Notre Dame Journal of Formal Logic
Volume III, Number 3, July 1962

PROGRAMMING THE FUNCTIONS OF FORMAL LOGIC

S. SUMMERSBEE and A. WALTERS

From time to time automatic defices are suggested which will simulate the operations which can be carried out on the truth-tables of formal logic. Specifically, if a formula $F(X_1 \ldots . X_n)$ constructed from the propositional variables $X_1 \ldots . X_n$ and certain logical connectives, is set into the machine it will calculate the truth-value of $F(X_1 \ldots . X_n)$ from the truth-values $x_1 \ldots . x_n$ ($x_i = T$ or F, $1 \leqq i \leq n$) of $X_1 \ldots . X_n$. Probably the best known of such devices is that of W. S. Jevons [1], while one of the most recent is designed to deal with many-valued logic [2].

There is no intrinsic reason for choosing the symbols "T" and "F" to represent the truth-values of a proposition, the symbols "1" and "0" will serve the same purpose. In such an event the truth-value of a formula $F(X_1 \ldots . X_n)$, determined by the truth-values $x_1 \ldots . x_n$ of $X_1 \ldots . X_n$, can be written in the form

$$m = \sum_{j=1}^{j=n} a_i \, 2^{j-1}$$

where $a_i = 0$ or $a_i = 1$. For example, the truth-table for Dpq is

TABLE I

p	q	Dpq
1	1	1
1	0	1
0	1	1
0	0	0

and the final column of the table can be regarded as having either the value 1110 (= 14) or the value 0111 (= 7) depending on the convention adopted. Alternatively, the values 0, 3, 5, 7 for m will give the truth-value of Dpq for given values of p and q; for example 5 would be rewritten as 101 and hence for $p = 1$ and $q = 0$, $Dpq = 1$.

Received September 17, 1961

Since many digital computers work in the binary scale of notation i.e., each integer is represented by

$$\sum a_i \, 2^{n-1}$$

where $a_i = 1$ or $a_i = 0$, it appears to be a simple step to interpret the operations of a digital computer as operations carried out in the propositional calculus. In addition, there are in most digital computers built in functions which correspond to the functions of the propositional calculus. For example, a machine having a function "logical and" would form what is essentially the truth-table entry for conjunction, given the truth-values of two propositions. Needless-to-say when such functions are built into digital computers their primary purpose is not that of compiling truth-table entries for the propositional calculus. The function "logical and", for example, is widely used in digital programming as an "eraser". Since this function will be used below in this form we give a description of this method of use and refer to it hereafter as "collate".

Suppose that we have under consideration a digital computer with registers which hold up to thirty-two binary digits and of one number in particular, say

$$100\ldots\ldots\ldots\ldots 01101110111 \qquad \text{(i)}$$

the five digits on the right hand end are to be removed and operated on in another part of the machine, then by having a number

$$000\ldots\ldots\ldots\ldots 00000011111 \qquad \text{(ii)}$$

in the machine (in a suitable place) and calling the operation "collate" the result will be

$$000\ldots\ldots\ldots\ldots 00010111$$

i.e. the five digits at the right-hand end of (i) are preserved (the rest are "erased")—they are in one-to-one correspondence with the five digits at the right-hand end of (ii). Hence although a logical operation has been carried out, its primary purpose has not been a "truth-value" computation. The fact that such operations are now standard functions for digital computers suggests that these machines might be programmed so that they function as if they were logical machines. Further more, the possible uses of a logical machine may be of some practical value. For example, in [2] Rose suggests the application of a many-valued logical machine to the economic use of factory scheduling; while the machine suggested by McCallum and Smith might be adapted to the solution of problems in circuit simplification.

In order to give our investigation a concrete form we choose first of all a digital computer with the following characteristics:

It has a "word-length" of thirty-two binary digits,
a single address system,
it has built in the logical function "conjunction" which when it is used as a truth-function operator will be called "conjunction", otherwise "collate",

it has built in the logical function "disjunction", which when it is used as a truth-function operator will be called "disjunction", otherwise "or",
it has a store of about one thousand words,
its operating speed is about six micro-seconds cycle time,
it "reads" its information from a punched paper tape, and prints its answers on a teleprinter, and can simultaneously punch paper tape.

(this describes approximately a machine built in the laboratory).

To give the investigation some direction we choose first of all the problem of simplifying truth-functions as described by Quine.

In principle the machine can deal with an unlimited number of propositional variables, but as a practical matter some limit must be placed on the number of variables operated on.

