6M-3938

Division 6 - Lincoln Laboratory
Massachusetts Institute of Technology
Lexington 73, Massachusetts

SUBJECT: THE LOGICAL STRUCTURE OF DIGITAL COMPUTERS
The Turing Machine

To: Class Registrants

Abstracts to: All Lincoln Division Heads and Group Leaders

From: W. A. Clark

Approved: W. A. Clark

Date: 5 October 1955

ABSTRACT: The logic of the Turing machine as a symbol manipulator is described and examples of counting and sorting are explained. A set of problems is included.

The research reported in this document was supported jointly by the Department of the Army, the Department of the Navy, and the Department of the Air Force under Air Force Contract No. AF 19(122)-458.

INTRODUCTION

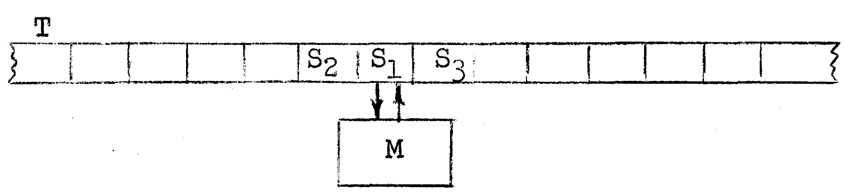
The subject of this course is the Logical Structure of Digital Computers. By "computer logic" one means the set of rules which the computer follows in carrying out its operations. Logical structure is to be distinguished from physical structure. The electronic components, wires, motors, and other hardware, comprising the physical structure of the computer, do no more than mechanize the operating rules defining the logical structure of the computer. It is completely irrelevant to the logic that the computer is built of vacuum tubes, or relays, or paper, as long as the rules are properly represented and followed.

The digital computer is essentially a symbol-manipulating machine. It accepts a set of symbols defining a problem to be solved and the data on which to operate. It then performs various operations on these symbols according to the rules defining its logical structure, and thereby produces a new set of symbols which comprise the solution to the problem. The rules thus take the form of a set of statements describing the manner in which certain symbols are to be replaced with new symbols. (Consider, for example, a particular sequence of five symbols. One useful rule in a computer dealing with this sequence would be: "If the first symbol is a '1', the second '+', the third '1', the fourth '=', and the fifth 'x', then replace the symbol 'x' with the symbol '2'.")

THE TURING MACHINE

A simple abstract model of the general symbol-manipulation process (and, therefore, of digital computer logical structure) was formulated by the British mathematician, Turing[1] as a conceptual aid in proving certain results in mathematical logic. He defined a class of symbol processing mechanisms which he called simply "automatic machines," but which are now generally known as "Turing machines."

The elements of the Turing machine are illustrated below:



[1] A. M. Turing: On Computable Numbers, with an Application to the Entscheidungs Problem, Proc. Lond. Math. Soc., series 2, V24, pp. 230-265, 1936.

A machine, M, having a finite number of internal configurations or states operates on an infinitely long tape, T, which is divided into cells. Each cell is capable of bearing one symbol from a specified, finite set of symbols, S_0, S_1, $S_2 \cdots S_n$, e.g., the alphabet, the digits, etc. The machine deals with one cell at a time (called the scanned cell) and can read the symbol in this cell and write a new symbol in its place, or move to the next cell to the right or to the left.

An operation is carried out concurrently with a jump from one machine state to another. This action is completely determined by the current state of the machine and the currently scanned symbol. Each move results in a new configuration of machine and tape in which the scanned symbol and machine state determine the next move, and so on.

A notation will now be described and some examples of Turing machines presented. This material will differ from that in Turing's original presentation, but the essential features are retained.

The operations to be discussed are:

1) Replace the scanned symbol, S_i, with the symbol S_j, abbreviated:

$$S_i : S_j$$

where i and j may have any particular values 0, 1, 2, $\cdots$ n. The symbol S_0 will represent blank tape to complete the description.

2) If the scanned symbol is S_i, move to the next cell on the right:

$$S_i : R$$

3) If the scanned symbol is S_i, move to the next cell on the left:

$$S_i : L$$

These operations can be abbreviated:

$$S_i : T_k \quad \begin{cases} T_k = \text{print } S_k & k = 0, 1, \cdots n \\ T_k = \text{move "R"} & k = n + 1 \\ T_k = \text{move "L"} & k = n + 2 \end{cases}$$

The rules by which the machine operates are then formulated in terms of these operations and the internal states of the machine. Each rule will be of the form:

<u>If</u> the machine is in state p $(p = 1, 2, \cdots M)$

<u>and</u> the scanned symbol is S_i $(i = 0, 1, \cdots N)$

<u>then</u> carry out operation T_k $(k = 0, 1, \cdots N + 2)$

<u>and</u> jump to state q $(q = 1, 2, \cdots M)$

which will be abbreviated to the quadruple:

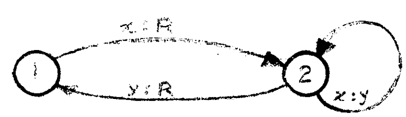
$$(p, S_i : T_k, q)$$

For example, the quadruple (3, x : R, 14) means "If the machine is in state 3 and the scanned symbol is x, move to the next cell on the right and jump to state 14."

The logical structure of the machine is thus specified by a finite set of quadruples of the above form.

The ordered pair of symbols <u>p, S_i</u> will be called a <u>determinant</u>, since it determines the <u>subsequent</u> move of the machine according to the remaining terms in the quadruple. To be consistent, the requirement is imposed that no two quadruples describing a given machine can have the same determinant.

The logical structure of a Turing machine may be represented conveniently as a network in which each node corresponds to a state of the machine and each directed branch between nodes corresponds to a jump between states. The branch is labeled with the operation which occurs during the jump. For example, the network

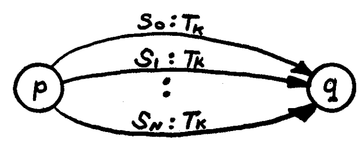


corresponds to the set of three quadruples

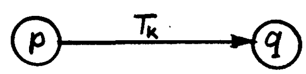
(1, x : R, 2)
(2, x : y, 2)
(2, y : R, 1)

A drawing of the network for a given machine is variously called a state diagram or transition diagram. Two conventions which simplify the drawing of transition diagrams for processes involving a large number of symbols S_0, S_1, S_2, • • • S_n are the following:

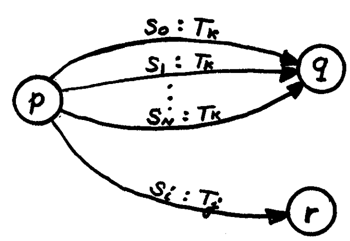
a) If all branches from a given state, p, lead to state q, and involve the same operation, T_k



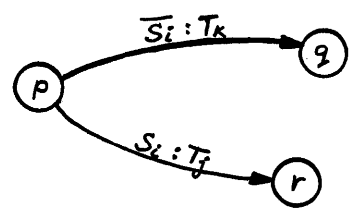
then the diagram may be abbreviated to



b) If all branches except the one for a particular scanned symbol, S_i, lead from state p to state q, and involve the same operation, T_k



then the diagram may be abbreviated to

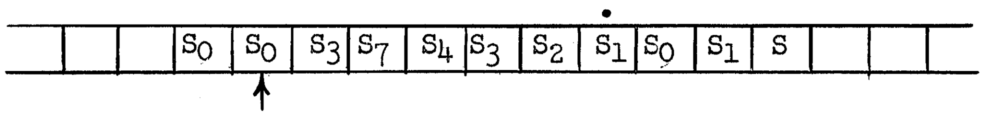


where $\overline{S_i}$ is read "not S_i."

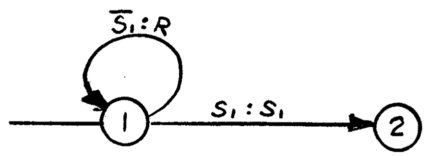
A few examples of Turing machines will now be given to illustrate the preceding definitions and concepts.

Example 1

Given a tape on which the symbols S_0, S_1, S_2, . . . S_n appear in any order and number. The machine starts in state 1 scanning any cell to the left of a cell holding the symbol, S_1



The machine is to "hunt" for the first cell to the right which holds S_1 and jump to state 2 when it is scanning this cell. The transition diagram is



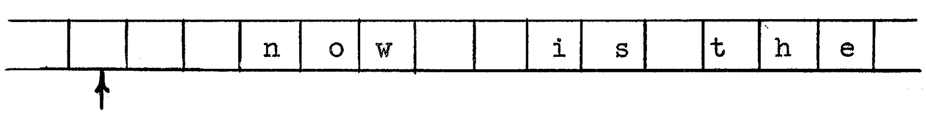
The machine remains in state 1 and scans the next cell on the right until S_1 is found, whereupon it jumps to state 2 with no change of symbol on the scanned cell.

Example 2

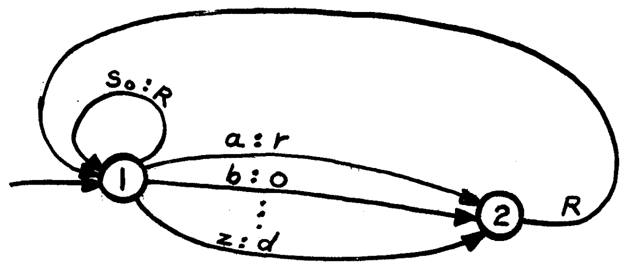
Consider a process of simple cryptographic encoding. Letters of the alphabet are to be scrambled according to the code

a → o
b → r
c → p
.
.
.
z → d

e.g., the word "cab" becomes "por", etc. The message to be encoded is printed on the tape with an arbitrary number of spaces between words.



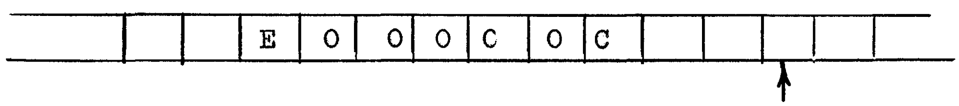
The machine starts in state 1, scanning the indicated cell. Its diagram is



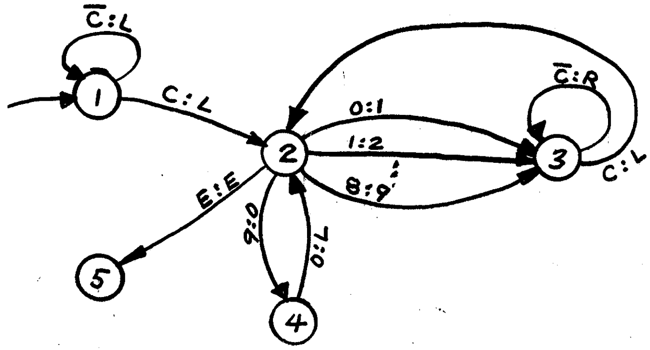
The symbol S_0 represents blank tape. The machine continues indefinitely, changing letters of the message to their equivalent code value.

Example 3

A block of five cells, each holding the digit 0, is separated from the rest of the tape by the symbol E on the left and C on the right.



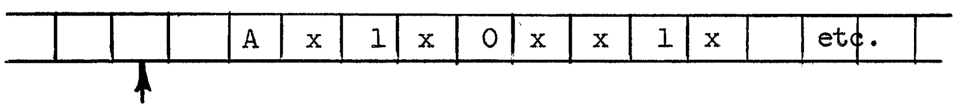
The machine starts in state 1, scanning a cell to the right of C. It is to print in succession the 5-digit decimal numbers from 0 to 99,999 on the marked block of cells, reset them to 0, and then jump to state 5. Its diagram is



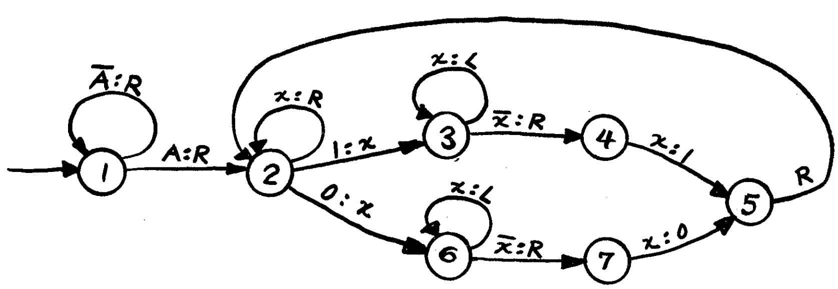
The machine finds the first digit following symbol C and jumps to state 2. If this digit is not 9, it is replaced with the next larger digit and the machine jumps to state 3. In state 3, the machine moves right until it finds C and backs up one cell, jumping back to state 2. When the digit is 9, it is replaced with 0 and the machine jumps to state 4 and moves left one cell (corresponding to a "carry" from one digit position to the next), returning to state 2 again. Clearly, this process results in the printing of the required sequence of numbers up to 99,999. At this point, the repeated sequence of transitions (2, 9 : 0, 4) (4, 0 : L, 2) resets the five cells to 0 and the process terminates with (2, E : E, 5).

Example 4

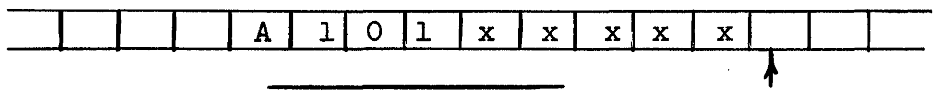
Consider a tape marked with A, 1, 0, and x in the following manner:



The 1's and 0's are intermixed with x's to the right of A in an arbitrary way. The machine starts in state 1, scanning a cell to the left of A, and is to compact the sequence of 1's and 0's into a block following A. The order of the 1's and 0's is to be retained. The diagram is



In state 1, the machine finds the first symbol to the right of A and jumps to 2. In state 2, the machine skips over cells holding x's and finds the nearest 0 or 1, replaces it with an x and jumps to state 6 or 3, respectively. In the case illustrated, the nearest non-x symbol is a 1, and the machine will go to state 3. In state 3, the machine finds the leftmost x, jumps to 4, prints a 1, and moves to the right, returning to state 2 via 5. Note that in taking either the upper or lower branch, (2, 1 : x, 3) or (2, 0 : x, 6) , the machine in effect "remembers" that the last symbol scanned was a 1 or 0, respectively. The reader should verify that the tape illustrated becomes:

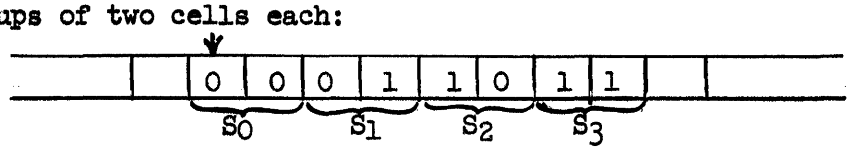


It is possible to restrict the Turing machine to two symbols, 0 and 1, without loss of generality.

