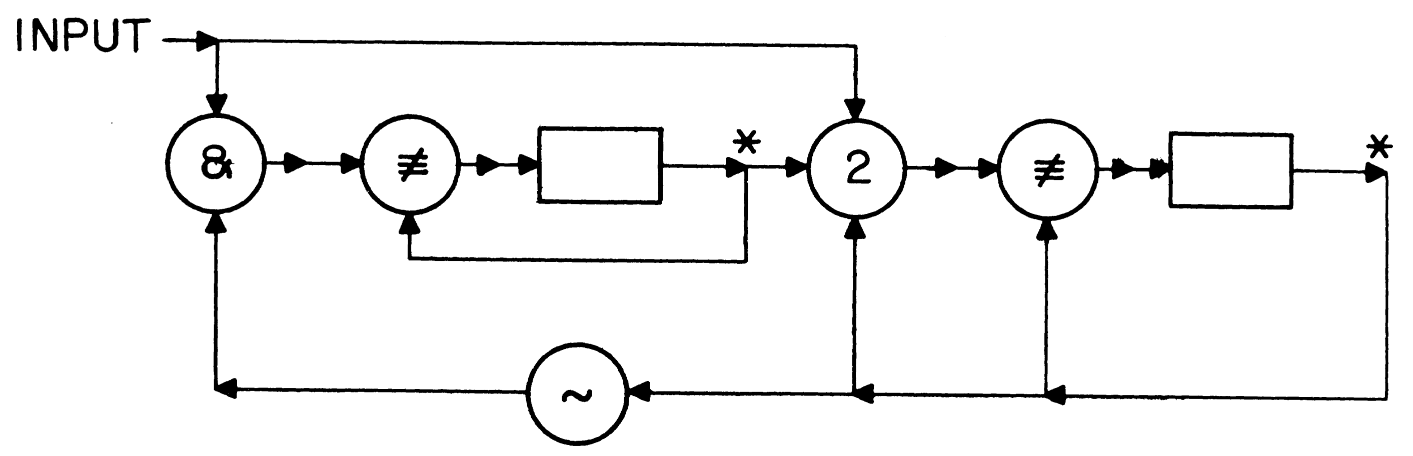

Fig. 5.1. Automaton of degree 2 (ternary counter).