Since the word-length of the machine is thirty-two, it is natural to choose either thirty-two to be the maximum number of variables, or five to be the maximum number of variables (since $2^5 = 32$). First consider the former case, and suppose the variables to be p_1, p_2, $p_3 \ldots p_{32}$, then if a truth-table were being drawn up for these variables it would appear as

TABLE II

p_1	p_2	$p_3 \ldots p_{31}$	p_{32}
0	0	0 0	0
0	0	0 0	1
0	0	0 1	0
0	0	0 1	1
	⋮	⋮	⋮
1	1	1 1	1

When the table is complete columns can be added in the usual way to give the truth-values of functions of up to thirty-two variables. For example, a column could be added giving the truth-values of the conjunction Kp_1p_2.

The machine, as stated above, is equipped to form the truth-values of the conjunction of two propositional variables, (or to collate two binary numbers) but in so doing it operates on the entire length of the register holding the truth-values. Hence under the scheme corresponding to Table I, 2^{32} registers would be required to hold the whole of the table. This far exceeds the capacity of the machine, but this difficulty can be avoided by using a single register to hold one line of the table at a time. For example, the first line of the table corresponds to an empty register, the second line to binary one, the third line to binary two, and so on; the last line corresponds to binary $(2^{32} - 1)$. Suppose that we wish to find the truth-value of a conjunction of two of the variables say p_1 and p_2, and we consider a line of the table—which will be in the register of the machine at one time—for example

TABLE III

p_1	p_2	$p_3 \cdots p_{31}$	p_{32}
.			.
.			.
.			.
.			.
1	*1*	*0*	*0*

To find the truth-value of the conjunction of p_1 and p_2 the entries in the truth-table under p_1 and p_2 are compared, and in the example given above Kp_1p_2 would have the value *1*. However, the machine can only collate different registers and consequently it is not possible to form the value of the conjunction of p_1 and p_2 directly. First, the value under p_1 is removed from the row of the table and put elsewhere (in another register). Second, the value under p_2 is removed from the row of the table and put elsewhere. The value of the conjunction of p_1 and p_2 for their particular values entered in the line of the table under consideration, is then found by collating the two registers chosen to hold the values of p_1 and p_2. Although this appears to be a long-winded procedure, it is systematic and it is not difficult to programme the machine to carry out the operations. The steps, once programmed, can be carried out for each line of the table, and the results preserved in another register i.e. the first digit in the register corresponds to the first entry in the column of the table for Kp_1p_2. The second digit corresponds to the entry for the second line of the table, and so on.

The operation can be broken down into two groups: first those which generate the line of the table under consideration, and second, those which operate on certain entries in the line to give the value of the function which the column of the table represents. In the event of the maximum possible number of variables being used i.e. *32*, part of the total calculation will be taken up generating successive lines of the table.

It was mentioned above that the cycle time of the machine under consideration was six micro-seconds, and allowing time for addition to take place serially the total time to generate one line of the table would be of the order of eighty micro-seconds, but the total time to generate every line of the table would be of the order of forty days. For this reason we attack the problem in another way.

Because there are thirty-two digits to each machine word we limit the number of variables (in the first instant) to five, since $2^5 = 32$. And rearrange the truth-table.

TABLE IV

p_1	*10101010101010101010101010101010*
p_2	*11001100110011001100110011001100*
p_3	*11110000111100001111000011110000*
p_4	*11111111000000001111111100000000*
p_5	*11111111111111110000000000000000*

In this arrangement each register represents one column of the truth-table hence to operate on five variables we need five registers. If there were a greater number of variables then some other arrangement would be needed since a single register is not long enough to hold the required number of entries. If two registers were placed (figuratively) end to end their doubled capacity would allow six propositional variables to be operated upon, and so on. If the number of variables were increased to thirty-two, there would have to be 2^{32} rows in the truth-table, and consequently about $1 \cdot 3 \times 10^8$ registers. No doubt ingenious programming could reduce this number, but at a corresponding increase in the time taken to complete a programme.

We shall say no more here about the size of machine required to handle any but few propositional variables, nor the time taken to complete a programme. Instead we now treat in more detail Quine's method for simplifying truth-functions, and confine ourselves to functions of five variables or less.

Quine's method for finding the simplest equivalent of any function of propositional variables falls fairly naturally into three fairly self-contained parts:

(a) given some function $Q(p_1 \ldots p_n)$ of the propositional variables $p_1 \ldots p_n$, form its perfect disjunctive normal form, [5]

(b) from the perfect disjunctive normal form derive a list of the "prime implicants" of $Q(p_1 \ldots p_n)$,

(c) from (a) and (b) used simultaneously derive the simplest equivalent of $Q(p_1 \ldots p_n)$.