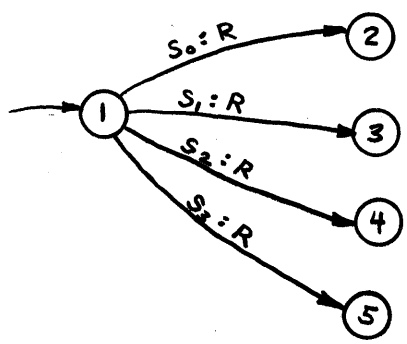
Consider a problem which is expressed in terms of four symbols: S_0, S_1, S_2, S_3. These can be encoded into groups of 1's and 0's in many ways. For example:

(1)	(2)
$S_0 = 000$	$S_0 = 00$
$S_1 = 001$	$S_1 = 01$
$S_2 = 010$	$S_2 = 10$
$S_3 = 100$	$S_3 = 11$

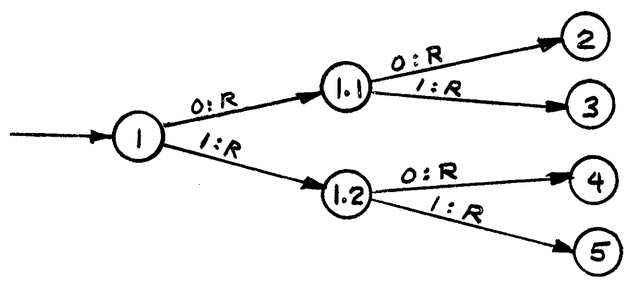
Case (2) will be discussed in more detail. The tape is divided into groups of two cells each:



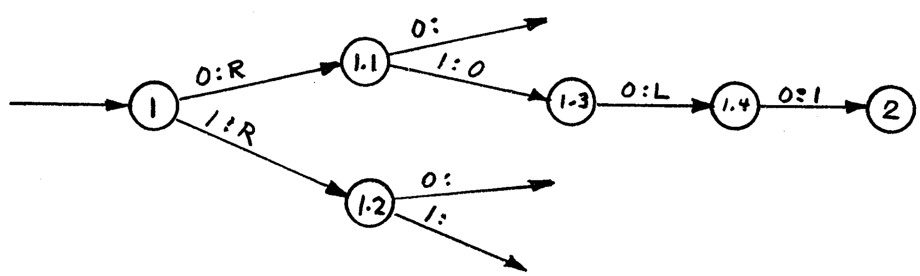
Then consider a section of the transition diagram of a machine dealing with S_0, S_1, S_2, S_3 which is of the form



This is equivalent to the following in which only 0 and 1 are used:



Similarly, a change of symbol, say $(1, S_1 : S_2, 2)$ can be represented as:



Note that the interpretation of the sequence of 1's and 0's depends on the direction of travel. This dependence could be eliminated by using a symmetrical code:

$$\begin{cases} S_0 = 000 \\ S_1 = 010 \\ S_2 = 101 \\ S_3 = 111 \end{cases}$$

We have now shown that the operations of any Turing machine can be reduced to the set

$$\begin{cases} \text{print } 0 \\ \text{print } 1 \\ \text{move right one cell} \\ \text{move left one cell} \end{cases}$$

This can be reduced still further. The only possible symbol-printing situations are:

$$\left.\begin{matrix}0 : 1\\1 : 0\end{matrix}\right\}$$

$$\left.\begin{matrix}0 : 0\\1 : 1\end{matrix}\right\}$$

The first two cases involve a change of information on the tape; the second two do not. We can define an operation "complement," abbreviated "C", to replace the two print operations. The cases involving no change of symbol are replaced with two complement operations done in sequence:

becomes

Thus, the set of Turing machine operations reduces to:

$$\left.\begin{matrix}C\\L\\R\end{matrix}\right\}$$

This can be reduced again to any of the three sets:

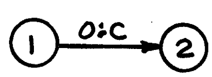
(1) $\left\{\begin{matrix}\text{CL}\\\text{R}\end{matrix}\right.$ CL complement and move left; R move right

(2) $\left\{\begin{matrix}\text{L}\\\text{CR}\end{matrix}\right.$ L move left; CR complement and move right

(3) $\left\{\begin{matrix}\text{CL}\\\text{CR}\end{matrix}\right.$ CL complement and move left; CR complement and move right

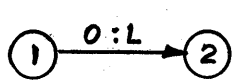
The proofs for (1) and (2) reduce to showing that "CL" can be broken into "C" and "L":

is the equivalent of



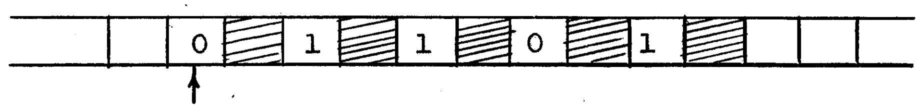
and

is the equivalent of

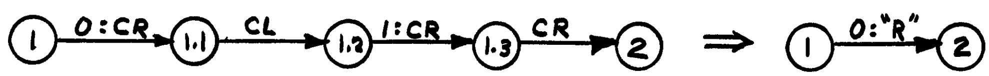


The other possible actions on 0 and 1 for cases (1) and (2) are proved in a similar manner.

Case (3) requires a different arrangement of symbols on the tape. Alternate cells are used to hold the symbols of the problem. The cells in between aid in "phasing" the complementing, but hold no significant information:



The equivalent forms are:



The other forms can be obtained directly from these.

PROBLEMS

1.1) A Turing machine tape is marked in the following manner:

			P	d_i	d_j	d_k	d_l	d_m			

Symbol "P" marks the beginning of a block of 5 different decimal digits, d_i, d_j, d_k, d_l, d_m. (e.g., 7 0 9 3 4).

Describe (draw a state-and-transition diagram of) a machine which will rearrange the digits in descending order on the 5 cells following "P" and move on to the right when finished. Any additional symbols and cells may be used in the process providing they are erased upon completion. The machine is to start on any cell to the left of "P".

1.2) Restate problem 1.1 in terms of a tape on which only the symbols "S_0" and "S_1" appear. (Invent a suitable code for the symbols "P", blank, 0, 1, · · · 9, etc., and describe the initial tape configuration). Redraw the state-and-transition diagram accordingly, using only the operations "complement" (change S_0 to S_1 or S_1 to S_0), "move right", and "move left", abbreviated "C","R", and "L", respectively.

1.3) Non-erasable tape can be defined as tape on which it is possible to write a symbol in a given cell only if the cell is blank. Show that it is possible for any Turing machine to use non-erasable tape.

6M-3938, Supplement 1

Division 6 - Lincoln Laboratory
Massachusetts Institute of Technology
Lexington 73, Massachusetts

SUBJECT: THE LOGICAL STRUCTURE OF DIGITAL COMPUTERS
The Universal Turing Machine

To: Class Registrants

Abstract to: J. C. Proctor, C. W. Farr

From: W. A. Clark

Date: 20 October 1955

Abstract: The complexity of a Turing machine can be measured by the number of quadruples defining its logical structure. At a certain level of complexity, it becomes possible to design a Turing machine which is universal in the sense that it can perform any calculation which any other Turing machine can perform. The Universal Turing Machine achieves this generality by having the ability to simulate other Turing machines, even those which are more complex. This simulation process is described in terms of the manipulation of the quadruples themselves. An example of a Universal Turing Machine is presented in detail, and a set of related problems is included.

W. A. Clark

WAC/jhk

The research reported in this document was supported jointly by the Department of the Army, the Department of the Navy, and the Department of the Air Force under Air Force Contract No. AF 19(122)-458.

THE UNIVERSAL TURING MACHINE

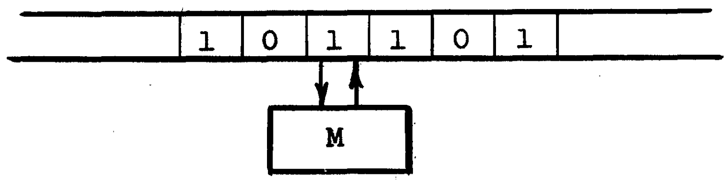
Preliminary Remarks

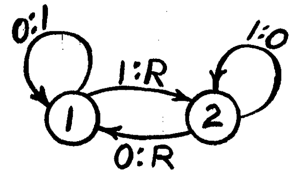
We have seen that a very general class of Turing machines can be defined which use a single tape on which only the symbols "0" or "1" appear in any cell ("0" corresponding to blank tape) and which perform only the operations "print 0," "print 1," "move right one cell," and "move left one cell." For convenience in subsequent reference, we will call these machines "Class A Turing machines" to distinguish them from other machines which use more symbols, different operations, more tapes, etc. Within a given class one might measure the complexity of a particular machine by the number of its internal states or, perhaps better, by the number of quadruples required to describe its logical structure. Thus, a 100-quadruple Class A Turing machine would be more complicated than a 10-quadruple Class A Turing machine.

It seems reasonable to attempt to relate a machine's complexity to its capability. For example, it is possible to combine a machine, M_1, which is capable of counting, with a machine, M_2, which is capable of ordering a set of numbers, and thereby obtain a more complicated machine, M_3, which is capable of both ordering and counting. It might be supposed that it is always possible to increase the generality of a machine by increasing its complexity in this way. The fact is, however, that there is a critical complexity beyond which no further increase in generality can be guaranteed! That is, at a certain level of complexity it becomes possible to design a Turing machine which is universal in the sense that it can perform any calculation which any other Turing machine can perform, even if the other machine is more complicated than the universal machine. The universal Turing machine achieves this generality by having the ability to simulate any machine whose calculation it is required to duplicate. The tape of the simulated machine appears as a designated sequence of cells on the tape of the universal machine. We will consider these points in more detail later.

Quadruple Manipulation

The simulation is itself a symbol manipulation process in which the symbols represent the set of quadruples describing the simulated machine. As an example of the manipulation of quadruples, consider the following simple Class A machine and the set of quadruples describing its logical structure:





(1, 0 : 1, 1)
(1, 1 : R, 2)
(2, 1 : 0, 2)
(2, 0 : R, 1)

This machine, starting in state 1, will print the sequence 10101010.. ..... Its operation will now be described in terms of the quadruples and scanned symbols:

Define <u>active</u> <u>quadruple</u> to be that quadruple which describes the action of the machine at any given point in the process. The <u>active</u> <u>determinant</u> is then the determinant found in the active quadruple (see page 3). The first term of the active quadruple will be called the <u>initial</u> <u>state</u>, and the last term, the <u>final</u> <u>state</u>. The second term of the active quadruple is, of course, the scanned symbol, and the third term is the specified operation.

In the illustration, if the machine is in state 1, the active quadruple is the second one in the list, namely:

(1, 1 : R, 2)

The machine will move one cell to the right and jump to the final state 2. State 2 thus becomes the initial state of the next machine action and the new scanned symbol is a "1". Therefore, the <u>next</u> active determinant will be:

<u>2, 1</u>

The next active quadruple can be found by again examining the list of quadruples and finding the quadruple which starts with the determinant 2, 1. In this case, it is the third determinant:

(2, 1 : 0, 2)

The machine will print a "0" and jump to the final state 2. The scanned symbol is now "0" and the next initial state is 2;

thus, the next active determinant will be

2, 0

and the next active quadruple is found to be

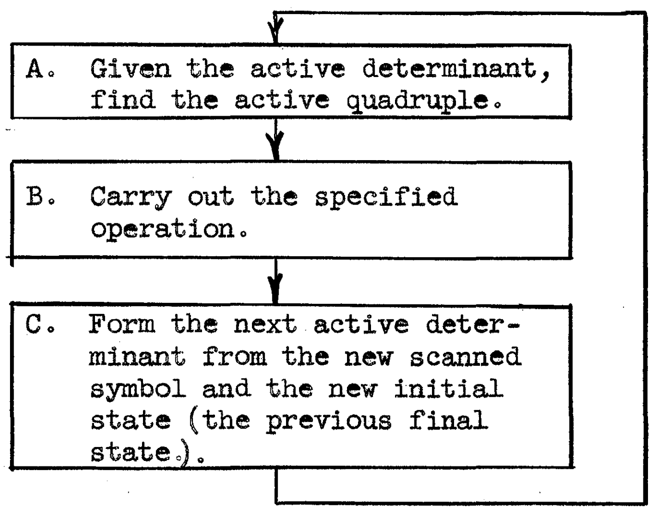
(2, 0 : R, 1)

etc.

This example illustrates the use of the set of quadruples in describing a sequence of machine actions.

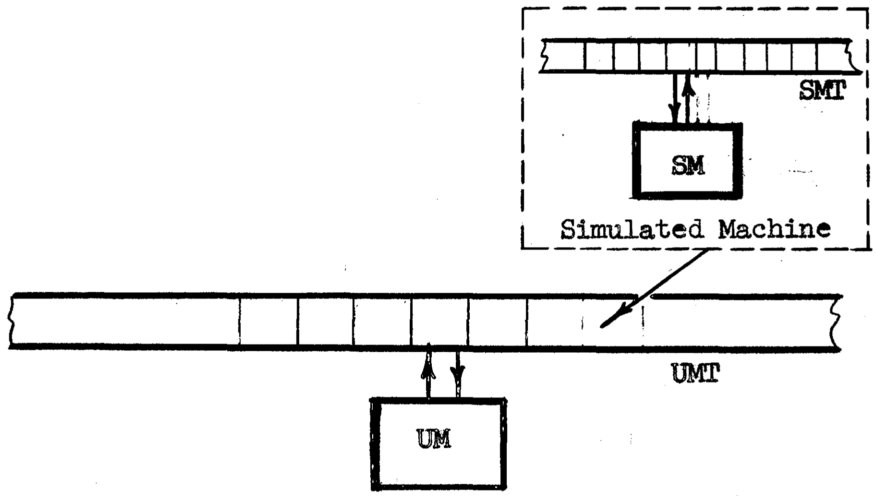
General Description of the Universal Turing Machine

The basic simulation process can be represented in the following way:



It is, of course, necessary to start this sequence with the first active determinant at A.

The Universal Turing Machine, UM, will be designed to carry out the above steps A, B, and C for any list of quadruples describing a given simulated machine, SM, which will be encoded in a suitable manner and printed on the universal machine tape, UMT. As we mentioned earlier, this tape will also hold a sequence of cells which correspond to the cells of the simulated machine tape, SMT. It will also require cells on which to print the active determinant symbols, and cells to mark significant points on the tape, e.g., the SMT scanned cell.



Universal Machine

Before UM begins its calculation, the quadruples defining SM are printed on UMT, the cells on UMT corresponding to SMT are marked to match the initial configuration of SMT, and the first active determinant is printed on UMT. UM is started in a specified initial state scanning a specified cell on UMT. It then carries out steps A, B, and C without end and prints the results of SM's calculation on the designated UMT cells corresponding to SMT. It is assumed that the set of symbols used by SM is included within the set of symbols used by UM.

Detailed Description of a Class A Universal Turing Machine

The general description of the previous section will now be related to a particular Class A Universal Turing Machine. It is seen that the first problem is that of finding a suitable code using the symbols 0 and 1 to represent quadruples, determinants, etc. The coding scheme presented here is the work of E. F. Moore who employed it in a description of a 3-tape universal machine[2]. The second problem is that of finding a suitable arrangement of the symbols on the universal machine tape. Finally, a description of the universal machine itself must be developed.

Consider first the coding of a Class A machine quadruple $(r, S_i : T_k, s)$. Each determinant must be one of the two forms:

---

[2]E. F. Moore: A Simplified Universal Turing Machine, Bell Telephone System Monograph 2098, presented at the Meeting of the Association for Computing Machinery, Toronto, Canada, Sept. 8, 1952.

r, 0

or

r, 1

where r takes on any integral value 1, 2, · · · M (for an M-state machine). The specified operation, T_k, is any one of the four forms:

0
1
R
L

and finally, the final state, s, is an integer from 1 to M.

The scheme proposed by Moore is the following:

Determinant:

Code	r, 0	as a block of	3r + 1 successive 1's
"	r, 1	" " " "	3r + 2 " 1's

Operation:

Code	0	as	0	immediately following determinant.
"	1	as	00	" " "
"	R	as	000	" " "
"	L	as	0000	" " "

Final State:

Code s as a block of 3s successive 1's immediately following the operation.

For example, the quadruple (1, 0 : 1, 1) would become:

111100111

and the quadruple (1, 1 : R, 2) would become:

11111000111111

A list of quadruples is coded by stringing together in any order the codes of member quadruples, separating one quadruple from the next by at least one 0. For example, the machine described on page 14

(1, 0 : 1, 1)
(1, 1 : R, 2)
(2, 0 : R, 1)
(2, 1 : 0, 2)

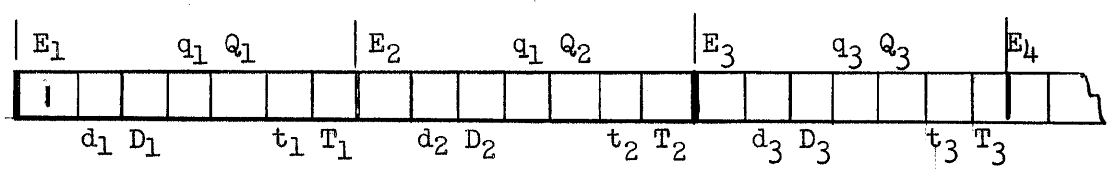
is completely (although not uniquely) coded by the sequence:

···000011110011101111111101111110001111111000111001111100011111100···

Notice that any block of ones in this sequence has a unique interpretation, e.g., a block of N ones represents a final state only if N is divisible by 3, etc. A block of zeros following a determinant represents an operation; a block of zeros following a final state is simply a separation.

We now proceed to describe the arrangement of symbols on the universal machine tape, UMT. UMT will need to be endless only on the right. The class of machines which the universal machine will simulate will also use tapes which are endless only on the right. This causes no restriction in the generality of these machines over that of doubly endless tape machines (see problem 2.1).

UMT will be divided into groups of 7 cells each. The quadruple list, active determinant, SMT cells, and various marking cells will be interleaved on UMT in the following manner:



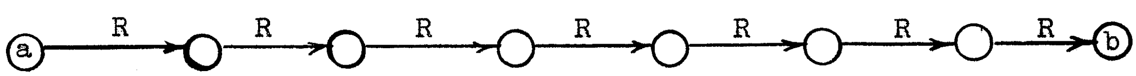
UMT

The E cells are used to mark the end of the tape (only E_1 holds a "1"; all other E cells hold "o". These are not changed.)

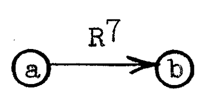
The D cells hold the active determinant.
The Q cells hold the list of quadruples.
The T cells correspond to the cells of the simulated machine tape.

The cells labeled d, q, and t are used to mark the following D, Q, and T cells, respectively. Only one cell of each will hold a "1" at any given point in the calculation. For example, a "1" in t_3 indicates that T_3 would be the scanned cell on SMT. The use of these marker cells will become clear later.