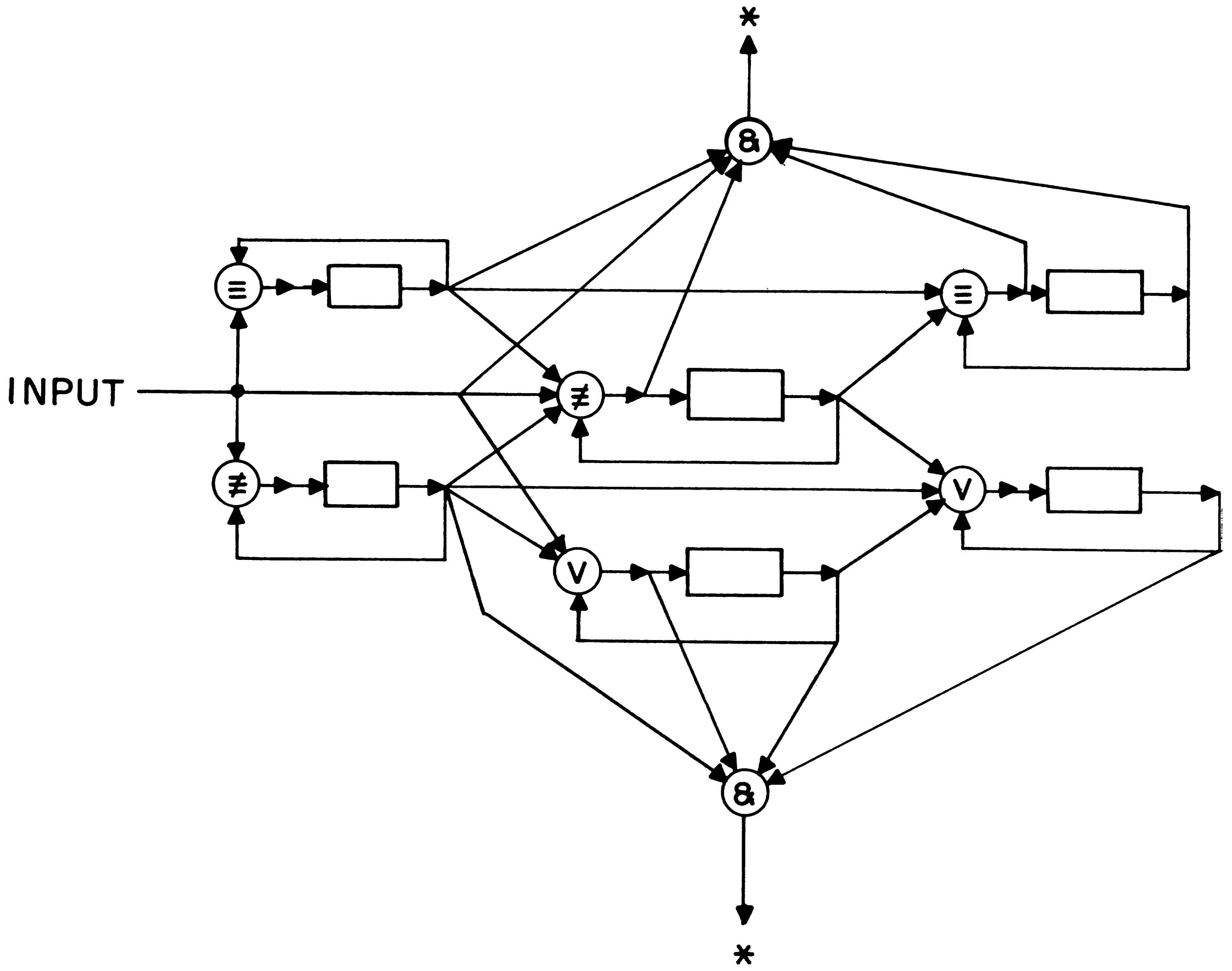

Fig. 5.2. Automaton of degree 1.

gree or less? (2) Is there some finite degree m such that all computations can be realized by automata of this degree or less? It might seem that the answer to both these questions is "no," since clearly in both cases there is no upper limit to the length of the tapes needed, and the tape designed in the preceding section is of infinite degree. However, it is possible to design a tape of finite degree by using isolated squares and moving the read-write head along the tape instead of shifting the information along the tape. To store a 1 in a square the automaton builds a delay cycle in which a pulse circulates; to store a 0 it builds a delay cycle whose successive states are 0. The fixed part of the machine can send out a long wire to do this and by means of another wire can sense the contents of the delay cycle. The contents of the square can be changed by destroying the delay cycle in it and rebuilding the appropriate one. When the fixed part of the automaton is neither reading from the square nor writing in the square, it would have no connection to it. In this way a generalized Turing machine may be designed which has a fixed degree.

There are other measures of feedback complexity which are of interest. Consider a fixed automaton with n unit delays $x_1, x_2, x_3, \ldots, x_n$. Let $f(x_i)$ be the number of delays which directly and immediate affect the delay element x_i, i.e., whose outputs drive the input of x_i through switch elements. [In Fig. 5.1, $f(x_1) = f(x_2) = 2$ since each delay drives itself and the other delay through a switch.] Define an average measure of feedback complexity F for the automaton by

$$F = \frac{\sum\limits_{i=1}^{n} f(x_i)}{n^2}$$

There are "directly entirely connected" automata for which F = 1,[35] Fig. 5.1 being an example. However, the amount of feedback complexity F in actual systems is usually much less than the maximum. F = 7/18 for Fig. 5.2 and F is small in a typical modern digital computer. (Note that such a computer is pretty nearly one maximal cycle, not counting the tapes.) This fact is relevant to our ability to understand a large digital computer. Because the ratio of actual feedback lines (from delays) to possible feedback lines is low the various parts of the computer have a large amount of local autonomy and may be understood in isolation from the rest of the computer. (Uniformities within these parts also make it easier for us to understand them.) It is worth noting that if neurons are represented by switches and delays, F is small for the human neural system.

Let us turn next to some problems which are unique to growing automata. The first is one that von Neumann considered: How complex does a self-reproducing automaton need to be?[36] In his model von Neumann used cells which were capable of twenty-nine states, and he estimated that about 200,000 cells would suf-

35. See Burks and Wang, op. cit., p. 291. Of course every automaton can be put in a normal form which has this property, but we are here interested in nets in "reduced form," in which all irrelevant (noneffective) switch input wires are deleted.