As stated above we treat this problem with the variables of Q limited to $1 < n \leq 5$. And suppose that we wish to find the disjunctive normal form of the function $DKKp_1p_2p_3Kp_3Np_4$, there are here four variables: $p_1p_2p_3p_4$. The machine can be programmed to generate the truth-table simply by setting the number 4 into it i.e. in four consecutive registers 1, 2, 3 and 4, the patterns

TABLE V

p_1	1010101010101010
p_2	1100110011001100
p_3	1111000011110000
p_4	1111111100000000

are generated. By setting the function $DKKp_1p_2p_3Kp_3Np_4$ into the machine the values for this function can be computed. For example, first the collate (machine) function is used between registers 1 and 2 to form Kp_1p_2, this in turn is collated with register 3 to form $KKp_1p_2p_3$; the "logical or" is then used between $KKp_1p_2p_3$ and Kp_3Np_4 to give the final entry to the table.

There are here perhaps two points worth noting:

(a) when any digital computer is used for numerical work, the problem to be solved is treated in two parts, first an appropriate numerical method

must be found to provide a solution to the problem and second, this method must be programmed for the machine to accept it. We are faced with a similar situation when applying the machine to the functions of logic viz., will the machine accept the functions as they are, or must they be changed to some standard form? A machine which is only equipped, for example with operations corresponding to disjunction and conjunction would have to have all problems transformed into some variation of these functions before programming could begin. For example, the function $CNKp_1p_2Kp_3KNp_3p_4$ would have to be transformed to $DKKp_1p_2p_3Kp_3Np_4$ before being programmed. We assume from now on that this has been done and consequently each function treated will consist of disjunctions of conjunctions.

(b) When the final values of the function have been calculated, each entry in the table may be checked to see if it is a "*1*", if it is then the machine can be made to print, say, the word "TRUE" against the function. If each entry is a "*0*" the machine can print "FALSE" against the function. Hence any given function $Q(p_1p_2p_3p_4p_5)$ of five, or less, propositional variables can be tested by the machine for universal validity (tautology) or universal invalidity. Further more, by making the machine select those entries in the function column of the table, which have the value "*1*", an assignment of truth-values to p_1, p_2, p_3, p_4, p_5 can be printed out for which $Q(p_1p_2p_3p_4p_5)$ takes the value *1*. The machine then corresponds to that suggested by McCallum and Smith in [3].

Let us suppose that the machine has made the necessary calculations and has filled an extra register S with the values of $DKKp_1p_2p_3Kp_3Np_4$ i.e. the registers correspond with

TABLE VI

p_1	*1010101010101010*
p_2	*1100110011001100*
p_3	*1111000011110000*
p_4	*1111111100000000*
$DKKp_1p_2p_3Kp_3Np_4$	*1100010001000100*

The machine may now examine each digit in register five, in turn. If the digit examined is a "*0*" the machine does nothing but passes on to the next digit.

When the machine finds a "*1*" - as it will do for the 1st, 2nd, 6th, 10th and 14th places in the table above - it then prints the corresponding variables if these are "*1s*", but preafixes the variable by an *N* if there is a *0* in the column, or rather in the same digit place in the registers *1*, *2*, *3* and *4*. For example, the machine would find a "*1*" in the first digit place of register five, it then examines the first digit places of registers one, two, three and four in turn. Each of these digit places contains a "*1*" and hence the machine would print "$p_1p_2p_3p_4$". It then examines the second digit place of register five and again finds a "*1*" so it examines the second digit places of registers one, two, three and four. In register one there is a "*0*" in the

second digit place, the digits in the other registers being "*1*s", the machine therefore prints "$p_1p_2p_3Np_4$", and so on. When each digit place of the fifth register has been examined what the machine has printed out is each clause of the disjunctive normal form of $DKKp_1p_2p_3Kp_3Np_4$. Hence given $Q(p_1p_2p_3p_4p_5)$ the machine will print out the disjunctive normal form of $Q(p_1p_2p_3p_4p_5)$.