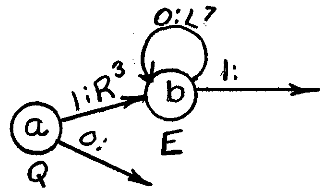
To go from cell Q_i to Q_{i+1} for example, it is necessary to slip over the intervening 6 cells. This process is diagrammed:



and will be abbreviated:



The process of locating the end of the tape will now be described. Suppose that UM is scanning some Q cell and is to find the cell E_1 if the scanned symbol is "1". The diagram is:



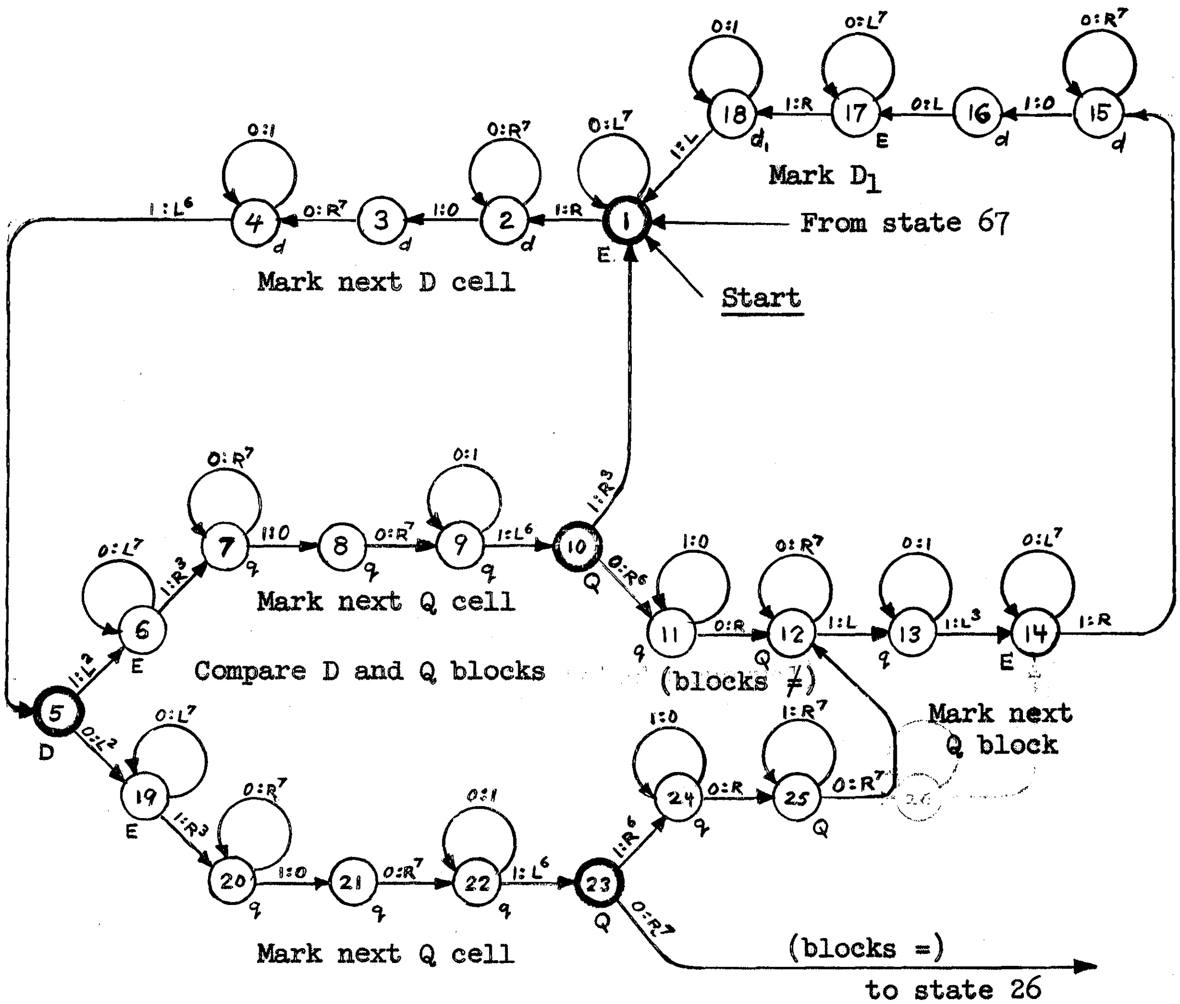
The letters Q and E below the state node in this diagram indicate which "phase" of the 7 tape phases the machine will end in after the transition to that state. The machine starts in state "a" scanning a Q cell, (i.e., in Q-phase). If the scanned symbol is a "1", the machine moves 3 cells to the right (to the nearest E cell) and jumps to state "b". Now in E phase, the machine scans successive E cells to the left until a "1" is found, which occurs on E_1 at the end of the tape.

UM will start in state 1 scanning E_1. The quadruple list for SM is printed on the Q cells beginning with Q_1 and q_1 holds a "1" (q_1 <u>marks</u> Q_1). The first active determinant is printed on D_1, D_2, D_3, etc., and d_1 holds a "1" (d_1 <u>marks</u> D_1). Finally, the T cells are marked according to SMT and the initial simulated scanned cell T_s will be marked (t_s holds a "1").

It will be noted that UM must move to the end of UMT before searching for a "1" on any marker cell in order to guarantee that the marked cell will not be missed.

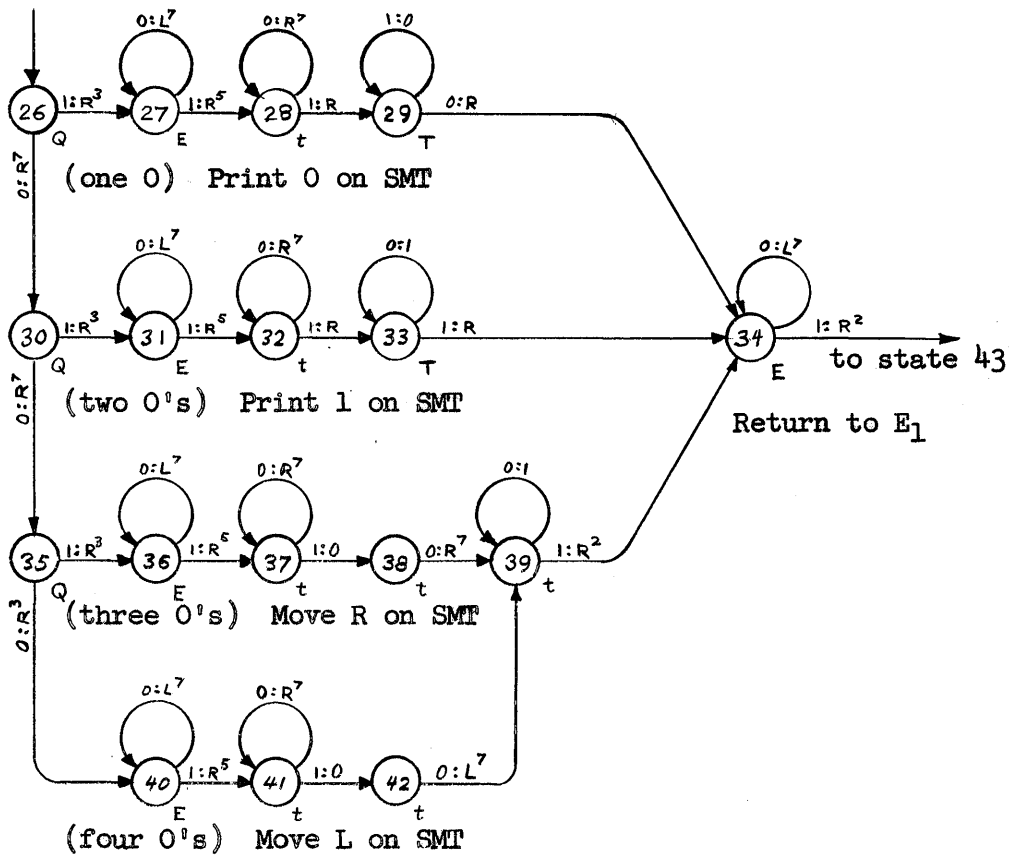
Parts A, B, and C of the operation of one particular Class A Universal Turing Machine will be described separately on the following pages.

A. Given the active determinant, find the active quadruple.



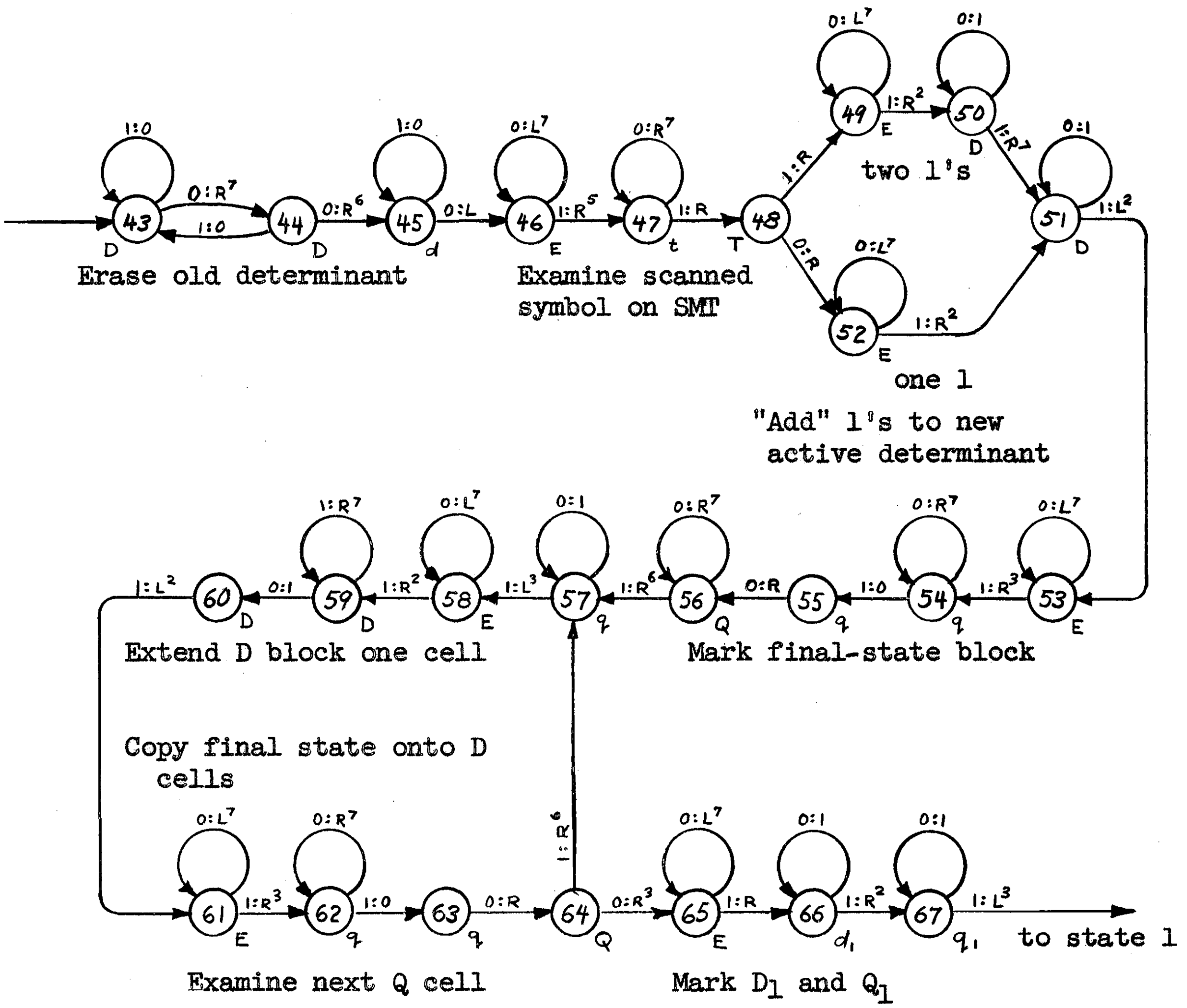
The next D cell is marked (a "1" is printed in the associated d cell) in states 2 to 4 and the previously marked D cell is examined (state 5). If it holds a "1", UM checks the currently marked Q cell (and marks the next Q cell) in state 10 and if it holds a "1", prepares to compare the next D and Q cells by returning to state 1. If the D and Q blocks are of unequal length, UM marks the next Q block of 1's, marks the beginning of the active determinant again, and starts the comparison with the new Q block by returning to state 1. When both D and Q cells hold 0's concurrently, the active quadruple has been found and UM jumps to state 26 to begin part B.

B. Carry out the specified operation.



The active quadruple has been located and UM determines which SM operation is specified by counting the number of successive 0's in the operation code. If one 0, UM finds the scanned cell on SMT and prints a "0"; if two 0's, a "1"; if three 0's, UM marks the T cell on the right of the currently marked T cell; if four 0's, UM marks the T cell on the left of the currently marked T cell. UM then returns to the end of the tape and moves on to part C.

C. Form the next active determinant from the new scanned symbol and the new initial state (the previous final state).



The old active determinant is erased (states 43 to 46) and the current scanned symbol on SMT is examined. If it is a "0", a "1" is printed on D_1 to start the next active determinant; if the scanned cell on SMT holds a "1", then a "1" is printed on both D_1 and D_2. UM then combines the block of "1's" which code the final state of the active quadruple with the one or two 1's now on the initial D cells, thus forming the new active determinant. D_1 and Q_1 are marked (1's printed on d_1 and q_1) and UM returns to state 1 to repeat A, B, and C for the next simulated transition.

PROBLEMS

2.1 Show that any problem which can be solved by a Turing machine using doubly infinite tape (infinitely long in both directions) can also be solved by a Turing machine using singly infinite tape.

2.2 Consider the class of Turing machines which use only the symbols "0" and "1" and which perform only the operations "print 0", "print 1", "move right one", "move left one", abbreviated 0, 1, R, L, respectively.

How many one-state machines of this class are there? Two-state? N-state?

2.3 A Turing machine calculation which never uses more than a finite length of tape might be called limited; otherwise non-limited. (For example, the machine (1,0:0,1) (1,1:R,1) performs a limited calculation if the tape holds a "0" to the right of the scanned cell.)

Write the set of quadruples for each one-state machine of the class defined in 2.2 which is non-limited for all possible arrangements of symbols on the tape.

2.4 Design a Universal Turing Machine using as few internal states as possible. The design may use any finite number of symbols to aid in reducing the number of states required. Make a count of the number of states and the number of quadruples.

6M-3938, Supplement 2

Division 6 - Lincoln Laboratory
Massachusetts Institute of Technology
Lexington 73, Massachusetts

SUBJECT: THE LOGICAL STRUCTURE OF DIGITAL COMPUTERS
Boolean Algebra

To: Class Registrants

Abstracts to: J. C. Proctor, C. W. Farr

From: W. A. Clark

Date: 2 November 1955

Abstract: The symbol-printing operations in a Class A Turing machine can be described in terms of the tape cells themselves. For example, a machine which performs the sequence "If cell A holds "1" or if cell B holds "0", print "1" on cell C" is described by the statement:

$$(A^1 \text{ or } B^0): C^1$$

The manipulative aspects of this notation can be exploited in demonstrating that the rules for printing symbols define a Boolean Algebra.

W. A. Clark

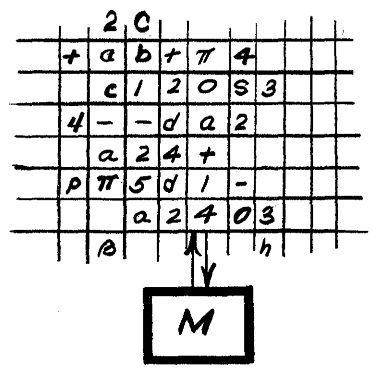
WAC/jhk

The research reported in this document was supported jointly by the Department of the Army, the Department of the Navy, and the Department of the Air Force under Air Force Contract No. AF 19(122)-458.

The Class A Universal Turing Machine described on the preceding pages provides us with one value for the critical complexity discussed earlier. A count of transitions shows that about 800 quadruples are required to describe this machine. Other universal machines using more symbols and fewer internal states have been designed (see also problem 2.4); Shannon has shown[3] that it is possible to design a two-state universal machine (and impossible to design a one-state universal machine) by using a large enough number of symbols.

Variations on the Turing Machine Theme

There are many possible variations to the Turing machine concept. In the example presented earlier, the machine operated on a linear array of symbols printed on an infinitely long tape. Another class of machine might be defined which operates on a two-dimensional array of symbols printed on an infinite plane.



The operations of this machine would include "move right," "move left," "move up," and "move down." Extension of these ideas to n-dimensional arrays readily follows.

A three-tape Turing machine was mentioned earlier[4]. In this case, the machine deals with three separate linear arrays scanning three cells simultaneously but operating on only one tape at a time. The transitions are described as sets of sextuples rather than quadruples, each determinant consisting of the initial state and the three scanned symbols. Again, extension to machines using more tapes or several n-dimensional arrays is possible.

Von Neumann has suggested a parts-manipulating machine analogous

[3] C. E. Shannon and others: "Automata Studies," Princeton University Press (to be published shortly.)

[4] E. F. Moore: Op. cit.

to the Turing symbol-manipulating machine. This machine would operate in an environment containing hardware of various kinds such as nuts, bolts, wire, vacuum tubes, etc., and would construct another machine from these parts. Again, it is possible to design a universal constructing machine capable of constructing anything that any other constructing machine can construct. A case of particular interest is that of a machine which constructs a copy of itself. With the current trend toward automation, some of these ideas are being applied in practical situations.