36. Shannon, op. cit., p. 127.

fice for self-reproduction.[37] The cells of our models are much more complicated, for a cell may contain any of about 2^{15} different elements, and each element is capable of several states. But counting states is not sufficient, because there are many ways in which states may be used; for example, a primitive element capable of transferring its structure to a neighboring cell on receipt of a single pulse is in certain respects more powerful than any of the primitives we have assumed.

One could compare growing automata constructed within the framework of different definitions of "automaton" by reducing both definitions to a common system. We will show how to make this reduction for our definition. For this purpose we will use a particular infinite fixed net of switch and delay elements to model or simulate our definition of automaton.[38] This infinite fixed net corresponds to the infinite two-dimensional array of discrete cells constituting the basis of each automaton in the sense of Section 4. You will recall that in Section 4 a particular automaton is defined by specifying the initial structure (i.e., the structure at time zero) of a finite number of cells; this structure would be represented in the infinite fixed net by a particular initial state of that net. In Section 4 the structure of an automaton at time t+1 is determined by the structure at time t and the complete state at t according to rules explained there; this succession of structures (and states) will be represented in the infinite fixed net as a succession of states of that net.

The infinite fixed net is constructed in the following way. There is a finite fixed net (located in a square) corresponding to each cell of our definition of automaton; these squares are repeated *ad infinitum* in the same two-dimensional way as are the cells of Section 4. We thus get an infinite structure which is locally heterogeneous but is homogeneous in the large.

We cannot take time to describe the finite fixed automaton corresponding to a cell, but we will indicate its main features. It will contain a register, a master switch, and switches and delays which will perform the scaffolding and computer functions of the original cell. Each fixed net of a square would have two inputs to, and two outputs from, each of the four neighboring squares; these inputs and outputs would be used for scaffolding wires and computer wires. A constructing sequence which was to affect a given square would be routed to the register of that square; the register would then determine the connections of the switches and delays by means of the master switch; in this way the fixed automaton of the square would simulate the original cell. Flip-flops would be used to store the switch settings of branched scaffolding elements; a switching

37. Kemeny, *op. cit.*, p. 66.
38. Note that this net is not itself an automaton in the sense of Section 4. Note also that though the net is infinite, at any finite time only a finite number of nodes have ever changed from their initial state; cf. the remark on infinite tapes in Section 2, note 8.
Little is known about infinite fixed automata, although special cases have been considered by Church, *op. cit.*, and John Holland, "A Universal Computer Capable of Executing an Arbitrary Number of Programs," unpublished.

sequence passing through a square would set the flip-flop correctly and then pass on to set the flip-flops representing other scaffolding branches. A destroying sequence would go into the register and restore the square to its original state.

Each particular automaton is defined within the framework of our original definition by stipulating the initial structure of a finite number of cells. As we remarked before, this operation corresponds to setting the initial states of the delay elements of a finite number of squares in the infinite fixed net. Changes of automaton structure in the original definition will then appear as changes of state in the infinite fixed net. Thus we are using some aspects of the behavior of the infinite fixed net to simulate the structure of the original automaton.

Von Neumann's definition of automaton could be simulated in a similar way. Then the complexity of von Neumann's design of a self-reproducing automaton could be compared with the complexity of the design we described by making a weighted count of the number of switches and delays used in the initial structure of each. By precisely defining a common reference system in this way, we can make the problem of minimal complexity of cell reproduction into a strictly logical problem. Experience with minimality problems suggests that it would be extremely difficult if not impossible to find the minimal solution and prove it minimal. One might, however, obtain an estimate of minimal complexity by using a computer to help work out many alternative constructions.

The idea of modeling a specific structure in an infinite uniform fixed net may have practical applications in future computers. There are under development radically new techniques for constructing solid-state computers. It is estimated that by one of these methods "the component density in a finished circuit could be 50 million per square inch per layer. ... It is conceivable that more than 10,000 layers per inch could be formed, giving a volume density of 5×10^{11} components per cubic inch."[39] Such computers might be built for the most part in a uniform way similar to that described above for the infinite fixed net used to simulate our definition of automaton. A particular problem could be solved by converting this net into a "structure" appropriate to that problem, in the way the infinite fixed net is structured to represent a particular growing automaton. For each new problem (or portion of a problem) the computer would simulate a different structure; in effect, it would in a sense become (during the solution of this problem) a special purpose computer especially designed to solve that problem. The common division of a computer into arithmetic units, controls, and storage units would be used only if this organization was best for the problem at hand. All this simulated restructuring would be done automatically, of course, by a programmed change of state of the uniform fixed net, and could be