However, we are not particularly interested at this stage in finding the disjunctive normal form of any function of $Q(p_1p_2p_3p_4p_5)$ of propositional variables. Instead of printing taking place the machine is made to preserve only that part of the table which corresponds to a "*1*" in the entry for $DKKp_1p_2p_3Kp_3Np_4$, i.e. table VI is abbreviated to

TABLE VII

p_1	*10000*
p_2	*11111*
p_3	*11010*
p_4	*11100*
$DKKp_1p_2p_3Kp_3Np_4$	*11111*

It will be recalled from above that each row of entries in the table represents a binary number in a register. For convenience the table may be transposed so that rows become columns and columns rows, viz., there will be four digits only in each of four new registers—and we note that at this stage there cannot be more than five digits in these registers—in the first register there will be the digits in the first places of each of the four registers (the first entry for p_4 is now in the fourth place of the first register, the entry for p_3 in the third place of that register etc.)

Each pair of rows in the new table, or each corresponding pair of registers, is compared and if they differ only in a single corresponding place, that entry with a "*1*" in it is deleted. For example, from Table VII it may be seen that columns one and two differ only in the first place, the entry is then treated as column two with column one deleted. The process is repeated until all that is left is entries which differ in more than one place (these entries correspond to Quine's "prime implicates").

These resulting entries are compared in turn with the entries corresponding to those of table VII and the common part printed as variables; for example, if the common part were *0* and *1* in the first and second places "Np_1p_2" would be printed. When a printing takes place a *1* is recorded in a register to serve as a counter. If a one is recorded, it is recorded in the register in the digit place corresponding to the column of the table. As each "prime implicant" is compared with the table when a common part is found, before printing takes place the count register is examined, if it already contains a "*1*" in that place, the machine does not print the common part but moves on to the next part. This process is repeated until the count register is full. The printed results then give a "simplest" equivalent to the function supplied to the machine in the first place. It may be noticed however, that the machine does not print *all* the simplest equivalents but only the first one it finds.

As mentioned above Rose has shown how a many-valued logical machine may be used for the solution of certain practical problems. The question naturally arises as to whether a digital machine may not be used to solve the same problems. We can see very simply what would be involved in attempting to solve all but the simplest variations of such problems on a digital computer, from the following considerations. Above we have discussed a two-valued logic and have found that except for extremely high speed machines the number of variables must be limited to about five although in principle a digital machine could be programmed to solve problems with any number of variables in a two-valued logic. In [2] Rose cites an example of a 4-valued logic with a table for disjunction.

TABLE VIII

DXY	*1*	*2*	*3*	*4*	*Y*
1	*1*	*1*	*1*	*1*	
2	*1*	*2*	*2*	*2*	
3	*1*	*2*	*3*	*3*	
4	*1*	*2*	*3*	*4*	
X					

It is clear that by representing the variables X and Y by two digits, the above table would appear as

TABLE IX

DXY	*00*	*01*	*10*	*11*	*Y*
00	00	00	00	00	
01	*00*	*01*	*01*	*01*	
10	*00*	*01*	*10*	*10*	
11	*00*	*01*	*10*	*11*	
X					

and could be stored in the machine in this form. Hence given values of X and Y the machine would select the appropriate value of DXY from the stored table.

A sub-routine for the machine could be written for each of the functions of the m-valued calculus treated, and the machine simply makes use of the appropriate sub-routine instead of supplying values directly from built-in functions. Once again the question arises as to the number of variables which can be reasonably allowed if the machine is to complete its calculations in "real" time. In the problems cited by Rose a practical case would demand a logic of about twenty-five values and about seven variables.* The

*What would be a "practical" case depends largely on personal interpretation. We have, in calculating the mentioned values, allowed for five lectures per day five days per week, and each lecturer to teach for twenty hours per week.

machine time required for solving such problems is many times greater than that required for solving corresponding problems in a two-valued logic. Similar considerations apply to the solution of problems in modal logic. [6]

REFERENCES

[1] W. S. Jevons: *On the Mechanical Performance of Logical Inference,* 1870.

[2] A. Rose: Many-valued Logical Machines, *Proc. of Camb. Phil. Soc.,* Volume 54, pp. 307-321, 1958.

[3] D. M. McCallum and J. B. Smith: Mechanised Reasoning, *Electronic Engineering,* April 1951.

[4] W. V. Quine: The Problem of Simplifying Truth Functions. *American Mathematical Monthly.* Vol. 59, No. 8, 1952.

[5] D. Hilbert and W. Ackerman: *Grundzüge der Theoretischen Logic,* Springer, Berlin 1931.

[6] G. H. Von Wright: *An Essay in Modal Logic,* N-Holland Publishing Co., Amsterdam. 1951.

The Computing Laboratory,
Letchworth College of Technology,
Letchworth, Hertfordshire, Gt. Britain