Summary Remarks

Before moving on to further discussion of the Turing machine and its relation to other topics, let us list some of the items which have been introduced in the preceding pages:

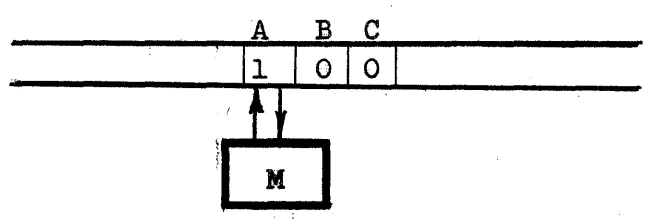
1. Logical structure
2. Operating rules
3. Stable states
4. Transitions
5. Symbol manipulation
6. Coding
7. Machine complexity
8. Simulation
9. Universality

These are all items will will be discussed in more detail during the remainder of the course. It is interesting that the conceptually simple Turing machine serves to introduce so many of the basic ideas in the subject of digital computers.

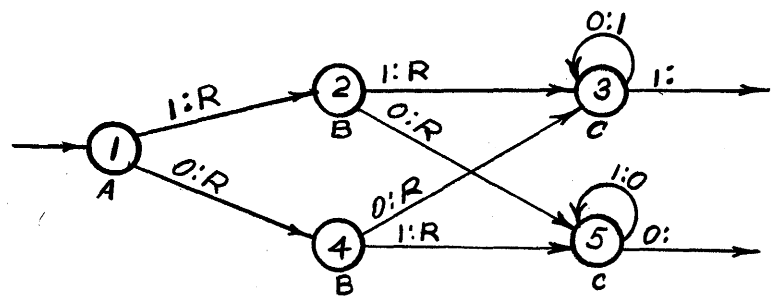
BOOLEAN ALGEBRA

We now proceed to develop a manipulative notation which enables us to describe the action of two-symbol machines in terms of the cells themselves. This will lead to a kind of symbol-printing algebra which will be shown to have the properties of Boolean algebra.

Consider the following simple Class A Turing machine which operates on cells labeled A, B, and C.



for which the transition diagram is



The machine prints "1" on C if A and B hold the same symbol and prints a "0" on C if A and B hold different symbols. The symbol finally appearing in cell C depends on the symbols in A and B. There are, of course, several kinds of dependence relations possible, and the machine illustrated mechanizes only one of these relations. A word description of the illustrated process in terms of basic Turing machine operations would consist of the following pair of statements:

1. If cell A holds "1" and cell B holds "1", <u>or</u> if cell A holds "0" and cell B holds "0", <u>then</u> print "1" on cell C.

2. If cell A holds "1" and cell B holds "0", <u>or</u> if cell A holds "0" and cell B holds "1", <u>then</u> print "0" on cell C.

A notation which simplifies this description is one which employs superscripts to denote the symbol held in a given cell:

A^{0} will mean "there is a '0' in cell A"

and A^{1} will mean "there is a '1' in cell A"

Then we might agree that

A^{0}: will mean "<u>if there is</u> a '0' in cell A"

and $:C^{1}$ will mean "<u>print</u> a '1' in cell C"

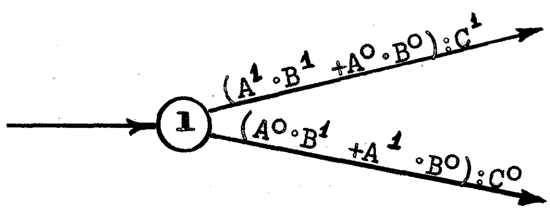
With these abbreviated forms, the statements describing the action of the illustrated machine become:

$$\left.\begin{array}{l}(A^1 \text{ and } B^1 \text{ or } A^0 \text{ and } B^0): C^1 \\ (A^0 \text{ and } B^1 \text{ or } A^1 \text{ and } B^0): C^0\end{array}\right\}$$

The remaining simplification involves replacing "and" and "or" with shorter symbols. We will choose "∘" to replace "and" and "+" to replace "or." With these changes the statements become:

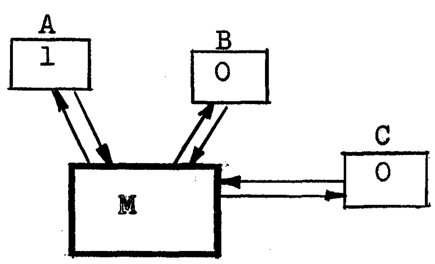
$$\left.\begin{array}{l}(A^1 \circ B^1 + A^0 \circ B^0): C^1 \\ (A^0 \circ B^1 + A^1 \circ B^0): C^0\end{array}\right\}$$

and the transition diagram can be redrawn in the form:



which is considerably simpler and more compact than the original diagram.

It is interesting to note that in this new description the operations "R" and "L" do not appear. It is no longer a requirement that the cells A, B, and C be adjacent cells in a linear array. In fact, the new transition diagram equally well describes the action of a discrete-state machine which deals with several independent cells simultaneously:



We will return to this idea later.

Table of Combinations

The possible outcomes and the corresponding conditions of A and B can be represented conveniently in a table. On the left are listed all combinations of symbols found in A and B, and on the right the resulting symbol in C:

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

Thus, the first line of the table corresponds to $A^0 \cdot B^0 : C^1$, etc. The number of lines, k, in such a table is given by $k=2^n$, where n is the number of cells determining the symbol to be printed. The number of different tables is 2^k, i.e.,

$$2^{2^n} = \text{number of functionally different machines which print "1" or "0" depending on the symbols held in n cells.}$$

It should be noted that directions for printing only the 1's are sufficient to determine the complete table. Thus, it is sufficient to describe the illustrated machine by:

$$(A^1 \cdot B^1 + A^0 \cdot B^0) : C^1$$

from which the table is written:

A	B	C
0	0	1
0	1	
1	0	
1	1	1

→

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

0's appearing in all unfilled positions. Similarly, directions for the printing of 0's are also sufficient and may result in simpler descriptions in some cases. For example,

$$A^0 \cdot B^0 : C^0 \qquad \text{and} \qquad (A^1 \cdot B^0 + A^0 \cdot B^1 + A^1 \cdot B^1) : C^1$$

both describe the same machine.

We will say that two cells, C and D, are <u>equivalent</u> if C holds a "1" whenever D holds a "1" and C holds "0" whenever D holds "0". Thus, from a table, e.g.,

A	B	C	D	E
0	0	1	1	0
0	1	0	0	0
1	0	0	0	1
1	1	1	1	1

it is seen that C is equivalent to D and E is equivalent to A. These will be written C=D and E=A, respectively.

Thus, given

$$x^1 : f^1$$

x	f
0	0
1	1

it is seen from the table that x=f. This could have been obtained from the statement $x^1 : f^1$ by dropping superscripts and replacing ":" with "=", and the converse is of course also true. Using this rule, we would also obtain $x^o = f^o$ from $x^o : f^o$.

From $x^o : f^1$ we would obtain $f = x^o$. The table is:

x	f
0	1
1	0

Inspection of the table shows that the symbol in cell f is the <u>complement</u> of the symbol in cell x, i.e., f=complement of x. Thus, x^o will be read "x complement" or "complement of x", the superscript "o" indicating the complementation.

From

$$x^o : f^1$$
$$f^o : g^1$$

x	f	g
0	1	0
1	0	1

we conclude that $x = (x^o)^o$, the double-complement rule.

Now we define two cells 1 and 0 in the following way: Cell 1 always holds the symbol "1" (see, for example, cell E_1 of the UM described earlier) and cell 0 always holds the symbol "0". Evidently:

$$1^o = 0 \qquad \text{and} \qquad 0^o = 1$$

Consider the following printing statements for cells f_1 through f_6:

$$(1^1 + 1^1) : f_1^1$$
$$(1^1 + 0^1) : f_2^1$$
$$(0^1 + 0^1) : f_3^1$$
$$1^1 \cdot 1^1 : f_4^1$$
$$1^1 \cdot 0^1 : f_5^1$$
$$0^1 \cdot 0^1 : f_6^1$$

The table of combinations is then

1	0	f_1	f_2	f_3	f_4	f_5	f_6
1	0	1	1	0	1	0	0
1	0	1	1	0	1	0	0
1	0	1	1	0	1	0	0
1	0	1	1	0	1	0	0

from which we conclude that

$$1+1=1 \qquad 1\cdot 1=1$$
$$1+0=1 \qquad 1\cdot 0=0$$
$$0+0=0 \qquad 0\cdot 0=0$$

These results illustrate properties of an arithmetic which is like ordinary arithmetic for the "dot" operation (multiplication) but unlike ordinary arithmetic for the "plus" operation (addition). We have described the fundamental operations of Boolean Algebra.

Evidently these operations are commutative, i.e., "x or y" is the same as "y or x" and "x and y" is the same as "y and x". Symbolically

$$x + y = y + x \qquad \text{and} \qquad x\cdot y = y\cdot x$$

The "and" and "or" operations are also associative. Using the parenthesis to denote grouping, the following statements hold:

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

Consequently, the order and grouping of symbols in any Boolean expression is arbitrary. The distributive properties of the "and" and "or" operations can be established by constructing the table of combinations for the forms:

$$(x^1\cdot y^1+x^1\cdot z^1):f_1^1 \qquad (x^1+y^1)\cdot(x^1+z^1):f_3^1$$
$$x^1\cdot(y^1+z^1):f_2^1 \qquad (x^1+y^1\cdot z^1):f_4^1$$

x	y	z	f_1	f_2	f_3	f_4
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Comparison of columns in the table shows that $f_1=f_2$ and $f_3=f_4$, i.e.,

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

As in ordinary algebra, then, one can "multiply" through a "sum" (recall the process of "factoring"). Unlike ordinary algebra, Boolean algebra permits one to "add" through a "product". These forms will occur quite often in subsequent work.

Memorandum 6M-3938, Supplement 3

Division 6 – Lincoln Laboratory
Massachusetts Institute of Technology
Lexington 73, Massachusetts

SUBJECT: THE LOGICAL STRUCTURE OF DIGITAL COMPUTERS
Boolean Algebra (continued)

To: Class Registrants

From: W. A. Clark

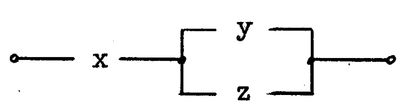
Date: 22 November 1955

Abstract to: J. C. Proctor, C. W. Farr

Abstract: Theorems in Boolean algebra can be proved by constructing a table of combinations for given expressions and finding equivalent entries in the table. The manipulative character of the algebra makes possible the proof of additional theorems without recourse to a table of combinations. Certain expressions are <u>dual</u> in form to one another. Examples:

$$\left.\begin{array}{r} x + 1 = 1 \\ x \circ 0 = 0 \end{array}\right\} \qquad \left.\begin{array}{l} x + yz = (x + y)(x + z) \\ x(y + z) = xy + xz \end{array}\right\}$$

Boolean algebra is also related to statements about paths in networks. A path between two nodes in the network can be associated with the symbol '1' and the absence of a path with the symbol '0'. A typical Boolean expression and its network representation follows:



$x \cdot (y + z)$

W. A. Clark

WAC/jhk

The research reported in this document was supported jointly by the Department of the Army, the Department of the Navy, and the Department of the Air Force under Air Force Contract No. AF 19(122)-458.

By using the general process of writing out the table of combinations for various expressions and then comparing columns to find equivalent forms, the theorems of Boolean algebra can be developed. As further examples, consider the following expressions:

$(x^1 + x^1): f_1^1$

$x^1 \cdot x^1: f_2^1$

x	f_1	f_2
0	0	0
1	1	1

Examination of the table shows that $f_1 = f_2 = x$, i.e.,

$$x + x = x$$
$$x \cdot x = x$$

and hence, by continuation

$$x + x + \ldots + x = x$$
$$x \cdot x \cdot \ldots \cdot x = x$$

Similarly, the expressions

$(x^1 + 1^1): f_1^1$

$x^1 \cdot 1^1: f_2^1$

$(x^1 + 0^1): f_3^1$

$x^1 \cdot 0^1: f_4^1$

0	1	x	f_1	f_2	f_3	f_4
0	1	0	1	0	0	0
0	1	1	1	1	1	0

lead to the conclusion that

$$x+1 = 1 \qquad x+0 = x$$
$$x \cdot 1 = x \qquad x \cdot 0 = 0$$

Notice that with the exception of the first one, these forms are like those of ordinary algebra.

Finally, from the expressions

$(x^1 + x^0): f_1^1$

$x^1 \cdot x^0: f_2^0$

x	f_1	f_2
0	1	0
1	1	0

we conclude that

$$x + x^0 = 1$$

$$x \cdot x^0 = 0$$

Additional theorems can also be obtained from the ones already established without recourse to a table of combinations. For example, it can be shown that

$$x + x \cdot y = x$$

by using the first of the distributive properties already proved:

$$\begin{aligned} x + xy &= x \cdot (1 + y) \\ &= x \cdot (1) \\ &= x \end{aligned}$$

Similarly, it can be shown that

$$x + x^{o}y = x + y$$

by using the second distributive property:

$$\begin{aligned} x + x^{o}y &= (x + x^{o}) \cdot (x + y) \\ &= (1) \cdot (x + y) \\ &= x + y \end{aligned}$$

The advantages of the manipulative character of the algebra in the above examples are apparent.

<u>The theorem of DeMorgan</u> relates an expression and its complement. Consideration of the forms:

$$(x^{1} + y^{1}) : f_1^{0}$$

$$x^{o} \cdot y^{o} : f_2^{1}$$

x	y	f_1	f_2
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

leads to the conclusion that since $f_1 = (f_1^{o})^{o}$ and $f_1 = f_2$,

$$(x + y)^{o} = x^{o} \cdot y^{o}$$

This can be extended to expressions involving more variables by using the associative property in the following way:

$$\begin{aligned} \{x + y + z\}^{o} &= \{x + (y + z)\}^{o} \\ &= x^{o} \cdot (y + z)^{o} \\ &= x^{o} \cdot y^{o} \cdot z^{o} \end{aligned}$$

A more general form of DeMorgan's theorem is the following:

$$f^{o}(x_1, x_2, \ldots x_n, \cdot, +) = f(x_1^{o}, x_2^{o}, \ldots x_n^{o}, +, \cdot)$$

that is, the complement of an expression involving certain variables and the "dot" and "plus" is obtained by replacing each variable by its complement and interchanging the "dot" and "plus". For example:

$$(x + y \cdot z)^{o} = x^{o} \cdot (y^{o} + z^{o})$$

PROBLEMS

3.1 Show that the complement of any Boolean function may be obtained by complementing all of the function variables and replacing "·" with "+" and "+" with "·". That is, show that

$$f^o(x_1, x_2, \ldots x_n, \cdot, +)$$
$$= f(x_1^o, x_2^o, \ldots x_n^o, +, \cdot)$$

3.2 Find the complements of the following functions:

$$f = ab + cd$$

$$f = (a + b)(c + d)(b^o + d)$$

$$f = (ab^oc + bc^od)^o + a^o$$

Check the results by showing that $f \cdot f^o = 0$ and $f + f^o = 1$ for each function.

3.3 Prove the following:

$$xy + x^oz = (x + z)(x^o + y)$$

$$xy + x^oz + yz = xy + x^oz$$

Duality

Using DeMorgan's theorem one can obtain new theorems from those already proved. For example, consider

$$x(y + z) = xy + xz$$

the first distributive property theorem. Complement both sides of the identity:

$$\{x(y + z)\}^{o} = \{xy + xz\}^{o}$$

$$x^{o} + y^{o}z^{o} = (x^{o} + y^{o})(x^{o} + z^{o})$$

which is recognized as the second distributive property theorem applied to complemented variables. Consequently, the two distributive theorems are related to each other; we say that they are dual in form to one another. For any Boolean expression, a dual expression can be obtained by complementation of the expression followed by complementation of all variables. For example:

$$x + x = x$$

Complementation of each side gives

$$(x + x)^{o} = x^{o}$$

$$x^{o} \cdot x^{o} = x^{o}$$

and final complementation of all variables yields

$$x \cdot x = x$$

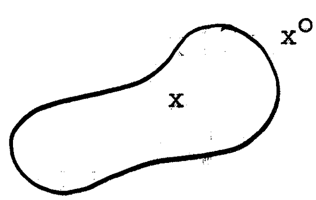
as the dual expression. Other expressions and their duals follow:

$$\left.\begin{aligned} x + 1 &= 1 \\ x \cdot 0 &= 0 \end{aligned}\right\} \qquad \left.\begin{aligned} x + x^{o} &= 1 \\ x \cdot x^{o} &= 0 \end{aligned}\right\}$$

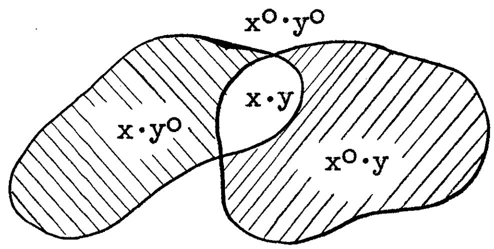
$$\left.\begin{aligned} x + xy &= x \\ x(x + y) &= x \end{aligned}\right\} \qquad \left.\begin{aligned} x + x^{o}y &= x + y \\ x(x^{o} + y) &= xy \end{aligned}\right\}$$

Venn Diagram

Another way of visualizing the expressions of Boolean algebra is to relate the states of cells to abstract areas. The area inside the following closed figure represents the state in which cell x holds '1' and the remaining area represents the state in which cell x holds '0':



By overlapping figures of this kind we obtain a Venn diagram(5) of Boolean expressions of the cells involved. For two variables, the Venn diagram is the following:



By relating the four distinct areas one can establish the theorems of Boolean algebra for two variables. For example, by joining the region $x \cdot y^{o}$ to the region $x \cdot y$, one obtains the region x. Symbolically:

$$x \cdot y^{o} + x \cdot y = x$$

Extension of the technique to more variables is, of course, possible.