39. Dudley Buck and Kenneth Shoulders, "An Approach to Microminiature Printed Systems," Notes from Summer Session Course in Advanced Theory of the Logical Design of Digital Computers, The University of Michigan, 1959, p. 2. This paper was presented to the Eastern Joint Computer Conference, Philadelphia, 1958.

done more or less continuously during the solution of a problem.[40]

There are many other concepts of growing automata worthy of investigation, of which we will mention a few. It is not necessary to employ a fixed framework of cells; one could allow new cells to spring up between old ones under the control of the automaton. This concept might be of help in describing and understanding a computation. Suppose a list of words is stored and at a later date many words are to be lengthened and new entries are to be inserted. This change could be conceived as an automatic process of lengthening the bins which hold the word, and the inserting of new bins between the old ones; the change of bins must of course be accompanied by an appropriate change of the switches which connect these bins to the rest of the automaton. In general, storage and computing facilities would be created wherever needed and in a form suited to the problem being solved.

In one of his models von Neumann had girders, sensing organs, and joining organs, as well as computing elements, floating freely in a lake. As Shannon remarks,[41] because of the complexity of motion which is thereby possible it would be exceedingly difficult to give a detailed design of a universal constructor in this model. One might modify the definition of automaton we have given (Section 4) in at least two ways to obtain some of the effects of von Neumann's "lake" model without all of the complexity. In the first case the result is a probabilistic automaton, while in the second case it is a deterministic automaton.

(1) We can assume as primitives a few basic types of cells, each with a number of internal states. These cells would be moving about on the surface of a lake in a random fashion. The random motion would cause cells to make contact with one another along the edges. Each cell would have the capacity to accept or reject other cells under the control of its own internal state. A particular automaton, such as a self-reproducing one, would begin life as a two-dimensional finite structure floating on the surface. The cells on its edges would contact other cells at random, and the automaton would accept or reject these new cells according to its needs. Stimulated by von Neumann's work, L. S. Penrose has investigated models of this sort, using cells with mechanical hooks and producing random motions by shaking the cells on a board.[42,43] His units are capable of only a few internal states, and the devices he begins with consist of only a few units.[44] Penrose did his work experimentally, that

40. Similarly, we can think of the program of a present-day general purpose computer as determining a temporary structure, and hence as converting the general purpose machine into a special purpose machine for the given problem. But the computer described above would have much more flexibility of structure than any existing computer.
41. Op. cit., pp. 126-127.
42. "Automatic Mechanical Self-Reproduction," pp. 92-117 of New Biology, No. 28, 1959, Penguin Books.
43. "Self-Reproducing Machines," Scientific American, 200, 105-114 (June, 1959).
44. Most of his models were one-dimensional, although he did some work on a two-dimensional model.

is, by constructing the units from wood and shaking them mechanically. This general type of model could be simulated on a digital computer.[45]

It would be instructive to make such simulations, starting with cells, rules for interconnections, and initially given automata, all of which are much more complicated than the units, rules, and initial structures Penrose used.

(2) Von Neumann's "lake model" and the variations of it just discussed are probabilistic automata. A second way of modifying our definition of automaton (Section 4) so as to obtain some of the power of von Neumann's lake model without all the complexity yields a deterministic automaton. The modification consists in adding a sensing element which would sense the structure and contents (state) of a cell. The sensing element would send sensing channels through the cells in a way similar to the way a scaffold is constructed by the constructing element. These sensing channels would not disturb the structure already in the cells, but would carry information about the cell being sensed back to the sensing element. By a process analogous to the construction procedure described in the previous section, an automaton could sense what element was contained in a cell and the state of that element at that time. This information could be used in various ways, e.g., it might be used to detect a fault which could later be repaired by the constructing element.

The question then arises: How much self-knowledge can an automaton acquire and keep? This question is of interest to the theory of self-organizing systems since a possible procedure in self-organization is for the system to have a picture of its structure and behavior and use this picture to guide itself. The automaton might be given this picture initially or it might be programmed to construct the picture by means of sensing elements and sensing channels. Clearly there are limits to how much knowledge an automaton can acquire and keep about itself. For example, a growing automaton might change faster than its sensing elements could sense these changes. There are also logical limitations. Thus if all the states of an automaton are used, a proper part of the system cannot have a completely detailed picture of the whole, for the whole, being larger than the proper part, would be capable of more states than this part.