Network Representation

The relationships of Boolean algebra can also be applied to statements about paths in a network of interconnected nodes.(6) The following definitions are used:

$$P \bullet\!\!-\!\!-\; x^{1} \;-\!\!-\!\!\circ\, Q$$

will mean "there is a path from P to Q if x holds '1' and there is no path from P to Q if x holds '0'. Similarly

$$P \circ\!\!-\!\!-\; x^{o} \;-\!\!-\!\!\circ\, Q$$

will mean "there is a path from P to Q if x holds '0' and no path if x holds '1'." Then the following correspondences hold:

$$P \circ\!\!-\!\!-\; 1^{1} \;-\!\!-\!\!\circ\, Q \Leftrightarrow P \circ\!\!-\!\!-\!\!-\!\!-\!\!\circ\, Q$$

$$P \bullet\!\!-\!\!-\; 0^{1} \;-\!\!-\!\!\circ\, Q \Leftrightarrow P \circ\!\!-\!\!-\quad -\!\!-\!\!\circ\, Q$$

so that a path is associated with the symbol '1' and the absence of a path is associated with the symbol '0'.

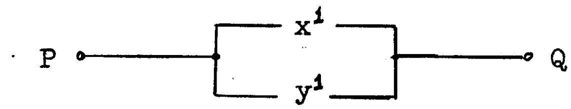
---

[5] H. Venn, Symbolic Logic, 1881

[6] C. E. Shannon, A Symbolic Analysis of Relay and Switching Circuits, Transactions of A.I.E.E., Vol. 57, 1938.

The relations "and" and "or" can be represented as follows:

$x \cdot y$



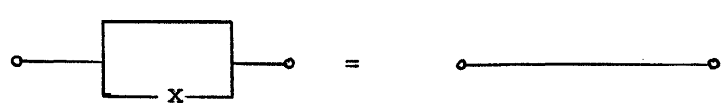
$x + y$

In the first case, there is a path from P to Q only if x holds '1' and y holds '1'. In the second case, there is a path from P to Q if x holds '1' or if y holds '1' (or both).

The theorems of Boolean algebra can then be represented as equivalence relations between suitable networks. Some of the theorems and their network representation follow (1's superscripts and node labels will be dropped):

$x \cdot x = x$

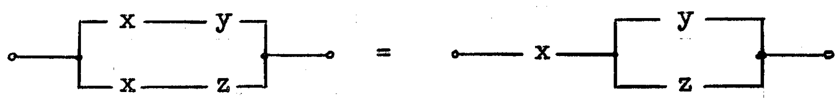
$x + x = x$



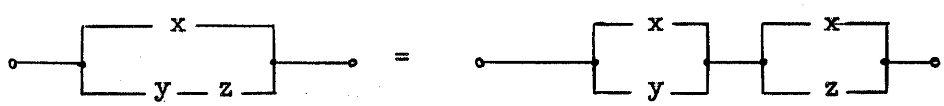
$x + 1 = 1$

$x \cdot x^o = 0$

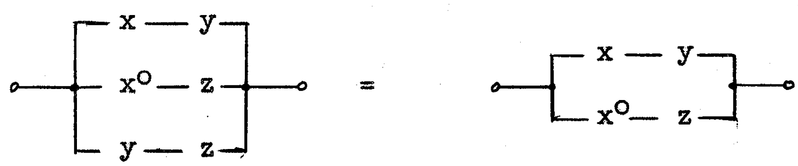
$x + x^o = 1$



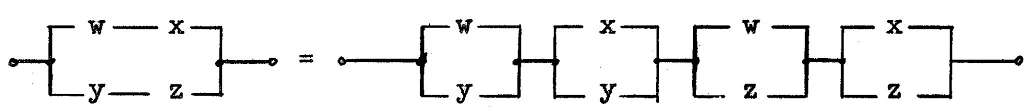
$$x \cdot y + x \cdot z = x(y + z)$$



$$x + y \cdot z = (x + y)(x + z)$$



$$xy + x^{o}z + yz = xy + x^{o}z$$



$$wx + yz = (w + y)(x + y)(w + z)(x + z)$$

For a more complete description of the application of Boolean algebra to the synthesis and simplification of networks of this kind, the reader is referred to the text:

Keister, Ritchie, Washburn, The Design of Switching Circuits, Van Nostrand, 1951.

Memorandum 6M-3938, S4

Division 6 – Lincoln Laboratory
Massachusetts Institute of Technology
Lexington 73, Massachusetts

SUBJECT: THE LOGICAL STRUCTURE OF DIGITAL COMPUTERS
Synthesis of Boolean Machines

To: Class Registrants

From: W. A. Clark

Date: 13 December 1955

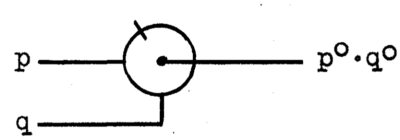
Abstracts to: J. C. Proctor, C. W. Farr

Abstract: It is possible to synthesize any of the two-cell Boolean machines using connective elements which implement the Sheffer stroke functions

$$p^o \cdot q^o \quad \text{or} \quad p^o + q^o$$

A convenient symbol is based on the matrix form of the table of combinations:

	f_1	q = 0	q = 1
p	0	1	0
	1	0	0



W. A. Clark

WAC/jhk

The research reported in this document was supported jointly by the Department of the Army, the Department of the Navy, and the Department of the Air Force under Air Force Contract No. AF 19(122)-458.

The four Boolean functions of one variable are defined by the following table:

x	0	$(x)^\circ$	x	1
0	0	1	0	1
1	0	0	1	1

Any Boolean expression involving only one variable must reduce to one of these four forms regardless of its apparent complexity. Examples:

$$(x + (x)^\circ)^\circ + x^\circ \circ (x \circ x \circ x + x) = 0$$

$$(((((x)^\circ)^\circ)^\circ)^\circ)^\circ = (x)^\circ$$

Design a Turing machine which, when supplied a tape bearing <u>any</u> Boolean expression involving one variable, will print the correct reduced form of the expression. The symbols appearing initially on the tape are:

$$x,\ ^\circ,\ +,\ \circ,\ (,\ ),\ =,\ b$$

Parentheses will always be written explicitly with any complement:

$(x)^\circ$ instead of x°

and "$\circ$" will always be written explicitly between factors

$x \circ (x + x \circ (x)^\circ)$

The expression is written one symbol to a cell and terminates with the symbol "="; all other cells are blank (symbol "b"). For example, the last expression appears as

														↓		
b	b	x	$\circ$	(	x	+	x	$\circ$	(	x	)	$^\circ$	)	=	b	b

The machine starts in state 1 scanning the cell which holds "=" and is to print the correct reduced form on the cells immediately following the "=" symbol. Any additional symbols may be used but must be erased upon completion.

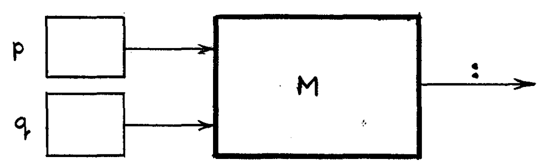
SYNTHESIS OF BOOLEAN MACHINES

The Boolean algebra description applies quite generally to two-symbol machines which manipulate information in several cells simultaneously according to fixed rules.* We will call these machines *Boolean machines*[7] and will now consider their synthesis and logical structure.

The simplest significant Boolean machine is a one-state, one-cell structure. In the more interesting cases, it is a complex machine with many internal states and many peripheral cells. What is desired in synthesizing these more complicated machines is a graphic description which emphasizes the logical relationships between the cells of a given machine and which thus augments the state and transition diagram describing the action of the machine. We will now proceed to make more explicit the possible forms of these logical relationships.

Two-Cell Machines

Consider first the one-state machine in which the printed symbol depends on the symbols initially held in two cells, p and q:



Sixteen different machines are possible; they are described by the 16 Boolean expressions, f_0, f_1, ... f_{15}, enumerated in the following table of combinations:

p q	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0 0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0 1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

It is noted that $f_0 = 0$ and $f_{15} = 1$. The Boolean expressions can be obtained by writing out the terms involving p and q which give rise to "1's" in the table:

---

*Refer page 27.

[7] I. S. Reed: "Some Mathematical Remarks on the Boolean Machine," Project Lincoln Technical Report #2, Dec. 1951

$f_0 = 0$	$f_8 = p \cdot q$
$f_1 = p^o \cdot q^o$	$f_9 = p^o \cdot q^o + p \cdot q$
$f_2 = p^o \cdot q$	$f_{10} = p^o \cdot q + p \cdot q$
$f_3 = p^o \cdot q^o + p^o \cdot q$	$f_{11} = p^o \cdot q^o + p^o \cdot q + p \cdot q$
$f_4 = p \cdot q^o$	$f_{12} = p \cdot q^o + p \cdot q$
$f_5 = p^o \cdot q^o + p \cdot q^o$	$f_{13} = p^o \cdot q^o + p \cdot q^o + p \cdot q$
$f_6 = p^o \cdot q + p \cdot q^o$	$f_{14} = p^o \cdot q + p \cdot q^o + p \cdot q$
$f_7 = p^o \cdot q^o + p^o \cdot q + p \cdot q^o$	$f_{15} = 1$

Many of these expressions can be simplified. For example:

$$\begin{aligned} f_3 &= p^o \cdot q^o + p^o \cdot q \\ &= p^o(q^o + q) \\ &= p^o \end{aligned}$$

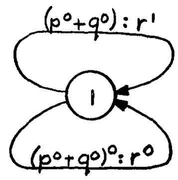
and

$$\begin{aligned} f_7 &= p^o \cdot q^o + p^o \cdot q + p \cdot q^o \\ &= p^o(q^o + q) + p \cdot q^o \\ &= p^o + p \cdot q^o \\ &= (p^o + p)(p^o + q^o) \\ &= p^o + q^o \end{aligned}$$

If these reductions are carried out for all such cases, the following list of minimum forms is obtained:

$f_0 = 0$	$f_6 = pq^o + p^oq$	$f_{11} = p^o + q$
$f_1 = p^oq^o$	$f_7 = p^o + q^o$	$f_{12} = p$
$f_2 = p^oq$	$f_8 = pq$	$f_{13} = p + q^o$
$f_3 = p^o$	$f_9 = p^oq^o + pq$	$f_{14} = p + q$
$f_4 = pq^o$	$f_{10} = q$	$f_{15} = 1$
$f_5 = q^o$		

The transition diagram for any of the one-state, two-cell machines can be obtained directly from this list. For example, the machine described by f_7 , M_7 has the diagram:



In order to represent the logical relationships

$$p^o + q^o$$

and

$$(p^o + q^o)^o = p \cdot q$$

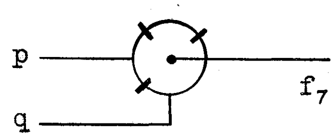
for M_7 in terms of the cells p and q in a graphic construction, we will introduce a symbolism based on the table of combinations. Note that the table for f_7 can be drawn in either the form:

p	q	f_7
0	0	1
0	1	1
1	0	1
1	1	0

or the matrix form:

	f_7	q = 0	q = 1
p	0	1	1
	1	1	0

The four positions in the matrix form can be related to the quadrants of a circle. Thus, the expression f_7 might be represented by a connective element of the following sort[8]:

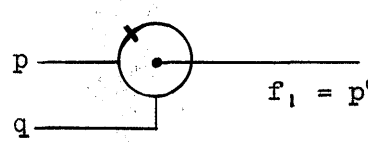


$f_7 = p^o + q^o$

the marks in the 1st, 2nd, and 3rd quadrants corresponding to the position of 1's in the matrix form for f_7 . A few other examples will help to illustrate the use of this symbol:

	f_1	q = 0	q = 1
p	0	1	0
	1	0	0

=

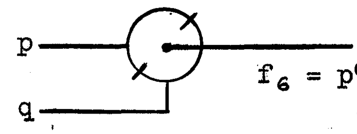


$f_1 = p^o \cdot q^o$

[8] J. D. Goodell: "The Foundations of Computing Machinery," Journal of Computing Systems, Vol. I, No. 1, June 1952.

f_6	q = 0	q = 1
p = 0	0	1
p = 1	1	0

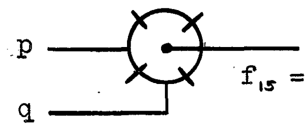
=



$f_6 = p^0 \cdot q + p \cdot q^0$

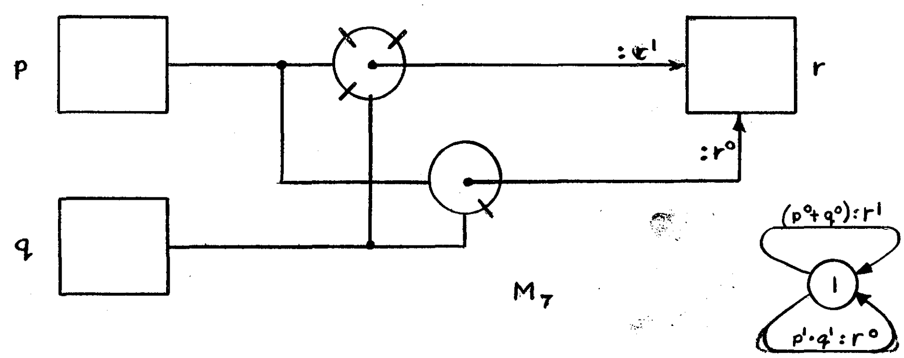
f_{15}	q = 0	q = 1
p = 0	1	1
p = 1	1	1

=

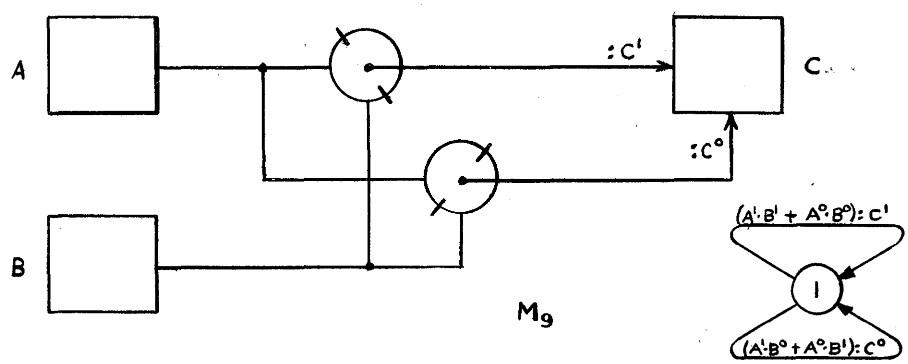


$f_{15} = 1$.

Using this symbol we can now construct any one-state, two-cell, binary machine in terms of the cells themselves and describe the action of the machine by means of the associated transition diagram. For example, the machine described by f_7 is:



and the machine described earlier (page 27), which corresponds to f_9 , is represented by:



The remaining 14 machines of this class can be constructed in a similar manner using the appropriate connective elements.

Universal Connective Elements

It is possible to put each of the expressions $f_0, f_1, \ldots f_{15}$ into a standard form which corresponds to that of f_1, namely the form:

$$(\ldots\ldots)^0 \cdot (\ldots\ldots)^0$$

$$(\ldots\ldots)^{\circ} \cdot (\ldots\ldots)^{\circ}$$

or its equivalent $\underline{f_1}$

$$\left[(\ldots\ldots) + (\ldots\ldots)\right]^{\circ}$$

where the empty parenthesis groups contain only combinations of the letters p and q which are also in the standard form. For example, $f_{14} = p + q$ can be written

$$f_{14} = \left\{\left[(p + q)^{\circ}\right] + \left[(p + q)^{\circ}\right]\right\}^{\circ}$$

Similarly, each of the sixteen expressions can be written in a form which corresponds to that of f_7

$$(\ldots\ldots)^{\circ} + (\ldots\ldots)^{\circ}$$

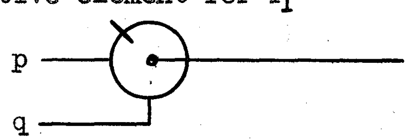
or its equivalent $\underline{f_7}$

$$\left[(\ldots\ldots) \cdot (\ldots\ldots)\right]^{\circ}$$

Again as an example, $f_{14} = p + q$ would be written

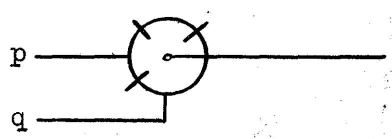
$$f_{14} = \left[(p)^{\circ} + (p)^{\circ}\right]^{\circ} + \left[(q)^{\circ} + (q)^{\circ}\right]^{\circ}$$

These expressions, f_1 and f_7, are known as the <u>Sheffer stroke functions</u>[9], and are sometimes written $f_1 = p \downarrow q$ and $f_7 = p \mid q$. The existence of these standard or universal forms means that any of the Boolean machines, M_0, M_1, . . . M_{15} can be synthesized using only the connective element for f_1



$f_1 = (p + q)^{\circ} = p^{\circ} \cdot q^{\circ}$

or the connective element for f_7



$f_7 = (p \cdot q)^{\circ} = p^{\circ} + q^{\circ}$

Accordingly, these elements might be called <u>universal connective elements</u>. To demonstrate this universality of these elements, we will now proceed to show how the sixteen machines M_0, M_1 . . . M_{15} are con-

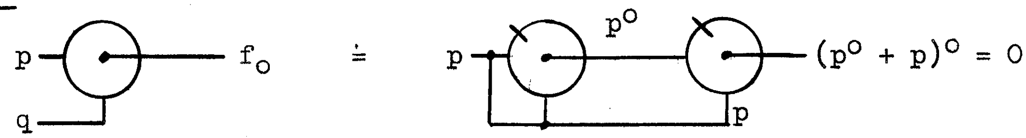
---

[9] H. M. Sheffer: "A Set of Five Independent Postulates for Boolean Algebra," Trans. Am. Math. Soc. XIV 1913

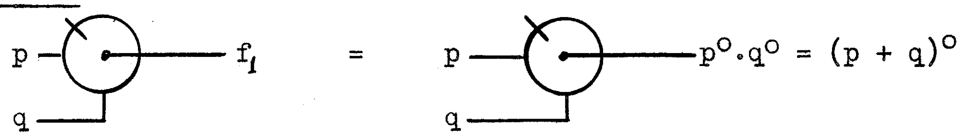
structured using only the element f_1 , and then we will show that any structure involving only f_1 can be replaced by another structure involving only f_7 . This demonstration is essentially the same as that presented by Goodell[10].

First, the sixteen forms are synthesized as follows:

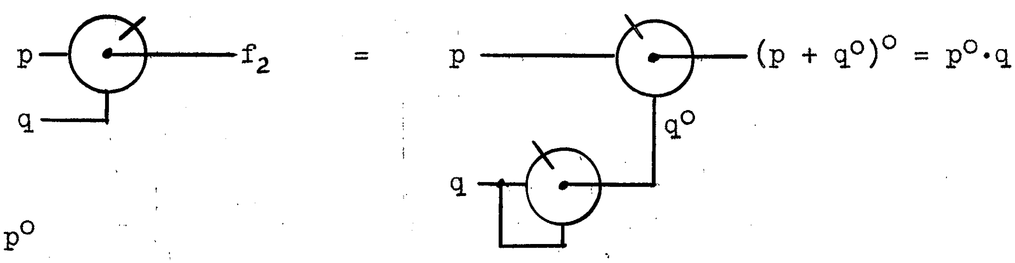
$f_0 = 0$



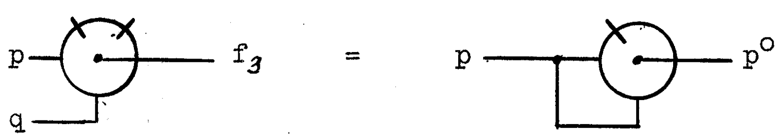
$f_1 = p^o \cdot q^o$



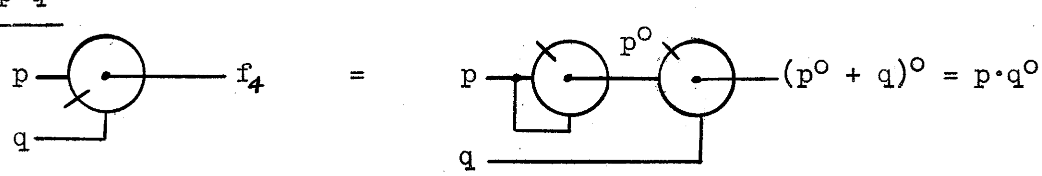
$f_2 = p^o \cdot q$



$f_3 = p^o$



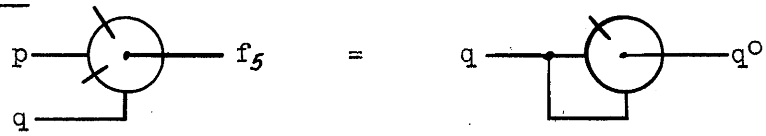
$f_4 = p \cdot q^o$



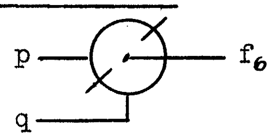
---

[10] J. D. Goodell: "The Foundations of Computing Machinery, Part II," The Journal of Computing Systems, Vol. 1, No. 2, January 1953.

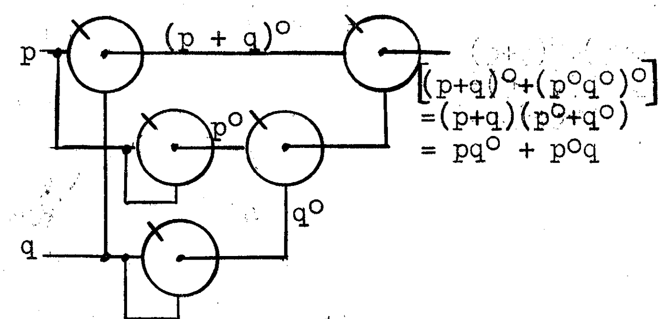
$f_5 = q^{\circ}$



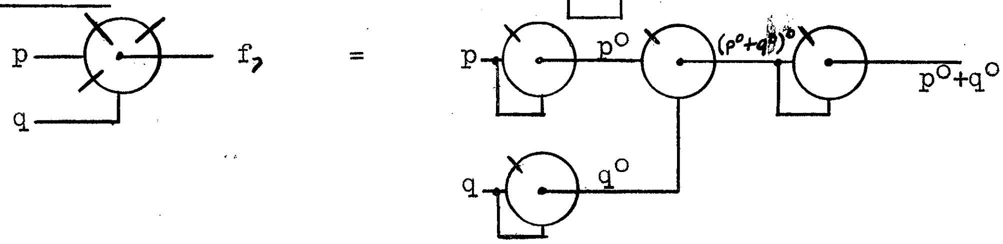
$f_6 = p \cdot q^{\circ} + p^{\circ} \cdot q$



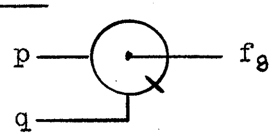
=



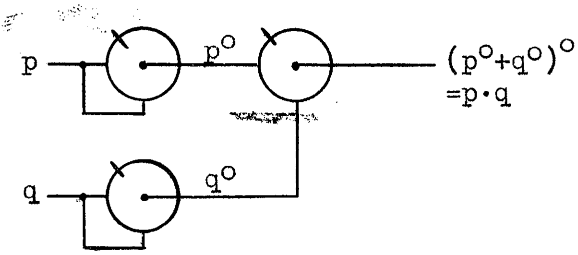
$f_7 = p^{\circ} + q^{\circ}$



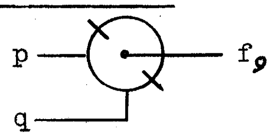
$f_8 = p \cdot q$



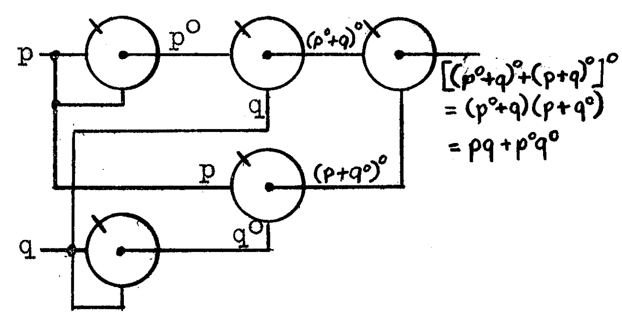
=



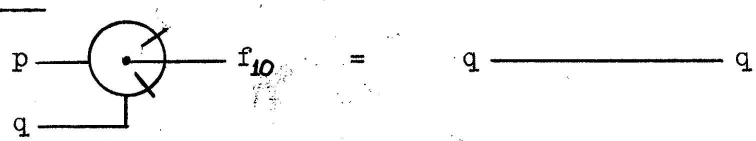
$f_9 = p \cdot q + p^{\circ} \cdot q^{\circ}$



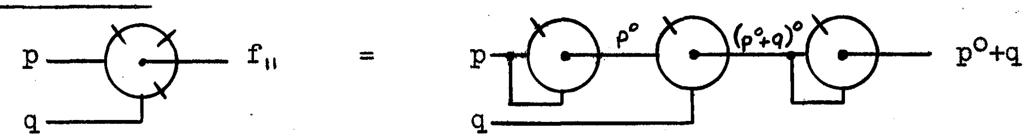
=



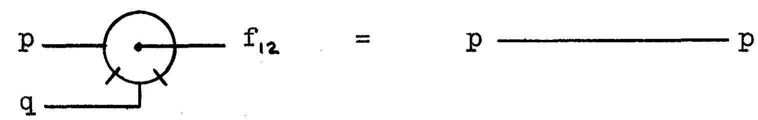
$f_{10} = q$



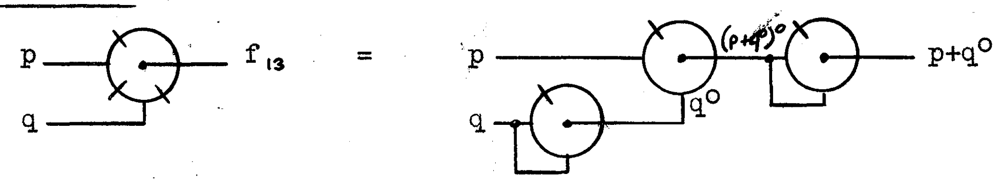
$f_{11} = p^o + q$



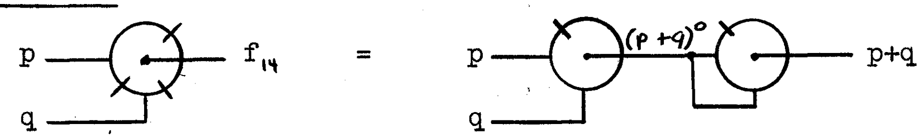
$f_{12} = p$



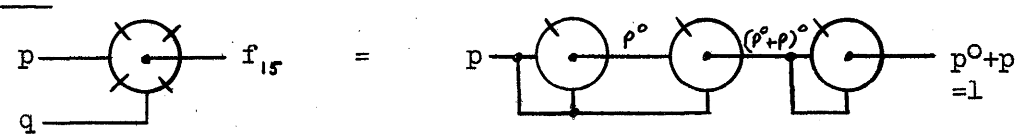
$f_{13} = p + q^o$



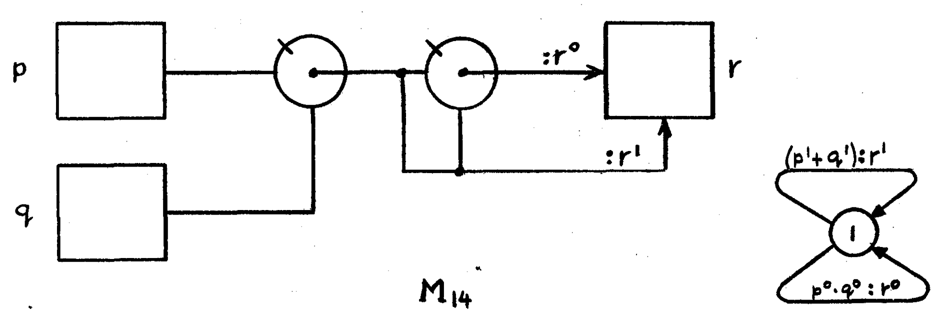
$f_{14} = p + q$



$f_{15} = 1$



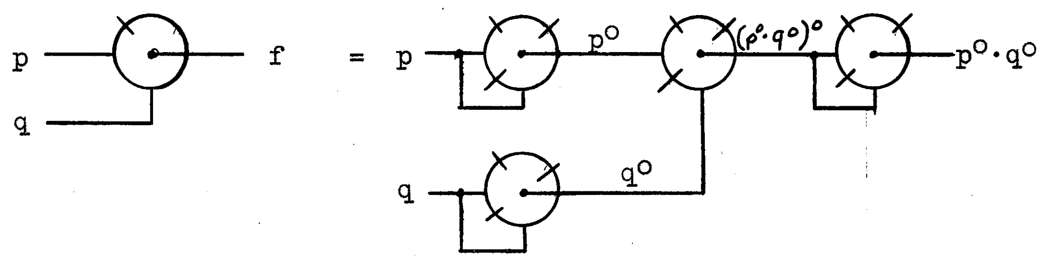
The corresponding 16 two-cell Boolean machines can be drawn directly using this list of f_1 -synthetic forms. For example, the machine M_{14} is:



M_{14}

Finally, any structure involving f can be replaced by a structure involving f_7 according to the following synthesis:

$f_1 = p^o \cdot q^o$

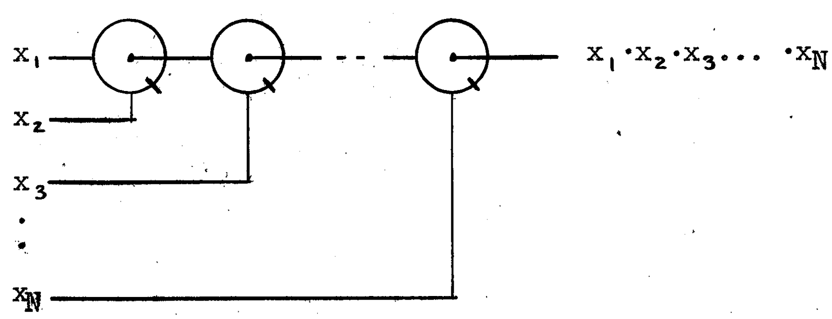


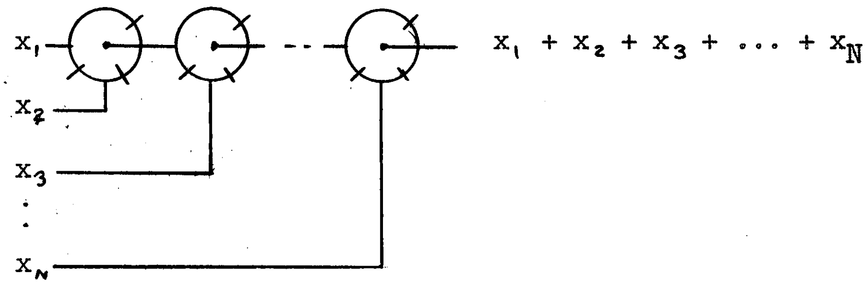
Hence, f_1 and f_7 connective elements are both universal for two-cell Boolean machines.

Although we will not make use of the universal properties of either element in subsequent work, it is of interest to note that the physical realizability of any Boolean machine can be established by showing that it is possible to construct a device which implements either f_1 or f_7.

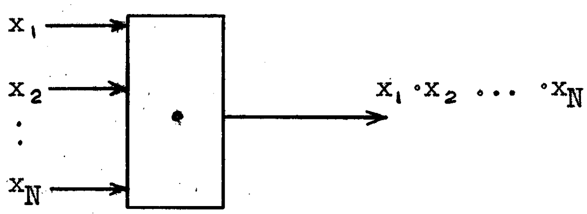
Boolean Machines Involving N Cells

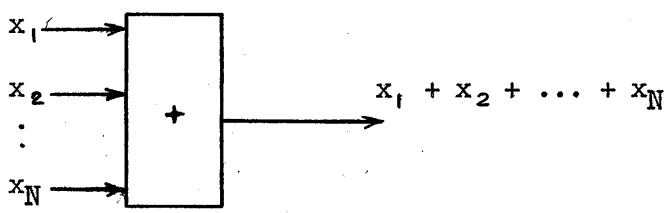
In discussing the synthesis of machines involving an arbitrary number of cells, it is convenient to define generalized connective elements for the "and" and "or" expressions. The circular symbol for the two-cell expressions can be generalized to more than two cells, but the drawing of matrices of more than two dimensions is difficult. By grouping two terms together at a time, the expressions for "and" and "or" involving N cells can, of course, be constructed using only the connective elements f_8 and f_{14}, respectively:



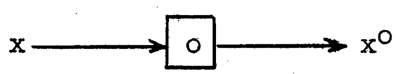


However, the connective elements we shall use in subsequent discussions of Boolean machines, regardless of how many cells are involved, are the following:

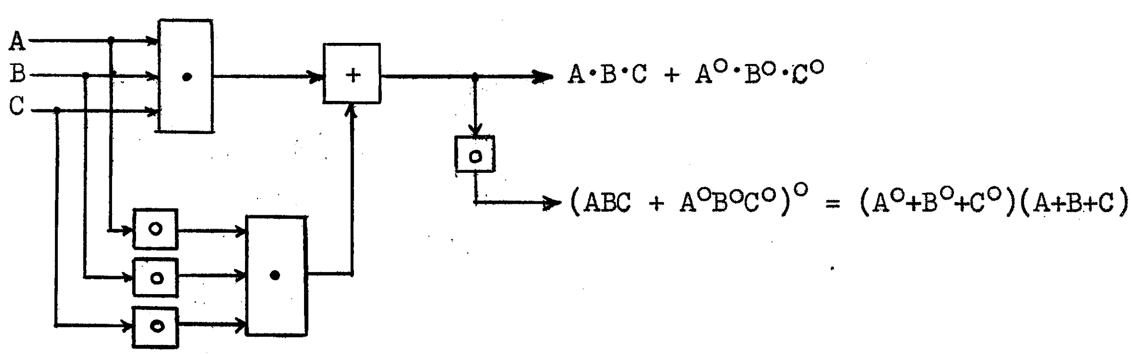




and, to complete the set of connective elements, a complementation element will be drawn in the following manner:



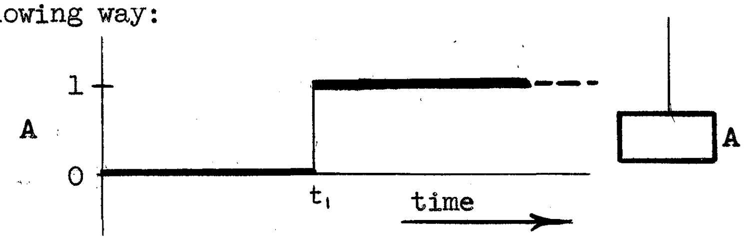
A composite example will illustrate the symbolism:



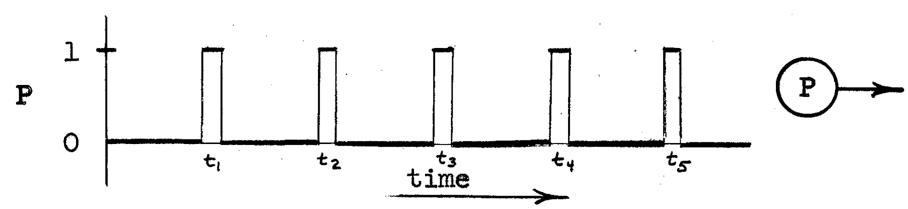
Extension of this symbolism to any Boolean expression follows from the definitions and from the fact that '·', '+' and '^{O}' form a complete set of connective elements.