The logical paradoxes of self-reference are relevant here. It is well known that some referential statements lead to contradictions. Thus the sentence, "This sentence is false," is false if true (because it says of itself that it is false) and true if false (because if it's false that it's false, then it is true), and hence both true and false. To resolve such paradoxes Russell proposed[46] a formal language in which no self-referential statements

45. The digital computer is (when working properly) a deterministic automaton, while the model simulated is probabilistic. The required randomness could be achieved by using a table of random numbers. The simulation will still be deterministic, but it will sufficiently approximate the probabilistic model to be satisfactory.

46. See B. Russell and A. N. Whitehead, Principia Mathematica, Ch. II of the Introduction ("The Theory of Logical Types").

could be formulated. The proposal that no self-referential statements be allowed in a formal system is much too strong, however, for there is clearly nothing wrong with the self-referential statements: (1) "This sentence is in English"; (2) "This sentence is written at least once." Indeed, both of these sentences are true. These sentences illustrate the fact that a sentence can refer to its physical properties without contradiction. There are also cases where a sentence can refer to its structural or syntactical properties without contradiction. The sentence, "This sentence is grammatical," is a case in point. Another case of a formula consistently referring to its own structure is the undecidable formula constructed by Godel in his proof of the incompleteness of arithmetic. This is a formula F which says, in a certain sense, "F is not provable."[47] Provability is a purely syntactical concept so this formula is referring to its own syntax, and it does so without contradiction.

Thus a sentence may contain a description of some of its own syntactical or structural properties. Note the resemblance of this to what we found in the preceding section, namely, that a self-reproducing automaton C contains within itself a description D(C) of its own structure. It does not, however, contain a description of its own state or contents (except for its initial state). This suggests that an automaton with a sensing element might be able to sense its own structure and construct and store a description of its structure. There is a problem in designing an automaton to do this, however.

Prima facie it might seem that an automaton could not store a description of its own structure because, however many cells it had, storage of the description would require more than that number of cells, in the manner of the Tristram

47. Kurt Godel, "Über formal unentscheidbare Sätze der _Principia Mathematica_ und verwandter Systeme," _Monatshefte für Mathematik und Physik_, _38_, 173-198 (1931).
In his lectures at the Institute of Advanced Study in 1934 (as reported in some mimeographed notes), Godel pointed out that his undecidable formula showed that Russell's solution of the paradoxes was too extreme because the undecidable formula (truly) says of itself that it is not provable. He agreed with Russell that there must be some limitation on what a sentence may say about itself and suggested that no sentence can properly talk about its own truth and falsity.
One may say more generally that a symbol (whether a word, phrase, or sentence) may properly refer to (or designate) itself as a physical object or to its physical properties and it may properly refer to its syntactical or structural properties and relations to other symbols, but it cannot refer to the relation between itself and what it refers to or designates. This latter relation is sometimes called the name relation, and is construed sufficiently broadly to include the relation between a sentence and its truth value as well as the relation between a proper name and what it names. See K. Reach, "The Name Relation and the Logical Antinomies," _The Journal of Symbolic Logic_, _3_, 97-111 (1938).

Shandy paradox.[48] This objection is of course not sound, because we may use indices, summation signs, and quantifiers in the description. You will recall that a certain master sequence is used to describe one square of the tape. In writing the description D(α) of the structure of automaton α one does not repeat this master sequence for each square of tape to be constructed, but rather uses the same sequence repeatedly in a manner familiar to all programmers. This can be done because the tape has a highly uniform structure.

It follows from these considerations that in order to produce a description of its own structure which is sufficiently small to be stored in that structure an automaton must detect some of the uniformities of its structure. It could, for example, systematically sense the structure of each of its cells and then analyze the results in order to find a more compact statement of them. Hence the amount of information a growing automaton can acquire and store about its own structure depends on what mechanical techniques there are for discovering uniformities of structure.

48. Tristram Shandy took two years to write the story of the first two days of his life. (Laurence Sterne, _The Life and Opinions of Tristram Shandy, Gentleman,_ Vol. 4, Ch. 13.)