The Time Element

It is necessary to make the distinction between printing a symbol and holding a symbol more precise. Printing occurs at definite times, namely those times at which transitions from state to state occur. On the other hand, symbol holding is a situation which persists in time until a new symbol is printed. For example, suppose cell A holds the symbol '0' and that during a machine transition occurring at time t_1 the symbol '1' is printed on A. We might represent the history of cell A in the following way:



To make the idea of transition timing more precise, we will introduce a new component cell into the machine. This new cell, which will be labeled P, will act as a clock for the rest of the machine. It will have the property of holding the symbol '1' when transitions are to occur and the symbol '0' otherwise:



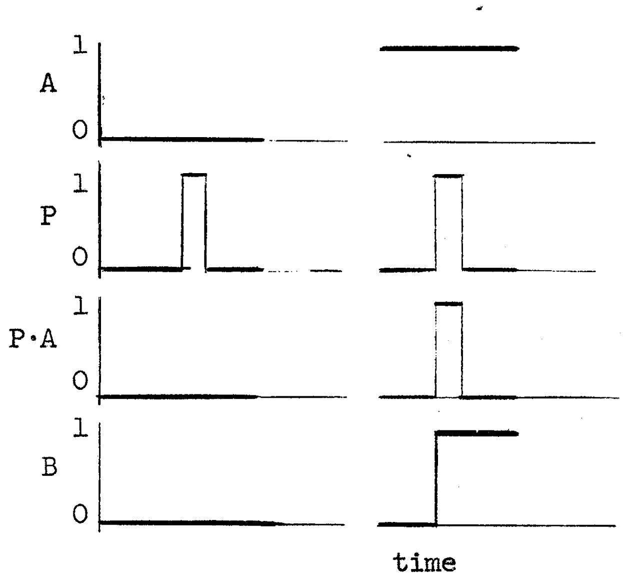
It is assumed that transitions and printing require a finite length of time and, consequently, P will hold '1' for a finite interval, although one which is not large compared to the interval between transitions. We will say that P acts as a source of pulses, and that a cell such as A is a source of symbol levels.

Printing can then be specified by forms of the following sort:

$$P^1 \cdot A^1 : B^1$$

$$P^1 \cdot (A^1 + B^1) : C^1$$

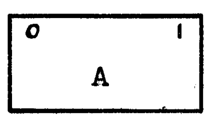
so that the act of printing is confined in time and coincides with the pulses occurring at t_1, t_2, etc. For example:



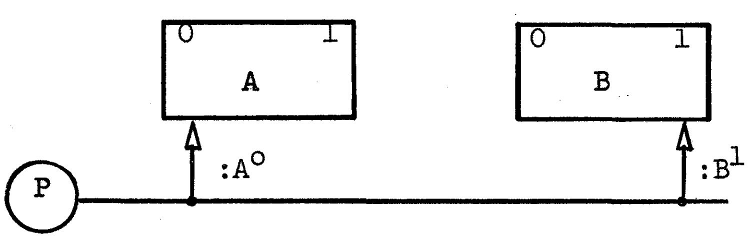
$P^1 \cdot A^1 : B^1$

In the case illustrated on the left, A holds '0' and no change of symbol occurs on B. In the case on the right, A holds '1' and a '1' is printed on B during the transition event.

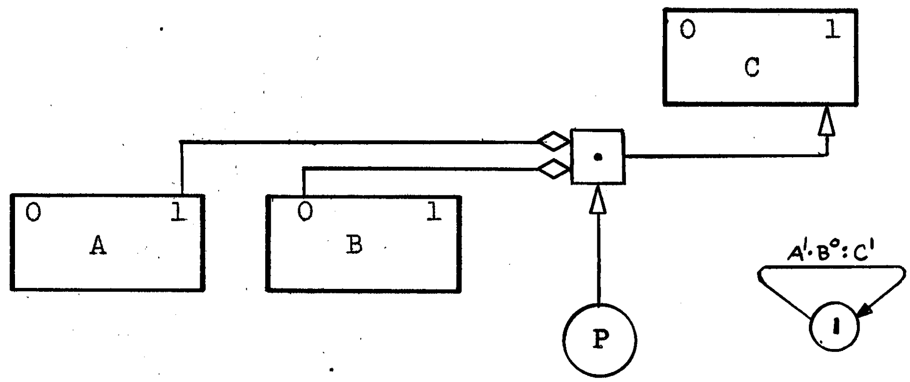
A refinement of the cell symbol will simplify drawing and emphasize the distinction between printing and holding functions. The cell label is written within the element representing the cell, and one half of the element is associated with the symbol '1' and the other with '0'.



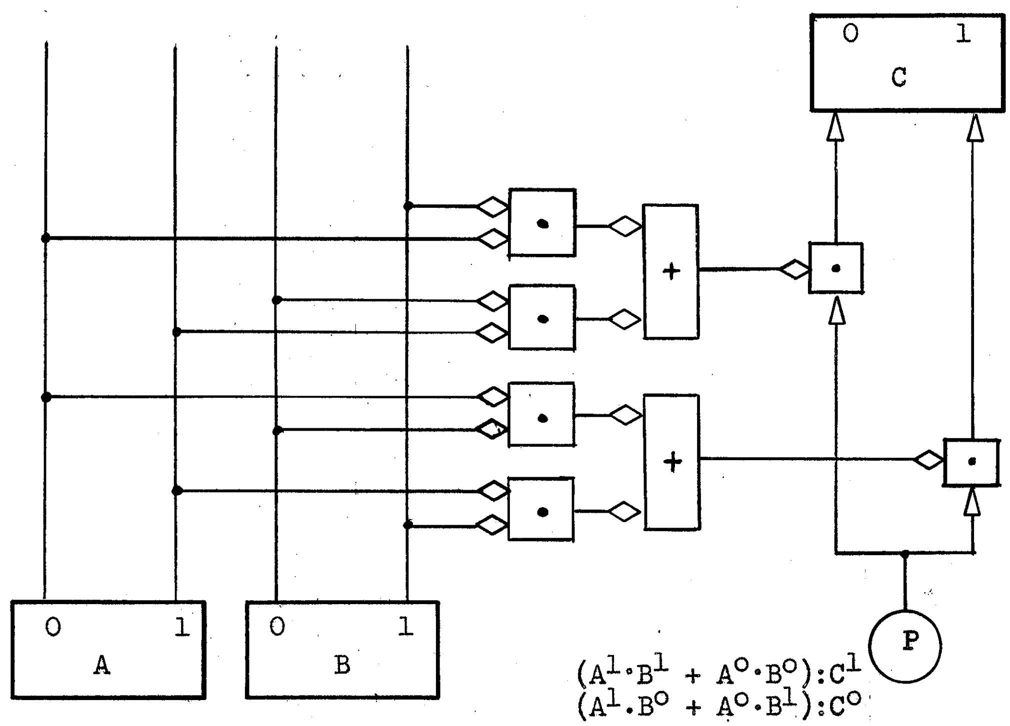
Then printing can be represented by one of the two forms:



A connective element will be associated with the appropriate sides of the participating cells. For example, the machine $P^1 \cdot A^1 \cdot B^0 : C^1$ is drawn:



The arrowhead will be used to denote symbol pulses and the diamond to denote symbol levels. One more example will illustrate these forms. The machine used to introduce Boolean algebra (page 27) is represented by the following drawing:

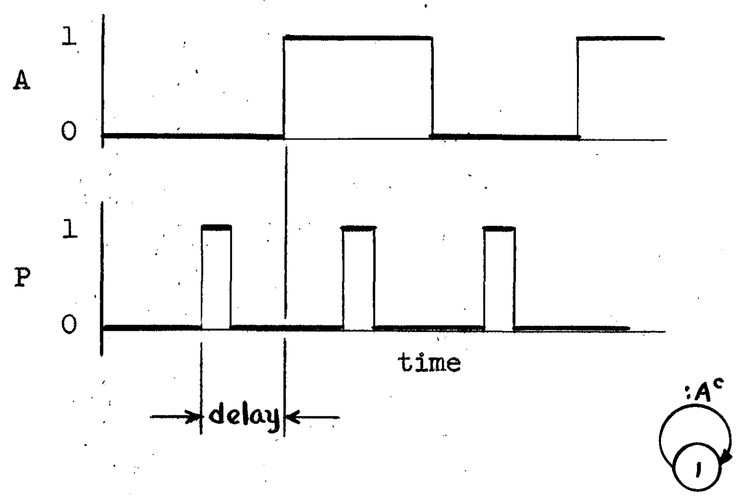


Complementation

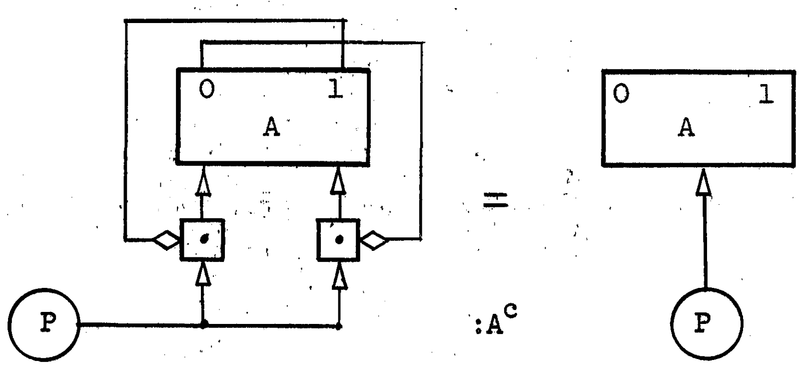
A case of special interest is that in which a cell participates in a symbol-printing operation on itself, e.g., the complementation operation:

$$\left.\begin{array}{l} P^1 \cdot A^1 : A^0 \\ P^1 \cdot A^0 : A^1 \end{array}\right\} : A^c$$

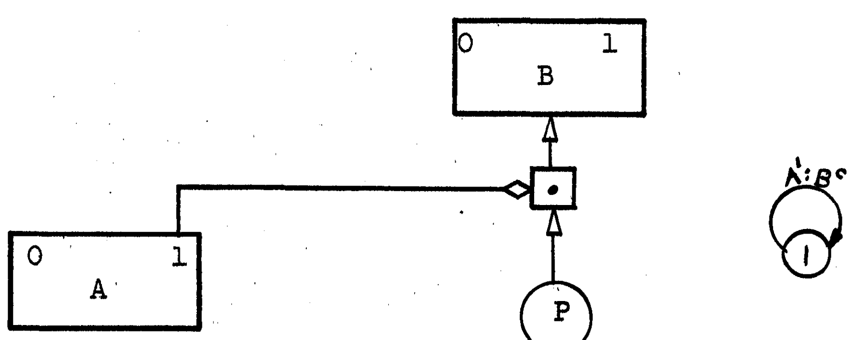
To avoid a logical indeterminacy it is necessary to redefine the printing operation so as to include a delay between the time at which the printing pulse occurs and the time at which the cell holds the printed symbol:



Complementation structures can then be simplified according to the following scheme:



where the pulse line drawn to the center of the cell element will indicate the complementation directly. As an example, the machine $A^1{:}B^c$ is



Memorandum 6M-3938, S5

Division 6 – Lincoln Laboratory
Massachusetts Institute of Technology
Lexington 73, Massachusetts

SUBJECT: THE LOGICAL STRUCTURE OF DIGITAL COMPUTERS
Synthesis of Boolean Machines (continued)

To: Class Registrants

From: W. A. Clark

Date: 9 January 1956

Abstracts to: J. C. Proctor, C. W. Farr

Abstract: Machines involving several states and cells can be synthesized according to a general procedure.

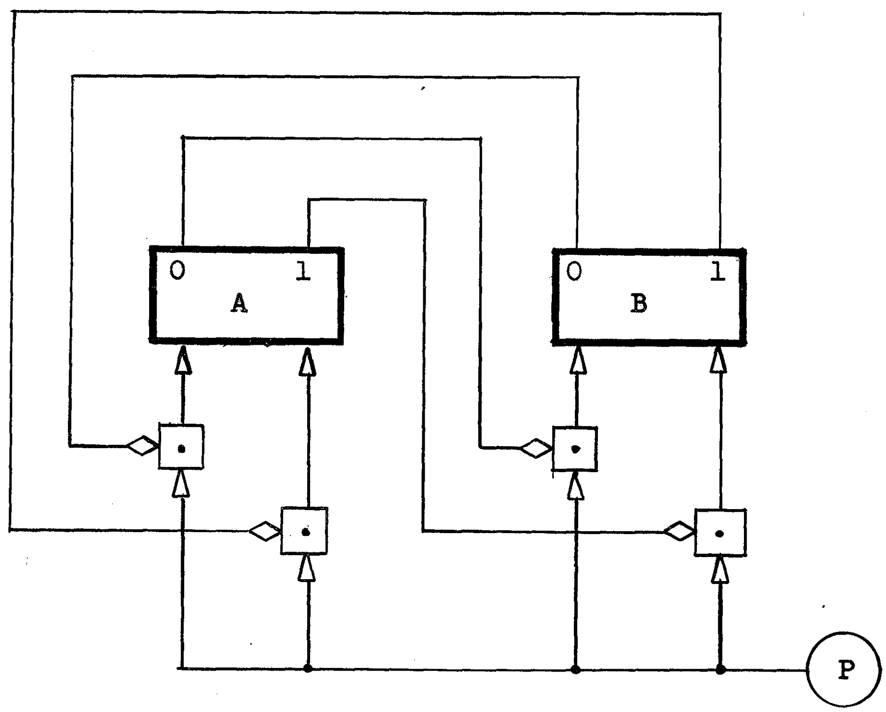
Complete configurations of the required machines, rather than internal states alone, are used in synthesizing the required structures. Examples of the synthesis of various counting structures are included.

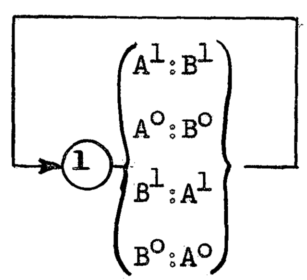
W. A. Clark

WAC/jhk

The research reported in this document was supported jointly by the Department of the Army, the Department of the Navy, and the Department of the Air Force under Air Force Contract No. AF 19(122)-458.

The existence of the printing delay also permits the construction of machines in which the symbols held in a set of cells are simultaneously interchanged. For two cells, A and B, an exchange of symbols is accomplished by the following structure:



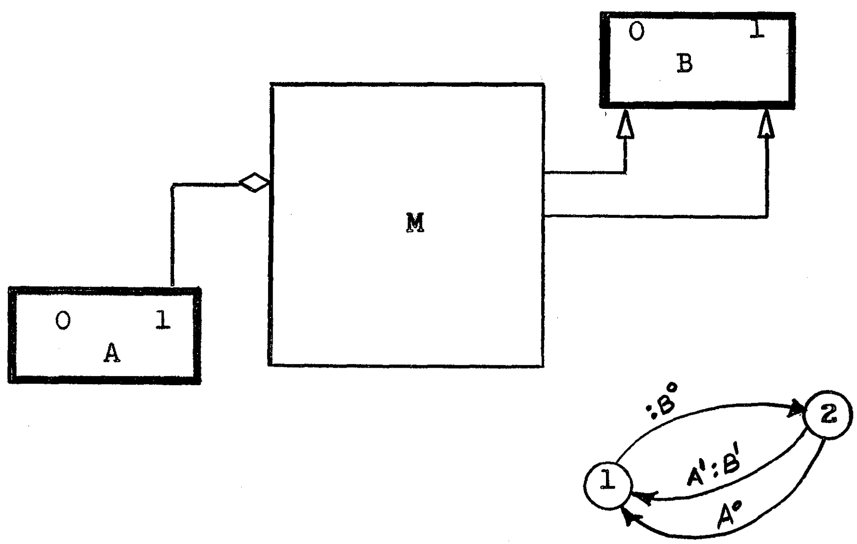


Interchange structures involving more cells may take many forms. A <u>shifting structure</u> is obtained if cell A's symbol is printed on cell B; B's on C; C's on D, etc. If the symbol in the last cell in this set of cells is printed on the first cell, a <u>cyclic interchange</u> is obtained and the set of cells takes the form of a closed ring.

Machines Involving Serial States

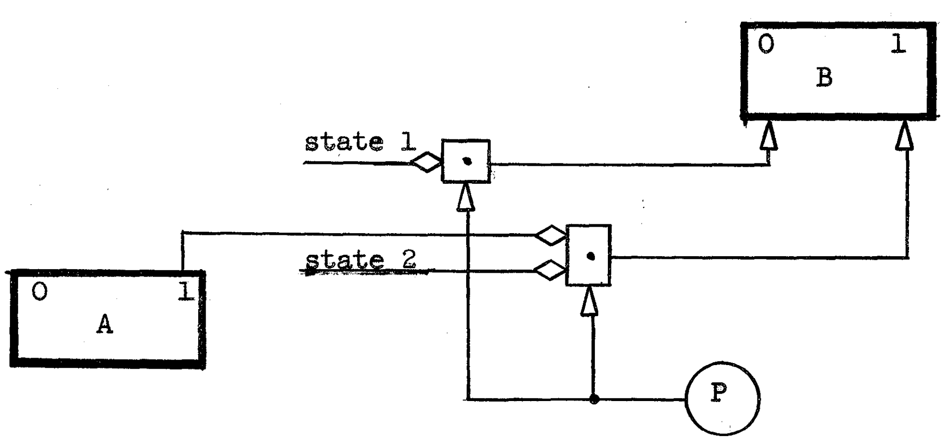
The Boolean machines discussed heretofore have all been one-state machines. We will now extend the synthesis procedure to machines involving more than one state.

Consider the following two-state machine, M:



M starts in state 1 and prints a '0' on B in the transition to state 2. From state 2, a '1' is printed on B if A holds '1' and M returns to state 1; no change in symbol occurs on B if A holds '0'. The net effect is that the symbol finally appearing in cell B is the same as the symbol in A; in other words, the symbol in A is transferred to B.

Evidently the printing requirements would be met if M supplied additional levels according to its state:

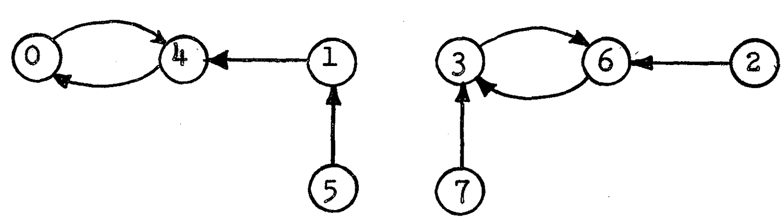


Complete Configurations

In one sense, cell K of the preceding section controls the action of the machine, M, in dealing with cells A and B. In another sense, cells K, A, and B are simply undifferentiated cells of a three-cell structure which is capable of assuming $2^3 = 8$ stable configurations and which jumps from one configuration to another in a manner which depends only on the current configuration itself. Thus, if the eight possible arrangements of symbols in K, A, and B are assigned configuration numbers according to the table:

K	A	B	#
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

then the action of the three-cell structure is described by the following transition diagram in which the nodes represent not states of K, but complete configurations:

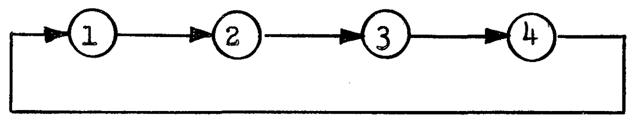


Note that the transitions in this kind of diagram are unconditional and that the final configuration depends only on the initial configuration. From the fact that M starts in state 1 (K^0), it follows that the complete structure starts in one of the four configurations, 0, 1, 2, or 3, depending on the initial arrangement of symbols on A and B. Consequently, the structure can never assume configurations 5 or 7; this can be seen from the configuration transition diagram.

It is worth mentioning at this point that in any physically realizable machine there always exists the chance that spurious transitions will occur. It might be said that one of the principal problems in engineering such a machine is that of making the probabilities of these spurious transitions suitably small.

A General Synthesis Procedure

Given the transition requirements for any structure in terms of complete configurations, it is possible to synthesize a suitable machine having the required transition diagram. As an example, suppose that it is desired to synthesize a machine which has the diagram:



The minimum number of two-symbol cells required is $\log_2 4 = 2$; these may be designated K_0 and K_1. Next, the state numbers 1, 2, 3, and 4 are coded in terms of the symbols in K_0 and K_1. The choice of code is <u>completely arbitrary</u>. The code we will choose for this example is the following:

K_1	K_0	#
0	0	1
0	1	2
1	1	3
1	0	4

We next write the Boolean expressions which will produce the required transitions. Consider first the situations which result in the <u>printing</u> of the symbol '1' on K_0. This occurs in the transition $1 \longrightarrow 2$ and again in the transition $2 \longrightarrow 3$. The configurations which <u>lead</u> to the printing of a '1' on K_0 are thus 1 and 2, and we may write:

$$P^1 \cdot (K_1^0 \cdot K_0^0 + K_1^0 \cdot K_0^1) : K_0^1$$

In a similar manner, the remaining printing expressions can be obtained. They are:

$$P^1 \cdot (K_1^1 \cdot K_0^1 + K_1^1 \cdot K_0^0) : K_0^0$$

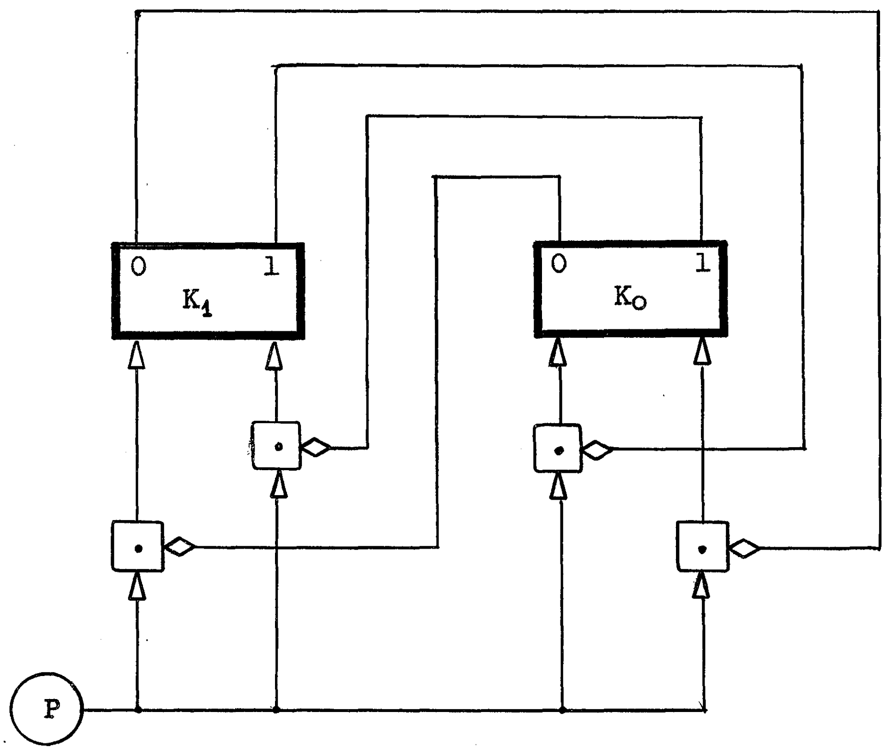
$$P^1 \cdot (K_1^0 \cdot K_0^1 + K_1^1 \cdot K_0^1) : K_1^1$$

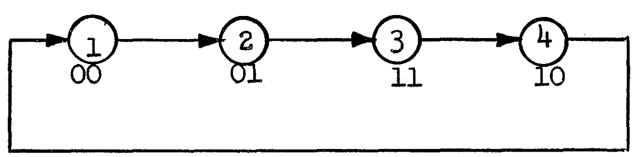
$$P^1 \cdot (K_1^0 \cdot K_0^0 + K_1^1 \cdot K_0^0) : K_1^0$$

These may be further reduced by factoring:

$P^1 \cdot K_1^0 \cdot (K_0^0 + K_0^1) : K_0^1$	$P^1 \cdot K_1^0 : K_0^1$
$P^1 \cdot K_1^1 \cdot (K_0^0 + K_0^1) : K_0^0$	$P^1 \cdot K_1^1 : K_0^0$
$P^1 \cdot K_0^1 \cdot (K_1^0 + K_1^1) : K_1^1$	$P^1 \cdot K_0^1 : K_1^1$
$P^1 \cdot K_0^0 \cdot (K_1^0 + K_1^1) : K_1^0$	$P^1 \cdot K_0^0 : K_1^0$

The required structure is thus:





The device, by definition, "counts" up to the number 4 and then starts over at 1.

A second example of the general synthesis procedure again deals with cyclic counting and with a coding scheme of special interest. First, a machine which counts cyclically through the numbers 0 through 7 will be synthesized, and then the results of the synthesis will be generalized to larger rings.

The eight configurations require three cells: K_2, K_1, and K_0. The particular code chosen will be a <u>binary numerical code</u> in which the configuration number is a sum of <u>powers of 2</u>, 2^i being the contribution to the sum of K_i holds a '1':

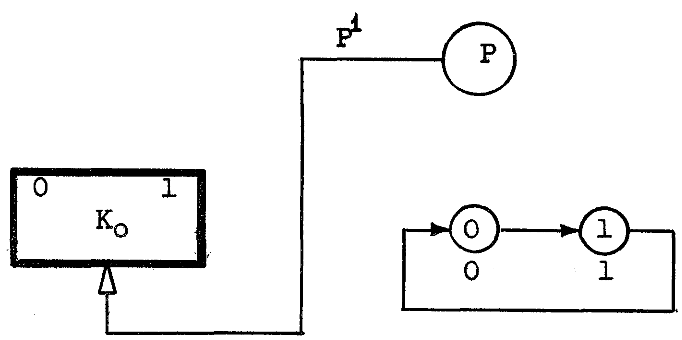
K_2	K_1	K_0	#
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Again, any consistent, complete coding scheme could have been chosen. The particular advantage of a binary numerical code is that it will yield a simple, iterative structure for the synthesized machine.

Instead of proceding as before, we will note instead the configurations which lead to changes of symbols[11].

We note first that K_0 is complemented during all transitions. We have immediately:

$$P^1 : K_0^c$$



K_1 is complemented during transitions from the configurations 1, 3, 5, and 7:

$$P^1 \cdot (K_2^0 K_1^0 K_0^1 + K_2^0 K_1^1 K_0^1 + K_2^1 K_1^0 K_0^1 + K_2^1 K_1^1 K_0^1) : K_1^c$$

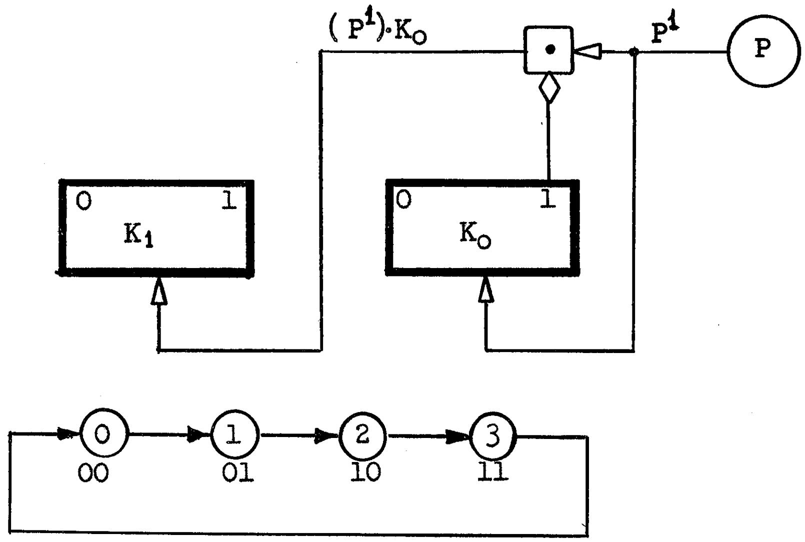
which factors to

$$P^1 \cdot K_0^1 \cdot (K_2^0 K_1^0 + K_2^0 K_1^1 + K_2^1 K_1^0 + K_2^1 K_1^1) : K_1^c$$

[11] Jeffrey, R. C., Reed, I. S.: "The Use of Boolean Algebra in Computer Design." MIT Digital Computer Lab, Engineering Note E-458-2, 15 April 1952.

and, since the expression within the parenthesis is the same as 1,

$$P^1 \cdot K_0^1 : K_1^c$$



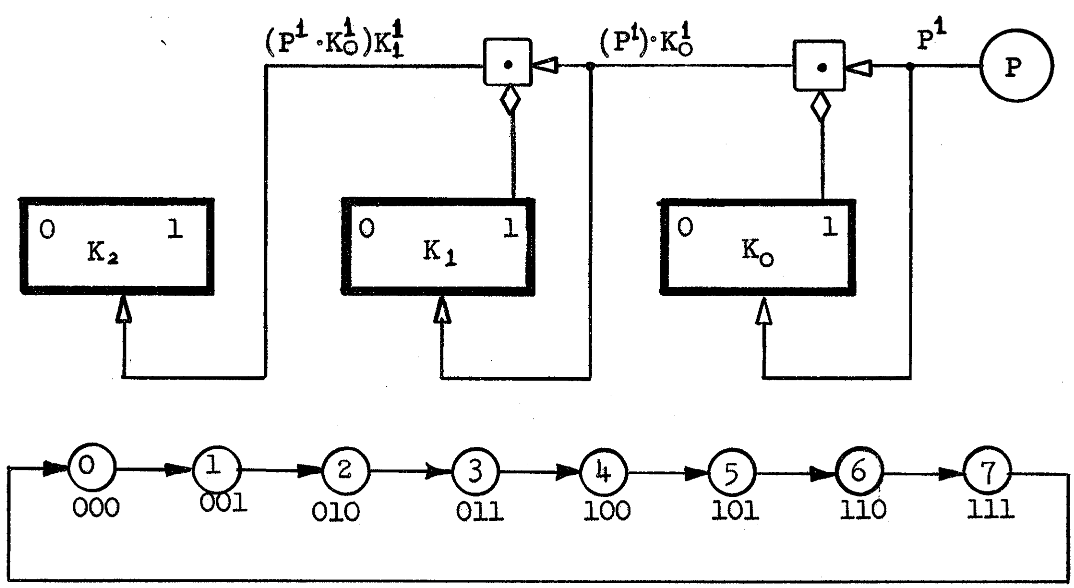
Finally, K_2 is complemented during transitions from configurations 3 and 7:

$$P^1 \cdot (K_2^0 K_1^1 K_0^1 + K_2^1 K_1^1 K_0^1) : K_2^c$$

or

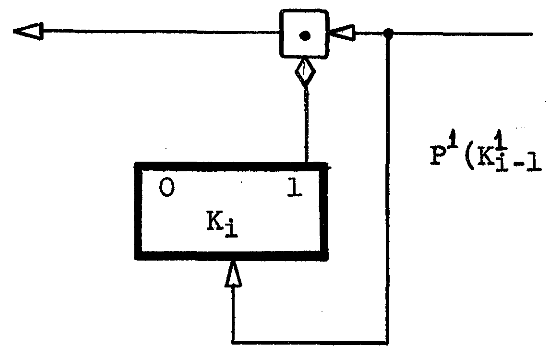
$$P^1 \cdot K_1^1 \cdot K_0^1 (K_2^0 + K_2^1) : K_2^c$$

$$P^1 \cdot K_1^1 \cdot K_0^1 \; : K_2^c$$



The generalization to structures involving more stages follows immediately:

The i^{th} stage is:



$P^1(K^1_{i-1} \cdot K^1_{i-2} \cdot \ldots \cdot K^1_0):K^c_i$

Iterated structures of this kind are desirable from the standpoint of simplicity of physical construction.

PROBLEMS

5.1 Design a one-state machine which will accomplish a cyclic interchange of symbols on the three cells, A, B, and C

$A^1:B^1$ $B^1:C^1$ $C^1:A^1$

$A^0:B^0$ $B^0:C^0$ $C^0:A^0$

using only the complement printing operation.

5.2 Design a machine to perform a cyclic interchange on A, B, and C as in 5.1, but with cells which do not include a printing delay. How many state are required?

5.3 Construct a two-cell, four-state machine which has the following transition diagram: